Reasoning on the response of logical signaling networks with answer set programming

Santiago Videla

ANNÉE 2014

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

En Cotutelle Internationale avec
Universität Potsdam, Allemagne

pour le grade de
DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention Informatique
École doctorale Matisse
présentée par

Santiago VIDELA

préparée à l'unité de recherche: UMR 6074 IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
Composante Universitaire: IFSIC

Reasoning on the response of logical signaling networks with Answer Set Programming

Thèse soutenue à Rennes
le 07/07/2014
devant le jury composé de:

Gregory BATT
Chargé de Recherche, Inria Rocquencourt / rapporteur
Mireille DUCASSE
Professeure, INSA de Rennes / examinatrice
Christine FROIDEVAUX
Professeure, Université Paris Sud / rapporteur
Torsten SCHAUB
Professeur, Université de Potsdam / co-directeur de thèse
Joachim SELBIG
Professeur, Université de Potsdam / examinateur
Anne SIEGEL
Directrice de Recherche, CNRS IRISA / directrice de thèse
Denis THIEFFRY
Professeur, ENS Paris / examinateur

Acknowledgements

First of all, I would like to thank Anne Siegel and Torsten Schaub, my two thesis advisors. Both Anne and Torsten, have trusted on me from the very beginning and without even knowing me. In fact, I must admit that at first I was rather scared of having two directors. Instead, nowadays I am convinced that this turned out to be the most valuable aspect during my experience as a doctoral student. Of course, this was only possible thanks to their constant goodwill for helping me to walk this path. Thus, once again: thank you very much!.

Special thanks goes for Carito Guziolowski and Julio Saez-Rodriguez who have played a crucial role during the whole thesis. Both were main contributors of the work presented in this thesis and help me to find challenging and relevant questions. Also, I thank to Federica Eduati, Jacques Nicolas, Martin Gebser, Roland Kaminski, Sven Thiele, and Thomas Cokelaer for their very valuable contributions. In addition, I would like to thank Alejandro Maass and all his research team for receiving me so kindly at the University of Chili. As well, I thank Alexander Bockmayr, Axel von Kamp, David Gilbert, Heike Siebert, Regina Samaga, and Steffen Klamt who I had the opportunity to meet and visit at their research groups for very interesting discussions.

I thank Gregory Batt, Mireille Ducasse, Christine Froidevaux, Joachim Selbig, and Denis Thieffry for having accepted to be members of the thesis committee, and for their time and interest in my work. I would like to acknowledge support from the the project ANR-10-BLANC-0218.

Big thanks to all my friends and colleagues in both Rennes and Potsdam who have made these last three years a very comfortable and enjoyable time. Finally, I of course thank to all my family and friends back in Argentina who have been always present despite the long distance.

Rennes, July 7th 2014 S. V.

Abstract

Deciphering the functioning of the so-called biological networks is one of the central tasks in systems biology. In particular, signal transduction networks are crucial for the understanding of the cellular response to external and internal perturbations. Notably, such networks are involved in biomedical processes and their control has a crucial impact on drug target identification and diagnosis. Importantly, in order to cope with the complexity of these networks, mathematical and computational modeling is required. Hence, the development of such modeling approaches is a major goal in the field. On the one hand, mathematical or quantitative approaches permit fine-grained analysis but are usually restricted to relatively small networks. On the other hand, computational or qualitative modeling allows for addressing large networks by relying on more abstract representations. Among various qualitative approaches, logic-based models are relatively simple yet able to capture interesting and relevant behaviors in the cell as several authors have shown during the last decade. In this context, researchers typically aim at modeling a given biological system by means of one logical model only. Afterwards, dynamical and structural analysis are conducted over *the* model. However, due to several factors, such as limited observability or the uncertainty in experimental measurements, it has been shown that *the* model is often non-identifiable. Hence, biological insights and novel hypotheses resulting from these analyses are likely to be incomplete, incorrect, or biased by methodological decisions.

In this thesis we propose a computational modeling framework in order to achieve more robust discoveries in the context of logical signaling networks. More precisely, we focus on modeling the response of logical signaling networks by means of automated reasoning using Answer Set Programming (ASP). ASP provides a declarative language for modeling various knowledge representation and reasoning problems. The basic idea of ASP is to express a problem in a logical format so that the models of its representation, the so-called answer sets, provide the solutions to the original problem. Moreover, available ASP solvers provide several reasoning modes for assessing the multitude of answer sets, among them, regular and projective enumeration, intersection and union, and multi-criteria optimization. Therefore, leveraging its rich modeling language and its highly efficient solving capacities, we use ASP to address three challenging problems in the context of logical signaling networks.

First we address the problem consisting of learning from an interaction graph and experimental observations, (Boolean) logical networks describing the immediate-early response of the system. Nowadays, for certain biological systems, a graph of causal interactions describing a large-scale signaling network can be retrieved from public databases [Guziolowski et al.,

2012] or by means of statistical methods and experimental data [Sachs et al., 2005]. However, functional relationships in signaling networks cannot be captured by means of graph theory only [Klamt et al., 2006b]. In this context, authors in [Saez-Rodriguez et al., 2009] have proposed a method to learn from an interaction graph and phosphorylation activities at a pseudo-steady state, Boolean logic models of immediate-early response fitting experimental data. Originally, a genetic algorithm implementation was proposed to solve the underlying optimization problem, and a software was provided, CellNOpt [Terfve et al., 2012]. Nonetheless, stochastic search methods cannot characterize the models precisely: they are intrinsically unable not just to provide a complete set of solutions, but also to guarantee that an optimal solution is found. Some variations of our problem were addressed using a mathematical programming approach [Mitsos et al., 2009, Sharan and Karp, 2013]. Despite their success to overcome some shortcomings of the genetic algorithm, such as performance and global optimality, the enumeration of all (nearly) optimal solutions was not considered. Combining both, multi-criteria optimization and enumeration provided by ASP, we are able to find all logic models explaining the experimental data equally well.

More generally, the inference of Boolean networks from time-series gene expression data has been addressed by several authors under different hypotheses and methods [Liang et al., 1998, Akutsu et al., 2000, Ideker et al., 2000, Lähdesmäki et al., 2003]. Recently, some of these methods have been compared in [Berestovsky and Nakhleh, 2013]. Nonetheless, overall, our work presents some significant differences. To start with, all of them are focused on gene regulatory networks and gene expression time-series data, whereas we work on signaling transduction networks and phosphorylation activities at a pseudo-steady state. Further, they work only with Boolean experimental observations which would correspond to adopt a binary discretization scheme in our framework. Moreover, except for the so-called *Best-Fit Extension Problem* [Lähdesmäki et al., 2003], they look for Boolean networks fully consistent with the time-series Boolean data. Meanwhile, herein we consider an objective function which describes the goodness of the model based on the numerical data that is subsequently optimized. Finally, all these contributions focus on a "local" inference in the following sense. They aim at learning the Boolean function for each node based on (local) input-output behaviors for such node. In contrast, our learning is based on (global) behaviors over the input-output layers in a network containing non-controllable/non-observable species.

Importantly, in contrast to the standard approach of modeling a signaling network by means of one logical model only, we show that there may be several thousands of feasible models. This fact motivates the second problem we address in this thesis which consists of finding an optimal experimental design in order to discriminate among all feasible logical networks. Broadly speaking, experimental design for model discrimination consists of finding an input that maximize the difference of the outputs of the rival models. In this context, we aim at finding the minimum number of experimental conditions allowing us to discriminate between every pair of input-output behaviors. Moreover, we adopt a criterion proposed before in the context of mathematical modeling in [Mélykúti et al., 2010]. Therein, authors argue that in principle, maximizing the difference between the outputs of two different models would ensure that even a noisy measurement has a good chance of discriminating between them.

Therefore, we adapt this idea to the context of our Boolean logic models and logical input-output behaviors. Also, we consider the minimization of the experiments' complexity in terms of the number of stimuli and inhibitions. Again, relying on multi-criteria optimization capacities of ASP, we can suggest the next round of experiments in order to refine the models at hand.

Most of the previous work on experimental design have been based on (semi-) quantitative modeling [Kremling et al., 2004, Vatcheva et al., 2005, Mélykúti et al., 2010, Stegmaier et al., 2013, Busetto et al., 2013]. Thus, in the context of computational modeling, existing approaches to experimental design are less established. It is worth noting that, in general, computational models provide certain predictive power which can be used to generate testable hypotheses and drive the experiments. Nevertheless, herein we refer to the specific problem consisting of automatically propose new experiments allowing to discriminate models at hand. To date, such a question has been addressed under various modeling hypotheses and methods [Ideker et al., 2000, Yeang et al., 2005, Barrett and Palsson, 2006, Szczurek et al., 2008, Sparkes et al., 2010]. Yet, their usefulness in practice remains an open question. Of special interest for us is the approach presented in [Sharan and Karp, 2013]. Therein, authors have addressed slight variations of our learning problem by means of mathematical programming. In addition, they sketched an algorithm for finding the most informative experiment to discriminate rival Boolean models but no implementation was provided. Nonetheless, compared to all aforecited contributions, our work presents certain differences and similarities. Except for [Szczurek et al., 2008], previous approaches aim at selecting exactly one experiment at each iteration. Therefore, only after the proposed experiment has been carried out in the laboratory and models have been (partially) discriminated, another experiment can be designed. In contrast, but similarly to [Szczurek et al., 2008], we aim at finding the smallest number of experiments to optimally discriminate all models at once. Furthermore, motivated by [Mélykúti et al., 2010], a distinct feature of our work is the criterion for optimality based on maximizing the sum of pairwise (output) differences. In general, previous methods have adopted an information-theoretic approach where the main design criterion is given by means of the so-called *Shannon entropy* [Shannon, 1948].

In principle, iterating over the loop of modeling and experimentation will yield more accurate logical models. Yet, there may be several models which cannot be discriminated using the available experimental capacities. In which case, instead of selecting a single model, we aim at reasoning over all of them. Thus, this leads to the third problem which consists of finding all minimal intervention strategies in order to control the biological system. That is, inclusion-minimal sets of activations and inhibitions forcing a set of target species into a desired steady state under various scenarios for all logical networks. Unfortunately, dedicated algorithms introduced in [Samaga et al., 2010] are computationally demanding due to the highly combinatorial mechanisms in logical networks. Therefore, they are limited to compute small intervention sets and fail to scale over large-scale networks. Importantly, in general, multiple interventions (or mutations) are necessary to cope with robustness and cellular complexity [Stelling et al., 2004]. Moreover, identified interventions should fulfill the desired goals in every feasible logical network. Concretely, the aforementioned limitations make

it hard to prove that the identified solutions are biologically robust to small variations of the system or its environment. Therefore, by reasoning over several sets of feasible logical networks we expect to find small yet robust intervention strategies.

More broadly, the problem of identifying "key-players" in logical signaling networks has been recently addressed in [Li et al., 2006, Abdi et al., 2008, Wang and Albert, 2011, Layek et al., 2011]. In contrast to our work, these contributions have rather focused on predicting what would happen if certain molecules fail. To that end, authors in [Li et al., 2006, Abdi et al., 2008, Layek et al., 2011] rely on digital circuits fault diagnosis engineering to identify the vulnerable molecules that play crucial roles in the dysfunction of signaling networks. In that context, a high vulnerability suggests that with high probability, the signaling network does not operate correctly if that particular molecule is dysfunctional ("stuck-at-0" or "stuck-at-1" in digital circuits terminology). It is worth noting that, due to computational limitations, in general authors have restricted their studies to small number of simultaneous faults. The approach presented in [Wang and Albert, 2011] does not relies on digital circuits but rather on standard graph theory. Therein, authors proposed two connectivity measures based either on the shortest paths, or on the so-called "elementary signaling modes". Then, using such measures their method provides a ranking of the nodes by the effects of their loss on the connectivity between the network's inputs and outputs. Importantly, despite the specific problem settings and computational approaches in these contributions, all of them have considered a single logical network describing the system. Notably, interventions allowing for accomplish certain goals in a given network are very likely to fail in another network which may describe the system equally well. Therefore, being able to address the same question but considering an ensemble of feasible models leads to more robust strategies.

We illustrate our approach to each of the aforementioned problems using real-world signaling pathways in human liver cells and publicly available experimental data. Interestingly, the computational performance is significantly improved with respect to dedicated algorithms to solve the same problems. But more importantly, the exhaustive nature of ASP allows us to find feasible solutions that were missing when using the existing methods. Also, thanks to its rich modeling language and efficient solving, ASP allows for reasoning over an ensemble of logical networks in order to achieve more robust discoveries. Altogether, the contribution of this thesis is on three different axes. On the modeling side, the proposed framework provides an unified computational modeling approach for reasoning on logical signaling networks. On the computational side, this thesis illustrates the potential of ASP to address hard combinatorial search and optimization problems related to qualitative modeling of biological systems. Finally, on the implementation side, we have presented a software package providing an interface to the ASP-based solutions in order to ease the accessibility for systems biologists, as well as the integration with available tools for simulation and analysis of logical models.

To conclude, the work presented in this thesis has raised several interesting questions for future research. In particular, our work have opened the way to an exhaustive characterization of feasible logical models for a given system. Now, we need to develop a proper modeling framework to interpret and take advantage of such a characterization. This must be necessarily

driven by available experimental technology allowing us to either confirm or refute generated hypotheses. For instance, phospho-proteomics assays like the one used for learning are performed over a population of cells and thus, it is unclear how to interpret the multitude of logical networks and their input-output behaviors. Given the existence of several logical input-output behaviors gathering different internal mechanisms, we need to elucidate if such mechanisms are a mere artifact of logical networks, or if they actually represent molecular mechanisms which in turn appear more or less often within each cell or at a population scale. Furthermore, despite the intrinsic uncertainty in biological systems, it is still rather hard to assess the amount of noise in measurements and how this impacts on our mathematical or computational models. This raises the question of identifiability and to what extent new experiments are able to yield more refined logical models. Thus, the method for experimental design requires to be validated and could help to tackle this issue. Finally, we find particularly interesting to explore further our approach for finding intervention strategies by reasoning over an ensemble of logical networks. In principle, such an approach allows systems biologists to draw insights under uncertainty and non-identifiability. Nonetheless, we need to elucidate whether such an ensemble of networks describes alternative pathways within a single cell or at the population scale. Overall, we look forward for experimental validation of our methods and findings. Notably, these questions comprise integrative modeling approaches considering multiple levels and time-scales of causation which pose very challenging goals. Towards this end, the development of hybrid reasoning systems leveraging the expressiveness of several technologies and modeling approaches appears as a very promising track for future research in computer science.

Key words: systems biology, logical signaling networks, answer set programming

Résumé

Décrypter le fonctionnement des réseaux biologiques est l'une des missions centrales de la biologie des systèmes. En particulier, les réseaux de signalisation sont essentiels pour la compréhension de la réponse cellulaire à des perturbations externes et internes. Notamment, ces réseaux sont impliqués en santé humaine et leur contrôle a un impact sur l'identification et la caractérisation de cibles de médicaments. Pour affronter la complexité de ces réseaux, modélisations mathématiques et informatiques sont nécessaires. D'une part, les approches mathématiques ou quantitatives permettent une analyse fine mais sont généralement limitées à des réseaux relativement petits. D'autre part, la modélisation informatique ou qualitative permet de traiter de grands réseaux, en s'appuyant sur d'autres représentations abstraites. Parmi les différentes approches qualitatives, les modéles logiques sont relativement simples, mais en mesure de capturer des comportements intéressants et pertinents dans la cellule. Dans ce cadre, les chercheurs essaient généralement de modéliser un système biologique particulier, en utilisant *un seul* modèle logique. Ensuite, des analyses dynamiques et structurales sont menées sur *ce* modèle. Toutefois, en raison de plusieurs facteurs, tels que l'observabilité limitée ou de l'incertitude dans les mesures expérimentales, il a été montré qu'un tel modèle est en pratique non identifiable dans de nombreux cas. En effet, les connaissances biologiques et les hypothèses découlant de ces analyses sont susceptibles d'être incomplètes, erronées, ou biaisées par des décisions prises au moment de l'identification ou de la construction du modèle.

Dans cette thèse, nous proposons un cadre de modélisation informatique afin d'obtenir des découvertes plus robustes dans le cadre de réseaux de signalisation modélisés sous forme logique. Plus précisément, nous nous concentrons sur la modélisation de la réponse des réseaux logiques au moyen du raisonnement automatisé à l'aide de Programmation par Ensembles-Réponses (*Answer Set Programming*, ASP). De manière générale, ASP fournit un langage déclaratif pour la modélisation de divers problèmes de représentation des connaissances et de raisonnement. L'idée de base de ASP est d'exprimer un problème sous la forme d'un programme logique de sorte que les modèles de sa représentation, les *answer sets* (ensembles-réponses), fournissent les solutions au problème initial. En outre, différents solveurs disponibles pour ASP offrent plusieurs modes de raisonnement pour l'évaluation de la multitude de réponses : énumération régulière et projective, intersection, union, et optimisation multi-critères. Par conséquent, en s'appuyant sur la richesse de son langage de modélisation et de ses capacités de résolution très efficaces, nous utilisons ASP pour répondre à trois problèmes difficiles dans le contexte des réseaux logiques de signalisation.

Premièrement, nous abordons le problème consistant à apprendre, à partir d'un graphe d'interaction et d'observations expérimentales, les réseaux booléens qui décrivent la réponse immédiate du système. Aujourd'hui, pour certains systèmes biologiques, un graphe des interactions causales qui décrivent un réseau de signalisation à grande échelle peut être extrait de bases de données publiques [Guziolowski et al., 2012] ou être appris à l'aide de méthodes statistiques et de données expérimentales [Sachs et al., 2005]. Cependant, les relations fonctionnelles dans les réseaux de signalisation ne peuvent pas être capturées seulement à l'aide d'analyse de graphes [Klamt et al., 2006b]. Dans ce contexte, les auteurs de [Saez-Rodriguez et al., 2009] ont proposé une méthode pour apprendre, à partir d'un graphe d'interaction et des activités de phosphorylation mesurées à un état pseudo-stationnaire de la cellule, les modèles logiques booléens de la réponse précoce compatible avec les données expérimentales. A l'origine, un algorithme génétique a été proposé pour résoudre le problème d'optimisation sous-jacent, et un logiciel a été fourni : CellNOpt [Terfve et al., 2012]. Cependant, les méthodes de recherche stochastiques ne peuvent pas caractériser les modèles intégralement : ces approches sont intrinsèquement incapables non seulement de fournir un ensemble complet de solutions, mais aussi de garantir qu'une solution optimale a été trouvée. Certaines variations de ce problème ont été abordées avec une approche de programmation linéaire [Mitsos et al., 2009, Sharan and Karp, 2013]. En dépit de leur succès à surmonter certaines insuffisances des algorithmes génétiques, comme la performance et la preuve de l'optimalité globale, l'énumération de toutes les solutions optimales (ou presque optimales) n'a pas été étudiée dans ces travaux. En combinant à la fois l'optimisation multi-critères et l'énumération fournie par ASP, nous montrons dans cette thèse qu'il est possible d'énumérer tous les modèles logiques expliquant les données expérimentales de manière sous-optimale.

Plus généralement, l'inférence de réseaux booléens à partir de données de séries temporelles d'expression génique a été abordée par plusieurs auteurs sous différentes hypothèses [Liang et al., 1998, Akutsu et al., 2000, Ideker et al., 2000, Lähdesmäki et al., 2003]. Récemment, certaines de ces méthodes ont été comparées dans [Berestovsky and Nakhleh, 2013]. Cependant, globalement, notre travail présente des différences significatives. Pour commencer, les références précédentes sont toutes concentrées sur les réseaux de régulation de gènes et des données de séries temporelles d'expression génique, tandis que nous travaillons sur la signalisation des réseaux de transduction et à l'aide de mesures d'activités de phosphorylation à un état pseudo-stationnaire. En outre, les approches d'apprentissage de réseaux fonctionnent uniquement avec des observations expérimentales booléennes, ce qui correspondrait à adopter un schéma de discrétisation binaire dans notre cadre. De plus, sauf pour le *Best-Fit Extension problem* [Lähdesmäki et al., 2003], les recherches consistent à identifier des réseaux booléens entièrement compatibles avec les données booléennes. Au lieu de cela, nous considérons ici une fonction objectif qui décrit la correction du modèle basée sur des données quantitatives. C'est cette fonction objectif numérique qui est ensuite optimisée. Enfin, toutes les contributions mentionnées ci-dessus portent sur une inférence "locale" dans le sens suivant : elles visent à l'apprentissage de la fonction booléenne pour chaque noeud sur la base des comportements entrées-sorties (locales) pour un tel noeud. En revanche, notre méthode d'apprentissage est basée sur les comportements (globaux) d'entrée-sortie dans un

réseau contenant des espèces non-controlables ou non-observables.

A la différence des approches classiques de modélisation d'un réseau de signalisation par un modèle logique seulement, on montre ainsi qu'il peut y avoir plusieurs milliers de modèles qui expliquent tous de manière sous-optimale les données entrées-sorties. Ce fait motive le deuxième problème que nous abordons dans cette thèse qui consiste à trouver un plan d'expérience optimal afin de discriminer les réseaux logiques possibles. D'une manière générale, le plan expérimental pour discriminer une famille de modèles consiste à trouver une entrée qui maximise la différence des sorties des modèles concurrents. Dans ce contexte, nous nous efforçons de trouver le nombre minimum de conditions expérimentales permettant de discriminer chaque paire de comportements entrée-sortie. En outre, nous adoptons un critère introduit dans une approche de modélisation quantitative [Mélykúti et al., 2010]. Dans ce travail, les auteurs font valoir que, en principe, en maximisant la différence entre les sorties de deux modèles différents, on garantit que même une mesure bruitée a une bonne chance de faire la distinction entre eux. Nous adaptons cette idée dans le contexte de nos modèles logiques booléens et les comportements logiques entrées-sorties. Aussi, nous cherchons à minimiser la complexité des expériences en termes de nombre de stimuli et d'inhibitions. Encore une fois, en s'appuyant sur les capacités d'optimisation multi-critères de l'ASP, nous pouvons identifier la meilleure série d'expériences afin d'affiner la famille de modèles disponibles à un moment donné.

La plupart des travaux antérieurs sur la conception de plans expérimentaux reposent sur des modélisations (semi-) quantitatives [Kremling et al., 2004, Vatcheva et al., 2005, Mélykúti et al., 2010, Stegmaier et al., 2013, Busetto et al., 2013]. Dans le contexte de la modélisation symbolique, les approches existantes sont moins établies. Il est intéressant de noter que, en général, les modèles formels s'appuient sur des fonctions de prédiction qui peuvent être utilisées pour générer des hypothèses testables et réaliser les expériences. Quand même, nous nous référons ici au problème spécifique consistant à proposer automatiquement de nouvelles expériences qui permettent de discriminer une famille de modèles. À ce jour, cette question a été abordée sous différentes hypothèses de modélisation et avec différentes méthodes [Ideker et al., 2000, Yeang et al., 2005, Barrett and Palsson, 2006, Szczurek et al., 2008, Sparkes et al., 2010]. Pourtant, leur application pratique reste une question ouverte. Une approche particulièrement intéressante est celle présentée dans [Sharan and Karp, 2013]. Dans ce travail, les auteurs ont étudié une variante de notre problème d'apprentissage par le biais de la programmation linéaire. Aussi, ils ont esquissé un algorithme pour trouver l'expérience la plus informative permettant de discriminer des modèles booléens rivaux. Cependant, aucune application n'a été fournie. Dans l'ensemble, par rapport à toutes les contributions cités précédemment, notre travail se positionne comme suit. Sauf pour [Szczurek et al., 2008], les approches précédentes visent à sélectionner exactement une expérience à chaque itération. Par conséquent, c'est seulement après que l'expérience proposée a été effectuée dans le laboratoire et que les modèles ont été (partiellement) discriminés, qu'une une autre expérience peut être conçue. En revanche, mais de façon similaire à [Szczurek et al., 2008], nous nous efforçons de trouver d'un seul coup le plus petit nombre d'expériences pour discriminer de manière optimale tous les modèles à la fois. De plus, motivé par [Mélykúti et al., 2010], une caractéristique de notre travail est

l'emploi d'un critère d'optimalité basé sur la maximisation de la somme des différences deux à deux. En général, les méthodes précédentes ont adopté une approche basé sur la théorie de l'information où le critère principal est donné au moyen de ce qu'on appelle *l'entropie de Shannon* [Shannon, 1948].

En principe, l'itération sur la boucle de modélisation et d'expérimentation donnera des modèles logiques plus précis et réduira la taille de la famille de modèles compatibles. Cependant, il peut y avoir à la fin plusieurs modèles qui ne peuvent être discriminés à l'aide des capacités expérimentales disponibles. Dans ce cas, au lieu de sélectionner un seul modèle, nous visons à raisonner sur le comportement de l'ensemble d'entre eux. Ainsi, ceci nous conduit à étudier un troisième problème qui consiste à trouver l'ensemble des stratégies d'intervention minimale afin de contrôler la réponse du système biologique. Plus précisément, nous cherchons à déterminer les ensembles d'activations et inhibitions forçant un ensemble d'espèces cibles à parvenir à un état d'équilibre attendu, dans le cadre de différents scénarios, et ceci pour tous les réseaux logiques d'une famille donnée. Malheureusement, les algorithmes dédiés introduits dans [Samaga et al., 2010] sont gourmands en calculs, en raison des mécanismes hautement combinatoires dans les réseaux logiques. Par conséquent, ces algorithmes sont limités à calculer de petits ensembles d'intervention et ne passent pas à l'échelle sur des réseaux de grande taille. Il faut noter que, en général, plusieurs interventions (ou mutations) sont nécessaires pour faire face à la robustesse et à la complexité cellulaire [Stelling et al., 2004]. Par ailleurs, les interventions identifiées doivent permettre au système de réaliser les objectifs attendus dans chaque réseau logique possible. Du fait de ces limitations, il est difficile de prouver que les solutions identifiées sont biologiquement robustes vis-à-vis de petites variations du système ou de son environnement. Par contre, en raisonnant sur une famille de réseaux logiques, nous espérons trouver des stratégies d'intervention robustes.

Plus généralement, le problème de l'identification des acteurs-clé dans les réseaux logiques de signalisation a été abordé dans [Li et al., 2006, Abdi et al., 2008, Wang and Albert, 2011, Layek et al., 2011]. Contrairement à notre travail, ces contributions ont plutôt mis l'accent sur la prédiction de ce qui se passerait si certaines molécules ne se comportaient pas comme attendu. À cette fin, les auteurs de [Li et al., 2006, Abdi et al., 2008, Layek et al., 2011] s'appuient sur le diagnostic de fautes dans des circuits pour identifier les molécules vulnérables qui jouent un rôle crucial dans le dysfonctionnement des réseaux de signalisation. Dans ce contexte, une grande vulnérabilité suggère que, avec une forte probabilité, le réseau de signalisation ne fonctionne pas correctement si cette molécule particulière est dysfonctionnelle ("stuck-at-0" ou "stuck-at-1" en terminologie des circuits digitaux). Il est intéressant de noter que, en raison des limites de calcul, en général, les auteurs ont limité leurs études à de petits nombres de défauts simultanés. L'approche présentée dans [Wang and Albert, 2011] ne s'appuie pas sur des circuits digitaux, mais plutôt sur la théorie des graphes. Dans ce travail, les auteurs ont proposé deux mesures de connectivité basées sur les chemins les plus courts, ou sur les dénommés *elementary signaling modes*. Puis, en utilisant ces mesures, leur méthode fournit un classement des noeuds via les effets de la perte de connectivité entre les entrées et les sorties du réseau. Il faut noter que, malgré les réglages spécifiques pour chaque problème et les approches informatiques de ces contributions, tous ont considéré un seul réseau logique

pour décrire le système. Or, comme nous l'avons mentionné plus haut, nous avons montré qu'il peut exister une famille de modèles importante qui peuvent tous expliquer aussi bien les données expérimentales. Les interventions qui permettent de réaliser certains objectifs à l'aide d'un réseau donné sont susceptibles d'échouer dans une autre réseau qui peut décrire le système tout aussi bien. Être capable, comme nous le faisons, de répondre à la même question sur le calcul des interventions permettant de forcer une réponse attendue, mais pour une famille de modèles, conduit finalement à des stratégies plus robustes.

Nous illustrons les approches répondant à chacun des problèmes mentionnés ci-dessus, en considérant un exemple de voies de signalisation dans des cellules hépatiques humaines et des données expérimentales disponibles publiquement. Dans tous les cas, les performances de calcul sont significativement améliorées par rapport à des algorithmes dédiés pour résoudre les mêmes problèmes. Mais encore plus important, la nature exhaustive d'ASP nous permet d'énumérer toutes les solutions admissibles qui pouvaient être omises lors de l'utilisation des méthodes existantes. Aussi, grâce à la richesse de son langage de modélisation et ses méthodes de résolution efficaces, ASP permet de raisonner sur un ensemble de réseaux logiques pour réaliser des découvertes plus robustes.

Au final, la contribution de cette thèse concerne trois axes différents. Du côté de la modélisation, le cadre proposé introduit une approche de modélisation formelle unifiée pour le raisonnement sur les réseaux logiques de signalisation. Sur le plan informatique, cette thèse illustre le potentiel d'ASP pour traiter des problèmes combinatoires difficiles de recherche et d'optimisation liés à la modélisation qualitative des systèmes biologiques. Enfin, du côté applicatif, nous avons conçu un logiciel fournissant une interface qui rend transparente l'utilisation d'ASP par des utilisateurs biologistes ou modélisateurs, et peut être intégré dans d'autres outils disponibles pour la simulation et l'analyse des modèles logiques.

Pour finir, le travail présenté dans cette thèse a soulevé plusieurs questions intéressantes pour de futures recherches. En particulier, nos travaux ont ouvert la voie à une caractérisation exhaustive des modèles logiques possibles pour un système donné. Maintenant, nous devons développer un cadre de modélisation approprié pour interpréter et tirer partie de cette caractérisation. Ce cadre de modélisation doit être nécessairement guidé par les technologies expérimentales disponibles, qui nous permettent de confirmer ou d'infirmer les hypothèses générées. Par exemple, les essais phospho-protéomiques comme celui utilisé pour l'apprentissage de réseaux sont réalisés sur une population de cellules. On ne sait donc pas comment interpréter la multitude de réseaux logiques et leurs comportements d'entrée-sortie au niveau d'une cellule individuelle. Etant donné l'existence de plusieurs comportements d'entrée-sortie logiques qui correspondent à différents mécanismes internes, nous devons élucider si ces mécanismes sont un simple artefact des modélisations logiques, ou s'ils représentent réellement les mécanismes moléculaires variables qui à leur tour apparaissent plus ou moins souvent au sein de chaque cellule ou à une échelle de la population. De plus, en dépit de l'incertitude intrinsèque dans les systèmes biologiques, il est encore assez difficile d'évaluer la quantité de bruit dans les mesures et comment cela influe sur nos modèles mathématiques ou formels. Cela soulève la question de l'identifiabilité et dans quelle mesure les nouvelles expériences peuvent permettre de construire des modèles logiques plus fins. La méthode de conception

de plans expérimentaux que nous proposons nécessite d'être validée et pourrait représenter un premier pas dans cette direction. Enfin, il serait particulièrement intéressant d'explorer plus notre approche permettant de trouver des stratégies d'intervention en raisonnant sur un ensemble de réseaux logiques. En principe, une telle approche devrait permettre de tirer des enseignements sur l'incertitude et la non-identifiabilité des systèmes étudiés. Cependant, nous devons à nouveau élucider si une telle famille de réseaux décrit des voies alternatives au sein d'une seule cellule ou à l'échelle de la population. Enfin, toutes ces questions s'intègrent dans un cadre de modélisation intégratif qui se positionne à différentes échelles de temps et de causalité. À cette fin, le développement de systèmes de raisonnement hybrides tirant parti de l'expressivité de plusieurs technologies et approches de modélisation apparaît comme une piste très prometteuse pour de futures recherches en informatique.

Mots clefs : biologie des systèmes, réseaux de signalisation logiques, *answer set programming*

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Systems biology and signaling networks

Systems biology

Systems biology is an interdisciplinary field aiming at the investigation and understanding of biology at a system and multi-scale level [Ideker et al., 2001, Kitano, 2002]. After biological entities have been identified in a specific environment, it remains to elucidate how they interact with each other in order to carry out a particular biological function. Thus, rather than focusing on the components themselves, one is interested on the nature of the links that connect them and the functionalities arising from such interactions. Notably, advances on high-throughput experimental technologies have been one of the main driving forces of systems biology. Such technologies have allowed biologists to study biological systems as a whole rather than as individual components. Nevertheless, the "reductionist" approach of molecular biology has been fundamental for the construction of the large catalogues of biological entities available nowadays. In fact, some authors have considered systems biology not as a new field of research but instead, as an approach to biomedical research combining "reductionist" and "integrationist" techniques [Kohl et al., 2010].

As it is often the case, the application of novel technologies has led to profound conceptual and philosophical changes in biology. From the early days of molecular biology, there exists the idea that the DNA sequence dictates most of cell actions, as the instructions in a computer program. Recently, together with the advent of systems biology, such a mechanistic understanding has been strongly revisited. Instead, an informatic perspective on the role of the genome has been established. From this point of view, the focus is on what the cell does with and to its genome products rather than on what the genome directs the cell to execute [Shapiro, 2009]. Then, for any biological system one can envision at least a three-way interaction between DNA products, the environment and the phenotype [Kohl et al., 2010]. In this scheme, the group of entities mediating between such interactions are the so-called biological networks. Deciphering the functioning of these complex networks is the central task of systems biology. Importantly, in order to cope with the increasing complexity of large-scale networks, mathematical and

computational modeling is required. Hence, the development of such modeling approaches is a major goal in the field.

From the early millennium, many efforts have been made to develop relevant formalisms and modeling frameworks to take into account the specificities of complex biological systems. Among them, one can distinguish between mathematical and computational modeling approaches [Fisher and Henzinger, 2007]. Essentially, mathematical (or quantitative) models are based on denotational semantics, that is, models are specified by mathematical equations describing how certain quantities change over time [Aldridge et al., 2006]. On the other hand, computational (or qualitative) models are based on operational semantics, that is, models are specified in terms of a sequence of steps describing how the states of an abstract machine relate to each other (not necessarily deterministically) [Melham, 2013]. Notably, each type of models provides a different level of abstraction enabling to address different kinds of questions. In fact, hybrid modeling precisely aims at exploiting the best of both worlds whenever possible [Henzinger, 1996]. In any case, it is clear that intuition is not enough to face the complexity of large-scale biological systems. Thus, systematic and elaborated methodological tools are required by (systems) biologists. Moreover, the development of such modeling frameworks is leading to a hypothesis-driven research in biology [Ideker et al., 2001, Kitano, 2002]. At first, due to the lack of information, multiple hypotheses are usually generated from prior knowledge and either mathematical, computational or hybrid modeling. Next, decision-making methods can be used to suggest new experiments in order to reduce ambiguous hypotheses [Kreutz and Timmer, 2009]. Finally, new experimental data is produced to test the generated hypotheses, the models are refined, and the loop is started over again. Interestingly, to some extent, this iterative process could be automatized allowing an autonomous scientific discovery [Sparkes et al., 2010].

Signaling networks

Among the biological networks mediating between genes, the environment and the phenotype, signal transduction networks are crucial for the understanding of the response to external and internal perturbations [Gomperts et al., 2009]. To be more precise, signal transduction occurs when an extracellular signaling molecule binds to a specific cell surface receptor protein. Such a binding causes a conformational change in the receptor that initiates a sequence of reactions leading to a specific cellular response such as growth, survival, apoptosis (cell death), and migration. Post-translational modifications, notably protein phosphorylation, play a key role in signaling. Importantly, signaling networks are involved in biomedical processes and their control has a crucial impact on drug target identification and diagnosis. Unfortunately, little is known still about the exact chaining and composition of signaling events within these networks in specific cells and specific conditions. For example, in cancer cells, signaling networks frequently become compromised, leading to abnormal behaviors and responses to external stimuli [Hanahan and Weinberg, 2011]. Thus, many current and emerging cancer treatments are designed to block nodes in signaling networks, thereby altering signaling cascades [Gom-

perts et al., 2009, Hainaut and Plymoth, 2012, Csermely et al., 2013]. Researchers expect that, advancing our understanding of how these networks are deregulated across specific environments will ultimately lead to more effective treatment strategies for patients. In fact, there is emerging experimental evidence that the combinatorics of interactions between cellular components in signaling networks is a primary mechanism for generating highly specialized biological systems [Papin et al., 2005]. In this context, phosphorylation assays are a recent form of high-throughput data providing information about protein-activity modifications in a specific cell type upon various combinatorial perturbations [Alexopoulos et al., 2010]. Therefore, moving beyond causal and canonical interactions towards mechanistic and specialized descriptions of signaling networks is a major challenge in systems biology. Importantly, cellular signaling networks operate over a wide range of timescales (from fractions of a seconds to hours). Thus, taking this into account often leads to significant simplifications [Papin et al., 2005, Macnamara et al., 2012].

Finally, we conclude this brief introduction to systems biology and signaling networks by noting that biological functionality is multilevel [Noble, 2010]. That is, signaling networks by no means could function in isolation from metabolic and regulatory processes. Hence, integrative modeling approaches considering these multiple levels of causation pose indeed a long-term goal in the field.

1.2 Computational modeling and methods

Computational modeling

The simplest abstraction to describe signaling networks is by means of standard graph theory. Nodes in the graph typically describe biological species whereas signed and directed edges represent causal relations among them (activatory or inhibitory effects). Nowadays, there exist public repositories such as Pathways Commons [Cerami et al., 2011], Pathways Interaction Database [Schaefer et al., 2009], and KEGG [Kanehisa et al., 2010] that contain curated knowledge about intracellular causal molecular interactions found in different cell types and species. Thus, on the one hand, one can query such databases in order to retrieve canonical cellular signaling networks [Guziolowski et al., 2012]. On the other hand, in order to build context-specific causal interaction graphs, machine learning was applied for the automated inference from experimental observations relying on Bayesian networks [Sachs et al., 2005, Needham et al., 2007]. Next, several graph-theoretical concepts can be used to understand structural properties of the biological system under study. However, functional relationships in signaling networks cannot be captured by means of graph theory only [Klamt et al., 2006b]. If two proteins modeled by nodes a and b have a positive effect on a third one c, this would be described in a graph by edges $a \rightarrow c$, $b \rightarrow c$. Nevertheless, it is unclear whether a or b can independently activate c, or if both are required. Therefore, due to the lack of mechanistic information, causal interaction graphs have null or very limited prediction power.

In order to qualitatively describe functional relations between species, numerous discrete dynamical systems approaches have been proposed. Among them, logic-based models have become very popular in the last few years [Réka and Wang, 2008, Morris et al., 2010, Wang et al., 2012, Saadatpour and Réka, 2013]. Generally speaking, species in the system are described by propositional variables and their relationships are captured by means of propositional formulas. Then, such a set of formulas determines the evolution of the system over discrete time steps. That is, the future state of each variables is determined by the current states of other variables through its corresponding formula. In this context, Boolean networks [Kauffman, 1969] provide a modeling approach able to capture interesting and relevant behaviors in the cell despite their simplicity [Bornholdt, 2008]. Notably, this simplicity is due to the high level of abstraction and synchronous update scheme in Boolean networks. That is, one needs to assume that all species in the system update their (discrete) state at the same time. In particular, this leads to deterministic dynamical behaviors. Nonetheless, it has been shown that, to some extent, the (early) response in signaling networks can be appropriately modeled with Boolean networks [Saez-Rodriguez et al., 2007, Samaga et al., 2009, Saez-Rodriguez et al., 2011]. In order to overcome some shortcomings in (synchronous) Boolean networks, more elaborate logical frameworks have been proposed. For instance, asynchronous automata networks [Thomas, 1973, 1991], and probabilistic Boolean networks [Shmulevich et al., 2002a] are often able to capture more complex (non-deterministic) dynamical behaviors [Shmulevich et al., 2002b, Sánchez et al., 2008, Calzone et al., 2010]. However, in order to capture such behaviors in large-scale networks, the computational cost is often significantly high due to the combinatorial explosion of possible states at simulation.

More broadly, other computational formalisms have been proposed to describe complex dynamical systems by incorporating (semi-) quantitative information, for instance, piecewise-linear models [de Jong et al., 2004], Petri nets [Chaouiya, 2007], π-calculus [Regev et al., 2001], and rule-based modeling [Hlavacek et al., 2006]. Piecewise-linear models rely on class of piecewise-linear differential equations to provide a coarse-grained description tailored to qualitative simulation. Although related to asynchronous automata networks, piecewise-linear models account for larger generality whereas their base mathematical formalism, namely, differential equations, facilitate the integration of quantitative data towards pure mathematical models. Petri nets provide a very intuitive (graphical) framework to represent production and consumption effects like in metabolic networks but, signaling events are not so easily modeled with standard settings. In fact, several extensions have been defined to increase the expressivity of standard Petri nets, such as stochastic, coloured, and continuous Petri nets [Heiner and Gilbert, 2011]. The π-calculus is an abstract process language developed for specifying concurrent computational systems [Milner, 1999]. In our context, the key idea is to model a biological system as a network of communicating processes (molecular species) through specific channels (binding sites). Furthermore, the so-called stochastic π-calculus [Priami et al., 2000] has been proposed to describe more accurate dynamical behaviors. Finally, in contrast to the aforementioned general purpose modeling frameworks, rule-based modeling relies on domain-specific languages to model biochemical networks [Hlavacek et al., 2006,

Calzone et al., 2006, Danos and Laneve, 2003]. Similarly to π-calculus, it allows modelers to take into account physicochemical parameters that capture details about proteins and their interactions at the level of binding sites. Notably, as one considers more proteins and binding sites, the number of possible protein complexes and combinations of protein modifications tends to increase exponentially [Papin et al., 2005]. Therefore, the explicit writing of all possible chemical reactions for signaling networks with more than a few proteins is out of reach. In fact, the main motivation for rule-based modeling is to provide a symbolic representation to face this combinatorial complexity, whereas various implementations allow to automatically generate a computational model to be executed.

Current challenges in logic-based modeling

In this thesis, we are interested in logic-based models, particularly Boolean networks and slight variations thereof. We note that, such modeling approach is among the simplest computational abstractions (after causal interaction graphs). Nevertheless, initially simple logic-models have been refined in an iterative fashion by adding layers of complexity yielding more precise descriptions [Wittmann et al., 2009, Samaga and Klamt, 2013]. This is especially useful in the case of poorly understood biological systems where kinetic information is scarce and hard to obtain at early stages of modeling. Thus, despite their high level of abstraction, logic-based models are well-suited to guide exploratory research where more detailed modeling frameworks are hard to adopt due to the lack of information. In this context, it is important to note that a large majority of the authors working with such kinds of models rely on *ad-hoc* methods to build them. In most cases, models are constructed manually based on information extracted from literature, experimental data, and human-expert knowledge. Notably, the manual identification of logic rules underlying the system being studied is often hard, error-prone, and time consuming. In this context, researchers typically aim at modeling a given biological system by means of one logical network only. Afterwards, dynamical and structural analysis (often computationally demanding) are conducted over *the* model, for instance, the identification of key-players [Abdi et al., 2008, Samaga et al., 2010, Wang and Albert, 2011, Layek et al., 2011] or logical steady states [Christensen et al., 2009, Saadatpour et al., 2011, Mendoza et al., 1999]. However, due to several factors, such as limited observability or the uncertainty in experimental measurements, *the* model is often non-identifiable. Hence, biological insights and novel hypotheses resulting from these analyses are likely to be incomplete, incorrect, or biased by methodological decisions. Therefore, automated learning of logic-based models is required in order to achieve unbiased and more robust discoveries.

Reverse engineering in systems biology consists of building mathematical and computational models of biological systems based on the available knowledge and experimental data. Towards the construction of predictive models, one can convert the generic prior knowledge (for example, canonical cell signaling networks) into a quantitative or qualitative model (for instance, a set of differential equations or a set of logic rules) that can be simulated or executed. Next, if enough experimental data is available, the model can be fitted to it in order to

obtain the most plausible models for certain environmental conditions or specific cell type. This is normally achieved by defining an objective fitness function to be optimized [Banga, 2008]. Optimization over quantitative modeling leads to continuous optimization problems. On the other hand, reverse engineering considering qualitative models typically give rise to combinatorial (discrete) optimization problems. Notably, this subject represents a very active area of research as illustrated by the successive "DREAM" challenges [Stolovitzky et al., 2007]. Importantly, methods for reverse engineering of biological systems are highly dependent on available (amount of) data, prior knowledge and modeling hypotheses. For instance, an inference method from gene expression data collected by DNA microarrays, may not be applicable to biochemical data like phosphorylation assays collected using xMAP Luminex technology. In particular, reverse engineering of logical models for signaling networks by confronting prior knowledge on causal interactions with phosphorylation activities has been first addressed in [Saez-Rodriguez et al., 2009]. Therein, authors have shown that the model is non-identifiable as soon as we consider the experimental error from measurements. Hence, rather than looking for *the* optimum logical model, one aims at finding (nearly) optimal models within certain tolerance. Interestingly, in the context of mathematical modeling, authors in [Chen et al., 2009] have elaborated upon the same argument. Clearly, an exhaustive enumeration of (nearly) optimal solutions would allow for identifying admissible logical models without any methodological bias. Furthermore, all subsequent analysis will certainly profit from having such a complete characterization of feasible models. That is, being able to address a given problem but considering an ensemble of logical models may lead to more robust solutions. In fact, this is in line with recent work showing that an ensemble of models often yields more robust predictions than each model in isolation [Kuepfer et al., 2007, Marbach et al., 2012]. Importantly, existing approaches, namely stochastic search and mathematical programming, are not well-suited to cope with this question in an exhaustive manner. Hence, there is an increasing demand of more powerful computational methods in order to achieve robust discoveries in the context of logic-based modeling.

Computational methods

Regardless of the modeling approach and specific biological question, numerous computational methods have been proposed to solve the underlying search and optimization problems. On the one hand, we find dedicated algorithms implementations either looking at exact solutions or by means of stochastic local search [Hoos and Stützle, 2005]. For instance, CellNetAnalyzer [Klamt et al., 2006a] implements various graph algorithms for structural and functional analysis of signaling networks whereas CellNOpt [Terfve et al., 2012] relies on a genetic algorithm for training logical models to experimental observations. By their algorithmic nature, such dedicated solutions allow for having control on *how* the problem is solved. However, in order to address large-scale problems, a significant amount of effort is needed for the development and maintenance of such algorithms. Also, it is often the case that, as soon as we introduce a slight modification in the problem specification, it is rather hard to adapt such dedicated algorithms to it. On the other hand, several problem solving technologies exist

which allow us to provide a specification, in some kind of machine-readable format, of *what* the problem is but without giving an algorithm to solve it. Afterwards, a general purpose solver is able to extract and (efficiently) traverse the search space looking for solutions. Popular methods found in the literature include, (mixed) integer linear programming ((M)ILP) [Mitsos et al., 2009, Sharan and Karp, 2013], model checking by means of temporal logic [Chabrier and Fages, 2003, Batt et al., 2005], satisfiability testing (SAT) [Dubrova and Teslenko, 2011, Guo et al., 2014], and constraint logic programing (CLP) [Soliman, 2011, Gay et al., 2011].[1] While each of these computational methods has proven to be very useful for solving challenging problems, in general, they provide different kinds of complementary features. Furthermore, in general, the performance of each solver is subject to specific parameters (search heuristics) and problems.

In what follows we introduce Answer Set Programming (ASP), the computational method adopted in this thesis. But before, let us highlight two distinct features compared to aforementioned methods. Firstly, ASP provides a fully declarative and executable modeling language. This is in contrast with other general solving technologies, such as (M)ILP and SAT, where the user is responsible of converting the problem specification (e.g., mathematical inequalities or Boolean formulas) and input data into the machine-readable format accepted by the solver. Typically, this is done *ad hoc* by means of some imperative programming language yielding an overhead to the solving process. Secondly, ASP provides several *built-in* reasoning modes, such as search, enumeration, and multi-objective optimization, among others. Notably, each of the previously mentioned methods (and extensions thereof) provides some but not all such functionalities. Thus, in order to achieve such automated reasoning modes, one needs to develop dedicated algorithms on top of the solver yielding again, an overhead to the process.

Answer Set Programming at a glance

Answer Set Programming (ASP; [Baral, 2003, Gebser et al., 2012a]) provides a declarative framework for modeling combinatorial problems in Knowledge Representation and Reasoning. The unique pairing of declarativeness and performance in state-of-the-art ASP solvers allows for concentrating on an actual problem, rather than a smart way of implementing it. The basic idea of ASP is to express a problem in a logical format so that the models of its representation provide the solutions to the original problem. Problems are expressed as logic programs and the resulting models are referred to as answer sets. Although determining whether a program has an answer set is the fundamental decision problem in ASP, more reasoning modes are needed for covering the variety of reasoning problems encountered in applications. Hence, a modern ASP solver, like *clasp* [Gebser et al., 2012b] supports several reasoning modes for assessing the multitude of answer sets, among them, regular and projective enumeration, intersection and union, and multi-criteria optimization. As well, these reasoning modes can be combined, for instance, for computing the intersection of all optimal models. This is accomplished in several steps. At first, a logic program with first-order variables is turned

[1] Note that aforecited works are only some, among many, examples of such methods applied in the field.

into a propositional logic program by means of efficient database techniques. This is in turn passed to a solver computing the answer sets of the resulting program by using advanced Boolean constraint technology. For optimization, a solver like *clasp* uses usually branch-and-bound algorithms but other choices, like computing unsatisfiable cores, are provided as well. The enumeration of all optimal models is done via the option `--opt-mode=optN`. At first an optimal model is determined along with its optimum value. This computation has itself two distinct phases. First, an optimal model candidate must be found and second, it must be shown that there is no better candidate; the latter amounts to a proof of unsatisfiability and is often rather demanding (because of its exhaustive nature). Then, all models possessing the optimum score are enumerated. Notice that this way one can enumerate all (strictly) optimal solutions. Nonetheless, we are often interested in (nearly) optimal answer sets as well. For a concrete example on how we address this in practice, we refer the reader to the encoding provided in Listing 3.2 and its solving in Listing 3.4.

Our encodings are written in the input language of *gringo* 4 series. Such a language implements most of the so-called *ASP-Core-2* standard.[2] In what follows, we introduce its basic syntax and we refer the reader to the available documentation for more details. An *atom* is a predicate symbol followed by a sequence of terms (e.g. `p(a,b)`,`q(X,f(a,b))`). A *term* is a constant (e.g. `c`, `42`) or a function symbol followed by a sequence of terms (e.g. `f(a,b)`, `g(X,10)`) where uppercase letters denote first-order variables. Then, a rule is of the form

$$h \texttt{ :- } b_1, \ldots, b_n.$$

where h (head) is an atom and any b_j (body) is a literal of the form a or `not` a for an atom a where the connective `not` corresponds to default negation. The connectives `:-` and `,` can be read as *if* and *and*, respectively. Furthermore, a rule without body is a *fact*, whereas a rule without head is an *integrity constraint*. A logic program consists of a set of rules, each of which is terminated by a period. An atom preceded with default negation, `not`, is satisfied unless the atom is found to be true. In ASP, the semantics of a logic program is given by the *stable models semantics* [Gelfond and Lifschitz, 1988]. Intuitively, the head of a rule has to be true whenever all its body literals are true. This semantics requires that each true atom must also have some derivation, that is, an atom cannot be true if there is no rule deriving it. This implies that only atoms appearing in some head can appear in answer sets.

We end this quick introduction by three language constructs particularly interesting for our encodings. First, the so called *choice rule* of the form,

$$\{h_1; \ldots; h_m\} \texttt{ :- } b_1, \ldots, b_n.$$

allows us to express choices over subsets of atoms. Any subset of its head atoms can be included in an answer set, provided the body literals are satisfied. Note that using a choice rule one can easily *generate* an exponential search space of candidate solutions. Second, a

[2] `http://www.mat.unical.it/aspcomp2013/ASPStandardization`

conditional literal is of the form

$$l : l_1, \ldots, l_n$$

The purpose of this language construct is to govern the instantiation of the literal l through the literals $l_1, \ldots, l_n$. In this respect, the conditional literal above can be regarded as the list of elements in the set $\{l \mid l_1, \ldots, l_n\}$. Finally, for solving (multi-criteria) optimization problems, ASP allows for expressing (multiple) cost functions in terms of a weighted sum of elements subject to minimization and/or maximization. Such objective functions are expressed in *gringo* 4 in terms of (several) optimization statements of the form

$$\texttt{\#opt}\{w_1@l_1, t_{1_1}, \ldots, t_{m_1} : b_1, \ldots, b_{n_1}; \ldots; w_k@l_k, t_{1_k}, \ldots, t_{m_k} : b_{1_k}, \ldots, b_{n_k}\}.$$

where $\#opt \in \{\text{“}\#minimize\text{”}, \text{“}\#maximize\text{”}\}$, $w_i, l_i, t_{1_i}, \ldots, t_{m_i}$ are terms and $b_{1_i}, \ldots, b_{n_i}$ are literals for $k \geq 0, 1 \leq i \leq k, m_i \geq 0$ and $n_i \geq 0$. Furthermore, w_i and l_i stand for an integer *weight* and priority *level*. Priorities allow for representing lexicographically ordered optimization objectives, greater levels being more significant than smaller ones.

Answer Set Programming for systems biology. Our work contributes to a growing list of ASP applications in systems biology. Almost a decade ago, Baral et al. have proposed applying knowledge representation and reasoning methodologies to the problem of representing and reasoning about signaling networks [Baral et al., 2004]. More recently, several authors have addressed the question of pruning or identification of biological networks using ASP. Durzinsky et al. have studied the problem consisting of reconstructing all possible networks consistent with experimental time series data [Durzinsky et al., 2011]. Gebser et al. have addressed the problem consisting of detecting inconsistencies and repairing in large biological networks [Gebser et al., 2011b, 2010]. Fayruzov et al. have used ASP to represent the dynamics in Boolean networks and find their attractors [Fayruzov et al., 2011]. Ray et al. have integrated numerical and logical information in order to find the most likely states of a biological system under various constraints [Ray et al., 2012]. Furthermore, Ray et al. have used an ASP system to propose revisions to metabolic networks [Ray et al., 2010]. Papatheodorou et al. have used ASP to integrate RNA expression with signaling pathway information and infer how mutations affect ageing [Papatheodorou et al., 2012]. Finally, Schaub and Thiele have first investigated the metabolic network expansion problem with ASP [Schaub and Thiele, 2009] and recently, their work has been extendend and applied in a real-case study by Collet et al. [Collet et al., 2013].

Altogether, this series of contributions illustrates the potential of ASP to address combinatorial search and optimization problems appearing in the field. Nonetheless, its strictly discrete nature poses interesting challenges for future work towards hybrid reasoning system allowing for qualitative and quantitative modeling.

1.3 Original contribution

The contribution of this thesis is on three different axes. Firstly, on the modeling axis we introduce a novel mathematical framework for characterizing and reasoning on the response of logical signaling networks. Secondly, on the methodological axis we contribute to a growing list of successful applications of Answer Set Programming in systems biology. Thirdly, on the implementation axis we present a software providing a complete pipeline for automated reasoning on the response of logical signaling networks. More precisely, the main contributions of this thesis are:

- a mathematical framework for reasoning on the response of logical signaling networks relying on propositional logic and fixpoint semantics together with a basic ASP representation (Chapter 2)
- an ASP formulation to address the problem consisting of learning from an interaction graph and experimental data, (Boolean) logical networks describing the immediate-early response of the system (Chapter 3)
- a proposal for finding optimal experimental designs to discriminate between rival logical models. Furthermore, an ASP formulation to solve this problem is provided (Chapter 4)
- an ASP formulation to address the problem consisting of finding all inclusion-minimal intervention strategies for an ensemble of feasible logical networks (Chapter 5)
- a software toolbox providing an interface to the ASP-based solutions for each of the problems mentioned above (Chapter 6)

A first approach to the subject presented in Chapter 3 was published in [Videla et al., 2012], and an extended version is under revision for a special issue of the journal Theoretical Computer Science. Nonetheless, results presented in Chapter 3 are mostly based on the work published in [Guziolowski et al., 2013]. Therein, we have also provided an informal description for the problem addressed in Chapter 4 which has been properly developed in this thesis. The work in Chapter 5 is an unpublished extension of the paper presented in [Kaminski et al., 2013]. Finally, the software described in Chapter 6 was introduced first in [Guziolowski et al., 2013] but the version presented in this thesis contains several improvements and new features.

1.4 Organization of the thesis

The dissertation is organized in seven chapters, including the introduction and the conclusion. In Chapter 2 we provide a precise definition of logical networks, their graphical representation and related mathematical notions used throughout this thesis. Next, we introduce a mathematical characterization of the response in logical networks based on a *single-step* operator and fixpoint semantics. Based on this mathematical characterization, we introduce a generic

representation using ASP. Interestingly, such a representation is relatively simple yet flexible enough to be extended and adapted for specific applications as we illustrate in the subsequent chapters. In Chapter 3 we address the problem consisting of learning from an interaction graph and phosphorylation activities at a pseudo-steady state, (Boolean) logical networks describing the immediate-early response of the system. We provide a characterization of this problem using the notions introduced in Chapter 2 and adapting our ASP representation accordingly. Also, we show that there are multiple (possibly many) logical networks compatible with the experimental observations. In particular, this leads to logical input-output predictions suffering a significant level of uncertainty. Thus, in Chapter 4 we introduce a method to suggest new experiments for discrimination of input-output behaviors and generate models providing more reliable predictions. We provide a characterization of this problem using the notions introduced in previous chapters and adapting our ASP representation accordingly. In Chapter 5 we address the problem consisting of finding inclusion-minimal intervention strategies in logical signaling networks. Towards this end, we revisit and extend a previous definition of this problem using the notions introduced in Chapter 2, and adapting our ASP representation. In Chapter 6 we present a software package providing an interface to the ASP-based solutions detailed in previous chapters. We describe its high-level design, main features, and usage. Finally, in Chapter 7 we discuss the strengths and weaknesses of our modeling and computational methods. Furthermore, we conclude with prospective lines of research and future directions to explore questions raised by our work.

2 Generic framework for reasoning on the response of logical networks

Here we provide a precise definition of logical networks, their graphical representation and related mathematical notions used throughout this thesis. Next, following the investigations of [Inoue, 2011] we introduce a mathematical characterization of the response in logical networks based on a *single-step* operator and fixpoint semantics. Based on this mathematical characterization, we introduce a generic representation using Answer Set Programming. Interestingly, such a representation is relatively simple yet flexible enough to be extended and adapted for specific applications as we illustrate in the subsequent chapters.

2.1 Preliminaries

2.1.1 Propositional logic and mathematical notation

Given a finite set V of propositional variables, we form propositional formulas from V with the connectives $\bot$, $\top$, $\neg$, $\vee$, and $\wedge$ in the standard way. Further, we consider (partial) truth assignments over V mapping formulas to truth values $\{\boldsymbol{t}, \boldsymbol{f}, \boldsymbol{u}\}$ according to Kleene's semantics [Kleene, 1950]. Clearly, two-valued assignments are restricted to range $\{\boldsymbol{t}, \boldsymbol{f}\}$ according to classical (Boolean) logic semantics. We recall the truth tables for classical (Boolean) and Kleene's logics in Table 2.1 and Table 2.2, respectively.

Let $f : X \to Y$, be a (partial) function mapping values $x \in X' \subseteq X$ to values $y \in Y$. We denote the set of values x such that $f(x)$ is defined, i.e. X', with $dom(f)$. We sometimes represent mappings extensionally as sets, viz. $\{x \mapsto f(x) \mid x \in dom(f)\}$, for checking containment, dif-

Table 2.1: Truth tables for classical (Boolean) logic.

$a \wedge b$		b	
		$\boldsymbol{t}$	$\boldsymbol{f}$
a	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$
	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$

$a \vee b$		b	
		$\boldsymbol{t}$	$\boldsymbol{f}$
a	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$
	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$

a	$\neg a$
$\boldsymbol{t}$	$\boldsymbol{f}$
$\boldsymbol{f}$	$\boldsymbol{t}$

Table 2.2: Truth tables for Kleene's logic.

$a \wedge b$		b		
		$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$
a	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$
	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{u}$

$a \vee b$		b		
		$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$
a	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$
	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$
	$\boldsymbol{u}$	$\boldsymbol{t}$	$\boldsymbol{u}$	$\boldsymbol{u}$

a	$\neg a$
$\boldsymbol{t}$	$\boldsymbol{f}$
$\boldsymbol{f}$	$\boldsymbol{t}$
$\boldsymbol{u}$	$\boldsymbol{u}$

ference, etc. To avoid conflicts when composing two-valued truth assignments, we define $A \circ B = (A \setminus \overline{B}) \cup B$ where $\overline{B} = \{v \mapsto \overline{s} \mid v \mapsto s \in B\}$ and $\overline{\boldsymbol{t}} = \boldsymbol{f}$, $\overline{\boldsymbol{f}} = \boldsymbol{t}$.

2.2 Characterizing the response of logical networks

2.2.1 Logical networks and interaction graphs

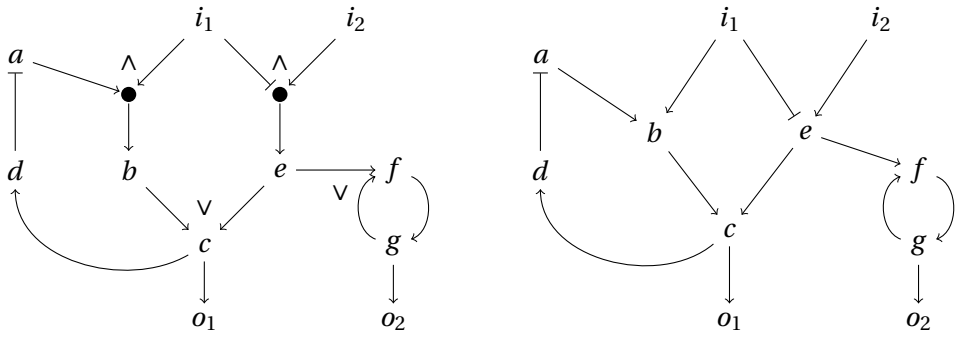
Logical networks. A *logical network* consists of a finite set V of propositional variables and a (partial) function ϕ mapping a variable $v \in V$ to a propositional formula $\phi(v)$ over V. The *logical steady states* of (V, ϕ) are given by truth assignments yielding identical values for v and $\phi(v)$ for all $v \in dom(\phi)$. Generally speaking, such logical networks can be seen as synchronous Boolean networks [Kauffman, 1969]. However, since we consider both, two- and three-valued logics, we refrain from using the term "Boolean". Without loss of generality, we assume only formulas in *disjunctive normal form*. [1] For illustration, let us consider the logical network consisting of the set V of species variables $\{i_1, i_2, a, \ldots, g, o_1, o_2\}$ along with the function ϕ defined as:

$$\phi = \left\{ \begin{array}{lll} a \mapsto \neg d & d \mapsto c & g \mapsto f \\ b \mapsto a \wedge i_1 & e \mapsto \neg i_1 \wedge i_2 & o_1 \mapsto c \\ c \mapsto b \vee e & f \mapsto e \vee g & o_2 \mapsto g \end{array} \right\}$$

Note that ϕ leaves the specification of the (input) variables i_1 and i_2 undefined. Furthermore, we represent logical networks as (signed) directed hypergraphs as shown in Figure 2.1a. A *(signed) directed hypergraph* is defined by a pair (V, H) with vertices V and (signed) directed hyperedges H; a (signed) directed hyperedge is a pair (S, t) where S is a finite, non-empty set of pairs (v_i, s_i) with $v_i \in V, s_i \in \{1, -1\}$ and $t \in V$. [2] Then, we say that the (signed) directed hypergraph (V, H) *represents* the logical network (V, ϕ) if and only if for every $v \in dom(\phi)$ and variable $w \in V$ that occurs positively (resp. negatively) in some conjunct ψ of $\phi(v)$, there is a hyperedge (S_ψ, v) with $(w, 1) \in S_\psi$ (resp. $(w, -1) \in S_\psi$); and vice versa. Following the example shown in Figure 2.1a, if we consider the mapping $\phi(e) = \neg i_1 \wedge i_2$, we need to verify the existence of the hyperedge $(S_{\neg i_1 \wedge i_2}, e)$ with $S_{\neg i_1 \wedge i_2} = \{(i_1, -1), (i_2, 1)\}$. Similarly, for the mapping $\phi(c) = b \vee e$, we need to verify the existence of the hyperedges (S_b, c) with $S_b = \{(b, 1)\}$ and (S_e, c) with $S_e = \{(e, 1)\}$.

[1] Also known as *sum-of-products* [Klamt et al., 2006b].

[2] More generally, a directed hyperedge is a pair (S, T) with $T \subseteq V$. We consider the particular case where T is a singleton. Directed hypergraphs are sometimes referred to as "AND/OR graphs" [Gallo et al., 1993]



(a) Exemplary logical network represented as a directed hypergraph. Directed hyperedges describe logical interactions

(b) Interaction graph underlying the logical network in (a). Directed edges describe causal interactions.

Figure 2.1: Graphical representation for logical networks and interaction graphs. Vertices represent biological entities and the interactions among them are represented as follows. Positive interactions are represented by an arrow ($\rightarrow$) whereas negative interactions are represented by a T-shape ($\dashv$).

Interaction graphs. Next, we introduce the notion of the interaction graph underlying a logical network (V,ϕ). An *interaction graph* (V,E,σ) is a signed and directed graph with vertices V, directed edges $E \subseteq V \times V$ and signature $\sigma \subseteq E \times \{1,-1\}$. Moreover, we say that $\Sigma_{(V,\phi)} = (V,E,\sigma)$ is the *underlying interaction graph* of (V,ϕ) if for every edge $(v,w) \in E$ with $((v,w),1) \in \sigma$ (resp. $((v,w),-1) \in \sigma$), the variable v occurs positively (resp. negatively) in the formula $\phi(w)$. Note that there is a *one-to-many* relation in the sense that the same graph (V,E,σ) corresponds to the underlying interaction graph $\Sigma_{(V,\phi)}$ for possibly many logical networks (V,ϕ). Now, we can rely on standard notions from graph theory to capture several concepts on logical networks. Recall that a path in a graph is a sequence of edges connecting a sequence of vertices. The length of a path is given by the number of edges whereas its sign is the product of the signs of the traversed edges. Herein, we consider only paths with length greater than zero. Thus, an edge (v,v) is required in order to consider the existence of a path from v to v. We say there is a positive (resp. negative) path from v to w in (V,ϕ) if and only if there is a positive (resp. negative) path from v to w in $\Sigma_{(V,\phi)}$. Furthermore, we say there is a positive (resp. negative) *feedback-loop* in (V,ϕ) if and only if for some $v \in V$ there is a positive (resp. negative) path from v to v in $\Sigma_{(V,\phi)}$.

2.2.2 Characterizing the response of the system

Clamping variables. Let (V,ϕ) be a logical network describing a biological system of interest. For capturing changes in the environment of such a biological system, for instance, due to an experimental intervention (over-expression or knock-out), we introduce the notion of clamping variables in the network for overriding their original specification. Towards this end, we define a *clamping* assignment C as a partial two-valued truth assignment over V. Then, we

define the mapping $\phi|_C$ as

$$\phi|_C(v) = \begin{cases} \top & \text{if } v \mapsto \boldsymbol{t} \in C \\ \bot & \text{if } v \mapsto \boldsymbol{f} \in C \\ \phi(v) & \text{if } v \in dom(\phi) \setminus dom(C) \end{cases}$$

yielding the modified logical network $(V, \phi|_C)$. Moreover, it is worth noting that $dom(\phi) \subseteq dom(\phi|_C)$. Let us illustrate this with our toy example in Figure 2.1a. Let C be the clamping assignment defined by $\{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$. Then, $\phi|_C$ is a complete mapping over V defined as

$$\phi|_C = \left\{ \begin{array}{lll} i_1 \mapsto \top & a \mapsto \neg d & d \mapsto c \\ i_2 \mapsto \bot & b \mapsto a \wedge i_1 & e \mapsto \neg i_1 \wedge i_2 \\ & c \mapsto b \vee e & f \mapsto e \vee g \end{array} \quad \begin{array}{lll} g \mapsto \bot \\ o_1 \mapsto c \\ o_2 \mapsto g \end{array} \right\}$$

In practice, clamping assignments are usually restricted to a subset of variables $X \subseteq V$. Moreover, certain variables in X may be further restricted to be clamped either to a single truth value, viz. $\boldsymbol{t}$ or $\boldsymbol{f}$, or not clamped at all. These restriction will be typically related to context-specific application settings, for instance, the kind of biological entity described by each variable and "real-world" experimental limitations over such entity.

Fixpoint semantics. Next, for capturing the synchronous updates in a logical network (V, ϕ), we define the following operator on either two- or three-valued (complete) truth assignments over V: [3]

$$\Omega_{(V,\phi)}(A) = \{v \mapsto A(\phi(v)) \mid v \in dom(\phi)\} \cup \{v \mapsto A(v) \mid v \in V \setminus dom(\phi)\}.$$

where A is extended to formulas in the standard way. Notice that the definition above captures the fact that unmapped variables in ϕ remain unchanged with respect to the value assigned in A. Furthermore, for capturing the trajectory of state A we define the iterative variant of $\Omega_{(V,\phi)}$ as

$$\Omega^0_{(V,\phi)}(A) = A \quad \text{and} \quad \Omega^{j+1}_{(V,\phi)}(A) = \Omega_{(V,\phi)}\left(\Omega^j_{(V,\phi)}(A)\right).$$

In biological terms, a sequence $(\Omega^j_{(V,\phi)}(A))_{j \in J}$ captures the signal propagation starting in state A. In particular, we are interested in the fixpoint of $\Omega_{(V,\phi)}$ reachable from certain initial assignment A. Importantly, the existence of such a fixpoint is not necessarily guaranteed. In general, it depends on the definition of A and the presence or absence of feedback-loops in (V, ϕ). But in case of existence, such a fixpoint describes a logical steady state which is interpreted as the response of the biological system described by (V, ϕ). To be more precise, the choice of A is related to how we model the absence of information in the context of either two- or three-valued logics. Hence, when we consider three-valued logics, we use the initial

[3] The interested reader may notice the resemblance to single-step operators for logic programs introduced in [Apt and Emden, 1982] and [Fitting, 1985] for two- and three-valued assignments respectively.

Table 2.3: Exemplary iterated application of $\Omega_{(V,\phi|_C)}$ for (V,ϕ) in Figure 2.1a, clamping assignment $C = \{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$ and initial assignment A with either $A = A_{\boldsymbol{u}}$ or $A = A_{\boldsymbol{f}}$.

	$\phi\|_C(v)$	$\top$	$\bot$	$\neg d$	$a \wedge i_1$	$b \vee e$	c	$\neg i_1 \wedge i_2$	$e \vee g$	$\bot$	c	g
	$v \in V$	i_1	i_2	a	b	c	d	e	f	g	o_1	o_2
3-valued	$\Omega^0_{(V,\phi\|_C)}(A_{\boldsymbol{u}})$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$
	$\Omega^1_{(V,\phi\|_C)}(A_{\boldsymbol{u}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{u}$
	$\Omega^2_{(V,\phi\|_C)}(A_{\boldsymbol{u}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{f}$
	$\Omega^3_{(V,\phi\|_C)}(A_{\boldsymbol{u}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{f}$
	$\Omega^4_{(V,\phi\|_C)}(A_{\boldsymbol{u}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{u}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{u}$	$\boldsymbol{f}$
2-valued	$\Omega^0_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\Omega^1_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\Omega^2_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\Omega^3_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\Omega^4_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$
	$\Omega^5_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$
	$\Omega^6_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$
	$\Omega^7_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$
	$\Omega^8_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$
	$\Omega^9_{(V,\phi\|_C)}(A_{\boldsymbol{f}})$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{f}$

assignment $A_{\boldsymbol{u}} = \{v \mapsto \boldsymbol{u} \mid v \in V\}$. Interestingly, in this context a fixpoint is reached regardless of the presence or absence of feedback-loops in the network. Moreover, such a fixpoint poses the property that each of its variables is assigned to $\boldsymbol{u}$ unless there is a cause to assign it to either $\boldsymbol{t}$ or $\boldsymbol{f}$. On the other hand, when we consider two-valued logics, we use the initial assignment $A_{\boldsymbol{f}} = \{v \mapsto \boldsymbol{f} \mid v \in V\}$. Unfortunately, in this context, the presence of negative feedback-loops typically avoids reaching a fixpoint [Remy et al., 2008, Paulevé and Richard, 2012].

Next, let us illustrate the iterated application of $\Omega_{(V,\phi|_C)}$ for our toy example in the context of both, two- and three-valued logics. Recall that we have defined $\phi|_C$ above for clamping assignment $C = \{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$. The resulting assignments from the computation of $\Omega^j_{(V,\phi|_C)}(A)$ with either $A = A_{\boldsymbol{u}}$ or $A = A_{\boldsymbol{f}}$ are shown in Table 2.3. Notably, when we consider three-valued logic, $\Omega^3_{(V,\phi|_C)}(A_{\boldsymbol{u}}) = \Omega^4_{(V,\phi|_C)}(A_{\boldsymbol{u}})$ results in the fixpoint

$$\{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, a \mapsto \boldsymbol{u}, b \mapsto \boldsymbol{u}, c \mapsto \boldsymbol{u}, d \mapsto \boldsymbol{u}, e \mapsto \boldsymbol{f}, f \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}, o_1 \mapsto \boldsymbol{u}, o_2 \mapsto \boldsymbol{f}\}\,.$$

Meanwhile, under two-valued logic we obtain $\Omega^1_{(V,\phi|_C)}(A_{\boldsymbol{f}}) = \Omega^9_{(V,\phi|_C)}(A_{\boldsymbol{f}})$ which leads to an oscillatory behavior for variables a, b, c, d and o_1. Notice that these variables correspond to the ones assigned to $\boldsymbol{u}$ in the fixpoint reached for $\Omega^3_{(V,\phi|_C)}(A_{\boldsymbol{u}})$. In this case, the oscillatory behavior is induced by the negative feedback-loop over the path a, b, c, d. Thus, one can verify that for example, if we remove the mapping $d \mapsto c$ from the definition of ϕ (leaving d undefined in ϕ),

then $\Omega^4_{(V,\phi|_C)}(A_{\boldsymbol{f}})$ would result in the fixpoint

$$\{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, a \mapsto \boldsymbol{t}, b \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}, d \mapsto \boldsymbol{f}, e \mapsto \boldsymbol{f}, f \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}, o_1 \mapsto \boldsymbol{t}, o_2 \mapsto \boldsymbol{f}\}\,.$$

In fact, in this thesis whenever we consider a logical network (V,ϕ) under two-valued logic, we enforce that (V,ϕ) is free of feedback-loops. Notably as detailed in Chapter 3, although not capable of capturing dynamical properties, this simplification guarantees the existence of a fixpoint while it allows us to characterize the so-called *immediate-early response* in signaling networks.

2.2.3 Logical networks and their response with Answer Set Programming

Logical networks. Let (V,ϕ) be a logical network. We represent the variables V as facts over the predicate `variable/1`, namely `variable(`v`)` for all $v \in V$. Recall that we assume $\phi(v)$ to be in disjunctive normal form for all $v \in V$. Hence, $\phi(v)$ is a set of clauses and a clause a set of literals. We represent formulas using predicates `formula/2`, `dnf/2`, and `clause/3`. The facts `formula(`v`,`$s_{\phi(v)}$`)` map variables $v \in V$ to their corresponding formulas $\phi(v)$, facts `dnf(`$s_{\phi(v)}$`,`s_ψ`)` associate $\phi(v)$ with its clauses $\psi \in \phi(v)$, facts `clause(`s_ψ`,`v`,1)` associate clause ψ with its positive literals $v \in \psi \cap V$, and facts `clause(`s_ψ`,`v`,-1)` associate clause ψ with its negative literals $\neg v \in \psi$. Note that each $s_{(\cdot)}$ stands for some arbitrary but unique name in its respective context here. Listing 2.1 shows the representation of our toy example logical network in Figure 2.1a.

Listing 2.1: Logical networks representation as logical facts

```
variable(i1). variable(i2). variable(o2). variable(o1). variable(a).
variable(b).  variable(c).  variable(d).  variable(e).  variable(f).
variable(g).

formula(a,0). formula(b,2). formula(c,1).  formula(d,4). formula(e,3).
formula(f,6). formula(g,5). formula(o1,4). formula(o2,7).

dnf(0,5). dnf(1,6). dnf(1,0). dnf(2,3). dnf(3,7).
dnf(4,1). dnf(5,2). dnf(6,4). dnf(6,6). dnf(7,4).

clause(0,b,1).  clause(1,c,1).  clause(2,f,1).  clause(3,a,1).
clause(3,i1,1). clause(4,g,1).  clause(5,d,-1). clause(6,e,1).
clause(7,i2,1). clause(7,i1,-1).
```

Clamping variables. The representation of clamping assignments is straightforward. Note that in the following we use 1 and -1 for truth assignments to $\boldsymbol{t}$ and $\boldsymbol{f}$, respectively. Let C be a clamping assignment over V, we represent the assignments in C as facts over the predicate `clamped/2`, namely `clamped(`v`,`s`)` with $s = 1$ if $C(v) = \boldsymbol{t}$ and $s = -1$ if $C(v) = \boldsymbol{f}$. The example clamping assignment $C = \{i_1 \mapsto \boldsymbol{t}, i_2 \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$ is shown in Listing 2.2.

Listing 2.2: Clamping assignment as logical facts

```
14 clamped(i1,1). clamped(i2,-1). clamped(g,-1).
```

Furthermore, we introduce two rules deriving predicates `eval/2` and `free/2`. The predicate `eval/2` captures the fact that clamped variables are effectively fixed to the corresponding evaluation. Finally, we use the predicate `free/2` to represent the fact that every variable not clamped in C, is subject to the corresponding mapping $\phi(v)$. Both rules are shown in Listing 2.3.

Listing 2.3: Clamped and free variables

```
15 eval(V,S)  :- clamped(V,S).
16 free(V,D)  :- formula(V,D); dnf(D,_); not clamped(V,_).
```

Two- and three-valued logics. Next, we describe how we model either two- or three-valued logics in ASP. In fact, the rule modeling the propagation of (positive) true values is the same for both logics. Essentially, we exploit the fact that formulas $\phi(v)$ are in disjunctive normal form. Hence, under both logics we derive `eval(`v`,1)` if v is not clamped and there exists a conjunct $\psi \in \phi(v)$ such that all its literals evaluate positively. The rule describing this is shown in Listing 2.4.

Listing 2.4: Positive propagation common to two- and three-valued logics

```
17 eval(V,1) :- free(V,D); eval(W,T) : clause(J,W,T); dnf(D,J).
```

Meanwhile, the propagation of (negative) false values depends on the type of logic under consideration. On one hand, when we consider two-valued logic, we use the rule shown in Listing 2.5 to derive `eval(`v`,`-1`)` if it cannot be proved that v evaluates positively, that is, `not eval(`v`,1)`.

Listing 2.5: Negative propagation for two-valued logic (with default negation)

```
18 eval(V,-1) :- variable(V); not eval(V,1).
```

On the other hand, when we consider three-valued logic, we use the rules shown in Listing 2.6. Notice that in this case, we derive `eval(`v`,`-1`)` only if it can be proved that all clauses $\psi \in \phi(v)$ evaluate negatively. A clause ϕ evaluates negatively if at least one of its literals evaluates negatively. Clauses evaluating negatively are represented with the predicate `eval_clause/2`.

Listing 2.6: Negative propagation for three-valued logic (with explicit proof)

```
18 eval_clause(J,-1) :- clause(J,V,S); eval(V,-S).
19 eval(V,-1)        :- free(V,D); eval_clause(J,-1) : dnf(D,J).
```

Correctness of the ASP representation. The following two propositions show the correctness of our ASP representation for finding the fixpoint under either two- or three-valued logics, respectively. We refer the reader to the appendix for detailed proofs. Let us introduce the used notation first. For a logical network (V,ϕ) and clamping assignment C over V, let us denote with $\tau((V,\phi),C)$ the set of logical facts as in Listing 2.1 and Listing 2.2. Furthermore, for $x = 3,\ldots,6$ let us denote with $\Pi_{2.x}$ the set of rules defined in Listing 2.x.

Proposition 2.2.1. *Let (V,ϕ) be a logical network without feedback-loops and let C be a clamping assignment over V.*
Then, there is an answer set X of $\tau((V,\phi),C)\cup\Pi_{2.3}\cup\Pi_{2.4}\cup\Pi_{2.5}$ such that $F = \{v \mapsto \boldsymbol{t} \mid \texttt{eval(v,1)} \in X\} \cup \{v \mapsto \boldsymbol{f} \mid \texttt{eval(v,-1)} \in X\}$ if and only if F is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{f}}$.

Proposition 2.2.2. *Let (V,ϕ) be a logical network and let C be a clamping assignment over V.*
Then, there is an answer set X of $\tau((V,\phi),C) \cup \Pi_{2.3} \cup \Pi_{2.4} \cup \Pi_{2.6}$ such that such that $F = \{v \mapsto \boldsymbol{t} \mid \texttt{eval(v,1)} \in X\} \cup \{v \mapsto \boldsymbol{f} \mid \texttt{eval(v,-1)} \in X\} \cup \{v \mapsto \boldsymbol{u} \mid \texttt{eval(v,1)} \notin X, \texttt{eval(v,-1)} \notin X\}$ if and only if F is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{u}}$.

2.3 Conclusion

In this chapter we have introduced the fundamental mathematical notions used throughout this thesis. More precisely, we have characterized the response of logical networks under either two- or three-valued logics based on fixpoint semantics. Furthermore, a representation using Answer Set Programming which allow for modeling logical networks and perform automated reasoning on their response has been provided. Notably, such a representation can be easily elaborated to consider specific application settings. For example, in Chapter 3 we extend our representation to learn logical networks from a given interaction graph confronting their response with experimental observations. Moreover, we consider several clamping assignments simultaneously instead of only one. In Chapter 4 we elaborate upon our representation in order to search for clamping assignments leading to different responses among several logical networks. In that context, the clamping assignments found are interpreted as experiments which would allow to discriminate the logical networks at hand. Finally, in Chapter 5 we adapt our representation again aiming at reasoning over a family of logical networks and finding clamping assignments leading to responses satisfying specific goals. Altogether, this constitutes a generic, flexible, and unified framework for modeling logical networks and perform automated reasoning on their response.

3 Learning Boolean logic models of immediate-early response

The manual identification of logic rules underlying a biological system is often hard, error-prone and time consuming. Therefore, automated inference of (Boolean) logical networks from experimental data is a fundamental question towards the construction of large-scale predictive models. This chapter addresses the problem consisting of learning from an interaction graph and phosphorylation activities at a pseudo-steady state, (Boolean) logical networks describing the immediate-early response of the system.

3.1 Introduction

In what follows, we briefly summarize the main biological hypotheses in [Saez-Rodriguez et al., 2009] providing the foundation for the concept of Boolean logic models of immediate-early response. Concretely, a *Boolean logic model of immediate-early response* is a logical network (V, ϕ) as defined in Chapter 2, without feedback-loops and using classical (Boolean) logics.

The main assumption under Boolean logic models as treated in [Saez-Rodriguez et al., 2009] is the following. The response of a biological system to external perturbations occurs at several time scales [Papin et al., 2005]. Thus, one can discriminate between fast and slow events. Under this assumption, at a given time after perturbation, the system reaches a state on which fast events are relevant, but slow events (such as protein degradation) have a relatively insignificant effect. In this context, we say that the system has reached a *pseudo-steady state* describing the early events or immediate-early response. Qualitatively, these states can be computed as logical steady states in the Boolean network (V, ϕ) [Klamt et al., 2006b]. In fact, the discrimination between fast and slow events has an important consequence. Since we focus on fast or early events, it is assumed that oscillation or multi-stability caused by feedback-loops [Remy et al., 2008, Paulevé and Richard, 2012] cannot happen until the second phase of signal propagation occurring at a slower time scale. Therefore, feedback-loops are not included in Boolean logic models of immediate-early response assuming that they will become active in a late phase [Macnamara et al., 2012]. Notably, it follows that starting from any initial state, a Boolean logic model of immediate-early response reaches a unique steady state or fixpoint

in polynomial time [Paulevé and Richard, 2012]. Thus, such modeling approach, although not capable of capturing dynamical properties, provides a relatively simple framework for input-output predictive models.

Learning Boolean logic models of immediate-early response. Nowadays, for certain biological systems, a graph of causal interactions describing a large-scale signaling network can be retrieved from public databases [Guziolowski et al., 2012] or by means of statistical methods and experimental data [Sachs et al., 2005]. However, functional relationships in signaling networks cannot be captured by means of graph theory only [Klamt et al., 2006b]. In this context and based on the assumptions described above, authors in [Saez-Rodriguez et al., 2009] have proposed a method to learn from an interaction graph and phosphorylation activities at a pseudo-steady state, Boolean logic models of immediate-early response fitting experimental data. Originally, a genetic algorithm implementation was proposed to solve the underlying optimization problem, and a software was provided, CellNOpt [Terfve et al., 2012]. Nonetheless, stochastic search methods cannot characterize the models precisely: they are intrinsically unable not just to provide a complete set of solutions, but also to guarantee that an optimal solution is found. Some variations of our problem were addressed using a mathematical programming approach [Mitsos et al., 2009, Sharan and Karp, 2013]. Despite their success to overcome some shortcomings of the genetic algorithm, such as performance and global optimality, the enumeration of all (nearly) optimal solutions was not considered.

More generally, the inference of Boolean networks from time-series gene expression data has been addressed by several authors under different hypotheses and methods [Liang et al., 1998, Akutsu et al., 2000, Ideker et al., 2000, Lähdesmäki et al., 2003]. Recently, some of these methods have been compared in [Berestovsky and Nakhleh, 2013]. Nonetheless, overall, our work presents some significant differences. To start with, all of them are focused on gene regulatory networks and gene expression time-series data, whereas we work on signaling transduction networks and phosphorylation activities at a pseudo-steady state. Further, they work only with Boolean experimental observations which would correspond to adopt a binary discretization scheme in our framework. Moreover, except for the so-called *Best-Fit Extension Problem* [Lähdesmäki et al., 2003], they look for Boolean networks fully consistent with the time-series Boolean data. Meanwhile, herein we consider an objective function which describes the goodness of the model based on the numerical data that is subsequently optimized. Finally, all these contributions focus on a "local" inference in the following sense. They aim at learning the Boolean function for each node based on (local) input-output behaviors for such node. In contrast, our learning is based on (global) behaviors over the input-output layers in a network containing non-controllable/non-observable species.

In the remainder of this chapter we provide a precise characterization of this problem using the notions introduced in Chapter 2 and adapting our Answer Set Programming representation accordingly. Furthermore, we validate our approach using real-world signaling pathways in human liver cells and publicly available experimental data.

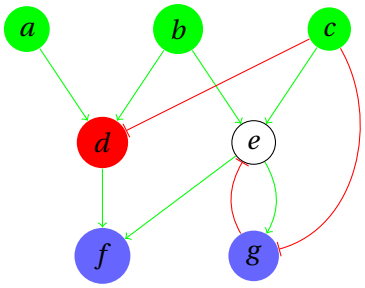
3.2 Problem

3.2.1 Prior knowledge network and phospho-proteomics dataset

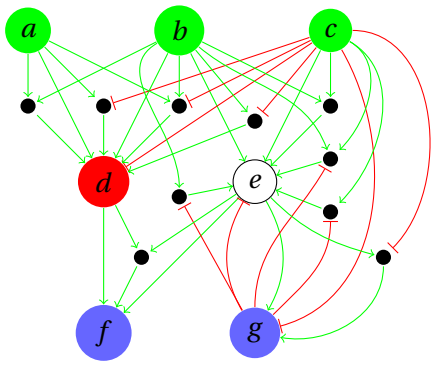
Prior knowledge network. A *prior knowledge network* is an interaction graph (V, E, σ) as defined in Chapter 2. In addition, we distinguish three special subsets of species in V namely, the *stimuli* (V_S), the *knock-outs* (V_K) and the *readouts* (V_R). Nodes in V_S denote extracellular ligands that can be stimulated and thus, we assume they have indegree equal to zero. Nodes in V_K denote intracellular species that can be inhibited or knocked-out by various experimental tools such as small-molecule drugs, antibodies, or RNAi. Finally, nodes in V_R denote species that can be measured by using an antibody. Notably, species in none of these sets, are neither measured, nor manipulated for the given experimental setup. Let us denote with V_U the set of such nodes. Then, except for V_R and V_K that may intersect, the sets V_S, V_K, V_R and V_U are pairwise mutually disjoint. An early simplification consists on compressing the PKN in order to collapse most of the nodes in V_U. This often results on a significant reduction of the search space that must be explored during learning. Thus, herein we assume a compressed PKN as an input and we refer the interested reader to [Saez-Rodriguez et al., 2009] for more details on this subject.

Phospho-proteomics dataset. Given a PKN (V, E, σ), the concept of an *experimental condition* over (V, E, σ) is captured by a clamping assignment over variables $V_S \cup V_K$. Recall that clamping assignments were defined in Chapter 2 as partial two-valued assignments. To be more precise, while variables in V_S can be clamped to either $\boldsymbol{t}$ or $\boldsymbol{f}$, variables in V_K can only be clamped to $\boldsymbol{f}$. Next, if C is an experimental condition and $v \in V_S$, then $C(v) = \boldsymbol{t}$ (resp. $\boldsymbol{f}$) indicates that the stimulus v is present (resp. absent), while if $v \in V_K$, then $C(v) = \boldsymbol{f}$ indicates that the species v is inhibited or knocked out. In fact, since extracellular ligands by default are assumed to be absent, for the sake of simplicity we can omit clampings to $\boldsymbol{f}$ over variables in V_S. Therefore, if C is an experimental condition and $v \in dom(C)$ then, either $v \in V_S$ and $C(v) = \boldsymbol{t}$, or $v \in V_K$ and $C(v) = \boldsymbol{f}$. Furthremore, the concept of an *experimental observation* under an experimental condition C is captured by a partial mapping $P_C : V_R \mapsto [0, 1]$. That is, $dom(P_C) \subseteq V_R$ denotes the set of measured readouts under the experimental condition C. If $v \in dom(P_C)$, then $P_C(v)$ represents the phosphorylation activity (at a pseudo-steady state) of the readout v under C. Notably, it is rather critical to choose a time point that is characteristic for the fast or early events in the biological system under consideration [Macnamara et al., 2012]. Since phosphorylation assays represents an average across a population of cells, the phosphorylation activity for each readout is usually normalized to $[0, 1]$. Finally, an *experimental dataset* ξ is a finite set of pairs (C_i, P_{C_i}) with experimental conditions C_i and experimental observations P_{C_i}. Further, we denote with N_ξ the *size* of ξ given by the number of measured readouts across all experimental conditions $i = 1, \ldots, n$, i.e., $N_\xi = \sum_{i=1}^{n} \left| dom(P_{C_i}) \right|$.

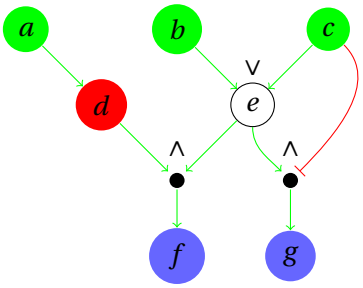
Let us illustrate the concepts described above with our toy example in Figure 3.1. Consider the PKN (V, E, σ) defined in Figure 3.1a. From the graph coloring, we have $V_S = \{a, b, c\}$, $V_K = \{d\}$



(a) Exemplary interaction graph describing an imaginary prior knowledge network (PKN).



(b) Hypergraph expansion describing all plausible logical interactions for the PKN in (a).



(c) An arbitrary Boolean logic model (V,ϕ) derived from the PKN in (a), i.e. pruning the hypergraph in (b). Logical interactions in (V,ϕ) correspond to the mapping $\phi = \{d \mapsto a; e \mapsto b \vee c; f \mapsto d \wedge e; g \mapsto e \wedge \neg c\}$.

Figure 3.1: Learning Boolean logic models of immediate-early response. The green and red edges correspond to activations and inhibitions, respectively. Green nodes represent ligands that can be experimentally stimulated (V_S). Red nodes represent species that can be inhibited or knocked out (V_K). Blue nodes represent species that can be measured (V_R). White nodes are neither measured, nor manipulated (V_U).

and $V_R = \{f,g\}$. Furthermore, let $\xi = ((C_1,P_{C_1}),\ldots,(C_4,P_{C_4}))$ be an example experimental dataset over (V,E,σ) defined by

$$\begin{aligned} C_1 &= \{a \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}\} & P_{C_1} &= \{f \mapsto 0.9, g \mapsto 0.0\} \\ C_2 &= \{a \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}, d \mapsto \boldsymbol{f}\} & P_{C_2} &= \{f \mapsto 0.1, g \mapsto 0.9\} \\ C_3 &= \{a \mapsto \boldsymbol{t}\} & P_{C_3} &= \{f \mapsto 0.0, g \mapsto 0.1\} \\ C_4 &= \{a \mapsto \boldsymbol{t}, b \mapsto \boldsymbol{t}\} & P_{C_4} &= \{f \mapsto 1.0, g \mapsto 0.8\}. \end{aligned} \tag{3.1}$$

In words, the experimental conditions $C_1,\ldots,C_4$ can be read as follows. In C_1, stimuli a and c are present, stimulus b is absent and d is not inhibited; in C_2, stimuli a,b,c are like in C_1 but d is inhibited; in C_3, only the stimulus a is present and d is not inhibited; and in C_4, stimuli a and b are present, stimulus c is absent and d is not inhibited. Experimental observations $P_{C_1},\ldots,P_{C_4}$ give (normalized) phosphorylation activities for readouts f and g under the corresponding experimental condition.

3.2.2 Boolean input-output predictions

Let (V,E,σ) be a PKN. Further, let $\xi = (C_i, P_{C_i})$ be an experimental dataset over (V,E,σ) with $i = 1,\ldots,n$. As detailed above, a Boolean logic model of immediate-early response is defined by a logical network (V,ϕ) without feedback-loops and using classical (Boolean) logics. Hence, now we can define the predictions (output) provided by a Boolean logic model of immediate-early response with respect to a given set of experimental conditions (input). Towards this end, we rely on our framework introduced in Chapter 2 and characterize the response of logical networks using fixpoint semantics. More precisely, for $i = 1,\ldots,n$ let F_i be the fixpoint of $\Omega_{(V,\phi|_{C_i})}$ reachable from $A_{\boldsymbol{f}} = \{v \mapsto \boldsymbol{f} \mid v \in V\}$. Notice that such a fixpoint always exists given that (V,ϕ) is free of feedback-loops. In words, each F_i describe the logical response (starting from $A_{\boldsymbol{f}}$) of (V,ϕ) with respect to the experimental condition or clamping assignment C_i. Next, we define a straightforward transformation from truth values to binary but numerical values. Such a transformation provides a more convenient notation in order to compare predictions and phosphorylation activities. The *Boolean prediction* of (V,ϕ) with respect to the experimental condition C_i is a function $\pi_i : V \to \{0,1\}$ defined as

$$\pi_i(v) = \begin{cases} 1 & \text{if } F_i(v) = \boldsymbol{t} \\ 0 & \text{if } F_i(v) = \boldsymbol{f}. \end{cases}$$

As we see below, during learning we aim at explaining the given experimental dataset ξ. Therefore, we are particularly interested in predictions with respect to the experimental conditions included in ξ and over the measured variables in each experimental condition. Nevertheless, predictions with respect to non-performed experimental conditions and/or over non-observed species can be useful to generate testable hypotheses on the response of the system.

As an example, consider the Boolean logic model (V,ϕ) from Figure 3.1c and the experimental condition C_2 from the dataset given in (3.1). Then, the clamped logical network $(V,\phi|_{C_2})$ is defined by the mapping

$$\phi|_{C_2} = \{a \mapsto \top; b \mapsto \bot; c \mapsto \top; d \mapsto \bot; e \mapsto b \vee c; f \mapsto d \wedge e; g \mapsto e \wedge \neg c\}.$$

Next, the fixpoint of $\Omega_{(V,\phi|_{C_2})}$ reachable from $A_{\boldsymbol{f}}$ can be computed as detailed in Chapter 2 yielding the assignment F_2 defined as

$$F_2 = \{a \mapsto \boldsymbol{t}, b \mapsto \boldsymbol{f}, c \mapsto \boldsymbol{t}, d \mapsto \boldsymbol{f}, e \mapsto \boldsymbol{t}, f \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}.$$

Finally, the Boolean prediction for (V,ϕ) under the experimental condition C_2 is given by

$$\pi_2 = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 0, e \mapsto 1, f \mapsto 0, g \mapsto 0\}.$$

3.2.3 Learning Boolean logic models

Search space. We aim at learning Boolean logic models from a PKN and an experimental dataset. In fact, any learned model has to be supported by some *evidence* in the prior knowledge. To be more precise, given a PKN (V,E,σ) we consider only Boolean logic models (V,ϕ) without feedback-loops and such that, for each variable $v \in V$, if w occurs positively (resp. negatively) in $\phi(v)$ then, there exists an edge $(w,v) \in E$ and $((w,v),1) \in \sigma$ (resp. $((w,v),-1) \in \sigma$). Towards this end, we consider a pre-processing step where the given PKN is expanded to generate a (signed) directed hypergraph describing all plausible logical interactions. For each $v \in V$ having non-zero indegree, let $Pred(v)$ be the set of its (signed) predecessors, namely, $Pred(v) = \{(u,s) \mid (u,v) \in E, ((u,v),s) \in \sigma\}$. Furthermore, let $\mathscr{P}(v)$ be the powerset of $Pred(v)$, namely, $2^{Pred(v)}$. Then, (V,H) is the (signed) directed hypergraph *expanded* from (V,E,σ) with nodes V and (signed) directed hyperedges H if for each $v \in V$, $(p,v) \in H$ whenever $p \in \mathscr{P}(v)$. Next, Boolean logic models must essentially result from pruning (V,H). Additionally, we impose two constraints related to the fact that our Boolean logic models essentially aim at providing a framework for input-output predictions.[1] Firstly, for any variable u defined in ϕ all variables $w \in \phi(u)$ must be *reachable* from some stimuli variable. That is, we consider only Boolean logic models (V,ϕ) such that for every $u \in dom(\phi)$ and $w \in \phi(u)$, either $w \in V_S$ or there exist $v \in V_S$ and a path from v to w in the underlying interaction graph $\Sigma_{(V,\phi)}$. Secondly, every variable u defined in ϕ must *reach* some readout variable. That is, we consider only Boolean logic models (V,ϕ) such that for every $u \in dom(\phi)$ there exist $v \in V_R$ and a path from u to v in the underlying interaction graph $\Sigma_{(V,\phi)}$. Finally, let us denote with $\mathbb{M}_{(V,E,\sigma)}$ the *search space* of Boolean logic models satisfying the conditions given above: evidence in (V,E,σ), no feedback-loops and reachability from/to stimuli/readouts.

In Figure 3.1 we show an exemplary PKN (Figure 3.1a) and the corresponding expanded (signed) directed hypergraph (Figure 3.1b). As described in Chapter 2, (signed) directed hypergraphs can be directly linked to logical networks. Thus, by considering each (signed) directed hyperedge in Figure 3.1b as either present or absent (and verifying the additional constraints with respect to feedback-loops and reachability from/to stimuli/readouts), one can generate the search space of Boolean logic models $\mathbb{M}_{(V,E,\sigma)}$.

Lexicographic multi-objective optimization. For a given PKN (V,E,σ), there are exponentially many candidate Boolean logic models (V,ϕ) having an evidence on it. Therefore, authors in [Saez-Rodriguez et al., 2009] put forward the idea of training Boolean logic models by confronting their corresponding Boolean predictions with phosphorylation activities at a pseudo-steady state. In this context, two natural optimization criteria arise in order to conduct the learning: (1) model accuracy (biologically meaningful), and (2) model complexity (Occam's razor principle). In fact, this is a typical scenario on automatized learning of predictive models [Freitas, 2004].

[1] The interested reader may notice the analogy with the elimination of nodes neither controllables nor (leading to) observables during the compression of the PKN described in [Saez-Rodriguez et al., 2009].

We now provide the precise formulation for each optimization criteria. Let (V,E,σ) be a PKN. Let $\xi = (C_i, P_{C_i})$ be an experimental dataset over (V,E,σ) with $i = 1,\ldots,n$. Let (V,ϕ) be a Boolean logic model having evidence in (V,E,σ) and let $\pi_1,\ldots,\pi_n$ be its Boolean predictions with each π_i defined under C_i. Firstly, based on the residual sum of squares (RSS) we define the *residual* (Θ_{rss}) of (V,ϕ) with respect to ξ as

$$\Theta_{rss}((V,\phi),\xi) = \sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} (P_{C_i}(v) - \pi_i(v))^2. \tag{3.2}$$

Secondly, for a given logical formula $\phi(v)$, let us denote its length by $|\phi(v)|$. Then, we define the *size* (Θ_{size}) of (V,ϕ) as

$$\Theta_{size}((V,\phi)) = \sum_{v \in dom(\phi)} |\phi(v)|. \tag{3.3}$$

A popular and relatively simple approach to cope with multi-objective optimization is to transform it into a single-objective optimization. Towards this end, one usually combines all criteria by defining a function using free parameters in order to assign different weights to each criterion. In fact, this is exactly the approach adopted in [Saez-Rodriguez et al., 2009]. Therein, a single-objective function is defined that balances *residual* and *size* using a parameter α chosen to maximize the predictive power of the model. Moreover, it has been shown that "predictive power" is best for $\alpha < 0.1$. However, as detailed in [Freitas, 2004], this approach suffers from known drawbacks. First, it depends on "magic values" for each weight often based on intuition or empirically determined. Second, it combines different scales of measurements that need to be normalized. Third, it combines non-commensurable criteria producing meaningless quantities. On the other hand, the lexicographic approach allows us to assign different priorities to different objectives in a qualitative fashion. Notably, in our context logic models providing high predictive power are significantly more relevant than the sizes of such models. Thus, the lexicographic approach is very convenient to cope with the multi-objective nature of our optimization problem. Yet another popular approach is to look for Pareto optimal models. However, this method will lead to a large number of models providing either none or very low predictive power. For example, consider the Boolean logic model (V,ϕ) with $\phi = \emptyset$, i.e. the *empty* model. Such a model is trivially consistent with any input PKN (V,E,σ) while it minimizes the objective function *size*, i.e. $\Theta_{size}((V,\phi)) = 0$. Therefore, (V,ϕ) is Pareto optimal although it does not provide any valuable information. Similarly, one can show that many other (*non-empty*) models will be Pareto optimal as well although they provide very low predictive power. Hence, Pareto optimality is not well suited for our problem. Notwithstanding, other multi-objective optimization methods (cf. [Marler and Arora, 2004]) could be investigated in the future. To conclude, our lexicographic multi-objective optimization consists of minimizing first Θ_{rss}, and then with lower priority Θ_{size}:

$$(V,\phi_{opt}) \in \underset{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}}{\operatorname{argmin}} \; (\Theta_{rss}((V,\phi),\xi), \Theta_{size}((V,\phi))). \tag{3.4}$$

Enumeration of (nearly) optimal models. Information provided by high-throughput data is intrinsically uncertain due to experimental errors. Therefore, one is not only interested in optimal models but in *nearly* optimal models as well. In this context, authors in [Saez-Rodriguez et al., 2009] have considered Boolean logic models minimizing their objective function within certain tolerance, e.g. 10% of the minimum. Next, they argue that all models found can explain the data similarly or equally well if one take into account the experimental error. Notice that, in the aforecited work the optimization is addressed using a genetic algorithm. Hence, "minimum" refers to the one found during the execution of the algorithm which is not necessarily the global minimum. Moreover, due to the incompleteness of stochastic search methods, it is very likely that certain solutions within the allowed tolerance are not found. In practice, one can execute the genetic algorithm several times in order to overcome this issue to some extent. Nonetheless, as we have shown in [Guziolowski et al., 2013], a significant number of models may be missing even after several executions. Similarly but in the context of quantitative modeling (based on ordinary differential equations) and using a simulated annealing algorithm, authors in [Chen et al., 2009] have elaborated upon the same argument. Interestingly, despite the fact that the model appears to be non-identifiable in both contexts, viz. qualitative and quantitative modeling, biologically relevant insights have been reported in the two aforecited studies. Notably, minimization over size in (3.4) is based on Occam's razor principle. On the one hand, one can consider that larger logic models overfit the available dataset by introducing excessive complexity [Saez-Rodriguez et al., 2009, Prill et al., 2011]. On the other hand, one can argue that it is actually necessary to consider such "spurious" links in order to capture cellular robustness and complexity [Stelling et al., 2004]. Therefore, let (V,ϕ_{opt}) be a Boolean logical model as defined in (3.4). Then, considering that tolerance over residual and size may yield biologically relevant models, we are particularly interested in enumerating all (nearly) optimal Boolean logic models (V,ϕ) such that,

$$\Theta_{rss}((V,\phi),\xi) \leq \Theta_{rss}((V,\phi_{opt}),\xi) + t_{rss} \qquad \Theta_{size}((V,\phi)) \leq \Theta_{size}((V,\phi_{opt})) + t_{size}$$

with t_{rss} and t_{size} denoting the tolerance over residual and size, respectively.

Logical input-output behaviors. Next, we introduce the notion of logical input-output behaviors. As we show below, in practice the enumeration of (nearly) optimal models often leads to a large number of logical networks, namely, (V,ϕ_j) with $j = 1,\ldots,m$ and $m \gg 1$. Notably, each ϕ_j is a different mapping from variables to propositional formulas. However, it may happen (and it often happens) that for all $v \in V_R$, several logical networks describe exactly the same response to every possible experimental condition (clamping assignments over variables $V_S \cup V_K$). In such a case, we say that those logical networks describe the same input-output behavior. To be more precise, recall that we consider a PKN (V,E,σ). Notice that in each experimental condition over (V,E,σ), every stimulus $v \in V_S$ and inhibitor $v \in V_K$, can be either clamped or not. Thus, let us denote with $\mathscr{C}$ the space of all possible clamping assignments or experimental conditions C_i over (V,E,σ). Notably, the number of possible clamping assignments is given by $|\mathscr{C}| = 2^{|V_S|+|V_K|}$. Then, let $(V,\phi_j),(V,\phi_{j'})$ be two

(nearly) optimal Boolean logic models. Furthermore, let F_i^j and $F_i^{j'}$ be the fixpoints of $\Omega_{(V,\phi_j|_{C_i})}$ and $\Omega_{(V,\phi_{j'}|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$, respectively. We say that (V,ϕ_j) and $(V,\phi_{j'})$ describe the same *logical input-output behavior* if and only if $F_i^j(v) = F_i^{j'}(v)$ for all $v \in V_R$ and $C_i \in \mathscr{C}$. Importantly, this abstraction allows us to group logical networks regardless of their "internal wirings" and focus on their input-output predictions. In practice, this also facilitates the analysis and interpretation of results whereas it provides a way to extract robust insights despite the high variability.

3.3 Learning Boolean logic models with Answer Set Programming

In order to express and solve the multi-objective optimization described in (3.4) by using ASP, one needs to discretize the function defined in (3.2). A very simple approach converts numerical data into binary data according to a threshold. Furthermore, we propose a finer multi-valued discretization scheme. In fact, the only non-integer variables in (3.2) are the experimental observations $P_{C_i}(v)$. Then, we approximate these values up to $\frac{1}{10^k}$ introducing a parametrized approximation function δ_k (e.g. using the floor or closest integer functions). Next, we define the discrete residual Θ_{rss_k} as

$$\Theta_{rss_k}((V,\phi),\xi) = \sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} \left[10^k \delta_k(P_{C_i}(v)) - 10^k \pi_i(v)\right]^2. \tag{3.5}$$

The minimizations of Θ_{rss} and Θ_{rss_k} may yield different Boolean logic models. Nonetheless, one can prove that finding all models minimizing Θ_{rss_k} within a certain tolerance allows us to find all models minimizing Θ_{rss} as well. To be more precise, one proves the following result.

Proposition 3.3.1. *Let (V,E,σ) be a PKN. Let ξ be an experimental dataset over (V,E,σ) with size N_ξ. Let $k \in \mathbb{N}$ define the discretization scheme. Let us denote with μ and μ_k, the corresponding minima for Θ_{rss} and Θ_{rss_k} over the space of models $\mathbb{M}_{(V,E,\sigma)}$ with respect to ξ:*

$$\mu = \min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} \Theta_{rss}((V,\phi),\xi) \qquad \mu_k = \min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} \Theta_{rss_k}((V,\phi),\xi).$$

Then $10^{-2k}\mu_k$ converges to μ when k increases, with an exponential speed:

$$\mu_k = 10^{2k}\mu + O(10^k).$$

Moreover, any Boolean logic model minimizing Θ_{rss}, also minimizes Θ_{rss_k} within the following tolerance t_k:

$$t_k = 2\sqrt{\frac{N_\xi}{\mu_k}} + \frac{N_\xi}{\mu_k}.$$

Notice that μ_k increases exponentially with k. Furthermore, in practice, μ_k is significantly greater than N_ξ provided that $k \geq 1$. Thus, the tolerance t_k is relatively small, for instance 0.1, i.e. 10% of μ_k. Next, we aim at enumerating all (nearly) optimal Boolean logic models (V,ϕ)

such that,

$$\Theta_{rss_k}((V,\phi),\xi) \leq \Theta_{rss_k}((V,\phi_{opt}),\xi) + t_{rss} \qquad \Theta_{size}((V,\phi)) \leq \Theta_{size}((V,\phi_{opt})) + t_{size}$$

with t_{rss} and t_{size} the tolerances over (discrete) residual and size, respectively.

3.3.1 Instance

Let (V,E,σ) be a PKN and let (V,H) be the directed hypergraph expanded from it. Recall that with $\mathscr{P}(v)$ we denote the powerset of the signed predecessors of $v \in V$, namely, $2^{Pred(v)}$. We represent the directed hypergraph (V,H) using predicates `node/2`, `hyper/3`, and `edge/3`. The facts `node(`v`,`$s_{\mathscr{P}(v)}$`)` map nodes $v \in V$ to their corresponding sets of signed predecessors $\mathscr{P}(v)$, facts `hyper(`$s_{\mathscr{P}(v)}$`,`s_p`,`l`)` associate $\mathscr{P}(v)$ with its sets $p \in \mathscr{P}(v)$ where l denotes their cardinalities, facts `edge(`s_p`,`v`,1)` associate the set p with $(v,1) \in p$, and facts `edge(`s_p`,`v`,-1)` associate the set p with $(v,-1) \in p$. Note that each $s_{(\cdot)}$ stands for some arbitrary but unique name in its respective context here. Facts over predicates `stimulus/1`, `inhibitor/1`, and `readout/1` denote nodes in V_S, V_K, and V_R respectively. Next, let $\xi = (C_i, P_{C_i})$ be an experimental dataset over (V,E,σ) with $i = 1,\ldots,n$. Recall that each C_i is a clamping assignment over variables in $V_S \cup V_K$. Then, we extend our representation of clamping assignments given in Chapter 2 in order to consider several experimental conditions simultaneously. Towards this end, we represent experimental conditions as facts over predicate `clamped/3`, namely `clamped(`i`,`v`,`$C_i(v)$`)` for all $v \in dom(C_i)$ and $i = 1,\ldots,n$. Finally, let k define the discretization scheme. We represent discretized experimental observations as facts over predicate `obs/3`, namely, `obs(`i`,`v`,`$10^k\delta_k(P_{C_i}(v))$`)` for all $v \in dom(P_{C_i})$ and $i = 1,\ldots,n$. We use the predicate `dfactor/1` to denote the discretization factor 10^k.

Using the discretization scheme provided by $k = 1$, Listing 3.1 shows the instance representation for our toy example. That is, the (signed) directed hypergraph in Figure 3.1b and the dataset given in (3.1).

Listing 3.1: Toy example input instance (`toy.lp`)

```
node(e,1).  node(d,2). node(g,3).
node(f,4).  node(a,5). node(b,6). node(c,7).

hyper(1,1,1). hyper(2,1,1). hyper(1,8,2).  hyper(2,13,2).
hyper(1,2,1). hyper(2,4,1). hyper(1,9,2).  hyper(2,11,2).
hyper(1,3,1). hyper(2,5,1). hyper(1,10,2). hyper(2,12,2).
hyper(3,5,1). hyper(4,6,1). hyper(3,14,2). hyper(1,16,3).
hyper(3,6,1). hyper(4,7,1). hyper(4,15,2). hyper(2,17,3).

edge(1,b,1).   edge(2,c,1).   edge(3,g,-1).  edge(4,a,1).
edge(5,c,-1).  edge(6,e,1).   edge(7,d,1).   edge(8,b,1).
edge(8,c,1).   edge(9,b,1).   edge(9,g,-1).  edge(10,c,1).
edge(10,g,-1). edge(11,a,1).  edge(11,b,1).  edge(12,a,1).
edge(12,c,-1). edge(13,b,1).  edge(13,c,-1). edge(14,e,1).
edge(14,c,-1). edge(15,d,1).  edge(15,e,1).  edge(16,b,1).
```

```
edge(16,c,1).  edge(16,g,-1). edge(17,a,1).  edge(17,b,1).
edge(17,c,-1).

clamped(1,a,1). clamped(1,c,1).                   obs(1,f,9).  obs(1,g,0).
clamped(2,a,1). clamped(2,c,1). clamped(2,d,-1). obs(2,f,1).  obs(2,g,9).
clamped(3,a,1).                                   obs(3,f,0).  obs(3,g,1).
clamped(4,a,1). clamped(4,b,1).                   obs(4,f,10). obs(4,g,8).

stimulus(a).  stimulus(b). stimulus(c).
inhibitor(d). readout(f).  readout(g).

dfactor(10).
```

3.3.2 Encoding

Next we describe our encoding for solving the learning of Boolean logic models as described in the previous section. Our ASP encoding is shown in Listing 3.2.

Listing 3.2: Logic program for learning Boolean logic models (`learning.lp`)

```
variable(V)               :- node(V,_).
formula(V,I)              :- node(V,I); hyper(I,_,_).
{dnf(I,J) : hyper(I,J,N)} :- formula(V,I).
clause(J,V,S)             :- edge(J,V,S); dnf(_,J).

path(U,V) :- formula(V,I); dnf(I,J); edge(J,U,_).
path(U,V) :- path(U,W); path(W,V).
:- path(V,V).
:- dnf(I,J); edge(J,V,_); not stimulus(V); not path(U,V) : stimulus(U).
:- path(_,V); not readout(V); not path(V,U) : readout(U).

exp(E)           :- clamped(E,_,_).
clamped(E,V,-1)  :- exp(E); stimulus(V); not clamped(E,V,1).
clamped(E,V)     :- clamped(E,V,_).
free(E,V,I)      :- formula(V,I); dnf(I,_); exp(E); not clamped(E,V).

eval(E,V, S) :- clamped(E,V,S).
eval(E,V, 1) :- free(E,V,I); eval(E,W,T) : edge(J,W,T); dnf(I,J).
eval(E,V,-1) :- not eval(E,V,1); exp(E); variable(V).

rss(D,V, 1, (F-D)**2) :- obs(E,V,D); dfactor(F).
rss(D,V,-1, D**2)     :- obs(E,V,D).

#minimize{L@1, dnf,I,J : dnf(I,J), hyper(I,J,L)}.
#minimize{W@2, rss,E,V : obs(E,V,D), eval(E,V,S), rss(D,V,S,W)}.

:- formula(V,I); hyper(I,J1,N); hyper(I,J2,M); N < M,
   dnf(I,J1);  dnf(I,J2); edge(J2,U,S) : edge(J1,U,S).

:- formula(V,I); dnf(I,J); edge(J,U,S); edge(J,U,-S).

```

```
32 #const maxsize = -1.
33 #const maxrss  = -1.
34
35 :- maxsize >= 0; maxsize + 1
36    #sum {L,dnf,I,J : dnf(I,J), hyper(I,J,L)}.
37
38 :- maxrss >= 0; maxrss  + 1
39    #sum {W,rss,E,V : obs(E,V,D), eval(E,V,S), rss(D,V,S,W)}.
40
41 #show formula/2.
42 #show dnf/2.
43 #show clause/3.
```

Guessing logical networks. Lines 1-4 define rules generating the representation of a logical network as described in Chapter 2. Line 1 simply projects node names to the predicate `variable/1`. In Line 2 every node $v \in V$ having non-zero indegree is mapped to a formula $\phi(v)$. Next, in Line 3 each set of signed predecessors $p \in \mathscr{P}(v)$ is interpreted as an abducible conjunctive clause in $\phi(v)$. Then, if $p \in \mathscr{P}(v)$ has been abduced, in Line 4 predicates `clause/3` are derived for every signed predecessor in p.[2] Let us illustrate this on our toy example. In order to describe the mappings $e \mapsto b \vee c$ and $g \mapsto e \wedge \neg c$, one would generate a candidate answer set with atoms `dnf(1,1)`, `dnf(1,2)` and `dnf(3,14)` (from Line 2 we derive `formula(e,1)` and `formula(g,3)`). Note that this also force to have atoms `clause(1,b,1)`, `clause(2,c,1)`, `clause(14,e,1)` and `clause(14,c,-1)`. Lines 6-8 eliminate candidate answer sets describing logic models with feedback-loops. Paths from u to v are represented over predicate `path/2` and derived recursively. Thus, the integrity constraint in Line 8 avoids self-reachability in the Boolean logic models. Next, the constraint in Line 9 ensures that for any variable u defined in ϕ all variables $w \in \phi(u)$ are reachable from some stimuli variable. Whereas the constraint in Line 10 guarantees that every variable u defined in ϕ reaches some readout variable. Notice that at this point, we have a representation of the search space of Boolean logic models $\mathbb{M}_{(V,E,\sigma)}$.

Fixpoint, residuals, and optimization. Lines 12-19 elaborate on the rules $\Pi_{2.3}$, $\Pi_{2.4}$ and $\Pi_{2.5}$ given in Chapter 2 in order to consider several clamping assignments simultaneously and compute the fixpoint for each of them accordingly. To be more precise, the response under each experimental condition is represented over predicates `eval/3`, namely `eval(`i`,`v`,`s`)` for experimental condition C_i if variable v is assigned to s. In Lines 21-22 we compute the possible differences (square of residuals) between Boolean predictions and the corresponding experimental observations. We denote such differences over predicate `rss/4`, namely `rss(`o`,`v`,`t`,`r`)` for a residual r with respect to the experimental observation o if the Boolean prediction for $v \in V$ is the truth value $t \in \{1,-1\}$. Note that such predicates are independent from every candidate answer set, that is, they can be deduced during grouding. For our

[2] Notice that predicates `clause/3` are only used for the sake of interpretation. One could simply replace Line 45 at the end of the encoding with `#show clause(J,V,S) : edge(J,V,S); dnf(_,J).` and remove Line 4.

example, due to the experimental condition C_2 we have `rss(1,f,1,81)`, `rss(1,f,-1,1)`, `rss(9,g,1,1)` and `rss(9,g,-1,81)`. Therefore, if the fixpoint for f under the experimental condition C_2 is 1, the residual would be 81, whereas if the fixpoint is -1, the residual is only 1. Analogously, but in the opposite way the same holds for g. Next, we describe our lexicographic multi-objective optimization. In Line 24 we declare with lower priority (`@1`) the minimization over the size of logic models (Eq. (3.3)). Meanwhile, in Lines 25 we declare, with higher priority (`@2`), the minimization of the residual sum of squares between the Boolean predictions and experimental observations (Eq. (3.5)).

Symmetry breaking. Lines 27-30 define two relatively simple symmetry-breaking constraints which are particularly relevant during the enumeration of (nearly) optimal solutions. Essentially, these integrity constraints eliminate answer sets describing "trivially" equivalent Boolean logic models with respect to their logical input-output behavior. The constraint in Lines 27-28 eliminate solutions by checking inclusion between conjunctions. For example, for two variables v and w, the formula $v \vee (v \wedge w)$ is logically equivalent to v and hence, the latter is preferred. The constraint in Line 30 simply avoids solutions having mappings in the Boolean logic models of the form $v \wedge \neg v$. Recall that contradictory causal interactions can be present in the PKN yielding such a formula. Notably, other logical redundancies could be considered as well. However, a complete treatment of redundancies would lead to the NP-complete problem known as *minimization of Boolean functions* [McCluskey, 1956]

Enumeration. Lines 32-39 define a rather "standard" mechanism in order to enumerate solutions within given boundaries. Lines 32-33 simply define two constants describing the boundaries for each optimization criterion which are by default set to -1. Lines 35-36 define an integrity constraint in order to eliminate solutions describing Boolean logic models (V,ϕ) if $\texttt{maxsize} \geq 0$ and $\texttt{maxsize} + 1 \leq \Theta_{size}((V,\phi))$. Analogously, Lines 37-39 define an integrity constraint in order to eliminate solutions describing Boolean logic models (V,ϕ) if $\texttt{maxrss} \geq 0$ and $\texttt{maxrss} + 1 \leq \Theta_{rss_k}((V,\phi),\xi)$.

The following result shows the correctness of our ASP representation. We denote with $\tau((V,E,\sigma),\xi,k)$ the set of facts describing the instance as in Listing 3.1, and with $\Pi_{3.2}$ the set of rules given in Listing 3.2.

Proposition 3.3.2. *Let (V,E,σ) be a PKN. Let $\xi = (C_i, P_{C_i})$ be an experimental dataset over (V,E,σ) and let k define the discretization scheme.*
Then, there is an answer set X of $\tau((V,E,\sigma),\xi,k) \cup \Pi_{3.2}$ such that

$$\phi_{opt} = \left\{ v \mapsto \bigvee_{p \in \mathscr{P}(v)} \left(\bigwedge_{(w,1)\in p} w \right) \wedge \left(\bigwedge_{(w,-1)\in p} \neg w \right) \;\middle|\; dnf(s_{\mathscr{P}(v)}, s_p) \in X, v \in V \right\}$$

if and only if $(V,\phi_{opt}) \in \arg\min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} (\Theta_{rss_k}((V,\phi),\xi), \Theta_{size}((V,\phi)))$.

3.3.3 Solving

Optimization. In Listing 3.3 we show the optimum answer set found for the toy instance described in Listing 3.1.[3] In this case, the optimum answer set is the thirteenth answer set inspected by the solver (`Answer: 13`). Such answer set describes the Boolean logic model given in Fig. 3.1c. Furthermore, the values for the optimization criteria are given ordered by their priorities (`Optimization: 88 7`). That is, 88 for the discretized residual sum of squares (Eq. (3.5)), and 7 for the model size (Eq. (3.3)).

Listing 3.3: Learning an optimum Boolean logic model

```
$ gringo toy.lp learning.lp | clasp --quiet=1
clasp version 3.0.2
Reading from stdin
Solving...
Answer: 13
formula(e,1) formula(d,2) formula(g,3) formula(f,4)\
dnf(1,1) dnf(1,2) dnf(2,4) dnf(3,14) dnf(4,15)\
clause(1,b,1) clause(2,c,1) clause(4,a,1)\
clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1)
Optimization: 88 7
OPTIMUM FOUND

Models       : 13
  Optimum    : yes
Optimization : 88 7
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Enumeration. Next, the enumeration capabilities of an ASP solver like *clasp* [Gebser et al., 2007] can be used to find not only one optimal model but all (nearly) optimal models as described earlier. Considering tolerance $t_{rss} = 8$ ($\sim 10\%$ of the optimum residual sum of squares) and size tolerance $t_{size} = 3$, we enumerate all models such that

$$\Theta_{rss_k}((V,\phi),\xi) \leq \Theta_{rss_k}((V,\phi_{opt}),\xi) + t_{rss} = 96 \qquad \Theta_{size}((V,\phi)) \leq \Theta_{size}((V,\phi_{opt})) + t_{size} = 10.$$

In this example, there are 5 (nearly) optimal Boolean logic models as we show in Listing 3.4.[4] Interestingly, even for this small example, the symmetry-breaking constraints make a significant difference. We note that running the same program but without the symmetry-breaking constraints, yields 17 Boolean logic models instead of only 5. Notably as we show below, in real-world problem instances, exploiting these symmetries significantly reduces the number

[3] Using the option `--quiet=1` only the last (optimum) answer set is printed. Notice that the solver prints all the atoms in the answer set in a single line but we have (manually) introduced breaklines to improve readability.

[4] Option `--opt-mode=ignore` tells the solver to ignore optimize statements; option `-n0` tells the solver to enumerate all solutions; and option `--quiet` avoids printing enumerated solutions.

of solutions (without missing any input-output behavior) and hence, it facilitates their post processing and interpretation.

Listing 3.4: Enumeration of all (nearly) optimal Boolean logic models

```
$ gringo toy.lp learning.lp -c maxrss=96 -c maxsize=10 |\
  clasp --opt-mode=ignore -n0 --quiet
clasp version 3.0.2
Reading from stdin
Solving...
SATISFIABLE

Models       : 5
Calls        : 1
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

3.4 Finding input-output behaviors with Answer Set Programming

Once we have enumerated all (nearly) optimal Boolean logic models with respect to certain tolerances, we can use ASP in order to identify the logical input-output behaviors they describe. Towards this end, we have developed a simple algorithm that (using ASP) systematically compares all pairs of models looking for at least one experimental condition, i.e. a clamping assignment, generating a different response over the readouts nodes. If such an experimental condition exists, the solver will return *SAT* (the logic program is *SATISFIABLE*), meaning that the two models at hand have a different input-output behavior. Otherwise, the solver will return *UNSAT* (the logic program is *UNSATISFIABLE*), meaning that they have the same logical input-output behavior. In what follows we show the ASP representation and the developed algorithm.

3.4.1 Instance

The representation of the problem instance is a straightforward extension from the one described in Listing 2.1 in order to describe a pair of logical networks. To be more precise, instead of having facts over predicates `formula/2`, we consider facts over predicates `formula/3` as follows. Let $(V,\phi_j),(V,\phi_{j'})$ be two (nearly) optimal Boolean logic models. The facts `formula(`j`,`v`,`$s_{\phi_j(v)}$`)` (resp. `formula(`j'`,`v`,`$s_{\phi_{j'}(v)}$`)`) map variables $v \in V$ to their corresponding formulas $\phi_j(v)$ (resp. $\phi_{j'}(v)$). Meanwhile, facts over predicates `variable/1`, `dnf/2` and `clause/3` remain the same as in Listing 2.1.

3.4.2 Encoding

Next we describe our encoding for deciding whether given two Boolean logic models, there exists at least one experimental condition or clamping assignment generating a different

response over the readout nodes. Our ASP encoding is shown in Listing 3.5.

Listing 3.5: Logic program for finding input-output behaviors (`behaviors.lp`)

```
1  {clamped(V, 1)} :- stimulus(V).
2  {clamped(V,-1)} :- inhibitor(V).
3   clamped(V,-1)  :- stimulus(V), not clamped(V,1).
4
5  model(M)    :- formula(M,_,_).
6  clamped(V)  :- clamped(V,_).
7  free(M,V,I) :- formula(M,V,I); not clamped(V).
8
9  eval(M,V, S) :- clamped(V,S); model(M).
10 eval(M,V, 1) :- free(M,V,I); eval(M,W,T) : clause(J,W,T); dnf(I,J).
11 eval(M,V,-1) :- not eval(M,V,1); model(M); variable(V).
12
13 diff :- eval(M1,V,S); eval(M2,V,-S); M1 < M2; readout(V); model(M1;M2).
14
15 :- not diff
```

Lines 1-3 generate (all) possible clamping assignments or experimental conditions. Next, analogously to what we have shown in Listing 3.2, Lines 5-11 elaborate on the the rules $\Pi_{2.3}, \Pi_{2.4}$ and $\Pi_{2.5}$ given in Chapter 2 in order to consider several logical networks simultaneously and compute for each of them, the corresponding fixpoint under the abduced clamping assignment. To be more precise, the response under the abduced clamping assignment is represented over predicates `eval/3`, namely `eval(`j`,`v`,`s`)` for the logical network j if variable v is assigned to s. Afterwards, Line 13 derives the constant predicate `diff` if there exists at least one readout v for which the fixpoints of each logical network do not agree. Notice the use of `M1 < M2` instead of `M1 != M2` in oder to avoid a duplicated (symmetric) constraint. Finally, we use an integrity constraint in Line 15 which forces the solver to exhaustively look for a clamping assignment allowing for the derivation of `diff`. See the following section for more details on how one should use this encoding and interpret its result.

The following result shows the correctness of our ASP representation. We denote with $\tau((V,\phi_j),(V,\phi_{j'}))$ the set of facts describing (V,ϕ_j) and $(V,\phi_{j'})$ as detailed above, and with $\Pi_{3.5}$ the set of rules given in Listing 3.5.

Proposition 3.4.1. *Let $(V,\phi_j),(V,\phi_{j'})$ be two Boolean logic models.*
Then, there is an answer set of $\tau((V,\phi_j),(V,\phi_{j'}))\cup\Pi_{3.5}$ if and only if there exist $C_i\in\mathscr{C}$, $v\in V_R$ such that $F_i^j(v)\neq F_i^{j'}(v)$.

3.4.3 Solving

The idea for finding logical input-output behaviors is to compare Boolean logic models by pairs using the encoding in Listing 3.5. Then, the program given by $\tau((V,\phi_j),(V,\phi_{j'}))$ together with the rules in Listing 3.5, is *SATISFIABLE* if and only if the constant predicate `diff` can be derived, i.e., there exists at least one clamping assignment generating a different response

over the readouts. Otherwise, the program is *UNSATISFIABLE* and hence, we can deduce that (V,ϕ_j) and $(V,\phi_{j'})$ have the same logical input-output behavior. The developed algorithm (in pseudo-code) implementing this idea is shown in Algorithm 1. We assume an auxiliary

Algorithm 1 Finding input-output behaviors over a set of $\mathscr{M}$ Boolean logic models

```
1:  function BEHAVIORS(M : set of Boolean logic models)
2:      B ← ∅                          ▷ representative models for each logical input-output behavior
3:      for (V, φ_j) ∈ M do
4:          found ← False
5:          for (V, φ_j') ∈ B do
6:              if GRINGO-CLASP((V, φ_j), (V, φ_j')) == UNSAT then
7:                  found ← True       ▷ j and j' have the same logical input-output behavior
8:                  break
9:              end if
10:         end for
11:         if not found then
12:             B ← B ∪ (V, φ_j)       ▷ j describes a new logical input-output behavior
13:         end if
14:     end for
15:     return B
16: end function
```

procedure GRINGO-CLASP, such that GRINGO-CLASP(j,j') translates (V,ϕ_j) and $(V,\phi_{j'})$ into the set of facts $\tau((V,\phi_j),(V,\phi_{j'}))$ and then it executes the grounder and solver combining $\tau((V,\phi_j),(V,\phi_{j'}))$ with the rules from Listing 3.5.[5] The algorithm keeps in the (initially empty) set $\mathscr{B}$, one "representative" Boolean logic model for each of the input-output behaviors found. Then, for every model (V,ϕ_j) in the set $\mathscr{M}$ of Boolean logic models, the algorithm calls GRINGO-CLASP$((V,\phi_j),(V,\phi_{j'}))$ for $(V,\phi_{j'}) \in \mathscr{B}$ (representative models of input-output behaviors found in previous iterations) until GRINGO-CLASP returns $UNSAT$ or until all models in $\mathscr{B}$ have been considered. If for some $(V,\phi_{j'})$, the call GRINGO-CLASP$((V,\phi_j),(V,\phi_{j'}))$ returns $UNSAT$, we set $found$ to ***True*** and we break the loop. Next, in Line 11 we skip the **if** block and we continue with the next model $(V,\phi_j) \in \mathscr{M}$. On the other hand, if GRINGO-CLASP$((V,\phi_j),(V,\phi_{j'}))$ returns SAT for all $(V,\phi_{j'}) \in B$, we enter in the **if** block of Line 11 and we add the model (V,ϕ_j) to the set representative Boolean logic models $\mathscr{B}$. Following with our example from the previous section, over the 5 (nearly) optimal Boolean logic models enumerated in Listing 3.4, we found 3 logical input-output behaviors. Furthermore, with a slight modification to the given algorithm, one can also find that in this example, one behavior is described by exactly one model whereas the other two behaviors are described by two models each. As we show in our empirical evaluation, the number of Boolean logic models describing the same input-output behavior can vary significantly in real-world examples. In fact, this raise the question of whether certain logical input-output behaviors should be considered as more relevant than others due to the number of Boolean logic models describing them.

[5] In fact, we have developed the library *pyzcasp* (available at `http://svidela.github.io/pyzcasp/`) providing this kind of functionality. Moreover, we rely on it in order to implement the software presented in Chapter 6.

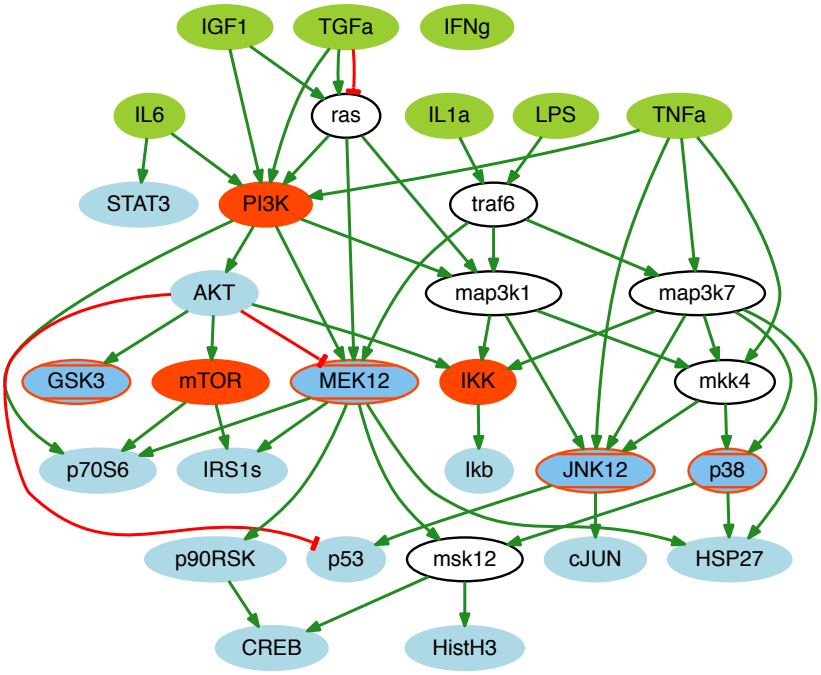


Figure 3.2: Prior knowledge network (V, E, σ) describing signaling pathways in human liver cells. It contains 31 nodes (V) describing biological species: 7 stimuli (V_S), 7 inhibitors (V_K), 15 readouts (V_R) and 6 neither controlled nor observed (V_U). Furthermore, 53 signed directed edges (E) describing activatory and inhibitory causal interactions yield 130 possible (signed) directed hyperedges.

3.5 Empirical evaluation

3.5.1 Real-world problem instance

Prior knowledge network. We evaluate our approach using real-world signaling pathways in human liver cells. The (compressed) PKN (V, E, σ), shown in Figure 3.2, was introduced in [Saez-Rodriguez et al., 2009] and here we use a variation from [Morris et al., 2011]. It contains 31 nodes (V) describing biological species: 7 stimuli (V_S), 7 inhibitors (V_K), 15 readouts (V_R), and 6 neither controlled nor observed (V_U). Notice that $V_K \cap V_R \neq \emptyset$, namely $V_K \cap V_R = \{GSK3, MEK12, JNK12, p38\}$. Furthermore, 53 signed and directed edges (E), describing activatory and inhibitory causal interactions, yield 130 possible (signed) directed hyperedges. Hence, considering each hyperedge as either present or absent, one should inspect 2^{130} $(\sim 1.3 \times 10^{39})$ possible Boolean logic models. Notice that after compression, all the outgoing edges from one stimulus, namely $IFNg$, have been removed since they did not lead to any readout node. Also, it is worth noting that another consequence of compression is the presence of both, a positive and a negative edge from $TGFa$ to ras. For a detailed description of the compression method, we refer the interested reader to [Saez-Rodriguez et al., 2009].

Phospho-proteomics dataset. We consider a publicly available phospho-proteomics dataset ξ described in [Alexopoulos et al., 2010]. This dataset contains measurements in HepG2 liver cancer cells characterizing the immediate-early response over the 15 readouts under 64 experimental conditions combining single- stimulus/inhibitor perturbations. That is, $\xi = ((C_1, P_{C_1}), \ldots, (C_{64}, P_{C_{64}}))$ where the 64 experimental conditions C_i result from considering all possible combinations of either 0 or 1 stimulus, combined with either 0 or 1 inhibitor. Phosphorylation activities P_{C_i} for the 15 readouts were measured using the xMAP technology (Luminex) at 30 minutes after perturbation and normalized to the range $[0,1]$. The time-point characteristic for the immediate-early response, viz. 30 minutes, was chosen based on preliminary experiments (considering several time-points) looking for the largest changes in protein modification states. The number of measurements or size of the dataset is given by $N_\xi = 858$. Notice that this does not correspond exactly to $64 \times 15 = 960$ due to bad-readouts in the experiments. Importantly, both Θ_{rss} and Θ_{rss_k} defined respectively in (3.2) and (3.5), are absolute values which normally increase with N_ξ. Thus, in order to provide an better picture of how well a Boolean logic model fits the available (amount of) observations, in what follows we report the normalized (discrete) residual sum of squares, i.e., mean squared error (MSE). Towards this end, for a Boolean logic model (V,ϕ) and dataset ξ, let us define Θ_{mse} and Θ_{mse_k} as

$$\Theta_{mse}((V,\phi),\xi) = \frac{\Theta_{rss}((V,\phi),\xi)}{N_\xi} \qquad \Theta_{mse_k}((V,\phi),\xi) = \frac{\Theta_{rss_k}((V,\phi),\xi)}{10^{2k} N_\xi}.$$

Notably, the range for both functions is the interval $[0,1]$ where lower values indicate a better fitness to the available data. Notice that Θ_{mse_k} can be computed "directly" from the output provided by the ASP solver whereas Θ_{mse} has to be computed in a post-processing step, e.g. by using any standard scripting programming language.

Problem instance. Next, we consider the problem instance given by the PKN (V,E,σ) and the dataset ξ as described above. Also, we adopt the discretization scheme δ_k provided by using the closest integer function and $k = 2$. That is, we approximate phosphorylation activities up to $\frac{1}{10^2}$, e.g. $\delta_2(0.586) = 59$. In what follows, we denote with the filename `extliver.lp`, the set of logic facts describing the problem instance as illustrated in Listing 3.1 for our toy example.

Listing 3.6: Learning all optimal Boolean logic models

```
$ gringo extliver.lp learning.lp |\
  clasp --conf=jumpy --opt-strategy=4 --opt-mode=optN --quiet
clasp version 3.0.1
Reading from stdin
Solving...
OPTIMUM FOUND

Models       : 18
  Optimum    : yes
  Optimal    : 16
Optimization : 427905 28
Calls        : 1
```

```
Time         : 0.170s (Solving: 0.02s 1st Model: 0.01s Unsat: 0.00s)
CPU Time     : 0.080s
```

3.5.2 Optimal Boolean logic models.

We can identify all optimal Boolean logic models for our real-world case study as shown in Listing 3.6. The command-line option `--opt-mode=optN` asks the solver to find an optimum solution and then continue looking for all solutions with the same "costs" for each optimization criterion. In this case, the solver had found 16 Boolean logic models (`Optimal: 16`) in 0.08 seconds (in total 18 answer sets were inspected (`Models: 18`)). In fact, for this particular instance there is no significant difference between finding an optimum solution and finding all of them. Let us denote with (V,ϕ_i) with $i = 1,\ldots,16$ the corresponding 16 optimal Boolean logic models found. Then, from the solver's output (`Optimization : 427905 28`), we have that $\Theta_{mse_k}((V,\phi_i),\xi) = 0.0499$, i.e. $427905 \times 10^{-4} \times N_{\xi}^{-1}$, and $\Theta_{size}((V,\phi_i)) = 28$ for all $i = 1,\ldots,16$. Interestingly, in this case $\Theta_{mse}((V,\phi_i),\xi) = 0.0499$ as well. That is, either using the original observations (rational numbers) or using the discrete observations (integer numbers), both MSEs agree in this case.[6]

3.5.3 Enumeration of (nearly) optimal Boolean logic models.

Next we evaluate the enumeration of (nearly) optimal Boolean logic models as described before. Let us denote with (V,ϕ_{opt}) any of the 16 optimal Boolean logic models (V,ϕ_i) with $i = 1,\ldots,16$. Our aim here (and more broadly in this thesis) is not to draw biological conclusions but to illustrate the potential of using the introduced framework based on the computational power of ASP and the available systems. Therefore, for the sake of such an illustration in the following we consider tolerances only with respect to the optimum RSS, i.e., $\Theta_{rss_k}((V,\phi_{opt}),\xi) = 427905$ whereas no tolerance is allowed with respect to the optimum size, i.e. $\Theta_{size}((V,\phi_{opt})) = 28$. To be more precise, we enumerate all Boolean logic models (V,ϕ) such that

$$\Theta_{rss_k}((V,\phi),\xi) \leq \Theta_{rss_k}((V,\phi_{opt}),\xi) + t_{rss} \qquad \Theta_{size}((V,\phi)) \leq \Theta_{size}((V,\phi_{opt}))$$

with t_{rss} equal to 0%, 2%, 4%, 6%, 8% or 10% of the optimum RSS, i.e. $\Theta_{rss_k}((V,\phi_{opt}),\xi)$. Importantly, from the Proposition 3.3.1 we can verify that in this case, considering a tolerance of 10% with respect to the minimum of Θ_{rss_k} is sufficient to guarantee that all models minimizing Θ_{rss} are found. Moreover, experimental error in phospho-protein measurements is often estimated to 10% [Chen et al., 2009, Saez-Rodriguez et al., 2009]. Hence, all Boolean logic models found in such a range of tolerances could not be distinguished experimentally. Results are shown in Table 3.1. We also report the identification of logical input-output behaviors. Note the large difference on computation times for enumeration of Boolean logic models (t_{enum}), and for

[6] Notably, for computing Θ_{mse}, beforehand one must run the solver without using the option `--quiet`.

Table 3.1: Enumeration of (nearly) optimal Boolean logic models. We report for each tolerance $\mathscr{T}$, the number of Boolean logic models found $\mathscr{M}$, the CPU time used by the ASP solver t_{enum}, the range of MSEs Θ_{mse_k} and Θ_{mse}, the range of sizes Θ_{size}, the number of logical input-output behaviors $\mathscr{B}$ and the CPU time used by the script implementing Algorithm 1.

$\mathscr{T}$	$\mathscr{M}$	t_{enum}	Θ_{mse_k}	Θ_{mse}	Θ_{size}	$\mathscr{B}$	t_{in-out}
0%	16	0.35s	0.0499	0.0499	28	1	0.560s
2%	144	0.32s	0.0499 - 0.0507	0.0499 - 0.0507	27 - 28	4	4.846s
4%	2150	0.68s	0.0499 - 0.0519	0.0499 - 0.0519	25 - 28	31	240s
6%	2306	0.73s	0.0499 - 0.0522	0.0499 - 0.0523	25 - 28	38	785s
8%	3524	0.96s	0.0499 - 0.0539	0.0499 - 0.0539	25 - 28	66	1296s
10%	5306	1.24s	0.0499 - 0.0546	0.0499 - 0.0546	25 - 28	91	2520s

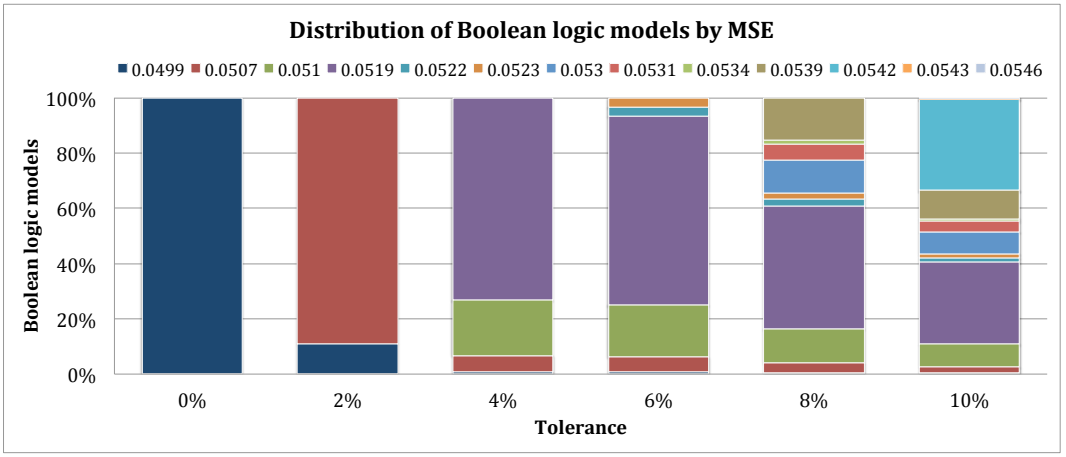


Figure 3.3: Distribution of Boolean logic models found for each tolerance according to their MSEs (Θ_{mse}). When 10% of tolerance is considered, two MSEs, viz. 0.0519 and 0.0542, gather 63% of the 5306 Boolean logic models.

identification of logical input-output behaviors (t_{in-out}). While the ASP solver can enumerate thousands of models in fractions of a second, identifying their input-output behaviors is still a rather demanding task as we increase the number of models. Notably, considering tolerances with respect to both criteria would imply an even larger number of models. This is not actually an issue for the enumeration but it is certainly an issue for the identification of logical input-output behaviors. Interestingly, the range of values for MSEs Θ_{mse_k} and Θ_{mse} do not show any significant difference. Hence, together with the theoretical result from Proposition 3.3.1, this suggest that in practice, the information lost due to discretization is negligible. In fact, in this case the models minimizing Θ_{mse} were already identified without considering tolerance. In Figure 3.3 we illustrate the distribution of Boolean logic models found for each tolerance according to their MSEs (Θ_{mse}). It is worth noting that as we increase the tolerance, more values for Θ_{mse} appear but some are clearly more common than others. For example, when we consider 10% of tolerance, two values for Θ_{mse}, viz. 0.0519 and 0.0542, gather 63% of the 5306 Boolean logic models.

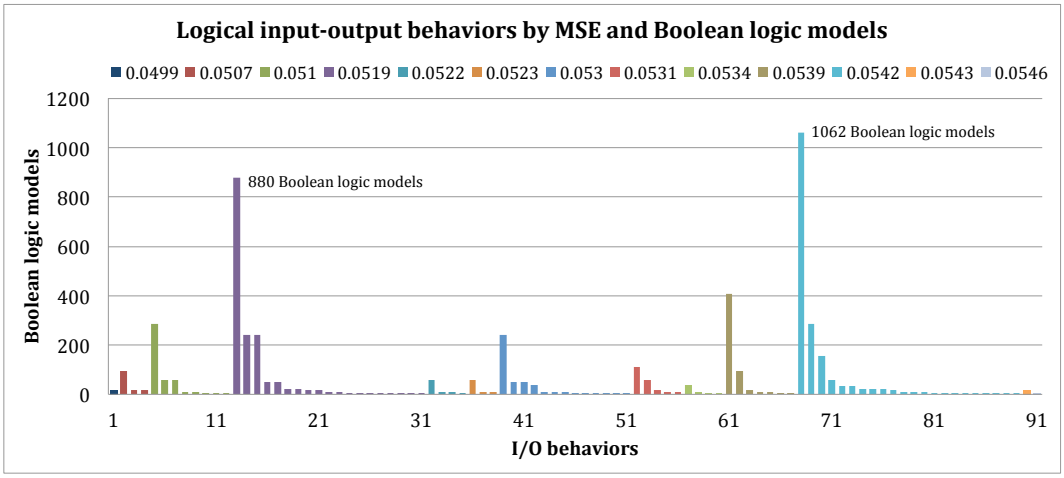


Figure 3.4: Logical input-output behaviors for 10% of tolerance. Behaviors are ordered (from left to right) first according to their MSE (colors), and then according to the number of models they gather (bars). The 2 most common behaviors describe the response of 1062 and 880 Boolean logic models having MSEs 0.0542 and 0.0519 respectively.

3.5.4 Analyzing logical input-output behaviors.

In order to further characterize the multitude of Boolean logic models found, one can focus on their logical input-output behaviors. As an illustration, we concentrate on the 91 behaviors found when considering 10% of tolerance. In Figure 3.4 we show the 91 input-output behaviors ordered by MSE and the number of Boolean logic models they gather. Interestingly, among the 91 logical input-output behaviors, two behaviors are significantly more common than the rest. Such behaviors result from 20% and 16% of the models respectively whereas all the others result from at most 7%. Hence, one can argue that they are more relevant to describe the system's response. Nevertheless, such a claim requires further biological validation which is out of the scope for this thesis.

Next, in Figure 3.5 we show for each tolerance, the evolution of "core predictions" versus the number of input-output behaviors. By "core predictions" we refer to the percentage of experimental conditions leading to exactly the same response in every logical input-output behavior. Recall that the total number of experimental conditions is given by the experimental setup at hand. In our case, since we have 7 stimuli and 7 inhibitors in the PKN (Figure 3.2), one would consider $2^{14} = 16348$ possible experiments (all combinatorial perturbations). However, by inspecting the Boolean logic models found, it is easy to see that stimulation of $IFNg$ or LPS, and inhibition of $GSK3$, is not informative for the input-output analysis. Hence, with 5 stimuli and 6 inhibitors, the number of "relevant" experimental conditions is given by $2^{11} = 2048$. For 0% of tolerance there is only one behavior and hence, there is 100% of core predictions. Then, as we increase the tolerance, the number of behaviors increases whereas the core predictions decreases down to 31%. On the one hand, this approach may provide a way to extract robust insights despite the high variability. On the other hand, it also provides a metric to asses the

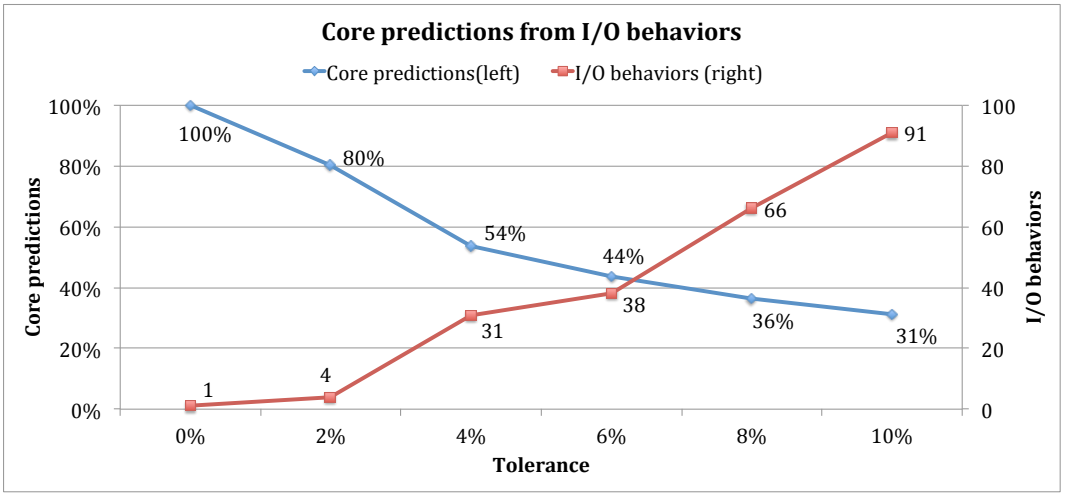


Figure 3.5: Core predictions versus number of input-output behaviors for each tolerance.

level of uncertainty provided by a given set of logical input-output behaviors.

3.5.5 Comparing with a meta-heuristic approach.

In [Guziolowski et al., 2013] we compared our approach with CellNOpt [Terfve et al., 2012], the existing tool to solve the same problem, but using a genetic algorithm. Stochastic search methods such as genetic algorithms, are intrinsically unable not just to provide a complete set of solutions, but also to guarantee that an optimal solution is found. Hence, typically one needs to combine solutions from multiple runs in order to increase the confidence. However, from multiple independent runs of CellNOpt (1000 runs with an average of 1000 seconds per run), only 20% of them have converged to Boolean logic models within the 10% of tolerance over the optimum, i.e. $\Theta_{mse} \leq 0.0549$. Notably, the 16 optimal models ($\Theta_{mse} = 0.0499$) were found by CellNOpt. Nevertheless, the genetic algorithm has retrieved only 51 out of the 91 logical input-output behaviors with an evident bias towards the most common ones. Hence, those behaviors described only by a few logical networks are very unlikely to be found with such stochastic approaches. Therefore, these results underscore the importance of computational methods allowing for an exhaustive characterization of feasible models.

3.6 Conclusion

In this chapter we have addressed the problem consisting of learning from an interaction graph and phosphorylation activities at a pseudo-steady state, (Boolean) logical networks describing the immediate-early response of the system. This problem has been first described in [Saez-Rodriguez et al., 2009] and a genetic algorithm implementation was proposed to solve the underlying optimization problem. To overcome some of the shortcomings intrinsic to stochastic search methods, mathematical programming approaches were presented

in [Mitsos et al., 2009, Sharan and Karp, 2013]. However, rather than looking for *the* optimum Boolean model, one is interested in finding (nearly) optimal models within certain tolerance. Importantly, previous methods, namely stochastic search and mathematical programming, are not able to cope with this question in an exhaustive manner. In this context, we have characterized the learning problem using the notions introduced in Chapter 2. Next, we have elaborated upon our basic ASP representation in order describe the underlying lexicographic multi-objective optimization. Furthermore, we illustrate the strengths of our approach using a real-world problem instance.

Notably, the efficient enumeration techniques provided by an ASP solver like *clasp*, allow us to explore exhaustively the space of feasible solutions in very short time. Therefore, in contrast with the usual approach of modeling a biological system by means of one Boolean network only, our work opens the way for exploring exhaustively the family of plausible networks explaining the available data equally well. Nevertheless, given the ability to enumerate a large number of Boolean models, the way to select among them arises in order to provide new insights to biologists. Towards this end, we have introduced the notion of logical behaviors which allows us to group logical networks regardless of their "internal wirings" and focus on their input-output predictions. In practice, this facilitates the analysis and interpretation of results whereas it provides a way to extract robust insights despite the high variability. At the same time, the existence of several input-output behaviors and the non-uniform distribution of logical networks among them, pose interesting questions for future work. For instance, in the next chapter we concentrate on the problem consisting of finding a set of experiments allowing to discriminate between every pair of logical input-output behaviors.

More generally, we need to develop a mathematical framework for modeling an ensemble of logical networks and/or their logical behaviors. For instance, all networks could be combined in order to define a single probabilistic Boolean network [Shmulevich et al., 2002a]. Furthermore, a relatively simple approach would be to define output predictions as a (deterministic) function of predictions from all input-output behaviors considering the distribution of logical networks among them. Interestingly, preliminary results have shown that, such predictions fit the experimental data similarly to each individual input-output behavior. However, it has been shown recently that an ensemble of models often yields more robust predictions than each model in isolation [Kuepfer et al., 2007, Marbach et al., 2012]. Hence, the presented approach in this chapter is a key contribution in order to achieve reliable and unbiased discoveries in the context of logic-based modeling of signaling networks.

4 Experimental design for discrimination of input-output behaviors

As we have shown in the previous chapter, if the inherent noise is considered during learning, there are multiple (possibly many) logical networks compatible with the experimental observations. Notably, this leads to several internal mechanisms for the system under study but more importantly, to several input-output behaviors as well. Hence, logical input-output predictions may suffer a significant level of uncertainty. This chapter introduces a method to suggest new experiments for discrimination of input-output behaviors and generate models providing more reliable predictions.

4.1 Introduction

Advances on high-throughput technologies have made possible the development of mathematical and computational models describing large-scale biological systems. Importantly, the development of such models allows biologists to test and validate hypotheses, as well as to predict non-observed behaviors. Nonetheless, it is often the case that several "rival" models explain the available experimental data equally well. Especially, in the context of reverse engineering where many (nearly) optimal models may result from an optimization procedure. For instance, as in the previous chapter where thousands of Boolean logic models fit the available dataset similarly well when we consider the experimental noise. Moreover, we have shown that such a large number of models is not only describing alternative mechanisms (internal wirings), but also different input-output behaviors. Therefore, predictions made with such models may suffer a significant level of uncertainty since for a given input, there are several possible system outputs. As mentioned earlier, we can consider several approaches in order to build an ensemble of models such that for a given input, only one output prediction is provided. Nonetheless, learning from experimentation is necessarily an iterative process. Hence, following the loop of hypotesis-driven research in biology [Ideker et al., 2001, Kitano, 2002], we need to look for the next round of experiments in order to refine the models at hand.

Historically, experiments have been designed based on the experience and intuition from experimentalists. However, since experiments are usually expensive and time-consuming, we

would like to know which experiments are more likely to bring new insights to the optimization process. Thus, a proper experimental design enables a maximum informative analysis of the experimental data. Towards this end, exploiting mathematical or computational models for designing the following experiments is a natural approach. We refer the reader to [Kreutz and Timmer, 2009] for a recent review on this subject. Therein, two kinds of "experimental designs" are described, namely, experimental design for parameter estimation and experimental design for model discrimination. In the following, we focus on a particular case of the latter.

Experimental design for discrimination of input-output behaviors. Motivated by our results from the previous chapter, we propose a method for experimental design in order to discriminate between several input-output behaviors. Broadly speaking, experimental design for model discrimination consists of finding an input that maximize the difference of the outputs of the rival models. Importantly, we restrict ourselves to the context of Boolean logic models of immediate-early response as described in Chapter 3. In this context, we aim at finding the minimum number of experimental conditions allowing us to discriminate between every pair of input-output behaviors. Moreover, we adopt a criterion proposed before in the context of mathematical modeling in [Mélykúti et al., 2010]. Therein, authors argue that in principle, maximizing the difference between the outputs of two different models would ensure that even a noisy measurement has a good chance of discriminating between them. Therefore, we adapt this idea to the context of our Boolean logic models and logical input-output behaviors. Also, we consider the minimization of the experiments' complexity in terms of the number of stimuli and inhibitions.

Most of the previous work on experimental design have been based on (semi-) quantitative modeling [Kremling et al., 2004, Vatcheva et al., 2005, Mélykúti et al., 2010, Stegmaier et al., 2013, Busetto et al., 2013]. Thus, in the context of computational modeling, existing approaches to experimental design are less established. It is worth noting that, in general, computational models provide certain predictive power which can be used to generate testable hypotheses and drive the experiments. Nevertheless, herein we refer to the specific problem consisting of automatically propose new experiments allowing to discriminate models at hand. To date, such a question has been addressed under various modeling hypotheses and methods [Ideker et al., 2000, Yeang et al., 2005, Barrett and Palsson, 2006, Szczurek et al., 2008, Sparkes et al., 2010]. Yet, their usefulness in practice remains an open question. Of special interest for us is the approach presented in [Sharan and Karp, 2013]. Therein, authors have addressed slight variations of the problem described in Chapter 3 by means of mathematical programming. In addition, they sketched an algorithm for finding the most informative experiment to discriminate rival Boolean models but no implementation was provided. Nonetheless, compared to all aforecited contributions, our work presents certain differences and similarities. Except for [Szczurek et al., 2008], previous approaches aim at selecting exactly one experiment at each iteration. Therefore, only after the proposed experiment has been carried out in the laboratory and models have been (partially) discriminated, another experiment can be designed. In contrast, but similarly to [Szczurek et al., 2008], we aim at finding the smallest

number of experiments to optimally discriminate all models at once. Furthermore, motivated by [Mélykúti et al., 2010], a distinct feature of our work is the criterion for optimality based on maximizing the sum of pairwise (output) differences. In general, previous methods have adopted an information-theoretic approach where the main design criterion is given by means of the so-called *Shannon entropy* [Shannon, 1948]. Finally, on the theoretical side, results regarding the number and complexity of experiments required for the exact identification of a Boolean genetic network have been reported in [Akutsu et al., 2003].

In the remainder of this chapter we provide a precise characterization of this problem using the notions introduced in Chapter 2 and adapting our Answer Set Programming representation accordingly. Furthermore, we validate our approach using realistic problem settings over the logical input-output behaviors learned in the previous chapter.

4.2 Problem

4.2.1 Logical input-output behaviors and search space of experiments

Logical input-output behaviors. In the following, we consider a given prior knowledge network (PKN) (V,E,σ) and an experimental dataset ξ as defined in Chapter 3. Then, we assume we can learn from (V,E,σ) and ξ, a finite family of (nearly) optimal Boolean logic models $\mathscr{M} = ((V,\phi_j))_{j\in J}$. Next, using the Algorithm 1 (Section 3.4.3) we can find the set of logical input-output behaviors $\mathscr{B} = \text{BEHAVIORS}(\mathscr{M})$. That is, $\mathscr{B}$ contains (exactly) one "representative" Boolean logic model $(V,\phi_j) \in \mathscr{M}$ for each input-output behavior found. Now, let us recall some notation already introduced in previous chapters. We denote with $\mathscr{C}$ the space of all possible clamping assignments or experimental conditions C_i over (V,E,σ). Notice that in principle, $i = 1,\ldots,2^{|V_S|+|V_K|}$, i.e. all combinatorial perturbations of stimuli and knockouts. Furthermore, we use F_i^j to describe the fixpoint of $\Omega_{(V,\phi_j|_{C_i})}$ reachable from $A_{\boldsymbol{f}} = \{v \rightarrowtail \boldsymbol{f} \mid v \in V\}$. Notably, since every $(V,\phi_j) \in \mathscr{B}$ describes a different input-output behavior, it holds that

$$\Big(\forall (V,\phi_j),(V,\phi_{j'}) \in \mathscr{B}, j \neq j' :: \exists C_i \in \mathscr{C}, v \in V_R :: F_i^j(v) \neq F_i^{j'}(v)\Big).$$

In words, for every pairs of different models $(V,\phi_j),(V,\phi_{j'}) \in \mathscr{B}$, there exists at least one experimental condition C_i and one readout v such that, the corresponding responses in each model differ on v, i.e. $F_i^j(v) \neq F_i^{j'}(v)$. In practice this means that performing the experiment C_i, one would be able to discriminate between (V,ϕ_j) and $(V,\phi_{j'})$. Except in the case when the measurement on v is around 0.5 and hence, both Boolean predictions would explain the experimental observation similarly well.

Search space of experiments. As mentioned above, we denote with $\mathscr{C}$ the space of all possible clamping assignments or experimental conditions C_i over (V,E,σ). Recall that, if C_i is an experimental condition and $v \in dom(C_i)$ then, either $v \in V_S$ and $C_i(v) = \boldsymbol{t}$, or $v \in V_K$ and

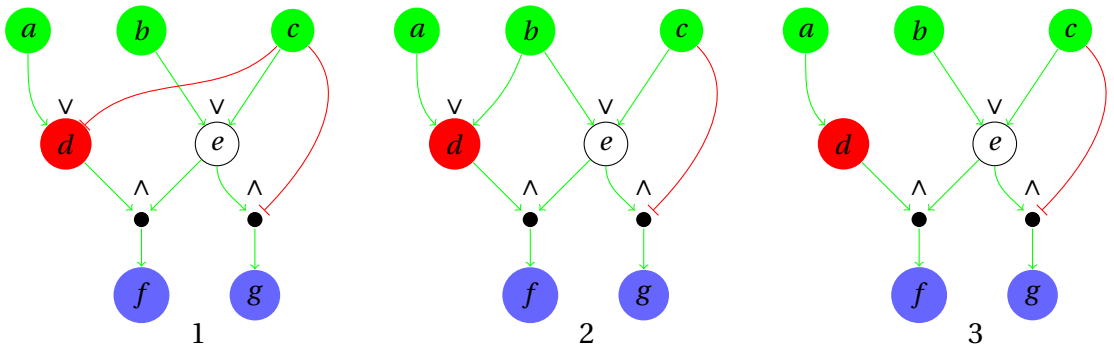


Figure 4.1: Representative Boolean logic models $(V,\phi_1),(V,\phi_2)$, and (V,ϕ_3) for 3 different logical input-output behaviors. The correspoinding mappings are: $\phi_1 = \{d \mapsto a \vee \neg c, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, $\phi_2 = \{d \mapsto a \vee b, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, and $\phi_3 = \{d \mapsto a, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$.

$C_i(v) = \boldsymbol{f}$. In theory, all combinatorial perturbations of stimuli and knockouts are possible. Thus, one would consider $i = 1,\ldots,2^{|V_S|+|V_K|}$. However, in practice, combining more than a few perturbations in the same experiment may be out of reach for current technology. In this context, let us denote with $|C_i|_{V_S}$ and $|C_i|_{V_K}$, the number of stimuli and inhibitors respectively clamped in C_i. That is, $|C_i|_{V_S} = |dom(C_i) \cap V_S|$ and $|C_i|_{V_K} = |dom(C_i) \cap V_K|$. Next, we consider a search space of experimental conditions parameterized by integers s and k in order to restrict ourselves to experiments having at most, s stimuli and k inhibitors, with $0 \leq s \leq |V_S|$ and $0 \leq k \leq |V_K|$. To be more precise, the *search space of experiments* with at most, s stimuli and k inhibitors, is defined as $\mathscr{C}(s,k) = \left\{C_i \mid C_i \in \mathscr{C}, |C_i|_{V_S} \leq s, |C_i|_{V_K} \leq k\right\}$. Notably, the total number of experimental conditions or clamping assignments in $\mathscr{C}(s,k)$ is given by

$$\sum_{i=0}^{s} \binom{|V_S|}{i} \times \sum_{i=0}^{k} \binom{|V_K|}{i}$$

where $\binom{n}{m}$ denotes the binomial coefficient, i.e., $\frac{n!}{m!(n-m)!}$. Notice that for some pairs of input-output behaviors in $\mathscr{B}$, there may not exist any experimental condition in $\mathscr{C}(s,k)$ yielding a different response between them. Especially, if either s or k are too small.

Let us illustrate the concepts introduced above with our toy example from Chapter 3. As mentioned in Section 3.4.3, over the 5 (nearly) optimal Boolean logic models enumerated in Listing 3.4, we found 3 logical input-output behaviors. In fact, in Figure 4.1 we show one "representative" Boolean logic model for each behavior. Let us denote these models by (V,ϕ_j) with $j = 1,2,3$. Then, then corresponding mappings are defined as

$$\begin{aligned}
\phi_1 &= \{d \mapsto a \vee \neg c, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\} \\
\phi_2 &= \{d \mapsto a \vee b, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\} \\
\phi_3 &= \{d \mapsto a, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}\,.
\end{aligned}$$

Furthermore, let us consider two experimental conditions in $\mathscr{C}(2,1)$, namely $C_1 = \{b \mapsto \boldsymbol{t}\}$ and

Table 4.1: Fixpoints F_i^j for $\Omega_{(V,\phi_j|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$ with $j = 1,2,3$ and $i = 1,2$. Logical networks (V,ϕ_j) are shown in Figure 4.1 whereas $C_1 = \{b \rightarrowtail \boldsymbol{t}\}$ and $C_2 = \{b \rightarrowtail \boldsymbol{t}, c \rightarrowtail \boldsymbol{t}\}$.

	$(V,\phi_j\|_{C_1})$							$(V,\phi_j\|_{C_2})$						
	a	b	c	d	e	f	g	a	b	c	d	e	f	g
(V,ϕ_1)	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$
(V,ϕ_2)	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$
(V,ϕ_3)	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{t}$	$\boldsymbol{f}$	$\boldsymbol{f}$

$C_2 = \{b \rightarrowtail \boldsymbol{t}, c \rightarrowtail \boldsymbol{t}\}$. That is, in C_1 only b is stimulated whereas in C_2 both b and c are stimulated. In both experimental conditions, a is not stimulated and d is not inhibited. The corresponding fixpoints F_i^j for $\Omega_{(V,\phi_j|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$ are shown in Table 4.1. We refrain from giving the detailed iterations for the sake of brevity. Nonetheless, one can verify that for each (V,ϕ_j) and C_i, the assignments F_i^j given in Table 4.1 satisfy that, for all $v \in dom(\phi_j)$ it holds $F_i^j(\phi(v)) = F_i^j(v)$. Furthermore, using C_1 and C_2 one can discriminate between every pair of input-output behvaiors, namely, $(1,2),(1,3)$ and $(2,3)$. To be more precise, the output responses for (V,ϕ_1) and (V,ϕ_2) differ on species f under C_2 ($F_2^1(f) \neq F_2^2(f)$). Between (V,ϕ_1) and (V,ϕ_3), the corresponding fixpoints also differ on species f but under C_1 ($F_1^1(f) \neq F_1^3(f)$). Finally, the output responses for (V,ϕ_2) and (V,ϕ_3) differ on f under both clamping assignments, C_1 ($F_1^2(f) \neq F_1^3(f)$) and C_2 ($F_2^2(f) \neq F_2^3(f)$).

4.2.2 Experimental design

Discriminating among every pair of behaviors. Let $\mathscr{B}$ be a finite set of input-output behaviors represented by Boolean logic models $((V,\phi_j))_{j\in J}$. In most cases, it happens that several experimental conditions must be considered together in order to discriminate among every pair of behaviors. Clearly, in practice one would like to perform as few experiments as possible. Therefore, our first criterion for experimental design consists of finding the minimum number of experimental conditions $C_i \in \mathscr{C}(s,k)$ with $0 \leq s \leq |V_S|$ and $0 \leq k \leq |V_K|$, which allow us to discriminate among every pair of logical input-output behaviors. To be more precise, we aim at finding the smallest $\varepsilon \geq 0$ such that there exist an ε-tuple $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}_1(s,k) \times \cdots \times \mathscr{C}_\varepsilon(s,k)$ satisfying:

$$\left(\forall (V,\phi_j),(V,\phi_{j'}) \in \mathscr{B}, j \neq j' :: \left(\exists C_i \in \{C_1,\ldots,C_\varepsilon\}, v \in V_R :: F_i^j(v) \neq F_i^{j'}(v)\right)\right). \tag{4.1}$$

In what follows, we denote with $\mathscr{C}^\varepsilon(s,k)$ the set of all ε-tuples $(C_1,\ldots,C_\varepsilon)$ satisfying (4.1).

Maximizing differences over readouts. Once we have identified that ε experimental conditions are sufficient in order to discriminate between all input-output behaviors, the next question is how to select among all possible $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}^\varepsilon(s,k)$. Towards this end, let us denote the Boolean logic models in $\mathscr{B}$ by $(V,\phi_1),\ldots,(V,\phi_n)$. Then, we define the *differences* (Θ_{diff}) generated by the experimental conditions $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}^\varepsilon(s,k)$ over the logical input-output

behaviors in $\mathscr{B}$ as

$$\Theta_{diff}(\mathscr{B},(C_1,\ldots,C_\varepsilon)) = \sum_{j=1}^{n-1}\sum_{j'=j+1}^{n}\sum_{i=1}^{\varepsilon}\sum_{v\in V_R}\begin{cases}1 & F_i^j(v)\neq F_i^{j'}(v)\\ 0 & \text{otherwise}\,.\end{cases} \tag{4.2}$$

Next, our second criterion for experimental design consists of finding ε-tuples $(C_1,\ldots,C_\varepsilon)\in \mathscr{C}^\varepsilon(s,k)$ such that the function Θ_{diff} is maximized,

$$(C_1,\ldots,C_\varepsilon)\in \underset{(C_1,\ldots,C_\varepsilon)\in\mathscr{C}^\varepsilon(s,k)}{\operatorname{argmax}} \left(\Theta_{diff}(\mathscr{B},(C_1,\ldots,C_\varepsilon))\right)\,. \tag{4.3}$$

Minimizing the complexity of experiments. Intuitively, the complexity or "cost" of an experimental condition C_i over (V,E,σ) can be related to the number of stimuli ($|C_i|_{V_S}$) and inhibitors ($|C_i|_{V_K}$), clamped in C_i. In fact, this intuition has been used before in [Akutsu et al., 2003] where it has been investigated the number and complexity of experiments required for the identification of a Boolean gene regulatory network. Moreover, we have already taken this notion of complexity into account when we restricted ourselves to experimental conditions in $\mathscr{C}(s,k)=\left\{C_i \mid C_i\in\mathscr{C}, |C_i|_{V_S}\leq s, |C_i|_{V_K}\leq k\right\}$. However, the selection of s and k aims at giving upper bounds to the number of stimuli and inhibitors. Now, we aim at finding the *simplest* experimental conditions among all ε-tuples $(C_1,\ldots,C_\varepsilon)$ maximizing Θ_{diff} as in (4.3). Towards this end, we define two functions over ε-tuples of experimental conditions counting the number of stimuli (Θ_{V_S}) and inhibitors (Θ_{V_K}) respectively,

$$\Theta_{V_S}((C_1,\ldots,C_\varepsilon)) = \sum_{i=1}^{\varepsilon}|C_i|_{V_S} \qquad \Theta_{V_K}((C_1,\ldots,C_\varepsilon)) = \sum_{i=1}^{\varepsilon}|C_i|_{V_K}\,. \tag{4.4}$$

Next, let us denote with Δ the set of all ε-tuples $(C_1,\ldots,C_\varepsilon)$ maximizing Θ_{diff} as in (4.2). Finally, we consider two additional optimization criteria in lexicographic order aiming at the identification of the *simplest* $(C_1,\ldots,C_\varepsilon)\in\Delta$,

$$(C_1,\ldots,C_\varepsilon)\in \underset{(C_1,\ldots,C_\varepsilon)\in\Delta}{\operatorname{argmin}} \left(\Theta_{V_S}((C_1,\ldots,C_\varepsilon)),\Theta_{V_K}((C_1,\ldots,C_\varepsilon))\right)\,. \tag{4.5}$$

Notice that we minimize first Θ_{V_S} and then, with lower priority Θ_{V_K}. But this is a rather arbitrary decision which can be revisited in practice. For instance, if for the system at hand, it is the case that combining several inhibitors is experimentally more complicated than combining several ligands. Alternatively, both functions could be combined into a single function, for example, $\Theta_{V_S\cup V_K}=\Theta_{V_S}+\Theta_{V_K}$. In fact, many other criteria could be taken into account. For example, we could assign a weight to every stimulus and inhibitor in order to describe its price. Thereafter, we can minimize the required budget for running the suggested experiments. Also, if certain stimuli and/or inhibitors are not compatible with each other, we could consider additional constraints in order to avoid such combinations. Nevertheless, these kinds of optimization criteria and constraints are related to very specific application settings such as, available technology, experimental tools (ligands, small-molecule drugs, antibodies, etc), budget, and

others. Instead, we aim at capturing generic problem settings which can be adapted to specific use cases by exploiting the elaboration tolerance of our methods.

Let us illustrate the definitions given above with our toy example from Figure 4.1 and experimental conditions $C_1 = \{b \mapsto \boldsymbol{t}\}$ and $C_2 = \{b \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}\}$. As we have shown in Table 4.1, using C_1 and C_2 we can discriminate between the three pairs of input-output behaviors. Hence, the condition given in (4.2) is satisfied for $\varepsilon = 2$ and the 2-tuple (C_1, C_2). In fact, in this case, $\varepsilon = 2$ is the minimum number of experiments required in order to satisfy (4.2). Next, we can count the number of differences among every pair of behaviors. As detailed above, we have $F_2^1(f) \neq F_2^2(f), F_1^1(f) \neq F_1^3(f), F_1^2(f) \neq F_1^3(f)$, and $F_2^2(f) \neq F_2^3(f)$. Therefore, $\Theta_{diff}(\{(V,\phi_1),(V,\phi_2),(V,\phi_3)\},(C_1,C_2)) = 4$. Finally, the complexity of the 2-tuple (C_1, C_2) is given by the two functions Θ_{V_S} and Θ_{V_K}. The number of stimuli is given by $\Theta_{V_S}((C_1,C_2)) = 3$, whereas the number of inhibitors is given by $\Theta_{V_K}((C_1,C_2)) = 0$.

4.3 Experimental design with Answer Set Programming

4.3.1 Instance

The representation of the problem instance is essentially the same as in Section 3.4.1 but applied to several networks instead of only a pair. To be more precise, let $\mathscr{B}$ be a finite set of input-output behaviors represented by Boolean logic models $(V,\phi_1),\ldots,(V,\phi_n)$. The facts `formula(`$j$`,`$v$`,`$s_{\phi_j(v)}$`)` map variables $v \in V$ to their corresponding formulas $\phi_j(v)$ with $j = 1,\ldots,n$. Meanwhile, facts over predicates `variable/1`, `dnf/2` and `clause/3` remain the same as in Listing 2.1. Also, as in Chapter 3, facts over predicates `stimulus/1`, `inhibitor/1`, and `readout/1` denote nodes in V_S, V_K, and V_R respectively. Finally, we consider two constants, namely, `maxstimuli=`s and `maxinhibitors=`k, in order to represent the search space of experimental conditions $\mathscr{C}(s,k)$. By default, we assign such constants to $|V_S|$ and $|V_K|$, respectively. Afterwards, at grounding time, we can use command-line options `-c maxstimuli=`s `-c maxinhibitors=`k in order to overwrite these values with arbitrary s and k.

Listing 4.1 shows the instance representation for our toy example. That is, the three input-output behaviors represented by the Boolean logic models $(V,\phi_1),(V,\phi_2)$, and (V,ϕ_3) shown in Figure 4.1.

Listing 4.1: Toy example input instance (`toy.lp`)

```
variable("f").  variable("e").  variable("a").  variable("g").
variable("b").  variable("c").  variable("d").

formula(1,"d",1). formula(1,"e",0). formula(1,"f",3). formula(1,"g",2).
formula(2,"d",4). formula(2,"e",0). formula(2,"f",3). formula(2,"g",2).
formula(3,"d",5). formula(3,"e",0). formula(3,"f",3). formula(3,"g",2).

dnf(0,2). dnf(0,0). dnf(1,7). dnf(1,8). dnf(2,14).
dnf(3,16).  dnf(4,0). dnf(4,7). dnf(5,8). dnf(4,8).

```

```
clause(0,"b",1).  clause(2,"c",1).  clause(7,"c",-1). clause(8,"a",1).
clause(14,"e",1). clause(14,"c",-1).  clause(16,"d",1). clause(16,"e",1).

stimulus("a").  stimulus("c").  stimulus("b").
readout("g"). readout("f").   inhibitor("d").

#const maxstimuli = 3.
#const maxinhibitors = 1.
```

4.3.2 Encoding

Next we describe our encoding relying on *incremental* ASP [Gebser et al., 2008], for finding an optimal experimental design as described in the previous section. The idea is to consider one problem instance after another by gradually increasing the number of experimental conditions such that, if the program is satisfiable at the step ε, then there exists an ε-tuple $(C_1,\ldots,C_\varepsilon)$ satisfying (4.1). Our ASP encoding is shown in Listing 4.2.

Listing 4.2: Logic program for finding an optimal experimenal design (`design.lp`)

```
#include <iclingo>.
#const imax = 20.

model(M) :- formula(M,_,_).

#program cumulative(k).

{clamped(k,V, 1) : clause(_,V,_), stimulus(V) } maxstimuli.
{clamped(k,V,-1) : clause(_,V,_), inhibitor(V)} maxinhibitors.
 clamped(k,V,-1) :- stimulus(V); not clamped(k,V,1).

clamped(k,V)  :- clamped(k,V,_).
free(k,M,V,I) :- formula(M,V,I); not clamped(k,V).

eval(k,M,V, S) :- clamped(k,V,S); model(M).
eval(k,M,V, 1) :- free(k,M,V,I); eval(k,M,W,T) : clause(J,W,T); dnf(I,J).
eval(k,M,V,-1) :- not eval(k,M,V,1); model(M); variable(V).

diff(k,M1,M2)   :- diff(k,M1,M2,_).
diff(k,M1,M2,V) :- eval(k,M1,V,S); eval(k,M2,V,-S);
                   M1 < M2; readout(V); model(M1;M2).

#minimize{1@1,clamped,k,V    : clamped(k,V,-1), inhibitor(V)}.
#minimize{1@2,clamped,k,V    : clamped(k,V, 1),  stimulus(V)}.
#maximize{1@3,diff,k,M1,M2,V : diff(k,M1,M2,V)}.

#program volatile(k).
#external query(k).

:- not diff(K,M1,M2) : K=1..k; model(M1;M2); M1<M2; query(k).

```

```
32 #show clamped/3.
33 #show diff/4.
```

Preliminaries Line 1 declares the required inclusion in order to use incremental solving embedded in *clingo* 4. Importantly, the remainder of the encoding must follow a specific structure in order to work properly. More precisely, we need to define two "subprograms", namely, `cumulative(k)`, and `volatile(k)`. To this end, we use the directive `#program` with a name, e.g., `cumulative`, and an optional list of parameters, e.g., `(k)`. Each subprogram comprise all rules up to the next such directive (or the end of file) and their grounding is controlled via the embedded script. Line 2 declares the constant `imax` which is used as the maximum number of incremental steps to be considered. Recall that, the number of incremental steps represents the number of experimental conditions. Thus, by default, we assign it to 20 but we can easily overwrite this value afterwards using the command-line option `-c imax=`e. Next, Line 4 defines auxiliary domain predicates `model/1`, namely, `model(`j`)` for all Boolean logic model $(V,\phi_j) \in \mathscr{B}$.

Experimental conditions, fixpoints, differences, and optimization. Lines 6-25 define the subprogram `cumulative(k)`. The purpose of this subprogram, is to declare all logic rules which have to be grounded at every incremental step `k`. As in Listing 3.2, we represent experimental conditions using predicates `clamped/3`. Thus, Lines 8-10 define rules in order to guess the experimental condition at step `k`. Note the usage of constants `maxstimuli` and `maxinhibitors` as upper bounds in the corresponding choice rules. Also, we restrict the choices to experimental conditions clamping a variable $v \in V_S \cup V_K$ only if v occurs in some clause. Otherwise, clamping the variable v does not make any "downstream" difference and hence, it is not useful in order to discriminate models. For instance, following our toy example, the experimental conditions $C_1 = \{b \mapsto \boldsymbol{t}\}$ and $C_2 = \{b \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}\}$ are represented by predicates `clamped(1,b,1)`, `clamped(2,b,1)`, and `clamped(2,c,1)`.[1] Lines 12-17 elaborate on the rules $\Pi_{2.3}, \Pi_{2.4}$ and $\Pi_{2.5}$ given in Chapter 2 in order to consider several clamping assignments and logical networks simultaneously, and compute the fixpoint for each of them accordingly. To be more precise, the response for each logical network under each experimental condition is represented over predicates `eval/4`, namely, `eval(`i`,`j`,`v`,`s`)` for a logical network (V,ϕ_j) and experimental condition C_i, if the variable v is assigned to s, i.e. $F_i^j(v) = s$. Let us illustrate this with our example. The corresponding predicates `eval/4` describing the response over variables f and g as in Table 4.1 are:

```
eval(1,1,"f", 1)  eval(1,1,"g",1)     eval(2,1,"f",-1)  eval(2,1,"g",-1)
eval(1,2,"f", 1)  eval(1,2,"g",1)     eval(2,2,"f", 1)  eval(2,2,"g",-1)
eval(1,3,"f",-1)  eval(1,3,"g",1)     eval(2,3,"f",-1)  eval(2,3,"g",-1)
```

[1] We note that permutations, e.g., `clamped(2,b,1)`, `clamped(1,b,1)`, and `clamped(1,c,1)`, are allowed in our encoding, but one can encode additional symmetry-breaking constraints in order to avoid them.

Then, in Lines 19-21 we derive predicates `diff/3` and `diff/4` in order to represent differences among every pair of logical input-output behavior for the experimental conditions under consideration. That is, we derive `diff(`i`,`j`,`j'`,`v`)` provided that for some models $(V,\phi_j),(V,\phi_{j'})$ with $j<j'$, we have that $F_i^j(v)\neq F_i^{j'}(v)$, i.e., `eval(`i`,`j`,`v`,`s`)` and `eval(`i`,`j'`,`v`,-`s`)`. For our example, we derive predicates `diff(1,1,3,"f")`, `diff(1,2,3,"f")`, `diff(2,1,2,"f")`, and `diff(2,2,3,"f")`. Hence, predicates `diff/3`, namely, `diff(`i`,`j`,`j'`)` indicate the existence of at least one (output) difference between (V,ϕ_j) and $(V,\phi_{j'})$ under experimental condition C_i. Finally, Lines 23-25 declare the three optimization criteria in lexicographic order. Notably, when grounding and solving `cumulative(k)` for successive values of `k`, the solver's objective functions are gradually extended accordingly. Lines 23-24 declare the minimization of functions Θ_{V_S} and Θ_{V_K}, as defined in (4.4). Notice the corresponding priority levels, namely `@1` and `@2`, meaning that as defined in (4.5), first we minimize Θ_{V_S}, and then with lower priority Θ_{V_K}. As discussed earlier, this is a rather arbitrary decision. However, in practice it can be revisited very easily thanks to the declarativeness of ASP. Finally, Line 25 declares with the greatest priority, namely, `@3`, the maximization of the function Θ_{diff} as defined in (4.2).

Discriminating among every pair of input-output behaviors. Lines 27-33 define the subprogram `volatile(k)`. The purpose of this subprogram, is to declare logic rules specific for each value of `k`. Typically, in the form of integrity constraints forcing the incremental solving to continue until such constraints are satisfied at a given step. Line 28 declares the external atoms `query(k)` for every incremental step `k`. Internally, the incremental solving embedded in *clingo* 4 assigns these atoms to either true or false in order to indicate the current step. Then, in Line 30 we define an integrity constraint in order to eliminate answer sets describing an ε-tuple $(C_1,\ldots,C_\varepsilon)$ such that (4.1) does not hold. In fact, the body of the integrity constraint is precisely the negation of the expression given in (4.1). Therefore, the incremental solving continues until its reach either the maximum number of steps `imax`, or the step ε such that there exists an ε-tuple satisfying (4.1).

The following result shows the correctness of our ASP representation. We denote with $\tau(\mathscr{B},s,k)$ the set of facts describing the instance as in Listing 4.1, and with $\Pi_{4.2}$ the set of rules given in Listing 4.2.

Proposition 4.3.1. *Let $\mathscr{B}$ be a finite set of input-output behaviors represented by Boolean logic models $((V,\phi_j))_{j\in J}$. Let ε, s, and k be three positive integers with $s\leq|V_S|$ and $k\leq|V_K|$.*
Then, there is an answer set X of $\tau(\mathscr{B},s,k)\cup\Pi_{4.2}$ such that $C_i=\{v\mapsto\boldsymbol{t}\mid v\in V_S, clamped(i,v,1)\in X\}\cup\{v\mapsto\boldsymbol{f}\mid v\in V_K, clamped(i,v,-1)\in X\}$ with $i=1,\ldots,\varepsilon$ if and only if,

1. *ε is the least number of clamping assignments for which (4.1) holds,*
2. *and $(C_1,\ldots,C_\varepsilon)\in\arg\min_{(C_1,\ldots,C_\varepsilon)\in\Delta}\left(\Theta_{V_S}((C_1,\ldots,C_\varepsilon)),\Theta_{V_K}((C_1,\ldots,C_\varepsilon))\right)$ with $\Delta=\arg\max_{(C_1,\ldots,C_\varepsilon)\in\mathscr{C}^\varepsilon(s,k)}\left(\Theta_{diff}(\mathscr{B},(C_1,\ldots,C_\varepsilon))\right)$ and $\mathscr{C}^\varepsilon(s,k)$ the set of all ε-tuples $(C_1,\ldots,C_\varepsilon)\in\mathscr{C}_1(s,k)\times\cdots\times\mathscr{C}_\varepsilon(s,k)$ satisfying (4.1).*

4.3.3 Solving

In Listing 4.3 we show the optimum answer set for the toy instance described in Listing 4.1. Such an answer set represents the experimental conditions $C_1 = \{b \mapsto \boldsymbol{t}\}$ and $C_2 = \{b \mapsto \boldsymbol{t}, c \mapsto \boldsymbol{t}\}$. Notice that, we can see the number of steps in the incremental solving by looking at the number of calls (`Calls: 2` or `Solving...` printed twice). Furthermore, the value for each optimization criterion is given in the corresponding order (`Optimization 8 3 0`). Actually, the grounder performs an internal transformation for maximize statements and hence, the reported value (8) for the first optimization criterion is not exactly as one would expect. In order to recover the optimum for Θ_{diff}, we need to count the number of predicates `diff/4` in the answer set. Thus, in this case, the optimum value for Θ_{diff} is 4. On the other hand, minimization statements are treated without any transformation. Therefore, the optimum value for Θ_{V_S} is 3, whereas the optimum value for Θ_{V_K} is 0, as reported directly by the solver.

Listing 4.3: Finding an optimum experimental design

```
$ clingo design.lp toy.lp --quiet=1
clingo version 4.3.1
Reading from design.lp ...
Solving...
Solving...
Answer: 1
clamped(1,"a",-1) clamped(1,"c",-1) clamped(1,"b",1)\
clamped(2,"a",-1) clamped(2,"c",1)  clamped(2,"b",1)\
diff(1,1,3,"f") diff(1,2,3,"f") diff(2,1,2,"f") diff(2,2,3,"f")
Optimization: 8 3 0
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 8 3 0
Calls        : 2
Time         : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.010s
```

4.4 Empirical evaluation

4.4.1 Real-world problem instance

Logical input-output behaviors. In Chapter 3, using real-world signaling pathways in human liver cells and a publicly available phospho-proteomics dataset, we have shown how we can learn Boolean logic models and their corresponding logical input-output behaviors. In particular, in Table 3.1 we have reported, for various levels of tolerance over fitness, the number of Boolean logic models and the number of input-output behaviors they describe. Recall that, in regards of the available experimental observations and their intrinsic uncertainty, such behaviors explain the data equally well. Therefore, now we are interested in identifying an

optimal experimental design, as described in Section 4.2, in order to discriminate among all input-output behaviors found for each tolerance, namely, 4 behaviors for 2% tolerance, 31 behaviors for 4% tolerance, 38 behaviors for 6% tolerance, 66 behaviors for 8% tolerance, and 91 behaviors for 10% tolerance.

Search space of experiments. The phospho-proteomics dataset used for learning contains 64 experimental conditions resulting from all possible combinations of either 0 or 1 stimulus, combined with either 0 or 1 inhibitor. Now, we consider the space of experimental conditions defined by $\mathscr{C}(3,2)$. That is, experiments having 0 to 3 stimuli, combined with 0 to 2 inhibitors. As explained earlier, on the one hand, this restriction reduces the size of the search space which is always convenient from the computational point of view. But also, on the other hand, considering experiments with more stimuli and/or inhibitors, does not appear to be very relevant in practice since it would not be possible to perform such experiments.

Problem instances. Next, we consider the problem instances given by each set of logical input-output behaviors listed in Table 3.1. Let us denote with $\mathscr{B}_m$ the corresponding set of m input-output behaviors, namely, $\mathscr{B}_4$, $\mathscr{B}_{31}$, $\mathscr{B}_{38}$, $\mathscr{B}_{66}$, and $\mathscr{B}_{91}$. Then, in what follows, we denote with the filename `behaviors-`m`.lp`, the set of logic facts describing each problem instance $\mathscr{B}_m$ as illustrated in Listing 4.1 for our toy example.

Listing 4.4: Finding an optimal experimental design to discriminate 4 behaviors

```
$ clingo design.lp behaviors-31.lp -c maxstimuli=3 -c maxinhibitors=2\
         --quiet=2 --conf=many -t 8
clingo version 4.3.1
Reading from design.lp ...
Solving...
Solving...
Solving...
Solving...
Solving...
Optimization: 28138 12 7
OPTIMUM FOUND

Models       : 19
  Optimum    : yes
Optimization : 28138 12 7
Calls        : 5
Time         : 119.240s (Solving: 116.15s 1st Model: 0.70s Unsat: 20.08s)
CPU Time     : 931.380s
Threads      : 8        (Winner: 0)
```

4.4.2 Optimal experimental designs

In Listing 4.4 we show the solving for the problem instance `behaviors-31.lp`. In this case, we find that at least five experimental conditions are required in order to discriminate between every pair of behaviors. Furthermore, for this problem we exploit the multi-threading capacities of *clingo* [Gebser et al., 2012b] using 8 threads: `--conf=many -t 8`. It is worth noting that, each thread uses a different set of pre-configured parameters which allow to traverse the search space with different heuristics. Actually, for small problem instances, there is no significant difference in performance if we compare single- and multi-threading solving. Nonetheless, for larger problem instances, for example `behaviors-66.lp` and `behaviors-91.lp`, the proof of optimality for a candidate answer set is a very demanding task. Hence, using multi-threading is very convenient for such cases. Note that, when we use multiple threads, the solver reports the overall CPU time (931.380s) for all threads and the real time (119.240s) for solving. Notably, in practice and from the user point of view, the real time is more relevant. As already mentioned for the toy example, from the solver's output we can directly read the corresponding values for Θ_{V_S} and Θ_{V_K}, in this case, 12 and 7 respectively. However, the value for Θ_{diff} has to be computed counting predicates `diff/4` in the optimum answer set. We refrain to show the complete answer set in Listing 4.4 for the sake of brevity. In this case, the optimum value for Θ_{diff} is 4412. That is, the designed experiments generate 4412 pairwise output differences over the 31 input-output behaviors.

In Table 4.2 we report results for all problem instances. As expected, as we consider larger sets of input-output behaviors, more experiments (ε) are required in order to discriminate all pairs. Interestingly, the computation time (t_ε) required to find such a minimum number of experiments is relatively small for all problem instances. Notice that, at that time the solver has found an answer set describing a candidate experimental design. Nevertheless, the computation time required to find an optimal experimental design increases several orders of magnitude for the largest instance. Approximately 18 hours of real time and 144 hours of CPU time summing all threads. Notably, for such an instance, the solver considers all possible sets of 7 experimental conditions among the search space $\mathscr{C}(3,2)$, i.e., $\sim 3.8 \times 10^{15}$. Thus, in practice, for even larger problem instances one should use a "reasonable" timeout and take the last answer set found as an approximation to the optimum. We also report the overall number of pairwise differences (Θ_{diff}), its mean (μ_{diff}) and standard deviation (σ_{diff}). For example, the optimal experimental design to discriminate between 38 behaviors, requires 5 experimental conditions which yield 5957 pairwise differences. Furthermore, the mean number of pairwise differences is 8.5 and the standard deviation is 4.9. It is worth noting that, in general, the standard deviation is relatively large. This indicates that certain pairs of behaviors would be "better" discriminated than others. In principle, if one aims at having a more uniform pairwise discrimination, an entropy-based design criterion would be more appropriate. However, such an approach requires numerical computations which are out of scope of our methods but also, they may compromise exhaustiveness and scalability.

Table 4.2: Input-output discrimination over the search space of experiments $\mathscr{C}(3,2)$. We report for each set of input-output behaviors $\mathscr{B}_m$ from Table 3.1, the minimum number of experimental conditions required to discriminate between every pair of behaviors (ε), the time for finding such a minimum (t_ε), the maximum number of pairwise differences (Θ_{diff}), the mean and standard deviation pairwise differences (μ_{diff} and σ_{diff}), the minimum number of stimuli (Θ_{V_S}), the minimum number of inhibitors (Θ_{V_K}), and the time for finding an optimum experimental condition (t_{opt}). Also, in parentheses, we report overall CPU time for the 8 threads.

$\mathscr{B}_m$	ε	t_ε	Θ_{diff}	μ_{diff}	σ_{diff}	Θ_{V_S}	Θ_{V_K}	t_{opt}
4	2	0.061s (0.110s)	21	3.5	1.7	3	0	0.061s (0.120s)
31	5	5.297s (20.85s)	4412	9.5	4.9	12	7	146.5s (1150.2s)
38	5	9.329s (41.81s)	5957	8.5	4.9	11	6	152.5s (1187.4s)
66	7	70.52s (397.2s)	23037	10.7	5.0	16	9	~ 5h (40h)
91	7	160.1s (1059s)	47232	11.5	5.2	16	9	~ 18h (144h)

4.4.3 Analyzing experimental designs

Next, we provide a further analysis of the optimal experimental design found to discriminate between the 91 input-output behaviors. Analogously, the same analysis can be done considering the experimental design found for each set of behaviors. In Figure 4.2 we show the optimal experimental design (Figure 4.2a) and several views of the pairwise differences generated by each experimental condition (Figures 4.2b, 4.2c, and 4.2d). Interestingly, all experimental conditions allow to discriminate between a large number of pairs of input-output behaviors (Figure 4.2b). Notice that, the total number of pairs among 91 behaviors is given by the binomial coefficient, $\binom{91}{2} = 4095$. Thus, each experimental condition in the optimal design discriminates between at least, 50% of the behaviors pairs. Next, we look at which specific readouts we generate differences (Figure 4.2c). On the one hand, for 2 readouts, viz., $HSP27$ and $p53$, we generate pairwise differences with all experiments. On the other hand, for all other readouts, we generate pairwise differences with at most 5 out of the 7 experiments. Moreover, for 4 readouts, viz., AKT, $p90RSK$, $MEK12$, and $GSK3$, we generate pairwise differences with only 1 experimental condition. More precisely, only experiment #3 generates differences over AKT and $GSK3$, and only experiment #1 generates differences for $p90RSK$ and $MEK12$. We note that, such correlations between readouts is explained by the fact that in all logic models (V,ϕ_j), AKT is the only regulator of $GSK3$, i.e., $\phi_j(GSK3) = AKT$, whereas $MEK12$ is the only regulator of $p90RSK$, i.e., $\phi_j(p90RSK) = MEK12$. Finally, we look at the overall pairwise differences generated with each experimental condition (Figure 4.2d). Percentages are computed over the 47232 pairwise differences generated with all experiments.

4.5 Conclusion

In this chapter we have addressed the problem consisting of finding an optimal set of experiments in order to discriminate between several logical input-output behaviors. Various authors have considered the problem of model discrimination in the context of computational

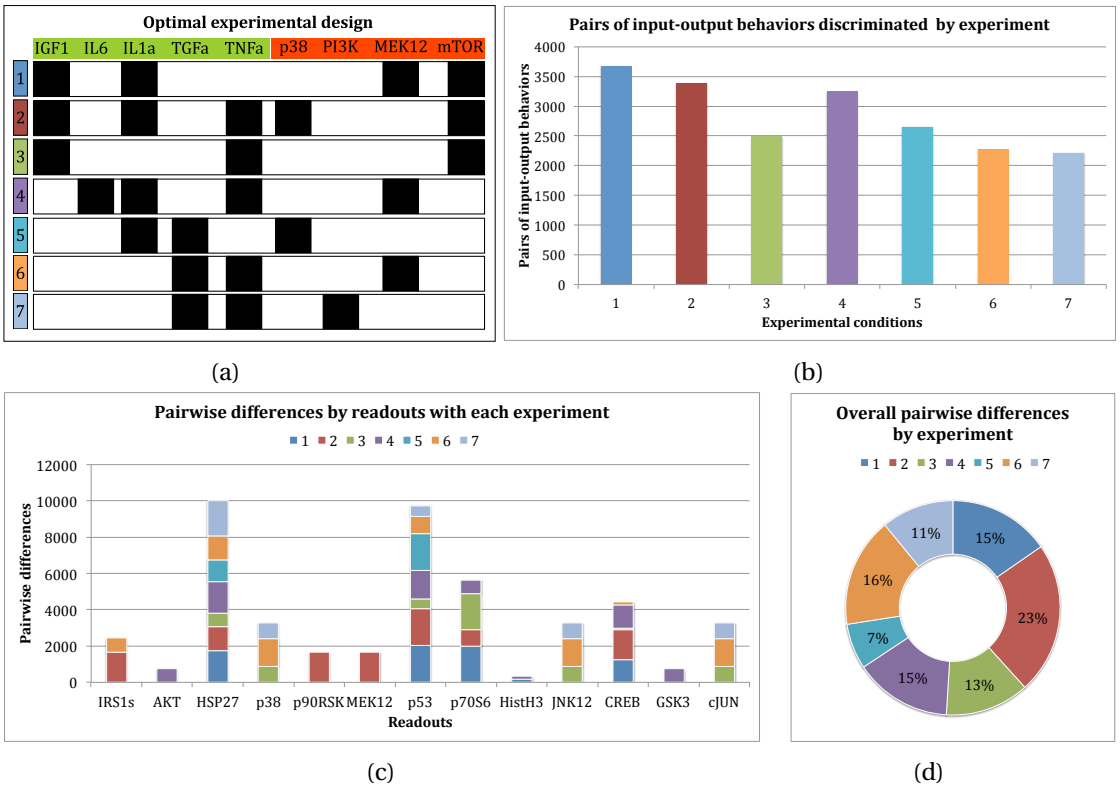


Figure 4.2: Optimal experimental design to discriminate between 91 input-output behaviors. (a) Description of each experimental condition. Black squares indicate the presence of the corresponding stimulus (green header) or inhibitor (red header). (b) Each column describes the number of pairs of behaviors (out of $\binom{91}{2} = 4095$) discriminated by each experimental condition. (c) Number of pairwise differences by readouts with each experimental condition. (d) Overall pairwise differences with each experimental condition.

modeling [Ideker et al., 2000, Yeang et al., 2005, Barrett and Palsson, 2006, Szczurek et al., 2008, Sparkes et al., 2010, Sharan and Karp, 2013]. Nevertheless, their usefulness in practice remains an open question. In contrast to the relatively popular entropy-based approach, and motivated by [Mélykúti et al., 2010], our main design criterion aims at maximizing the number of pairwise (output) differences between rival logical behaviors. Thus, in principle, our work is complementary to previous approaches to experimental design in the context of logical models [Ideker et al., 2000, Sharan and Karp, 2013]. Therein, the aim is to suggest the most informative experiment to discriminate among a set of previously inferred Boolean networks. Instead, we aim at finding the least set of experiments allowing to discriminate every pair of logical behaviors. Notably, both approaches share the vision of an iterative process combining modeling and experimentation for learning accurate logical models.

As in the previous chapter, we have elaborated upon our basic ASP representation (Chapter 2) in order describe the underlying lexicographic multi-objective optimization. Moreover, in this case, we have exploited the incremental and multi-threading solving features of the ASP

solver. This emphasizes the flexibility of ASP as a powerful and general-purpose framework for problem solving. Furthermore, we illustrated the usage of our approach with the sets of input-output behaviors learned in Chapter 3 and suggested optimal experiments towards more accurate logical models of immediate-early response.

Several questions can be investigated in the future. Firstly, we need to develop proper artificial benchmarks in order to evaluate optimal experimental designs. In fact, the generation of biologically meaningful networks and datasets, is a key step towards realistic performance assessment of reverse engineering methods [Marbach et al., 2009]. Secondly, additional knowledge could be considered. For instance, the number of Boolean logic models for each input-output behavior was not taken into account. Also, information about preferences or incompatibilities among stimuli and/or inhibitors could help to prune the search space. Thirdly, in an iterative process of learning and experimentation, it is not clear which strategy may yield more predictive models after a few iterations. On the one hand, we could adopt a "cautious" strategy where we start by identifying a few input-output behaviors for a small tolerance, e.g. 2%, and in the following iterations, we increase the tolerance as we also increase the confidence in learned behaviors. On the other hand, with a "brave" strategy, at every iteration, we should aim at identifying input-output behaviors for a large tolerance, e.g. 10%, and look for experiments to discriminate among possibly many behaviors. Moreover, yet another factor to consider is to take into account certain tolerance over model size during learning which often leads to more input-output behaviors. Altogether, the work presented in this chapter contributes to the loop of hypothesis-driven research in biology [Ideker et al., 2001, Kitano, 2002] in the context of logic-based modeling.

5 Minimal intervention strategies

In previous chapters we have shown how can we iteratively learn and refine an ensemble of logical networks describing the cellular response. Next, a major challenge is how to control it by means of therapeutic interventions. Importantly, progress in this area may have a crucial impact on bio-medical research, drug target identification and diagnosis. This chapter addresses the problem consisting of finding minimal intervention strategies in logical signaling networks. That is, inclusion-minimal sets of activations and inhibitions forcing a set of target species into a desired steady state under various scenarios.

5.1 Introduction

As mentioned in the seminal paper for systems biology [Kitano, 2002], a major challenge in this field is how to systematically control the state of the cell. From an application viewpoint, this means selecting appropriate drugs in order to force the system to reach a state with properties that were specified a priori. In previous chapters we have shown how can we iteratively learn and refine an ensemble of logical networks describing the cellular response. In principle, iterating over such a loop of modeling and experimentation will yield more accurate logical models. Yet, there may be several networks which cannot be discriminated using the available experimental capacities. In which case, instead of selecting a single logical network, in this chapter we aim at reasoning over all of them.

Based on earlier work [Klamt, 2006] on metabolic networks, the notion of *minimal intervention sets* for signaling networks was introduced in [Samaga et al., 2010] and dedicated algorithms were developed to compute them. Intuitively, an "intervention set" represents a set of *knock-ins* and *knock-outs*.[1] Examples for knock-ins are mutations leading to constitutively activated species or a continuous stimulation with external signals, whereas knock-outs may correspond to gene knock-outs or inhibition of certain species by various experimental tools such as small-molecule drugs, antibodies, or RNAi. Then, authors in [Samaga et al., 2010], were particularly

[1] It is worth noting that, our notion of *clamping assignments* introduced in Chapter 2 was originally motivated as an abstraction (closer to standard logic terminology) of intervention sets.

interested in enumerating all minimal intervention sets leading to a specific steady state in a given logical signaling network. Notably, advances on this subject may have a crucial impact on bio-medical research, drug target identification and diagnosis. For instance, one could ask for possible therapeutic interventions which would induce apoptosis (cell death) in cancer cells [Layek et al., 2011]. Furthermore, for some human diseases, including cancer, a single gene does not cause the disorder but instead, the disease may result from the abnormal behavior of several molecules in different pathways [Abdi et al., 2008]. Thus, the method presented in this chapter can also be used as a diagnostic tool to identify causes (mutations) leading to observed cellular responses.

Finding robust minimal intervention strategies. Unfortunately, dedicated algorithms introduced in [Samaga et al., 2010] are computationally demanding due to the highly combinatorial mechanisms in logical networks. Therefore, they are limited to compute small intervention sets and fail to scale over large-scale networks. Importantly, in general, multiple interventions (or mutations) are necessary to cope with robustness and cellular complexity [Stelling et al., 2004]. Moreover, as we have shown in Chapter 3, if the inherent experimental noise is considered there are many logical networks compatible with a given dataset of experimental observations. Thus, identified interventions should fulfill the desired goals in every feasible logical network. Concretely, the aforementioned limitations make it hard to prove that the identified solutions are biologically robust to small variations of the system or its environment. Therefore, by reasoning over several sets of feasible logical networks we expect to find small yet robust intervention strategies.

More broadly, the problem of identifying "key-players" in logical signaling networks has been recently addressed in [Li et al., 2006, Abdi et al., 2008, Wang and Albert, 2011, Layek et al., 2011]. In contrast to our work, these contributions have rather focused on predicting what would happen if certain molecules fail. To that end, authors in [Li et al., 2006, Abdi et al., 2008, Layek et al., 2011] rely on digital circuits fault diagnosis engineering to identify the vulnerable molecules that play crucial roles in the dysfunction of signaling networks. In that context, a high vulnerability suggests that with high probability, the signaling network does not operate correctly if that particular molecule is dysfunctional ("stuck-at-0" or "stuck-at-1" in digital circuits terminology). It is worth noting that, due to computational limitations, in general authors have restricted their studies to small number of simultaneous faults. The approach presented in [Wang and Albert, 2011] does not relies on digital circuits but rather on standard graph theory. Therein, authors proposed two connectivity measures based either on the shortest paths, or on the so-called "elementary signaling modes". Then, using such measures their method provides a ranking of the nodes by the effects of their loss on the connectivity between the network's inputs and outputs. Importantly, despite the specific problem settings and computational approaches in these contributions, all of them have considered a single logical network describing the system.

In the remaining of this chapter, first we revisit the problem defined in [Samaga et al., 2010]

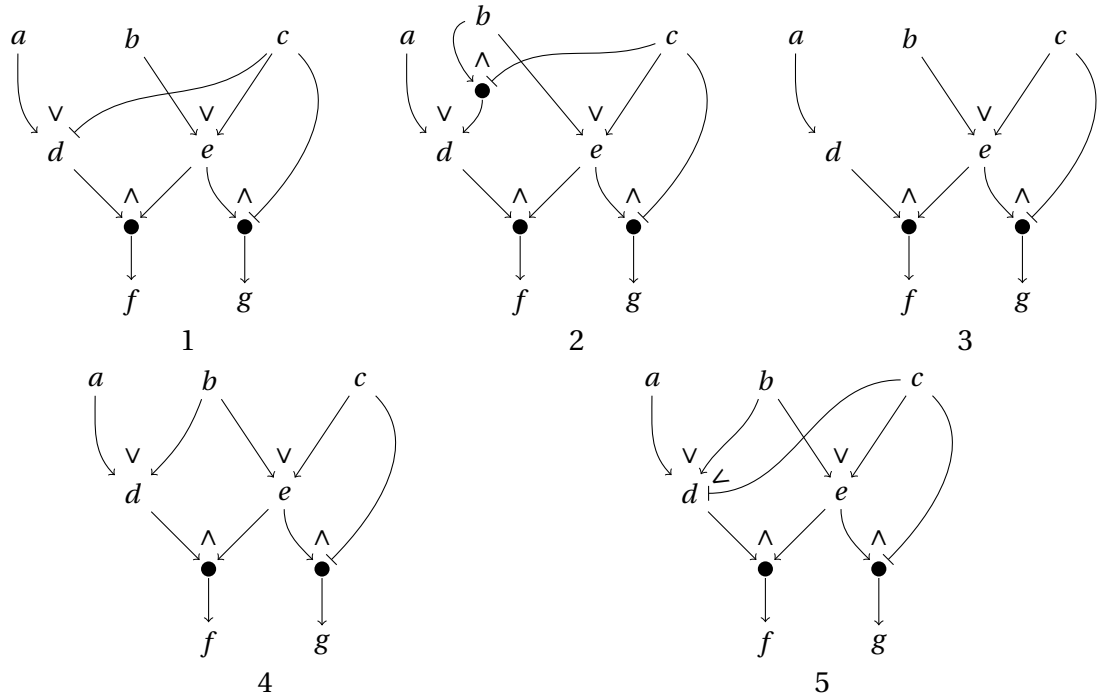


Figure 5.1: Alternative logical networks (V,ϕ_1), (V,ϕ_2), (V,ϕ_3), (V,ϕ_4), and (V,ϕ_5) describing a given system equally well. The correspoinding mappings are: $\phi_1 = \{d \mapsto a \vee \neg c, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, $\phi_2 = \{d \mapsto a \vee (b \wedge \neg c), e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, $\phi_3 = \{d \mapsto a, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, $\phi_4 = \{d \mapsto a \vee b, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$, and $\phi_5 = \{d \mapsto a \vee b \vee \neg c, e \mapsto b \vee c, f \mapsto d \wedge e, g \mapsto e \wedge \neg c\}$.

using the notions introduced in Chapter 2. Then, we adapt our Answer Set Programming (ASP) representation accordingly. Of special interest is here ASP's expressive power to address problems of elevated complexity, in particular, for computing all inclusion-minimal solutions. Finally, we illustrate our approach using the families of (nearly) optimal logic models learned in Chapter 3.

5.2 Problem

5.2.1 Intervention scenarios and strategies

Intervention sets, side constraints and goals. Given a logical signaling network, the aim of an intervention strategy is to identify a set of therapeutic interventions that leads to a steady state satisfying a given goal under some side constraints. In fact, the concepts of an *intervention* (I), *goal* (G), and side *constraints* (C) can be captured as partial two-valued assignments. Moreover, both intervention sets and side constraints are considered clamping assignments as defined in Section 2. To be more precise, given a logical network (V,ϕ), an *intervention scenario* is a pair (G,C) of partial two-value assignments over V where C is considered also as a clamping assignment, and an *intervention set* is a clamping assignment I

over a set of intervention variables $X \subseteq V$.

Intervention strategies. Let (V,ϕ) be a logical network, let (G,C) be an intervention scenario, and $X \subseteq V$ be a set of intervention variables. An intervention set I over X is an *intervention strategy* for (G,C) with respect to (V,ϕ), if for some $j \geq 0$, we have that

$$\Omega^{j}_{(V,\phi|_{C\circ I})}(A_u) = \Omega^{j+1}_{(V,\phi|_{C\circ I})}(A_{\boldsymbol{u}}) \qquad G \subseteq \Omega^{j}_{(V,\phi|_{C\circ I})}(A_{\boldsymbol{u}})$$

with $A_{\boldsymbol{u}} = \{v \mapsto \boldsymbol{u} \mid v \in V\}$. In words, $\Omega^{j}_{(V,\phi|_{C\circ I})}(A_{\boldsymbol{u}})$ is a steady state of the clamped network $(V,\phi|_{C\circ I})$ satisfying the goal conditions in G. Notice the composition of $C \circ I$ indicating that clampings in the intervention set I overwrite clampings in the side constraints C. Recall that for two-valued truth assignments A,B we defined the composition of assignments $A \circ B = (A \setminus \overline{B}) \cup B$ where $\overline{B} = \{v \mapsto \overline{s} \mid v \mapsto s \in B\}$ and $\overline{\boldsymbol{t}} = \boldsymbol{f}$, $\overline{\boldsymbol{f}} = \boldsymbol{t}$.

Finally, the *intervention set problem* consists in deciding whether there is an intervention strategy for an intervention scenario (G,C) with respect to a logical network (V,ϕ).

For illustration, let us consider the toy logical network (V,ϕ_1) in Figure 5.1 along with the intervention scenario $(G,C) = (\{d \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}, \{e \mapsto \boldsymbol{t}\})$. In this example, the intervention scenario requires the inhibition of d and g, given that e is stimulated. Next, the intervention set $I = \{a \mapsto \boldsymbol{f}, c \mapsto \boldsymbol{t}\}$ where a is inhibited and c is stimulated, satisfies the scenario yielding the steady state $\{a \mapsto \boldsymbol{f}, b \mapsto \boldsymbol{u}, c \mapsto \boldsymbol{t}, d \mapsto \boldsymbol{f}, e \mapsto \boldsymbol{t}, f \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$. Therefore, I is an intervention strategy for (G,C) with respect to (V,ϕ_1). In fact, I is also an intervention strategy for (G,C) with respect to (V,ϕ_2) and (V,ϕ_3) in Figure 5.1 as well. But now, if we consider the toy logical network (V,ϕ_4) in Figure 5.1 along with the same intervention scenario. The previous intervention set I yields the steady state $\{a \mapsto \boldsymbol{f}, b \mapsto \boldsymbol{u}, c \mapsto \boldsymbol{t}, d \mapsto \boldsymbol{u}, e \mapsto \boldsymbol{t}, f \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}$. Notably, since $\phi_4(d) = a \vee b$, and b is neither constrained nor intervened, d remains *undefined* in this case (see Table 2.2). Similarly, the same happens with respect to (V,ϕ_5). Therefore, while the intervention set $I = \{a \mapsto \boldsymbol{f}, c \mapsto \boldsymbol{t}\}$ is an intervention strategy for (V,ϕ_1), (V,ϕ_2), and (V,ϕ_3), when we consider either (V,ϕ_4) or (V,ϕ_5), the inhibition of d in our intervention goal is not fulfilled. In this toy example, it is relatively easy to see that the intervention set $I' = \{a \mapsto \boldsymbol{f}, b \mapsto \boldsymbol{f}, c \mapsto \boldsymbol{t}\}$ is an intervention strategy for (G,C) with respect to all logical networks (V,ϕ_i) with $i = 1,\ldots,5$. Clearly, the same analysis for several intervention scenarios with respect to thousands of large-scale logical networks is out of reach to do it by hand.

5.2.2 Enumeration of minimal (bounded) intervention strategies

Authors in [Samaga et al., 2010], were particularly interested in enumerating all minimal (bounded) intervention strategies with respect to a single logical network. However, as we illustrate in this thesis, and other authors have shown by considering real-world networks and data [Saez-Rodriguez et al., 2009, Chen et al., 2009], it often happens that *the* model is non-identifiable. Therefore, as several logical networks can describe a given biological

system equally or similarly well, identified intervention strategies should fulfill all intervention scenarios in every possible logical network. Importantly, as we have shown with the toy example above, intervention strategies with respect to a logical network are very likely to fail in another network which may describe the system similarly well. Notably, this may have a strong impact on the robustness of identified solutions. In order to overcome this issue, we extend the problem settings in order to consider an ensemble of logical networks, e.g. resulting from the enumeration of (nearly) optimal Boolean logic models described in Chapter 3. This way, we are able to reason over all possible networks without any a priori bias and towards more robust insights.

Now, let us define further intervention strategies relying on a finite family $(V,\phi_i)_{i\in N}$ of logical networks, a finite family $(G_j,C_j)_{j\in J}$ of intervention scenarios and k some positive integer.

- A *multi-scenario intervention strategy* for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ is an intervention strategy for each (G_j,C_j) wrt (V,ϕ_i) for each $j\in J$ and $i\in N$.
- A *bounded intervention strategy* for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ and k is a multi-scenario intervention strategy for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ of cardinality $k'\leq k$.
- A *minimal bounded intervention strategy* for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ and k is a $\subseteq$-minimal multi-scenario intervention strategy for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ of cardinality $k'\leq k$.

In what follows, we focus on the enumeration of all minimal (bounded) intervention strategies for given families of intervention scenarios $(G_j,C_j)_{j\in J}$ and logical networks $(V,\phi_i)_{i\in N}$.

5.3 Minimal intervention strategies with Answer Set Programming

5.3.1 Instance

Once again, the representation of the problem instance is an extension from the one described in Listing 2.1 in order to describe a finite family of logical networks and clamping assignments. To be more precise, instead of having facts over predicates `formula/2`, we consider facts over predicates `formula/3` as in Chapter 4. Let $(V,\phi_i)_{i\in N}$ be a finite family of logical networks. The facts `formula(`i`,`v`,`$s_{\phi_i(v)}$`)` map variables $v\in V$ to their corresponding formulas $\phi_i(v)$ for each $i\in N$. Meanwhile, facts over predicates `variable/1`, `dnf/2` and `clause/3` remain the same as in Listing 2.1. We use facts over predicate `candidate/1` to denote the intervention variables that can be part of an intervention set. This allows us to control on which variables interventions are permitted, for example one can exclude interventions over constrained or goal variables. Next, we represent the family of intervention scenarios $(G_j,C_j)_{j\in J}$ using predicates `scenario/1`, `goal/3`, and `constrained/3`. The facts `scenario(`j`)` denote the scenarios to consider. The facts `goal(`j`,`v`,`s`)` with $s=1$ (resp. $s=-1$) if $G_j(v)=\boldsymbol{t}$ (resp. $G_j(v)=\boldsymbol{f}$) and `constrained(`j`,`v`,`s`)` with $s=1$ (resp. $s=-1$) if $C_j(v)=\boldsymbol{t}$ (resp. $C_j(v)=\boldsymbol{f}$)

denote the respective intervention goals and side constraints in each scenario (G_j, C_j). Notice that both predicates, `goal/3` and `constrained/3` are a straightforward elaboration upon predicates `clamped/2` given in Listing 2.2.

Listing 5.1 shows the instance representation of our toy example. That is, the five logical networks in Figure 5.1 together with the intervention scenario $(G_1, C_1) = (\{d \mapsto \boldsymbol{f}, g \mapsto \boldsymbol{f}\}, \{e \mapsto \boldsymbol{t}\})$. Notably, for the sake of understanding, we consider a toy example with a single intervention scenario. But in general, this instance representation and the logic program given below, support several scenarios.

Listing 5.1: Toy example problem instance (`toy.lp`)

```
variable(a). variable(b). variable(c). variable(d).
variable(e). variable(f). variable(g).

formula(1,d,7). formula(1,e,0). formula(1,f,3).  formula(1,g,2).
formula(2,d,4). formula(2,e,0). formula(2,f,3).  formula(2,g,2).
formula(3,d,5). formula(3,e,0). formula(3,f,3).  formula(3,g,2).
formula(4,d,1). formula(4,e,0). formula(4,f,3).  formula(4,g,2).
formula(5,d,6). formula(5,e,0). formula(5,f,3).  formula(5,g,2).

dnf(5,8). dnf(6,8).  dnf(2,14). dnf(0,2).  dnf(0,0). dnf(6,7).  dnf(4,9).
dnf(7,8). dnf(6,0).  dnf(1,8).  dnf(1,0).  dnf(4,8). dnf(3,16). dnf(7,7).

clause(16,d,1). clause(0,b,1).  clause(14,e,1). clause(16,e,1).
clause(9,c,-1). clause(8,a,1).  clause(9,b,1).  clause(2,c,1).
clause(7,c,-1). clause(14,c,-1).

scenario(1).
constrained(1,e,1). goal(1,d,-1). goal(1,g,-1).

candidate(a). candidate(b). candidate(c). candidate(f).
```

5.3.2 Encoding

Next we describe our encoding for solving the minimal intervention set problem as described earlier. Our ASP encoding is shown in Listing 5.2.

Listing 5.2: Logic program for finding intervention strategies (`control.lp`)

```
goal(T,S)         :- goal(_,T,S).
goal(T)           :- goal(T,_).
constrained(Z,E) :- constrained(Z,E,_).
constrained(E)    :- constrained(_,E).
model(M)          :- formula(M,_,_).

satisfy(M,V,W,S) :- formula(M,W,D); dnf(D,C); clause(C,V,S).
closure(M,V,T)    :- model(M); goal(V,T).
closure(M,V,S*T) :- closure(M,W,T); satisfy(M,V,W,S); not goal(V,-S*T).
closure(V,T)      :- closure(_,V,T).
```

```
11
12 { intervention(V,S) : closure(V,S) , candidate(V) }.
13 :- intervention(V,1); intervention(V,-1).
14 intervention(V) :- intervention(V,S).
15
16 eval(M,Z,V,S) :- scenario(Z); intervention(V,S); model(M).
17 eval(M,Z,E,S) :- model(M); constrained(Z,E,S);
18                  not intervention(E).
19 free(M,Z,V,D) :- formula(M,V,D); scenario(Z);
20                  not constrained(Z,V); not intervention(V).
21
22 eval_clause(M,Z,C,-1) :- clause(C,V,S); eval(M,Z,V,-S); model(M).
23
24 eval(M,Z,V, 1) :- free(M,Z,V,D); dnf(D,C);
25                   eval(M,Z,W,T) : clause(C,W,T).
26 eval(M,Z,V,-1) :- free(M,Z,V,D); eval_clause(M,Z,C,-1) : dnf(D,C).
27
28 :- goal(Z,T,S); model(M); not eval(M,Z,T,S).
29
30 #const maxsize=0.
31 :- maxsize>0; maxsize + 1 { intervention(X) }.
32
33 #show intervention/2.
```

Guessing an intervention set. In Lines 1-5 we define auxiliary domain predicates used in the remainder of the encoding. Lines 7-10 deserve closer attention since they allow us to reduce significantly the search space of candidate solutions. Note that, each candidate variable could be intervened positively, negatively, or not intervened. Then, if n is the number of candidate variables for interventions, this leads to 3^n possible intervention sets. We incorporate a preprocessing step introduced in [Samaga et al., 2010] that prunes variable assignments that can never be part of an intervention strategy. The idea is to inductively collect all assignments that, in principle, could be used to support a goal. First we gather all assignments that make a literal in a clause true and associate it with variable of the associated DNF (Line 7). Starting from the assignments that can satisfy a goal literal directly (Line 8), we inductively consider variable assignments (Line 9) that can support the assignments collected so far. Finally, in Line 10 we project all "relevant" interventions regardless of the logical network. Let us illustrate this with our toy example. In order to satisfy `goal(1,d,-1)` with respect to (V,ϕ_1), one would never consider to intervene the variable b. Since b does not reach d in (V,ϕ_1), an intervention on b is meaningless in order to inactivate d. However, when we look at (V,ϕ_2), (V,ϕ_4), and (V,ϕ_5), the variable b reaches d positively in those networks. Hence, a negative intervention on b could help to satisfy `goal(1,d,-1)` in such cases. Next, we use a choice rule in Line 12 to generate candidate solutions. For example, one could generate the intervention set consisting of `intervention(a,-1)`, `intervention(b,-1)`, and `intervention(c,1)`. We only choose interventions collected in the preprocessing step above. It is worth noting that, even for such a small problem instance, the search space is reduced

significantly. In our example, we have only 4 candidate variables for interventions, namely, a, b, c, and f. On the one hand, without pruning, we should evaluate $3^4 = 81$ intervention sets. On the other hand, with the preprocessing, we are able to detect (at grounding time) that only interventions on a, b, and c are relevant since f does not reach any goal. Moreover, only a negative intervention is relevant for both a and b. Notably, this leads to $2^2 \times 3^1 = 12$ intervention sets to be evaluated. Next, the integrity constraint in Line 13 eliminates contradictory interventions, e.g. `intervention(c,1)` and `intervention(c,-1)`. Whereas Line 14 simply projects the intervention set to the intervened variables regardless of their signature.

Fixpoint per logical network and intervention scenario. Lines 16-26 elaborate on the rules from Listings 2.3, 2.4 and 2.6 given in Section 2 in order to consider several logical networks simultaneously and compute the fixpoint for each of them accordingly. To be more precise, we need to describe which variables are clamped (in all networks) according to the side constraints C_j in each scenario j and the intervention set I, namely, $(V, \phi_i|_{C_j \circ I})$. Towards this end, we use the predicate `eval/4`, namely `eval(`i`,`j`,`v`,`s`)` to represent that in the network (V, ϕ_i) and intervention scenario (G_j, C_j), the variable v is clamped to value s. Following the previous example, this will generate predicates `eval(`i`,1,e,1)`, `eval(`i`,1,a,-1)`, `eval(`i`,1,b,-1)`, and `eval(`i`,1,c,1)` with $i = 1, \ldots, 5$. The remaining rules are adapted accordingly in order to compute for each logical network (V, ϕ_i) and intervention scenario (G_j, C_j), the corresponding fixpoint of $\Omega_{(V, \phi_i|_{C_j \circ I})}$. For our example, we can see how the positive intervention over c is propagated as follows. Since the variable d is not intervened, its formula is "free" in all scenarios and logical networks, namely, `free(1,1,d,7)`, `free(2,1,d,4)`, `free(3,1,d,5)`, `free(4,1,d,1)`, and `free(5,1,d,6)`. Furthermore, let us illustrate the case for the logical network (V, ϕ_5). In such a network, we have $\phi_5(d) = a \vee b \vee \neg c$, which in turn is represented by predicates `dnf(6,8)`, `dnf(6,0)`, `dnf(6,7)`, `clause(8,a,1)`, `clause(0,b,1)`, and `clause(7,c,-1)`. Then, from the rule in Line 22 we derive for $i = 1, \ldots, 5$, predicates `eval_clause(`i`,1,8,-1)`, `eval_clause(`i`,1,0,-1)`, and `eval_clause(`i`,1,7,-1)`. Finally, in Line 26 we can derive predicates `eval(`i`,1,d,-1)` with $i = 1, \ldots, 5$.

Goals satisfaction. Line 28 declares an integrity constraint in order to eliminate answer sets describing intervention sets that do not satisfy all goals in every logical network and intervention scenario. Notably, following with our example, `goal(1,d,-1)` is satisfied with respect to all logical networks. Similarly, `goal(1,g,-1)` is satisfied as well. Therefore, the intervention set described by predicates `intervention(a,-1)`, `intervention(b,-1)`, and `intervention(c,1)`, is an intervention strategy with respect to all networks and interventions scenarios. Finally, the statements in Line 30 and 31 allows us to optionally bound the problem by considering only intervention sets up to a given size.

The following result shows the correctness of our ASP representation. We denote with $\tau((V, \phi_i)_{i \in N}, (G_j, C_j)_{j \in J}, k)$ the set of facts describing the instance as in Listing 5.1, and with

$\Pi_{5.2}$ the set of rules given in Listing 5.2.

Proposition 5.3.1. *Let $(V,\phi_i)_{i\in N}$ be a finite family of logical networks. Let $(G_j,C_j)_{j\in J}$ be a finite family of intervention scenarios and k some positive integer.*
Then, there is an answer set X of $\tau((V,\phi_i)_{i\in N},(G_j,C_j)_{j\in J},k)\cup\Pi_{5.2}$ such that $I=\{v\mapsto \boldsymbol{t}\mid intervention(v,1)\in X\}\cup\{v\mapsto \boldsymbol{f}\mid intervention(v,-1)\in X\}$ if and only if I is a bounded intervention strategy for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ and k.

5.3.3 Solving

Normally ASP solvers allow for computing cardinality-minimal solutions whereas we are interested in finding $\subseteq$-minimal solutions. In [Kaminski et al., 2013] we have shown how one can overcome this limitation by means of meta-programming and disjunctive logic programs [Gebser et al., 2011a] or by using a specialized solver like *hclasp* [Gebser et al., 2013]. However, herein we leverage the functionality recently introduced in *clasp* 3 series which allow for computing $\subseteq$-minimal solutions *out-of-the-box* by incorporating the features from *hclasp*. Importantly, this requires to use very specific command-line options for *clasp*. We refer the reader to *clasp*'s documentation for more details.

In Listing 5.3 we show the only intervention strategy found for the intervention scenario $(G_1,C_1)=(\{d\mapsto \boldsymbol{f},g\mapsto \boldsymbol{f}\},\{e\mapsto \boldsymbol{t}\})$ with respect to the five logical networks in Figure 5.1.

Listing 5.3: Enumeration of all minimal intervention strategies

```
$ gringo control.lp toy.lp |\
  clasp --dom-pref=32 --dom-mod=6 --heu=domain -n0
clasp version 3.0.3
Reading from stdin
Solving...
Answer: 1
intervention(a,-1) intervention(b,-1) intervention(c,1)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

5.4 Empirical evaluation

5.4.1 Real-world problem instance

Logical networks. In previous chapters we have shown how can we learn and refine an ensemble of logical networks in an iterative fashion. In particular, in Chapter 3, using real-world signaling pathways in human liver cells and a publicly available phospho-proteomics dataset, we have learned Boolean logic models for various levels of tolerance over fitness

(see Table 3.1). Importantly, in regards of the available experimental observations and their intrinsic uncertainty, such logical networks explain the data equally well. Thus, in Chapter 4, we have presented a method in order to find an optimal experimental design to discriminate all models at hand. In principle, iterating over this loop of modeling and experimentation will yield more accurate logical models. Nevertheless, there may be several models which cannot be discriminated using the available experimental capacities. Therefore, instead of selecting a single model, now we are interested in reasoning over all networks, as described in Section 5.2, in order to find all minimal intervention strategies for each tolerance, namely, 16 networks for 0% tolerance, 144 networks for 2% tolerance, 2150 networks for 4%, 2306 networks for 4%, 3524 networks for 8% tolerance, and 5306 networks for 10% tolerance.

Intervention scenario. Recall that we have learned Boolean logic models from the prior knowledge network in Figure 3.2 describing signaling pathways in human liver cells. In principle, one could consider as intervention scenarios, any combination of goals and side constraints over all species in the network. However, in practice we are interested in finding intervention strategies for biologically relevant scenarios. In this case, we consider a single intervention scenario given by $(G,C) = (\{cJUN \mapsto \boldsymbol{f}, CREB \mapsto \boldsymbol{f}\}, \emptyset)$. That is, we require to inactivate both species $cJUN$ and $CREB$ without any side constraints. Note that we choose this intervention scenario for illustration purpose but at the same time, it is motivated by biological knowledge about the role these two species play in the signaling pathways under consideration.

Problem instances. Next, we consider the problem instances given by the intervention scenario $(G,C) = (\{cJUN \mapsto \boldsymbol{f}, CREB \mapsto \boldsymbol{f}\}, \emptyset)$ with respect to each set of logical networks listed in Table 3.1. Let us denote with $\mathscr{M}_i$ the corresponding set of i logical networks, namely, $\mathscr{M}_{16}$, $\mathscr{M}_{144}$, $\mathscr{M}_{2150}$, $\mathscr{M}_{2306}$, $\mathscr{M}_{3524}$, and $\mathscr{M}_{5306}$. Then, in what follows, we denote with the filename `networks-i.lp`, the set of logic facts describing each problem instance as illustrated in Listing 5.1. Furthermore, we assume all variables or species, except for $cJUN$ and $CREB$, as candidates to be part of an intervention strategy. Eventually, we could restrict ourselves to more specific set of candidate variables which would reduce the search space. Finally, for each problem instance, we consider the enumeration of all inclusion-minimal intervention strategies, either bounded by the set cardinality k with $k = 2,3,4,5$, or unbounded.

5.4.2 Minimal intervention strategies

Listing 5.4: Enumeration of all minimal intervention strategies wrt 2306 logical networks

```
$ clingo control.lp networks-2306.lp -n0 --quiet\
         --dom-pref=32 --dom-mod=6 --heu=domain
clingo version 4.4.0
Reading from control.lp ...
Solving...
SATISFIABLE
```

```
Models        : 78
Calls         : 1
Time          : 10.641s (Solving: 1.89s 1st Model: 0.03s Unsat: 0.04s)
CPU Time      : 10.640s
```

In Listing 5.4 we show the solving for the unbounded problem instance `networks-2306.lp`. In this case, we find 78 inclusion-minimal intervention strategies satisfying our scenario with respect to the 2306 logical networks in $\mathscr{M}_{2306}$. It is worth noting the different computation times reported by the solver in this case. Recall that the process for problem solving with ASP is divided into *grounding* and *solving*. In fact, for these instances, the search space of possible intervention sets is relatively small compared to other instances reported in the next section. In this cases, the number of candidate variables to be intervened is only 29. This would require to evaluate $3^{29} =\sim 6.8 \times 10^{13}$ possible intervention sets. However as explained above, during grounding we can reduce this number significantly. The precise reduction depends on each set of logical networks but ranges from 2048 to $\sim 10^6$. Notably, these are rather small instances compared to other cases where the number of possible intervention sets is $\sim 6 \times 10^{47}$. However, the fact that we consider many logical networks simultaneously, leads to a more demanding grounding. Therefore, for these cases, most of the computation time is actually during grounding rather than solving. For example, in Listing 5.4 the CPU time is around 10.5 seconds, whereas only 2 seconds are strictly during search.

Next, in Table 5.1 we report results for all problem instances. For each set of logical networks, namely, $\mathscr{M}_{16}, \mathscr{M}_{144}, \mathscr{M}_{2150}, \mathscr{M}_{2306}, \mathscr{M}_{3524}$, and $\mathscr{M}_{5306}$, we report the number of inclusion-minimal intervention strategies up to a given size $k = 2,3,4,5$, or unbounded (∞). Also, in order to emphasize the computation time during grounding or solving, we show the total CPU time reported by *clingo* but also, in parentheses, the time strictly during solving. As expected, both grow as we increase the number of logical networks. However, for the largest set of logical networks, that is $\mathscr{M}_{5306}$, total CPU time is only 27.37 seconds, whereas around 6 seconds are needed for solving. Interestingly, we have found exactly the same intervention strategies for $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$, for $\mathscr{M}_{2150}$ and $\mathscr{M}_{2306}$, and for $\mathscr{M}_{3524}$ and $\mathscr{M}_{5306}$. This is supported by results

Table 5.1: Enumeration of all inclusion-minimal intervention strategies up to a given size $k = 2,3,4,5$, or unbounded (∞). For each set of logical networks ($\mathscr{M}_i$), we report the number of intervention strategies (I) and the CPU time (t_{enum}). Also, in parentheses, we report the time strictly during solving.

	≤ 2		≤ 3		≤ 4		≤ 5		∞	
$\mathscr{M}_i$	I	t_{enum}	I	t_{enum}	I	t_{enum}	I	t_{enum}	I	t_{enum}
16	7	0.04s (0.001s)	14	0.04s (0.001s)	14	0.03s (0.001s)	14	0.03s (0.001s)	14	0.04s (0.001s)
144	7	0.36s (0.005s)	14	0.36s (0.008s)	14	0.36s (0.008s)	14	0.36s (0.008s)	14	0.36s (0.008s)
2150	1	8.39s (0.239s)	18	9.1s (0.806s)	70	9.86s (1.624s)	78	9.82s (1.648s)	78	9.93s (1.771s)
2306	1	9.44s (0.326s)	18	9.87s (0.945s)	70	10.55s (1.713s)	78	10.54s (1.716s)	78	10.68s (1.884s)
3524	1	14.19s (0.355s)	16	15.3s (1.42s)	56	16.83s (2.877s)	98	17.51s (3.69s)	104	17.91s (4.042s)
5306	1	22.41s (0.757s)	16	23.62s (2.159s)	56	26.33s (4.84s)	98	26.81s (5.356s)	104	27.37s (5.912s)

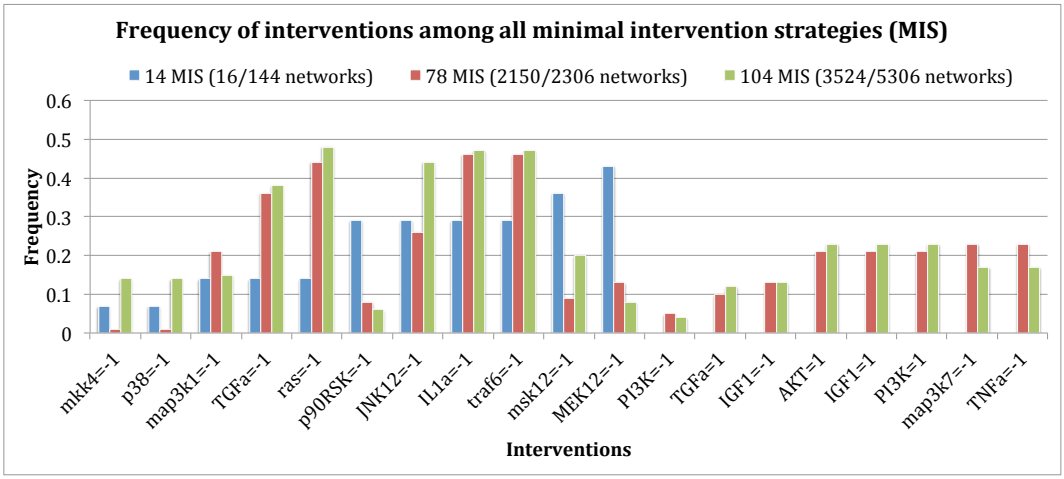


Figure 5.2: Frequency of single interventions. Vertical bars describe the frequency of each single intervention among all minimal intervention strategies with respect to different sets of logical networks. All interventions occur in less than 50% of the strategies yet, some interventions are clearly more frequent than others. Pairs of families $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$ (blue bars), $\mathscr{M}_{2150}$ and $\mathscr{M}_{2306}$ (red bars), and $\mathscr{M}_{3524}$ and $\mathscr{M}_{5306}$ (green bars), have the same strategies and thus we plot them together.

reported in Chapter 3 where such pairs of families shown similar number of input-output behaviors (Table 3.1) and MSEs distributions (Figure 3.3).

Now, in order to have a closer look at what interventions are found for each case, in Figure 5.2 we show the frequency of each single intervention among all strategies for the unbounded case (∞ column in Table 5.1). We note that, all single interventions occur in less than 50% of the strategies. This suggests that, there is no evident species in order to control the system, but instead, several combinatorial mechanisms require to combine multiple interventions together. Nonetheless, some interventions are clearly more frequent than others. But still, while a given intervention, e.g., $MEK12 \mapsto \boldsymbol{f}$, occurs in almost half of the strategies with respect to $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$, with respect to larger sets of networks, its frequency drops to 10%. Analogously, but in the opposite way, the same happens with $ras \mapsto \boldsymbol{f}$. Notably, interventions over certain species are found only when we consider larger sets of logical networks (red and green bars). In fact, as shown in Table 5.1, while only 14 intervention strategies are found with respect to $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$, if we consider either $\mathscr{M}_{3524}$ or $\mathscr{M}_{5306}$, we found up to 104 inclusion-minimal intervention strategies. Therefore, reasoning over a large family of feasible logical networks allows us to identify alternative intervention strategies. Furthermore, this poses the question of whether certain strategies are more robust than others and how to identify them.

5.4.3 Towards small yet robust intervention strategies

Nowadays, in signaling networks, more than four or five interventions combining different stimuli and inhibitors can be considered as a long-term technological perspective. Therefore,

in practice, small intervention strategies are preferred in terms of experimental feasibility. However, too small strategies are very likely to fail due to robustness and cellular complexity [Stelling et al., 2004]. Hence, we need to find a balance between small (and feasible) yet robust intervention strategies to control the cell. Our approach to achieve such a challenge is by reasoning over all feasible logical networks. In Figure 5.3 we show the 14 intervention strategies found with respect to $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$. Each path from the node "SCENARIOS CONSTRAINTS" to the node "SCENARIOS GOALS", describes a different strategy where species are intervened according to their coloring: red for inhibitions and green for activations. Nonetheless, in this case all strategies consist only of inhibitions. For instance, $\{MEK12 \mapsto \boldsymbol{f}, IL1a \mapsto \boldsymbol{f}\}$ and $\{MEK12 \mapsto \boldsymbol{f}, JNK12 \mapsto \boldsymbol{f}, msk12 \mapsto \boldsymbol{f}\}$, are two different intervention strategies of size 2 and 3 respectively. Now, if we restrict ourselves to logical networks in $\mathscr{M}_{16}$ or $\mathscr{M}_{144}$, these 14 intervention strategies are, in principle, all equally valid. But, if we consider a larger set of feasible networks, e.g., $\mathscr{M}_{5306}$, we can rule out more than half of these strategies before going to the laboratory. More precisely, out of the 14 strategies shown in Figure 5.3, only 6, the ones highlighted with stronger red, are conserved among all logical networks in $\mathscr{M}_{5306}$. That is, the intervention strategies valid with respect to all $\mathscr{M}_i$ are:

$$\left\{\begin{array}{l} \{MEK12 \mapsto \boldsymbol{f}, map3k1 \mapsto \boldsymbol{f}\} \\ \{MEK12 \mapsto \boldsymbol{f}, JNK12 \mapsto \boldsymbol{f}, msk12 \mapsto \boldsymbol{f}\} \\ \{MEK12 \mapsto \boldsymbol{f}, JNK12 \mapsto \boldsymbol{f}, p38 \mapsto \boldsymbol{f}\} \\ \{MEK12 \mapsto \boldsymbol{f}, JNK12 \mapsto \boldsymbol{f}, mkk4 \mapsto \boldsymbol{f}\} \\ \{p90RSK \mapsto \boldsymbol{f}, JNK12 \mapsto \boldsymbol{f}, msk12 \mapsto \boldsymbol{f}\} \\ \{p90RSK \mapsto \boldsymbol{f}, map3k1 \mapsto \boldsymbol{f}, msk12 \mapsto \boldsymbol{f}\} \end{array}\right\}$$

Intuitively, this way we are able to detect earlier those intervention strategies that are too specific and do not take into account alternative pathways. Therefore, by reasoning over several sets of feasible logical networks we may find small yet robust intervention strategies. Also, it is worth noting that, species $msk12$, $mkk4$, and $map3k1$ are neither measured, nor manipulated in the dataset used for learning each set of logical networks. Hence, the fact that we found strategies involving such species suggests the necessity of further experimentations in order to elucidate their role in the biological system at hand.

5.4.4 Comparing with a dedicated algorithm

In [Kaminski et al., 2013] we have evaluated the performance and scalability of our ASP-based solution over four real-world and biologically relevant benchmarks. Three of them were used in [Samaga et al., 2010] and their corresponding (manually curated) logical networks were recently published [Saez-Rodriguez et al., 2007, Samaga et al., 2009]. Further, the fourth benchmark consisted of an unpublished logical network provided by Axel von Kamp and Steffen Klamt. While the authors in [Samaga et al., 2010] restricted their study to a maximum cardinality of 3, we extended this limitation to a maximum cardinality of 10 or no limit at all. Importantly, the search space for these problem instances is significantly larger than the ones

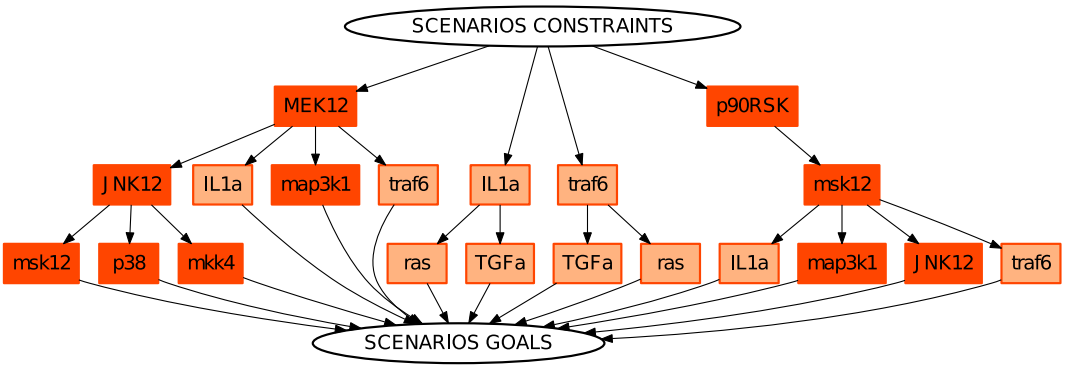


Figure 5.3: Robust intervention strategies. We plot the 14 intervention strategies found with respect to $\mathscr{M}_{16}$ and $\mathscr{M}_{144}$. Each path from the node "SCENARIOS CONSTRAINTS" to the node "SCENARIOS GOALS", describes a different strategy where species are intervened according the their coloring: red for inhibitions and green for activations. Out of the 14 strategies only 6, the ones highlighted with stronger red, are conserved among all logical networks in $\mathscr{M}_{5306}$.

shown above. More precisely, the number of candidate intervention sets to be considered by the solver range from 10^{11} to 10^{47}. Nonetheless, experiments have shown that our approach outperforms the dedicated algorithm in up to four orders of magnitude (for small strategies still feasible for the algorithm). This was not very surprising since such an algorithm is based on a standard breadth-first search using additional techniques for search space reduction. More importantly, using ASP we were able to search for significantly larger intervention strategies or even solve the unbounded problem despite the very large search space. Similarly to what we have shown above, if we consider a small number of interventions, the number of solutions is in the order of tens. But, if a larger number of interventions is allowed, we found hundreds or even thousands of feasible solutions. Note that, all benchmarks we evaluated in [Kaminski et al., 2013] consisted of a single and manually curated logical network. Hence, we expect that our approach for reasoning over several alternative logical networks (as they become available) can help to rule out solutions and identify robust strategies satisfying intervention scenarios with respect to all networks.

5.5 Conclusion

In this chapter we have addressed the problem consisting of finding intervention strategies in logical signaling networks. That is, inclusion-minimal sets of activations and inhibitions forcing a set of target species into a desired steady state under various scenarios. The identified interventions can be exploited to control the biological system or interpreted as causes for an abnormal cellular response (diagnosis).

In this context, dealing with large-scale and (possibly many) alternative networks describing the same biological system, leads to challenging combinatorial problems that require advanced solving technologies. In fact, previous work on this subject consists of dedicated

algorithms and special purpose search space reduction techniques for coping with combinatorial explosion [Samaga et al., 2010]. More broadly, other authors have elaborated upon concepts of electronic circuit fault diagnosis engineering [Li et al., 2006, Abdi et al., 2008, Layek et al., 2011] and standard graph theory [Wang and Albert, 2011], for the identification of essential components in the network. Importantly, despite the specific problem settings and computational approaches in these contributions, all of them have considered a single logic network describing the system. However, as we illustrate in this thesis, due to limited observability or the uncertainty in experimental measurements, it often happens that a single model is non-identifiable when we use reverse engineering methods. Thus, identified interventions (or essential components) should fulfill the desired goals in every feasible logical network. Notably, this may lead to more complex but also more robust solutions. As mentioned earlier, large-scale (e.g. > 10) interventions combining different stimuli and inhibitors, can be considered as a long-term technological perspective. Nevertheless, in general, we did not find intervention strategies of size bigger than 10. On the one hand, this suggests that extending the set of interventions may not be interesting for the systems we have considered in our benchmarks. On the other hand, knowing that a large number of interventions are required to reach certain state could help to understand (at least theoretically) systems' robustness and complexity. Hence, being able to compute both small and large admissible intervention strategies over an ensemble of feasible logical networks, appears as an interesting and distinct feature of our approach.

Both computational and biological perspective tracks are open. On the computational side, despite the successful performance in all benchmarks, a precise estimation of the empirical complexity appears to be non trivial and very specific to each case. Given a problem instance, the number of candidate solutions to be considered by the solver can be computed analytically. However, experiments have shown that this is not the only parameter determining the empirical complexity. The number of goal variables and their location in the networks, the number of intervention scenarios, and the number of networks and their structural properties may have a strong impact on the computational efforts required in practice. On the biological side, in the light of such a large number of solutions, the way to select among them arises. In fact, we expect that our approach introduced in [Kaminski et al., 2013] and extended here for reasoning over several alternative logical networks, will help to rule out solutions and identify small yet robust intervention strategies. Therefore, the work presented in this chapter is a key methodological contribution allowing systems biologists to draw robust insights even under uncertainty.

6 Software toolbox: caspo

In practice, problem solving using ASP involves three main steps. First, certain input data must be converted to logic facts describing the corresponding problem instance. Second, the problem instance needs to be combined with the specific set of logic rules encoding the problem to be solved. Third, answer sets returned by the ASP solver must be interpreted according to the representation used in the encoding. Hence, a software package providing an interface to the ASP-based solutions detailed in previous chapters would ease the accessibility for end-users, as well as the integration with available tools for simulation and analysis of logical models. This chapter presents such a software tool and illustrates its main features.

6.1 Introduction

In practice, interactions graphs, experimental datasets, logical networks, and similar knowledge in systems biology is often (publicly) available in different kinds of formats. Clearly, converting such a knowledge from their corresponding format to a set of logic facts, e.g., Listing 3.1, Listing 4.1, or Listing 5.1, is a tedious and error-prone task if do it by hand. In fact, this can be easily automated by using any popular scripting language, e.g. python.[1] Analogously, the resulting answer sets from the ASP solver can be converted back to "standard" formats for subsequent analysis with available tools or even visualized in order to facilitate their interpretations. Notably, over the last few years, many simulation and analysis software tools for logical models have been developed, among them, ADAM [Hinkelmann et al., 2010], BoolNet [Mussel et al., 2010], BooleanNet [Albert et al., 2007], Cell Collective [Helikar et al., 2011], CellNetAnalyzer [Klamt et al., 2006a], CellNOpt [Terfve et al., 2012], ChemChains [Helikar and Rogers, 2008], GNA [Batt et al., 2012], GINsim [Naldi et al., 2008], SimBoolNet [Zheng et al., 2010], and SQUAD [Cara et al., 2006]. The underlying methods of these tools are very heterogeneous, including, *ad hoc* algorithms, metaheuristics, and symbolic model checking by means of temporal logic and specialized tools, such as NuSMV [Cimatti et al., 2002]. Moreover, in general, users and developers of such tools, do not have the expertise to deal with ASP

[1] `http://www.python.org/`

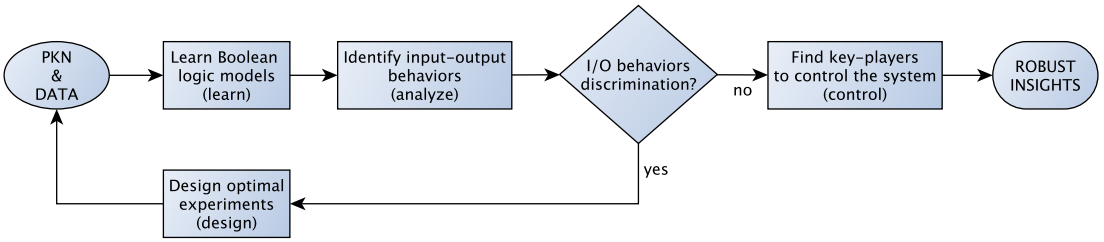


Figure 6.1: Pipeline for reasoning on the response of logical signaling networks using **caspo**. At first, all (nearly) optimal Boolean logic models are learned by confronting prior knowledge on causal interactions with a phosphorylation dataset. Then, supported by insights from experts and various validation methods, we can decide if the ensemble of Boolean models needs further refinements. In such a case, optimal experiments can be designed in order to discriminate the set of behaviors at hand. By combining the previous dataset with the new observations, the learning is performed again yielding a new ensemble of models. After several iterations, the "high-quality" ensemble of logical networks can be used to find (robust) intervention strategies and generate novel hypotheses.

encodings and systems. Thus, a software package providing an interface to the ASP-based solutions detailed in previous chapters would ease the accessibility for end-users, as well as the integration with available tools for simulation and analysis of logical models.

Software toolbox: caspo. In order to encapsulate our ASP-based solutions, we have implemented the python package **caspo** which is freely available for download.[2] More broadly, **caspo** is part of BioASP, a collection of python packages leveraging the computational power of ASP for systems biology.[3] The aim of **caspo** is to implement a pipeline for automated reasoning on logical signaling networks providing a powerful and *easy-to-use* software tool for systems biologists. In Figure 6.1 we show a schematic view of such a pipeline. At first, all (nearly) optimal Boolean logic models and their corresponding logical input-output behaviors, are learned as shown in Chapter 3. Then, supported by insights from experts and various validation methods, we can decide if the ensemble of Boolean models needs further refinements. If this is the case, we look for an optimal experimental design in order to discriminate the set of input-output behaviors as presented in Chapter 4. By combining the initial dataset with the new experimental observations, the learning is performed again yielding a new ensemble of models. After several iterations of this cycle, a "high-quality" ensemble of logical networks can be used to find intervention strategies as shown in Chapter 5. Each section of this pipeline is implemented in **caspo** by means of *subcommands*, namely, *learn*, *analyze*, *design*, and *control*. Notably, this in line with the loop of hypotesis-driven research in biology [Ideker et al., 2001, Kitano, 2002]. Moreover, this iterative process could be automatized allowing an autonomous scientific discovery [Sparkes et al., 2010].

[2] `http://bioasp.github.io/caspo/`
[3] `http://bioasp.github.io/`

Except for CellNOpt and BoolNet, all tools mentioned above are mainly focused on the characterization of dynamical properties emerging from the simulation of a single logical model provided by the user. In the case of CellNOpt, the focus is on the training of logical models by using metaheuristics, whereas BoolNet includes two algorithms, namely, *Best-Fit Extension* [Lähdesmäki et al., 2003] and *REVEAL* [Liang et al., 1998], for the reconstruction of gene regulatory Boolean networks from time-series gene expression profiles. In addition, various simulation analyses are provided by both packages as well. The tool CellNetAnalyzer implements the identification of intervention strategies as described in Chapter 5 but with respect to a single logical network and limited to small strategies (less than 4 interventions). Furthermore, several tools allow the user to introduce artificial perturbations and simulate the system's response. Nevertheless, all tools aim at modeling a given biological system by means of one logical network only. Next, dynamical and structural analysis (often computationally demanding) are conducted over *the* model. In this context, a distinct feature of **caspo** is the fact that it aims at having an exhaustive characterization of feasible models. Either via automated learning, or directly provided by the user. Afterwards, instead of selecting a single model, the goal is to provide automated reasoning over the ensemble of models at hand. First to find optimal experimental designs for models discrimination, and second for finding minimal intervention strategies valid in all networks. Notably, towards this end, **caspo** strongly relies on ASP which is another distinct feature from the methodological point of view.

In the remainder of this chapter we present the software tool **caspo** describing its high-level design and main features. Furthermore, we illustrate how end-users can easily go through the pipeline shown in Figure 6.1 by using each of the provided subcommands.

6.2 Software design

6.2.1 High-level software design

Internal modules. The architecture of **caspo** is shown in Figure 6.2. The design comprises three different layers of internal modules. A first layer providing the interface with end-users of the software. Currently this layer has only the *console* module which implements the entry-points for **caspo** subcommands from the command-line. In the future, this layer could be extended with a module for a graphical user interface. A middle layer comprises various modules, namely, *learn*, *design*, *control*, *analyze*, and *visualize*, each of them implementing functionalities for the corresponding subcommand. The module *learn* implements the learning of Boolean logic models of immediate-early response as described in Chapter 3. The module *design* implements the experimental design to discriminate input-output behaviors as described in Chapter 4. The module *control* implements the identification of intervention strategies as described in Chapter 5. The module *analyze* implements the search of logical input-output behaviors among a family of Boolean logic models as described in Chapter 3. Furthermore, relatively simple statistical analysis is provided in this module as well. Finally, the module *visualize* implements basic visualization features for logical networks and inter-

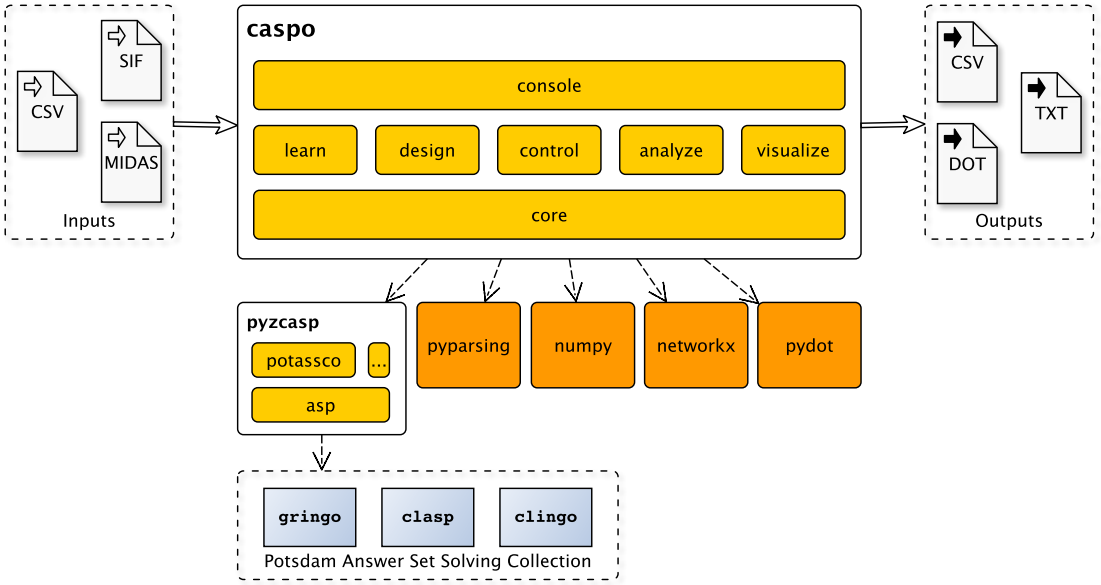


Figure 6.2: High-level software design. Modules of **caspo** are divided into three different layers. First layer comprises the *console* module which implements functionalities related to the usage of **caspo** from the command-line. Entry-points for **caspo** subcommands are implemented here. Second layer comprises various modules which implement different sections of the pipeline for automated reasoning, namely, *learn* for learning Boolean logic models (Chapter 3), *design* for designing new experiments (Chapter 4), *control* for finding intervention strategies (Chapter 5), *analyze* for identifying input-output behaviors (Chapter 3), and *visualize* for basic visualization features. Third layer comprises the *core* module which implements "low-level" functionalities for the whole software. Main external dependecies are shown as well.

vention strategies. Interestingly, all features implemented by the aforementioned modules are independent from the command-line usage of **caspo**. This way, we aim at providing not only a software for end-users, but also a powerful toolbox to be used by other developers to build their own software. A third layer, namely, *core*, implements "low-level" functionalities common to all other modules in **caspo**.

External dependencies. In Figure 6.2 we also shown the main external dependencies of **caspo**. On the one hand, to build **caspo**, we rely on relatively standard python packages, such as, *pyparsing*[4], *numpy*[5], *networkx*[6], and *pydot*[7]. Notably, all of them are open-source projects under different free software licenses. For more details on each package, we refer the reader to the corresponding website. On the other hand, we highlight the external dependency on

[4] `http://pyparsing.wikispaces.com/`
[5] `http://www.numpy.org/`
[6] `http://networkx.github.io/`
[7] `http://code.google.com/p/pydot/`

the python package *pyzcasp*[8]. In fact, *pyzcasp* was developed within this thesis in order to build **caspo** on top of it. However, it was developed in such a way to provide a general purpose framework to build on top of ASP tools. Therefore, it is not only of interest for **caspo** or biological applications, but for any other project aiming at the integration of python and ASP. Broadly speaking, the goal of *pyzcasp* is to provide an abstraction layer over ASP systems. Notably, such an abstraction allows us to have a clear separation of responsibilities, whereas it facilitates the maintenance of both, the application, in our case, **caspo**, and the communication with third-party ASP tools. In practice, problem solving using ASP involves three main steps. At first, certain input data, normally stored in different files with different formats, need to be converted to logic facts describing the corresponding problem instance. Second, the problem instance needs to be combined with the specific set of logic rules encoding the problem to be solved. It is worth noting that, sometimes the same problem can be addressed with different ASP tools which in turn, may accept different input languages and options. Hence, in such cases, we may need to select, among several encodings for the same problem, the appropriate encoding for a specific ASP tool. Next, the corresponding ASP tools are executed with the problem instance and encoding as input. Third, answer sets returned by the ASP solver must be interpreted according to the representation used in the encoding. Essentially, the package *pyzcasp* provides a framework for dealing with these three steps in a robust way within a python environment. Currently, it supports the usage of most common tools from Potassco: the Potsdam Answer Set Solving Collection.[9] Nevertheless, support for other ASP tools is planned in the future.

6.2.2 Generic workflow for subcommands

The workflow for main **caspo** subcommands, namely, *learn*, *analyze*, *design*, and *control*, is shown in Figure 6.3. Essentially, each of these subcommands is a particular use case of the more general workflow for problem solving using ASP as detailed earlier. In all cases, some input data is provided by the user in one or more files. In general, each subcommand deals with different file formats, but all of them need to convert the input data to logic facts describing the corresponding problem instance. Next, the problem instance together with the specific problem encoding are given to the corresponding ASP tools for solving. Furthermore, in some cases, very specific parameters are given to the solver in order to obtain right solutions or improve performance. For instance, in order to enumerate all inclusion-minimal answer sets as we do it when looking for intervention strategies, the ASP solver *clasp* must be executed with parameters `--dom-pref=32 --dom-mod=6 --heu=domain -n0` (see Listing 5.3). Once the ASP solving has finished, the resulting answer sets are loaded into memory as python objects for further analysis. In the simplest case, solutions are written to output files in standard formats. However, in some cases, additional "business logic" is implemented for each subcommand. For instance, the identification of input-output behaviors using Algorithm 1 (Section 3.4.3), or first find the optimum and then, with a second solver call, enumerate all solutions within

[8] `http://svidela.github.io/pyzcasp/`
[9] `http://potassco.sourceforge.net/`

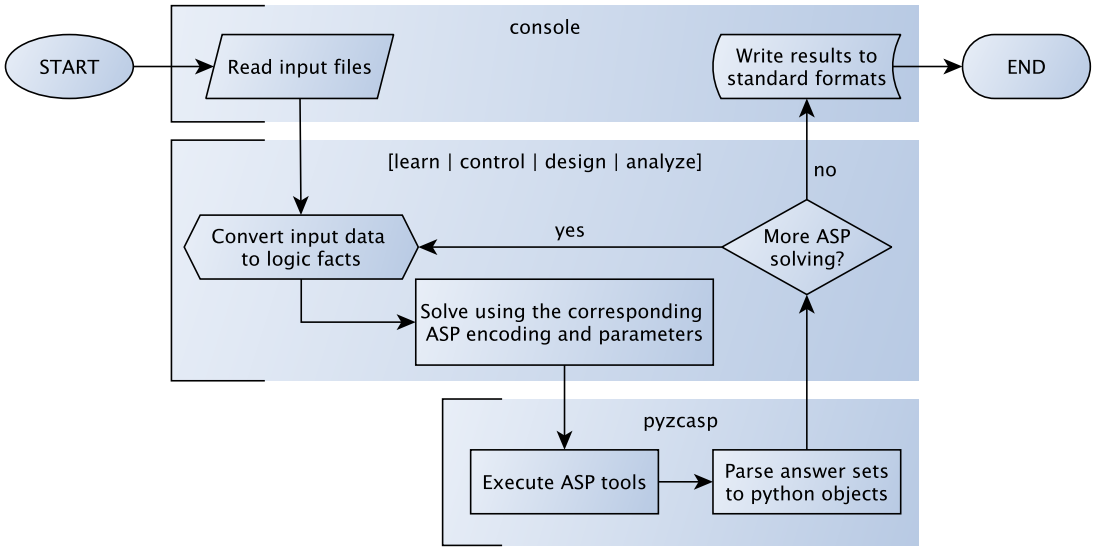


Figure 6.3: Workflow for **caspo** subcommands. For clarity, we distinguish three execution levels. First, the command-line (end-user) level where input and output files are handled. Second, the subcommands (application) level where input data is converted to logic facts, combined with a specific logic program, and given to the next level for launching ASP tools. Normally, some kind of "business logic" is implemented at this level, for instance, the identification of input-output behaviors using Algorithm 1 (Section 3.4.3), or first find the optimum and then enumerate solutions within certain tolerance. Third, the ASP solving (external) level where ASP tools are executed using system calls and the resulting answer sets are loaded into memory as python objects, which in turn are provided to the application level for further analysis.

certain tolerance. Altogether, **caspo** subcommands provide high-level automated reasoning on logical signaling networks by leveraging the computational power and reasoning modes of ASP systems.

6.3 Usage

6.3.1 Usage for end-users

In what follows we illustrate how end-users can easily go through the pipeline shown in Figure 6.1 by using each of the subcommands provided by **caspo**. To start with, in Listing 6.1 we show how to ask for help on the usage of **caspo** from the command-line.

Listing 6.1: Help message for **caspo**.

```
$ caspo --help
usage: caspo [-h] [--quiet] [--out O] [--version]
             {control,visualize,design,learn,test,analyze} ...
```

```
Reasoning on the response of logical signaling networks with ASP

optional arguments:
  -h, --help            show this help message and exit
  --quiet               do not print anything to standard output
  --out O               output directory path (Default to './out')
  --version             show program's version number and exit

caspo subcommands:
  for specific help on each subcommand use: caspo {cmd} --help

  {control,visualize,design,learn,test,analyze}
```

Learning (nearly) optimal Boolean logic models. The learning of Boolean logic models as described in Chapter 3 is provided by the subcommand *learn*. In Listing 6.2 we show the corresponding help message for this subcommand. In this case, there are three required arguments, namely, the prior knowledge network (PKN), the phospho-proteomics dataset, and the characteristic time-point for the immediate-early response. The PKN must given in the so-called *simple interaction format* (SIF).[10] Typically, lines in a SIF file specify a source node, an interaction type, and one or more target nodes. In our context, we restrict ourselves to SIF files where the interaction type is either 1 or -1, and there is only one target node. Hence, this way we can easily represent the interaction graph describing the PKN at hand. Note that the SIF file does not provide information about which nodes are stimuli, inhibitors or readouts. Currently, such information must be extracted from the dataset file which we describe next. The phospho-proteomics dataset must given in the so-called *minimum information for data analysis in systems biology* (MIDAS) format.[11] For more details on this file format we refer the reader to [Saez-Rodriguez et al., 2008]. In essence, a MIDAS file is a *comma-separated values* (CSV) file with a simple naming convention in order to identify columns as: (1) experimental perturbations, (2) time points for data acquisition, or (3) phosphorylation activities describing the response of the system. Then, each row in the file describes an experimental condition together with the phosphorylation activities at a certain time-point. Finally, the third required argument is the characteristic time-point for the immediate-early response. Clearly, the given time point must be present in the MIDAS file or an exception is raised otherwise. In addition to the three required arguments, several optional arguments can be provided as shown in Listing 6.2.

Listing 6.2: Help message for *learn* subcommand

```
$ caspo learn --help
usage: caspo learn [-h] [--clingo C] [--fit F] [--size S] [--factor D]
                   [--discretization T]
                   pkn midas time
```

[10] http://wiki.cytoscape.org/Cytoscape_User_Manual/Network_Formats
[11] http://www.cellnopt.org/doc/cnodocs/midas.html

```
positional arguments:
  pkn                   prior knowledge network in SIF format
  midas                 experimental dataset in MIDAS file
  time                  time-point to be used in MIDAS

optional arguments:
  -h, --help            show this help message and exit
  --clingo C            clingo solver binary (Default to 'clingo')
  --fit F               tolerance over fitness (Default to 0)
  --size S              tolerance over size (Default to 0)
  --factor D            discretization over [0,D] (Default to 100)
  --discretization T    discretization function: round, floor, ceil
                        (Default to round)
```

Next, in Listing 6.3 we show the execution of the *learn* subcommand for our toy example. In this case, we use `--fit 0.1 --size 5` in order to enumerate all (nearly) optimal Boolean logic models with a tolerance of 10% over the fitness and 5 over the size. The only output in this case is a CSV file, namely, `networks.csv` describing all logical networks found during enumeration with ASP. The first row or header of this file contains all possible (signed) directed hyperedges for the given PKN. It is worth recalling that, we interpret a (signed) directed hyperedge as the conjunction of its source (signed) nodes acting over the target node. Furthermore, we interpret several hyperedges with the same target node as the disjunction of the corresponding conjunctions of their sources nodes. For example, if the PKN contains edges $a \rightarrow c$ and $b \dashv c$, then the header will have columns $a = c$,$!b = c$, and $a+!b = c$. Next, each row in the file describes a different logical network by specifying which hyperedges are present, and which ones are absent. Hence, let (V,ϕ_1) and (V,ϕ_2) be two logical networks with $\phi_1(c) = a \vee \neg b$ and $\phi_2(c) = a \wedge \neg b$. Such logical networks are described in `networks.csv` as:

$$\begin{array}{cccc} a=c, & !b=c, & a+!b=c, & \dots \\ 1, & 1, & 0, & \dots \\ 0, & 0, & 1, & \dots \\ & \vdots & & \dots \end{array}$$

Such a matrix representation of logical networks is relatively simple and very easy to share with third-party software. In fact, some of the following subcommands use this kind of file format as an input.

Listing 6.3: Running *learn* subcommand

```
$ caspo learn pkn.sif dataset.csv 10 --fit 0.1 --size 5
Running caspo learn...

Wrote out/networks.csv
```

Identifying logical input-output behaviors. The identification of logical input-output behaviors as described in Chapter 3 is provided by the subcommand *analyze*. Actually, as we show below, this subcommand implements various additional features apart from the identification of input-output behaviors. In this case, there are no required arguments but instead, **caspo** performs all available computations for the given inputs. In particular, in order to identify logical input-output behaviors among a family Boolean logic models we need to provide two inputs. Of course, the family of logical networks is required. But also, we need to provide again the dataset and time point used during learning. There are two reasons for this. First, as with the SIF file for the PKN, the information about inputs, i.e., stimuli or inhibitors, and outputs, i.e., readouts, is not given in the networks file. Thus, we use the MIDAS file to extract such information. Second, some of the additional computations use the experimental observations in the dataset, e.g., computing MSEs for every logical network. In Listing 6.4 we show the corresponding help message for this subcommand.

Listing 6.4: Help message for *analyze* subcommand

```
$ caspo analyze --help
usage: caspo analyze [-h] [--clingo C] [--networks N] [--midas M T]
                     [--strategies S]

optional arguments:
  -h, --help       show this help message and exit
  --clingo C       clingo solver binary (Default to 'clingo')
  --networks N     logical networks in CSV format
  --midas M T      experimental dataset in MIDAS file and time-point
  --strategies S   intervention stratgies in CSV format
```

Next, in Listing 6.5 we show the execution of the *analyze* subcommand for our toy example. Let us briefly describe the output files from this subcommand.

- `networks-stats.csv`: frequency of each logical interaction and mutually exclusive/inclusive pairs.
- `networks-mse-len.csv`: two additional columns to the input file `networks.csv` with the corresponding MSE and size for each network.
- `behaviors.csv`: exactly one representative logical network for each behavior found.
- `behaviors-mse-len.csv`: two additional columns to the file `behaviors.csv` with the corresponding MSE and number of Boolean logic models for each input-output behavior.
- `variances.csv`: the variance for each output prediction over all behaviors.
- `core.csv`: experimental conditions for which all behaviors agree on the given response.
- `summary.txt`: a summary as printed in the terminal.

Based on the information generated by the *analyze* subcommand, one could decide whether additional experiments are required or not. Note that, eventually, one may find an unique logical input-output behavior. In such a case, there is nothing else to discriminate. However, this is not necessarily the only criterion to break the loop of learning and experimentation. In fact, finding an unique input-output behavior seems very unlikely from our experiments. Hence, when several input-output behaviors exists, we can assess them, for example, by means of their core predictions and variances.

Listing 6.5: Running *analyze* subcommand

```
$ caspo analyze --networks out/networks.csv --midas dataset.csv 10
Running caspo analyze...

Wrote out/networks-stats.csv
Wrote out/networks-mse-len.csv

Searching input-output behaviors... \
            3 behaviors have been found over 5 logical networks.

Wrote out/behaviors.csv
Wrote out/behaviors-mse-len.csv
Wrote out/variances.csv
Wrote out/core.csv
Wrote out/summary.txt

caspo analytics summary
=======================
Total Boolean logic networks: 5
Total I/O Boolean logic behaviors: 3
Weighted MSE: 0.1100
Core predictions: 87.50%
```

Finding an optimal experimental design. The optimal experimental design as described in Chapter 4 is provided by the subcommand *design*. In Listing 6.6 we show the corresponding help message for this subcommand. In this case, there are two required arguments, namely, the representative logical networks for each input-output behavior and the phospho-proteomics dataset. The representative logical networks must be given in a CSV file as already described. In particular, after the execution of the *analyze* subcommand, the output file `behaviors.csv` can be used directly as an input here. Again, as explained before, this subcommand requires the dataset in a MIDAS file in order to extract the information about inputs, i.e., stimuli or inhibitors, and outputs, i.e., readouts.

Listing 6.6: Help message for *design* subcommand

```
$ caspo design --help
usage: caspo design [-h] [--clingo C] [--stimuli S] [--inhibitors I]
                    [--experiments E]
                    networks midas
```

```
positional arguments:
  networks          logical networks in CSV format
  midas             experimental dataset in MIDAS file

optional arguments:
  -h, --help        show this help message and exit
  --clingo C        clingo solver binary (Default to 'clingo')
  --stimuli S       maximum number of stimuli per experiment
  --inhibitors I    maximum number of inhibitors per experiment
  --experiments E   maximum number of experiments (Default to 20)
```

Next, in Listing 6.7 we show the execution of the *design* subcommand for our toy example. In this case, we use `--stimuli 3 --inhibitors 2` in order to restrict the experimental design to experiments having at most, 3 stimuli and 2 inhibitors. The only output in this case is a CSV file, namely, `opt-design-0.csv` describing all experimental conditions in the optimal experimenal design found with ASP. Columns represent either stimuli or inhibitors, and rows or lines specify whether the corresponding species is clamped (1) or not (0) in that experimental condition. Once all suggested experiments are carried out in the laboratory, new experimental observations are added to the existing dataset. Next, the subcommands *learn* and *analyze* introduced earlier, can be used again for learning a refined family of Boolean logic models and its corresponding logical input-output behaviors. Notably, this loop can continue until we find an unique input-output behavior, or we consider that the current ensemble of logic models is accurate enough.

Listing 6.7: Running *design* subcommand

```
$ caspo design out/behaviors.csv dataset.csv --stimuli 3 --inhibitors 2
Running caspo design...

Wrote out/opt-design-0.csv
```

Enumerating minimal intervention strategies. The enumeration of all minimal intervention strategies as described in Chapter 5 is provided by the subcommand *control*. In Listing 6.8 we show the corresponding help message for this subcommand. In this case there are two required arguments, namely, the family logical networks and the intervention scenarios. The family of logical networks must be given in a CSV file as already described, e.g. the output file `networks.csv` from the *learn* subcommand. The intervention scenarios must also be given in a CSV file. As with the MIDAS file, we adopt a simple naming convention in order to identify columns as side constraints or goals. Then, each row specifies whether the corresponding species is clamped positively (1), negatively (-1), or not clamped (0) in that intervention scenario.

Listing 6.8: Help message for *control* subcommand

```
$ caspo control --help
```

```
usage: caspo control [-h] [--size M] [--allow-constraints]
                     [--allow-goals] [--gringo G] [--clasp C]
                     networks scenarios

positional arguments:
  networks              logical networks in CSV format
  scenarios             intervention scenarios in CSV format

optional arguments:
  -h, --help            show this help message and exit
  --size M              maximum size for interventions strategies
                        (Default to 0 (no limit))
  --allow-constraints   allow intervention over side constraints
                        (Default to False)
  --allow-goals         allow intervention over goals (Default to False)
  --gringo G            gringo grounder binary (Default to 'gringo')
  --clasp C             clasp solver binary (Default to 'clasp')
```

Next, in Listing 6.9 we show the execution of the *control* subcommand for our toy example. The only output in this case is a CSV file, namely, `strategies.csv` describing all inclusion-minimal intervention strategies found with ASP. Recall that each intervention strategy guarantees that all scenarios are satisfied in every logical network. The format of this output file is very simple. Columns represent all variables in the logical networks and rows specify whether the corresponding species is clamped positively (1), negatively (-1), or not clamped (0) in that intervention strategy.

Listing 6.9: Running *control* subcommand

```
$ caspo control out/networks.csv scenarios.csv
Running caspo control...

Wrote out/strategies.csv
```

In fact, now we can use the *analyze* subcommand again with the intervention strategies as input. Currently, only basic statistical analysis is implemented for intervention strategies. Nonetheless, further analysis may be implemented in the future. The output file `strategies-stats.csv` contains the frequency of each single intervention (either positive or negative), and mutually exclusive/inclusive pairs. Notice that, since we do not use neither `--networks` nor `--midas` arguments, the result is different from the previous call to this subcommand shown in Listing 6.5 Notably, if we use arguments `--networks`, `--midas`, and `--strategies` in the same call, all analyses are performed with such a call.

Listing 6.10: Running *analyze* subcommand (again)

```
$ caspo analyze --strategies out/strategies.csv
Running caspo analyze...

Wrote out/strategies-stats.csv
Wrote out/summary.txt
```

```
caspo analytics summary
=======================
Total intervention strategies: 3
```

Visualization support. Basic visualization features are provided by the subcommand *visualize*. In Listing 6.11 we show the corresponding help message for this subcommand. Similarly to the *analyze* subcommand, in this case there are no required arguments. The goal of this subcommand is to provide relatively simple visualization for the outcome from other subcommands. More precisely, it is possible to visualize a prior knowledge network in a SIF file with the corresponding experimental setup given in a MIDAS file. Also, visualization of logical networks as (signed) directed hypergraphs is provided for either a random sample or all networks in given CSV file. Moreover, the union (overlapping) of all logical networks is also provided. Finally, intervention strategies given in a CSV file can be visualized as well. In all cases, we choose to use DOT files as output format. Such a format is a quite widespread way of describing graph structures. Importantly, various programs can process a DOT file and render it in graphical form using several file formats, e.g., PDF, PNG, SVG and PS among others. Notably, most of such programs are included in the Graphviz package which is freely available.[12]

Listing 6.11: Help message for *visualize* subcommand

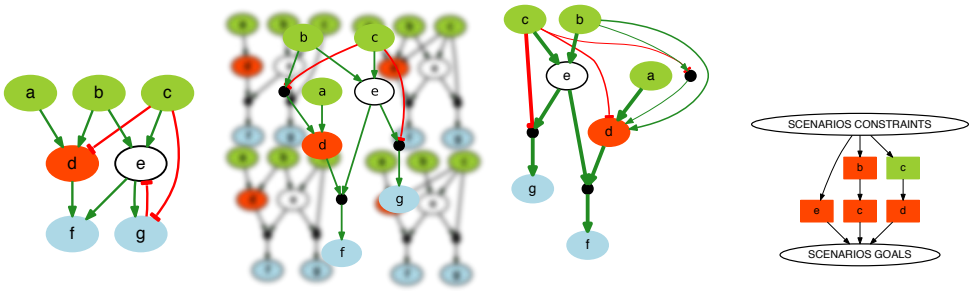
```
$ caspo visualize --help
usage: caspo visualize [-h] [--pkn P] [--midas M] [--networks N]
                       [--sample R] [--union] [--strategies S]

optional arguments:
  -h, --help      show this help message and exit
  --pkn P         prior knowledge network in SIF format
  --midas M       experimental dataset in MIDAS file
  --networks N    logical networks in CSV format
  --sample R      visualize a sample of R logical networks
                  (Default to all)
  --union         visualize the union of logical networks
                  (Default to False)
  --strategies S  intervention stratgies in CSV format
```

Next, in Listing 6.12 we show the execution of the *visualyze* subcommand for our toy example. Let us briefly describe the output files from this subcommand which we also shown in Figure 6.4.

- `pkn.dot`: PKN with the corresponding experimental setup (Figure 6.4a).
- `network-`i`.dot`: every logical network (V, ϕ_i) (Figure 6.4d).

[12] `http://graphviz.org/`



(a) (b) (c) (d)

Figure 6.4: Visualization provided by **caspo**. (a) Toy prior knowledge network (`pkn.dot`). (b) Logical networks (V, ϕ_i) as (signed) directed hypergraphs (`network-`i`.dot`). (c) Union of all logical networks where the thickness of hyperedges correspond to their frequencies (`networks-union.dot`). (d) All inclusion-minimal intervention strategies (`strategies.dot`).

- `networks-union.dot`: union (overlapping) of all logical networks (Figure 6.4b).
- `strategies.dot`: all intervention strategies (Figure 6.4c).

Listing 6.12: Running *visualize* subcommand

```
$ caspo visualize --pkn pkn.sif --midas dataset.csv \
                  --networks out/networks.csv --union \
                  --strategies out/strategies.csv

Wrote out/pkn.dot
Wrote out/network-1.dot to out/network-5.dot
Wrote out/networks-union.dot
Wrote out/strategies.dot
```

6.3.2 Usage for developers

In the previous section we have detailed how end-users can use **caspo** without any programming expertise but only by using the available subcommands. Notably, such a level of usability is fundamental in order to allow (systems) biologists to profit from our software in their daily research. Nevertheless, it may be the case that one needs to integrate different software tools in a larger framework. Towards this end, we have developed **caspo** in such a way that other developers can access to all its features programmatically. For the sake of brevity, we refrain from giving a detailed description of all classes and methods available and refer the interested reader to the publicly available repository where **caspo** is maintained.[13] It is worth noting that, this allows other software tools to take advantage of the computational power of ASP at a higher level of abstraction and looking at the solving process as a *black-box* procedure managed by **caspo**. Moreover, the component-based design of **caspo** aims at providing a

[13] `http://github.com/bioasp/caspo`

seamless interface with third-party software. More precisely, it is not only possible to use **caspo** programmatically, but also, developers can plug their own components and overwrite default implementations. Altogether, apart from being an *easy-to-use* software tool for end-users, **caspo** can be used by developers as a toolbox by leveraging the computational power of ASP for building their own software.

6.4 Conclusion

In this chapter we have presented a software toolbox providing an interface to the ASP-based solutions detailed in previous chapters. Altogether, our software implements a complete pipeline for automated reasoning on logical signaling networks providing a powerful and *easy-to-use* software tool for systems biologists. Each section of this pipeline is implemented by means of *subcommands*, namely, *learn*, *analyze*, *design*, *control*, and *visualize*. In fact, using a toy example, we have illustrated how end-users can easily go through the pipeline using such subcommands. Moreover, the component-based design of **caspo** aims at providing a seamless interface with third-party software. Therefore, apart from being an *easy-to-use* software for end-users, **caspo** can be used by developers as a toolbox by leveraging the computational power of ASP for their own software.

In this context, the development of standard formats would ease the integration of different tools by taking advantage of complementary features from each software. Typically, such tools are developed independently by different research groups around the world. Hence, it is rather hard to converge towards standards and common practices. In fact, an extension to the popular *systems biology markup language* (SMBL) [Hucka et al., 2003] has been recently proposed as a more appropriate and specific standard for describing logic-based models, the so-called *SBML Qualitative Models* (SBML qual) [Chaouiya et al., 2013]. Therefore, in the future we plan to support both import and export features via SBML qual in **caspo**. Also, given the active research and development in the ASP community, we expect to extend and enrich the implemented pipeline for automated reasoning as soon as new developments come to light. It is worth noting that, compared to other available software tools, **caspo** is still in its infancy. Thus, based on the feedback from users, developers and research community in general, we hope to continue improving the presented software in order to make it an atractive and useful alternative for systems biologists in the near future.

7 Conclusion and prospective

In previous chapters we have presented a computational modeling approach for the construction, refinement, and control of logical signaling networks. In what follows we summarize the main contribution of this thesis and discuss about its novelty and limitations. In particular, we discuss the strengths and weaknesses of our modeling and computational methods. We conclude with prospective lines of research and future directions to explore questions raised by our work.

7.1 Contribution

On the modeling side, we have presented a mathematical framework for reasoning on the response of logical signaling networks relying on propositional logic and fixpoint semantics. Despite its relative simplicity, such a framework has proven to be generic and flexible enough to describe several interesting problems in the field. In particular, we have characterized three challenging combinatorial optimization problems related to logical signaling networks. In Chapter 3 we have revisited the problem consisting of learning from an interaction graph and phosphorylation activities at a pseudo-steady state, (Boolean) logical networks describing the immediate-early response of the system. Next, in Chapter 4 we have proposed a novel problem specification for finding optimal experimental designs to discriminate between rival logical models. Finally, in Chapter 5 we have revisited the problem consisting of finding intervention strategies in logical signaling networks. Moreover, in order to increase robustness, we extended this problem for reasoning over several alternative logical networks. Notably, these three problems (or slight variations thereof) have been previously described by other authors using different formalisms. In contrast, the proposed framework provides an unified computational modeling approach for reasoning on logical signaling networks.

On the computational side, we have addressed each of the aforementioned problems by means of Answer Set Programming (ASP). The distinctive features of ASP have allowed us to model and solve each problem in a uniform and efficient way. More precisely, in Chapter 2 we have provided a basic ASP representation to describe logical networks and their response in

terms of fixpoint semantics. Next, we have benefited from the elaboration tolerance of ASP in order to adapt such a representation, with relatively little effort, to each specific problem in subsequent chapters. Interestingly, the computational performance is significantly improved with respect to state-of-the-art algorithms to solve the same problems. But more importantly, the exhaustive nature of ASP have allowed us to find feasible solutions that were missing when using existing methods. Notably, this poses new challenges related to the analysis and interpretation of possibly many solutions. For instance, in Chapter 3, in the light of a large number of feasible logical networks, we have introduced the concept of logical input-output behaviors which allows us to focus on predictions rather than alternative mechanisms. In fact, the proposal of experimental design presented in Chapter 4 was motivated by the existence of several input-output behaviors and the necessity to discriminate among them. Yet, there may be several models which cannot be discriminated using the available experimental settings. Thus, instead of selecting a single model, in Chapter 5 we aimed at reasoning over all of them in order to find more robust interventions strategies. Altogether, this thesis illustrates the potential of ASP to address hard combinatorial search and optimization problems related to qualitative modeling of biological systems. In fact, our work contributes to a growing list of ASP applications in systems biology which show its increasing impact on the field.

On the implementation side, in Chapter 6 we have presented a software toolbox providing an interface to the ASP-based solutions for each of the discussed problems. Essentially, the goal of such a software is to encapsulate the workflow for problem solving with ASP in our specific context. To this end, we have used a popular and high-level scripting language such as python. Notably, this will ease the accessibility for end-users, as well as the integration with available tools for simulation and analysis of logical models. Altogether, our software implements a complete pipeline for automated reasoning on logical signaling networks providing a powerful and easy-to-use software tool for systems biologists.

7.2 Discussion and prospective

Introduced by Stuart Kauffman 45 years ago, Boolean networks are a particular case of dynamical systems used for modeling biology. Notably, due to their binary states and synchronous updates, Boolean networks are among the simplest and more abstract dynamical systems. On the one hand, such a crude simplification of biological reality is a limitation in terms of the kinds of systems we can describe. On the other hand, is precisely because of their simplicity that Boolean networks are attractive for systems biologists. Typically, in the biological literature signaling networks are described by biologists as “pathways cartoons”. In fact, such cartoons are commonly found in pathway databases as well. In this context, while graph-based models are the natural approach for static analysis, logic-based models, such as Boolean networks provide a framework still intuitive for biologists but tailored to relatively complex dynamic analysis. Of course, as soon as biologists gain a deeper understanding of a particular system, such simplistic models may become obsolete and more elaborate formalisms should come into play. However, nowadays, the lack of quantitative information at systems level suggests

that, for some time to come, high-level computational abstractions will be crucial for pushing the boundaries of biological knowledge. Moreover, despite their limitations, such simple and intuitive modeling frameworks can help to reduce the existing gap between theoretical and experimental research in systems biology.

We have adopted ASP as our computational method. Notably, like any other method, ASP has its own strengths and weaknesses. On the one hand, thanks to its origins in the area of knowledge representation and reasoning, ASP provides a simple yet rich and fully declarative modeling language. Furthermore, in the last decade, the development of highly efficient solvers have made of ASP an attractive framework for problem solving, yielding comparable results to integer linear programming or Boolean constraint solving. In this context, the added value of ASP is two-fold (at least). First, it provides a high-level executable modeling language. In practice, this allows for very compact problem representations that can be tested and refined *on-the-fly*. Second, modern solvers implement various reasoning modes, such as enumeration, union and intersection, multi-objective optimization, and combination thereof. Notably, this allows us to address a multitude of problems with minimal effort or *ad hoc* programming overhead. On the other hand, two aspects (at least) of ASP may limit its usage in systems biology. First, its strictly discrete nature forbids us to take into account numerical or quantitative information. Sometimes and to some extent, available data can be discretized. Nonetheless, in general, ASP is rather tailored to combinatorial problems with constraints over variables with either Boolean, or small domains. Second, the two phases in the process for problem solving with ASP, namely, grounding and solving, sometimes yield a bottleneck which requires advanced modeling skills or even makes impossible to solve large-scale instances. Altogether, taking into account strengths and weaknesses, ASP is a relatively young technology which has proven to be very useful and efficient to solve numerous challenging problems in several application domains. Furthermore, the fact that it is an emergent and very active area of research suggests that, in the years to come, its usage will become more popular in the field of systems biology.

The work presented in this thesis has raised several interesting questions for future research. In particular, our work have opened the way to an exhaustive characterization of feasible logical models for a given system. Now, we need to develop a proper modeling framework to interpret and take advantage of such a characterization. This must be necessarily driven by available experimental technology allowing us to either confirm or refute generated hypotheses. For instance, phospho-proteomics assays like the one used for learning in Chapter 3 are performed over a population of cells and thus, it is unclear how to interpret the multitude of logical networks and their input-output behaviors. Given the existence of several logical input-output behaviors gathering different internal mechanisms, we need to elucidate if such mechanisms are a mere artifact of logical networks, or if they actually represent molecular mechanisms which in turn appear more or less often within each cell or at a population scale. Furthermore, despite the intrinsic uncertainty in biological systems, it is still rather hard to assess the amount of noise in measurements and how this impacts on our mathematical or computational models. This raises the question of identifiability and to what extent new experiments are able

to yield more refined logical models. Thus, the method for experimental design presented in Chapter 4 requires to be validated and could help to tackle this issue. Finally, we find particularly interesting to explore further the approach introduced in Chapter 5 for finding intervention strategies by reasoning over an ensemble of logical networks. Again, this is a key methodological contribution allowing systems biologists to draw insights under uncertainty and non-identifiability. Nonetheless, we need to elucidate whether such an ensemble of networks describes alternative pathways within a single cell or at the population scale. Overall, we look forward for experimental validation of our methods and findings.

We conclude this thesis by noting that, integrative modeling approaches considering multiple levels and time-scales of causation pose a very challenging goal in systems biology. Towards this end, we envision a hybrid modeling framework combining (non-deterministic) logic-based, stochastic, and continuous approaches. Notably, in order to achieve all these challenges, we believe that more sophisticated computational methods are needed in order to integrate qualitative and quantitative knowledge in a uniform and robust manner. In particular, hybrid reasoning systems leveraging the expressiveness of several technologies and modeling approaches appears as a very promising track for future research in computer science. In fact, very recent developments in the ASP systems used throughout this thesis allow for an interplay between declarative logic programming and imperative "traditional" programming. This opens innumerable possibilities to combine available technologies into a unified environment for problem solving. Hopefully, advances on this subject will foster the usage of knowledge representation and reasoning methodologies in systems biology towards a better understanding of life.

A Proofs

A.1 Data discretization schemes

Let (V,E,σ) be a PKN. Let $\xi = ((C_i, P_{C_i}))_{i\in N}$ be an experimental dataset over (V,E,σ) with size N_ξ. Let $k \in \mathbb{N}$ define the discretization scheme. Let us denote with μ and μ_k, the corresponding minima for Θ_{rss} and Θ_{rss_k} over the space of models $\mathbb{M}_{(V,E,\sigma)}$ with respect to ξ:

$$\mu = \min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} \Theta_{rss}((V,\phi),\xi) \qquad \mu_k = \min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} \Theta_{rss_k}((V,\phi),\xi).$$

Then $10^{-2k}\mu_k$ converges to μ when k increases, with an exponential speed:

$$\mu_k = 10^{2k}\mu + O(10^k).$$

Moreover, any Boolean logic model minimizing Θ_{rss}, also minimizes Θ_{rss_k} within the following tolerance t_k:

$$t_k = 2\sqrt{\frac{N_\xi}{\mu_k}} + \frac{N_\xi}{\mu_k}.$$

Proof. Let (V,ϕ) be any Boolean logic model having evidence in (V,E,σ). Let $\pi_1,\ldots,\pi_n$ be the n Boolean predictions of (V,ϕ) with each π_i defined under C_i. The difference Θ_{rss} and $10^{-2k}\Theta_{rss_k}$ over (V,ϕ) with respect to ξ is given by:

$$\left|[\Theta_{rss} - 10^{-2k}\Theta_{rss_k}]((V,\phi),\xi)\right| = \left|\sum_{i=1}^{n}\sum_{v\in dom(P_{C_i})} \left[P_{C_i}(v) - \pi_i(v)\right]^2 - \left[\delta_k(P_{C_i}(v)) - \pi_i(v)\right]^2\right|$$

For each triplet (P_{C_i},π_i,v) let $\gamma_i(v) = \delta_k(P_{C_i}(v)) - \pi_i(v)$ and $\lambda_i(v) = P_{C_i}(v) - \delta_k(P_{C_i}(v))$. Hence:

$$\begin{aligned}\left|[\Theta_{rss} - 10^{-2k}\Theta_{rss_k}]((V,\phi),\xi)\right| &= \left|\sum_{i=1}^{n}\sum_{v\in dom(P_{C_i})} \left[\gamma_i(v) + \lambda_i(v)\right]^2 - \gamma_i(v)^2\right| \\ &= \left|\sum_{i=1}^{n}\sum_{v\in dom(P_{C_i})} \lambda_i(v)\,\left[2\gamma_i(v) + \lambda_i(v)\right]\right|\end{aligned}$$

Recall that $N_\xi = \sum_{i=1}^{n} \left| dom(P_{C_i}) \right|$. From the Cauchy-Schwarz inequality we have that

$$\sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} \left| \gamma_i(v) \right| \leq \sqrt{N_\xi} \sqrt{\sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} \gamma_i(v)^2} = \sqrt{N_\xi} \sqrt{10^{-2k} \Theta_{rss_k}((V,\phi),\xi)}.$$

Notice that from the discretization scheme we have that $\lambda_i(v) < 10^{-k}$ for every (i,v). It follows that:

$$\begin{aligned} \left| [\Theta_{rss} - 10^{-2k}\Theta_{rss_k}]((V,\phi),\xi) \right| &\leq 10^{-k} \left(\sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} 2 \left| \gamma_i(v) \right| + \sum_{i=1}^{n} \sum_{v \in dom(P_{C_i})} \left| \lambda_i(v) \right| \right) \\ &\leq 10^{-k} \left(2\sqrt{N_\xi} \sqrt{10^{-2k}\Theta_{rss_k}((V,\phi),\xi)} + 10^{-k} N_\xi \right). \end{aligned}$$

We deduce that

$$\begin{aligned} \mu &\leq 10^{-2k}\mu_k + 10^{-k} \left(2\sqrt{N_\xi} \sqrt{10^{-2k}\mu_k} + 10^{-k} N_\xi \right) \qquad \text{(A.1)} \\ &\leq 10^{-2k}\mu_k \left(1 + 2 \times 10^{-k} \sqrt{\frac{N_\xi}{10^{-2k}\mu_k}} + 10^{-2k} \frac{N_\xi}{10^{-2k}\mu_k} \right) = 10^{-2k}\mu_k \left(1 + \underbrace{2\sqrt{\frac{N_\xi}{\mu_k}} + \frac{N_\xi}{\mu_k}}_{=t_k} \right) \end{aligned}$$

With a similar reasoning introducing $\gamma'_i(v) = P_{C_i}(v) - \pi_i(v)$ instead of $\gamma_i(v)$, we have the following relation.

$$\begin{aligned} \left| [\Theta_{rss} - 10^{-2k}\Theta_{rss_k}]((V,\phi),\xi) \right| &= \left| \sum_{i=1}^{n} \sum_{v \in dom(\omega_i)} \lambda_i(v) \left[2\gamma'_i(v) - \lambda_i(v) \right] \right| \\ &\leq 10^{-k} \left(2\sqrt{N_\xi} \sqrt{\Theta_{rss}((V,\phi),\xi)} + 10^{-k} N_\xi \right). \end{aligned}$$

Therefore,

$$\begin{aligned} \mu_k &\leq 10^{2k}\mu + 10^{k} \left(2\sqrt{N_\xi}\sqrt{\mu} + 10^{-k} N_\xi \right) \\ &\leq 10^{2k}\mu + 10^{k} \underbrace{\left(2\sqrt{N_\xi}\sqrt{\mu} + N_\xi \right)}_{=B} \end{aligned}$$

Introducing this inequality in (A.1), we deduce that:

$$\begin{aligned} 10^{2k}\mu &\leq \mu_k + 10^k\left(2\sqrt{N_\xi}\sqrt{10^{-2k}\mu_k} + 10^{-k}N_\xi\right) \\ &\leq \mu_k + 10^k\left(2\sqrt{N_\xi}\sqrt{\mu + 10^{-k}B} + N_\xi\right) \\ &\leq \mu_k + 10^k\left(\underbrace{2\sqrt{N_\xi}\sqrt{\mu + B} + N_\xi}_{=C}\right) \end{aligned}$$

Altogether, we have that there exists $D = \max\{B, C\}$, which is independent from k, such that

$$\left|10^{2k}\mu - \mu_k\right| \leq D10^k.$$

□

A.2 Correctness of ASP encodings

First, let us introduce standard mathematical notation for dealing with logic programs. A propositional logic program over a set $\mathscr{A}$ of ground atoms is a finite set of rules of the form

$$h \leftarrow b_1, \ldots, b_n.$$

where h (head) is an atom and any b_j (body) is a literal of the form a or $\sim a$ for an atom a where the connective $\sim$ corresponds to default negation. Next, for a rule r, we denote its head h with $head(r)$ and its body $\{b_1, \ldots, b_n\}$ with $body(r)$. Furthermore, we denote the positive literals b_j (of the form a) with $body(r)^+$ and the negative literals b_j (of the form $\sim a$) with $body(r)^-$. A set $X \subseteq \mathscr{A}$ of ground atoms is a *model* of a propositional logic program Π, if $head(r) \in X$ whenever $body(r)^+ \subseteq X$ and $body(r)^- \cap X = \emptyset$ for every $r \in \Pi$.

Next, let us recall the formal definition of answer sets. In ASP, the semantics is given by the *stable models semantics* [Gelfond and Lifschitz, 1988]. In fact, we view rules with first-order variables as schemes representing their sets of ground instances. To this end, the reduct, Π^X, of logic program Π relative to a set X of atoms is defined by

$$\Pi^X = \{(head(r) \leftarrow body(r)^+)\theta \mid r \in \Pi, (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \rightarrow \mathscr{A}\}$$

where $var(r)$ is the set of all variables that occur in a rule r and θ is a ground substitution for the variables in r. Then, X is an answer set of Π if and only if X is a $\subseteq$-minimal model of Π^X. Note that Π_X is a positive logic program, i.e., for all rules $r \in \Pi^X$ it holds $body(r)^- = \emptyset$. In fact, in what follows we rely on an operational characterization for computing the stable model of

positive programs. For a positive program P and a set of atoms X, we define

$$T_P X = \{head(r) \mid r \in P \text{ and } body(r) \subseteq X\} .$$

Iterated applications of T_P are written as T_P^j for $j \geq 0$, where

$$T_P^0 X = X \quad \text{and} \quad T_P^{i+1} = T_P T_P^i X \text{ for } i \geq 0 .$$

Then, $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$ is the smallest fixpoint of T_P and defines the stable model of P.

A.2.1 Basic ASP representation

For a logical network (V, ϕ) and clamping assignment C over V, let us denote with $\tau((V,\phi), C)$ the set of logical facts as in Listing 2.1 and Listing 2.2. More precisely,

$$\begin{aligned}
\tau((V,\phi), C) = &\{variable(v). \mid v \in V\} \\
&\cup \{formula(v, s_{\phi(v)}). \mid v \in dom(\phi)\} \\
&\cup \{dnf(s_{\phi(v)}, s_\psi). \mid v \in dom(\phi), \psi \in \phi(v)\} \\
&\cup \{clause(s_\psi, w, 1). \mid v \in dom(\phi), \psi \in \phi(v), w \in \psi \cap V\} \\
&\cup \{clause(s_\psi, w, -1). \mid v \in dom(\phi), \psi \in \phi(v), \neg w \in \psi\} \\
&\cup \{clamped(v, 1). \mid v \in dom(C), C(v) = \boldsymbol{t}\} \\
&\cup \{clamped(v, -1). \mid v \in dom(C), C(v) = \boldsymbol{f}\}
\end{aligned} \tag{A.2}$$

where each $s_{(\cdot)}$ stands for some arbitrary but unique name in its respective context here.

Classical (Boolean) logic

First, we define $\Pi_{A.3}$ by combining Listings 2.3, 2.4, and 2.5

$$\Pi_{A.3} = \left\{ \begin{array}{l} eval(V,S) \leftarrow clamped(V,S). \\ free(V,D) \leftarrow formula(V,D), dnf(D,\_), \sim clamped(V,\_). \\ eval(V,1) \leftarrow free(V,D), dnf(D,J), eval(W,T) : clause(J,W,T). \\ eval(V,-1) \leftarrow variable(V), \sim eval(V,1). \end{array} \right\} \tag{A.3}$$

and we prove the Proposition 2.2.1.

Let (V,ϕ) be a logical network without feedback-loops and let C be a clamping assignment over V.
Then, there is an answer set X of $\tau((V,\phi), C) \cup \Pi_{A.3}$ such that $F = \{v \mapsto \boldsymbol{t} \mid eval(v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(v,-1) \in X\}$ if and only if F is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{f}}$.

Completeness. In what follows we denote $\Pi = \tau((V,\phi), C) \cup \Pi_{A.3}$. Let $F = \Omega^k_{(V,\phi|_C)}(A_{\boldsymbol{f}}) =$

$\Omega^{k+1}_{(V,\phi|_C)}(A_{\boldsymbol{f}})$ for some $k \geq 0$. We consider the following set of atoms X. To start with, the set of atoms $\tau((V,\phi),C)$ is included in X. Also, the set of atoms $\{free(v, s_{\phi(v)}) \mid v \in dom(\phi) \setminus dom(C)\}$ is included in X. Finally, the set $\{eval(v,1) \mid F(v) = \boldsymbol{t}\} \cup \{eval(v,-1) \mid F(v) = \boldsymbol{f}\}$ is also included in X. We need to show that X is a $\subseteq$-minimal model of Π^X. We do so by inspecting each rule in Π^X.

First, we note that X includes all facts in $\tau((V,\phi),C)$. Each of these facts belongs also to Π^X. Thus, any set Y of atoms excluding at least one of them, cannot be a model of Π^X. Next, consider the first rule in $\Pi_{A.3}$. Notably, all its grounded instantiations belong to Π^X. Furthermore, $clamped(v,s) \in X$ only if $v \in dom(C)$, and either $s = 1$ and $C(v) = \boldsymbol{t}$, or $s = -1$ and $C(v) = \boldsymbol{f}$. Notably, if $C(v) = \boldsymbol{t}$ then $F(v) = \boldsymbol{t}$, whereas if $C(v) = \boldsymbol{f}$ then $F(v) = \boldsymbol{f}$. Therefore, all grounded instantiations of the first rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,s)$ with $v \in dom(C)$ and $s \in \{1,-1\}$ cannot be a model of Π^X. Next, consider the second rule in $\Pi_{A.3}$. Its grounded instances belong to Π^X only if $clamped(v,s) \notin X$ with $v \in dom(\phi)$, $s \in \{1,-1\}$. Since $free(v, s_{\phi(v)}) \in X$ for all $v \in dom(\phi) \setminus dom(C)$, all such rules are satisfied by X, and any set excluding from X at least one atom over predicate $free/2$ cannot be a model of Π^X. Now, let us consider the third rule in $\Pi_{A.3}$. Note that all its grounded instantiations belong to Π^X. Furthermore, if $v \in dom(\phi) \setminus dom(C)$ then, $F(v) = \boldsymbol{t}$ iff there exists $\psi \in \phi(v)$ such that $F(\psi) = \boldsymbol{t}$. Then, $F(w) = \boldsymbol{t}$ for all $w \in \psi \cap V$, and $F(w) = \boldsymbol{f}$ for all $\neg w \in \psi$. Hence, the set of atoms

$$\{free(v, s_{\phi(v)}), dnf(s_{\phi(v)}, s_\psi)\}$$
$$\cup\{clause(s_\psi, w, 1), eval(w,1) \mid w \in \psi \cap V\} \cup \{clause(s_\psi, w, -1), eval(w,-1) \mid \neg w \in \psi\}$$

is included in X. Therefore, all grounded instantiations of the third rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,1)$ with $v \in dom(\phi) \setminus dom(C)$ cannot be a model of Π^X. Finally, for the fourth rule only ground instantiations such $eval(v,1) \notin X$ with $v \in V$ are included, which implies $F(v) = \boldsymbol{f}$. Furthermore, if $v \in V \setminus dom(C)$ we have two cases. If $v \notin dom(\phi)$, all grounded instantiations of the fourth rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,-1)$ cannot be a model of Π^X. If $v \in dom(\phi) \setminus dom(C)$ then, $F(v) = \boldsymbol{f}$ iff for all $\psi \in \phi(v)$ there exists w such that, either $F(w) = \boldsymbol{f}$ with $w \in \psi \cap V$, or $F(w) = \boldsymbol{t}$ with $\neg w \in \psi$. Hence, the set X includes either

$$\{free(v, s_{\phi(v)}), dnf(s_{\phi(v)}, s_\psi), clause(s_\psi, w, 1), eval(w,-1)\}$$

or

$$\{free(v, s_{\phi(v)}), dnf(s_{\phi(v)}, s_\psi), clause(s_\psi, w, -1), eval(w,1)\}\,.$$

Therefore, any set excluding from X at least one atom over the predicate $eval/2$ cannot be a model of Π^X.

Hence, we have investigated all rules in Π and shown that their ground instances in Π^X are satisfied by X. Moreover, we have checked that any set excluding from X at least one atom is not a model of Π^X. Hence, X is a $\subseteq$-minimal model of Π^X and thus an answer set of Π. □

Soundness. Let X be an answer set of $\Pi = \tau((V,\phi),C) \cup \Pi_{A.3}$. Let us briefly describe the reduct Π^X. Notably, the set of facts $\tau((V,\phi),C)$ is included, i.e., $\tau((V,\phi),C) \subseteq \Pi^X$. Ground instances of first and third rules in $\Pi_{A.3}$ are also included since they are positive. For the second rule, only ground instantiations such $\mathit{clamped}(v,s) \notin X$ with $v \in dom(\phi), s \in \{1,-1\}$ are included. Similarly, for the fourth rule only ground instantiations such $\mathit{eval}(v,1) \notin X$ with $v \in V$ are included. Next, by definition, we know that X is a $\subseteq$-minimal model of Π^X. Furthermore, $Cn(\Pi^X) = \bigcup_{i\geq 0} T^i_{\Pi^X}\emptyset = X$. In what what follows, let $F' = \Omega^k_{(V,\phi|_C)}(A_{\boldsymbol{f}}) = \Omega^{k+1}_{(V,\phi|_C)}(A_{\boldsymbol{f}})$ for some $k \geq 0$.

Next, for all $v \in V$ and $j \geq 2$ we show by induction that if $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$ then, either $s = 1$ and $F'(v) = \boldsymbol{t}$, or $s = -1$ and $F'(v) = \boldsymbol{f}$. Note that $T^0_{\Pi^X}\emptyset = \emptyset$ and $T^1_{\Pi^X}\emptyset = \tau((V,\phi),C)$.

Case $(j = 2)$**.** *By definition,* $T^2_{\Pi^X}\emptyset = T_{\Pi^X}T^1_{\Pi^X}\emptyset = T_{\Pi^X}\tau((V,\phi),C)$*. Then, it is easy to see that* $\mathit{eval}(v,1) \in T^2_{\Pi^X}\emptyset$ *iff* $\mathit{clamped}(v,1) \in \tau((V,\phi),C)$*. Furthermore,* $\mathit{clamped}(v,1) \in \tau((V,\phi),C)$ *also implies* $v \in dom(C)$ *and* $C(v) = \boldsymbol{t}$*. Also, we can see that* $\mathit{eval}(v,-1) \in T^2_{\Pi^X}\emptyset$ *iff either* $\mathit{clamped}(v,-1) \in \tau((V,\phi),C)$*, or* $\mathit{variable}(v) \in \tau((V,\phi),C)$ *and* $\mathit{eval}(v,1) \notin X$*. Moreover, in this case* $\mathit{clamped}(v,-1) \in \tau((V,\phi),C)$ *also implies* $v \in dom(C)$ *and* $C(v) = \boldsymbol{f}$*.*

Next, if we assume $\mathit{eval}(v,1) \in T^2_{\Pi^X}\emptyset$*, then we have* $\phi|_C(v) = \top$*. Therefore,* $\Omega^k_{(V,\phi|_C)}(A_{\boldsymbol{f}})(v) = \boldsymbol{t}$ *for all* $k \geq 1$*. If we assume* $\mathit{eval}(v,-1) \in T^2_{\Pi^X}\emptyset$*, then we have either* $\phi|_C(v) = \bot$*, or* $v \in V \setminus dom(C)$*. If* $\phi|_C(v) = \bot$*, or* $v \notin dom(\phi|_C)$ *it is easy to see that* $\Omega^k_{(V,\phi|_C)}(A_{\boldsymbol{f}})(v) = \boldsymbol{f}$ *for all* $k \geq 1$*.*

For the remaining case, i.e, $v \in dom(\phi) \setminus dom(C)$*, we provide a rather informal argument due to the fact that* $\Omega_{(V,\phi|_C)}$ *is non-monotonic over two-valued assignments. Thus, it is hard (and does not worth) to write a precise constructive proof for finding the iterative step for which* v *reaches its fixpoint in* $\Omega_{(V,\phi|_C)}$ *starting from* $A_{\boldsymbol{f}}$*. However, since there are no feedback-loops in* (V,ϕ)*, we know that all variables must reach their fixpoint at some iterative step. Moreover, since* $\mathit{eval}(v,1) \notin X$*, there must be a set of atoms* $S \subseteq X$ *such that the body of all instantiations with respect to* v *of the third rule in* Π_2 *are false. Furthermore, provided that there are no feedback-loops in* (V,ϕ)*, the derivation of all atoms in* S *must be independent from the atom* $\mathit{eval}(v,-1)$*. Then, it is safe to assume that if* $\mathit{eval}(w,s) \in S$*, then either* $s = 1$ *and* $F'(w) = \boldsymbol{t}$*, or* $s = -1$ *and* $F'(w) = \boldsymbol{f}$*. Notably, the difficulty to conclude this more precisely is due to the fact that there may exist* $\mathit{eval}(w,1) \in S$ *such that* $\mathit{eval}(w,1) \notin T^2_{\Pi^X}\emptyset$*, i.e.* $\mathit{eval}(w,1)$ *is a consequence in a later iterative step. Again, due to the absence of feedback-loops which ensures the independence of* S *from* $\mathit{eval}(v,-1)$*, we can capture such atoms in the inductive case below. Finally, it is easy to see that for all* $\psi \in \phi(v)$ *we have* $F'(\psi) = \boldsymbol{f}$ *and thus,* $F'(v) = \boldsymbol{f}$*.*

Case $(j > 2)$**.** *By definition,* $T^{j+1}_{\Pi^X}\emptyset = T_{\Pi^X}T^j_{\Pi^X}\emptyset$*. Thus, if* $\mathit{eval}(v,s) \in T^{j+1}_{\Pi^X}\emptyset$*, either* $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$*, or* $\mathit{eval}(v,s)$ *is a consequence of* $T^j_{\Pi^X}\emptyset$ *by applying rules in* Π^X*. If* $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$*, by inductive hypothesis we have, either* $s = 1$ *and* $F'(v) = \boldsymbol{t}$*, or* $s = -1$ *and* $F'(v) = \boldsymbol{f}$*. Otherwise, we only need to consider* $s = 1$ *since for all* $j \geq 2$ *it cannot be the case that* $\mathit{eval}(v,-1) \in T^{j+1}_{\Pi^X}\emptyset$ *if*

eval$(v,-1) \notin T^{j}_{\Pi^X}\emptyset$. *Thus,* $v \in dom(\phi)$ *and there must exist* $\psi \in \phi(v)$ *such that* $Y \subseteq T^{j}_{\Pi^X}\emptyset$ *with*

$$Y = \{\mathit{free}(v, s_{\phi(v)}), \mathit{dnf}(s_{\phi(v)}, s_\psi)\} \cup \{\mathit{clause}(s_\psi, w, 1), \mathit{eval}(w,1) \mid w \in \psi \cap V\} \cup \{\mathit{clause}(s_\psi, w, -1), \mathit{eval}(w,-1) \mid \neg w \in \psi\}\,.$$

Next, by inductive hypothesis we have $F'(w) = \boldsymbol{t}$ *for all* $w \in \psi \cap V$, *and* $F'(w) = \boldsymbol{f}$ *for all* $\neg w \in \psi$. *Then, we also have* $F'(\psi) = \boldsymbol{t}$. *Therefore, provided that* $\phi|_C(v)$ *is in disjunctive normal form, we have* $F'(\phi|_C(v)) = F'(v) = \boldsymbol{t}$.

Hence, it follows that $F' = \{v \mapsto \boldsymbol{t} \mid \mathit{eval}(v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid \mathit{eval}(v,-1) \in X\} = F$ is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{f}}$. □

Kleene's (three-valued) logics

Next, we define $\Pi_{A.4}$ by combining Listings 2.3, 2.4, and 2.6.

$$\Pi_{A.4} = \left\{ \begin{array}{r} \mathit{eval}(V,S) \leftarrow \mathit{clamped}(V,S). \\ \mathit{free}(V,D) \leftarrow \mathit{formula}(V,D), \mathit{dnf}(D,\_), \sim \mathit{clamped}(V,\_). \\ \mathit{eval\_clause}(J,-1) \leftarrow \mathit{clause}(J,V,S), \mathit{eval}(V,-S). \\ \mathit{eval}(V,1) \leftarrow \mathit{free}(V,D), \mathit{dnf}(D,J), \mathit{eval}(W,T) : \mathit{clause}(J,W,T). \\ \mathit{eval}(V,-1) \leftarrow \mathit{free}(V,D), \mathit{eval\_clause}(J,-1) : \mathit{dnf}(D,J). \end{array} \right\} \tag{A.4}$$

and we prove the Proposition 2.2.2.

Let (V,ϕ) be a logical network and let C be a clamping assignment over V.
Then, there is an answer set X of $\tau((V,\phi),C) \cup \Pi_{A.4}$ such that $F = \{v \mapsto \boldsymbol{t} \mid \mathit{eval}(v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid \mathit{eval}(v,-1) \in X\} \cup \{v \mapsto \boldsymbol{u} \mid \mathit{eval}(v,1) \notin X, \mathit{eval}(v,-1) \notin X\}$ if and only if F is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{u}}$.

Completeness. In what follows we denote $\Pi = \tau((V,\phi),C) \cup \Pi_{A.4}$. Let $F = \Omega^{k}_{(V,\phi|_C)}(A_{\boldsymbol{u}}) = \Omega^{k+1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})$ for some $k \geq 0$. We consider the following set of atoms X. To start with, the set of atoms $\tau((V,\phi),C)$ is included in X. Also, the set of atoms $\{\mathit{free}(v, s_{\phi(v)}) \mid v \in dom(\phi) \setminus dom(C)\}$ is included in X. The set $\{\mathit{eval}(v,1) \mid F(v) = \boldsymbol{t}\} \cup \{\mathit{eval}(v,-1) \mid F(v) = \boldsymbol{f}\}$ is also included in X. Furthermore, if $v \in dom(\phi)$ and $F(v) = \boldsymbol{f}$, for all $\psi \in \phi(v)$ atoms of the form $\mathit{eval\_clause}(s_\psi, -1)$ are included in X. We need to show that X is a $\subseteq$-minimal model of Π^X. We do so by inspecting each rule in Π^X.

First, we note that X includes all facts in $\tau((V,\phi),C)$. Each of these facts belongs also to Π^X. Thus, any set Y of atoms excluding at least one of them, cannot be a model of Π^X. Next, consider the first rule in $\Pi_{A.4}$. Notably, all its grounded instantiations belong to Π^X. Furthermore, $\mathit{clamped}(v,s) \in X$ only if $v \in dom(C)$, and either $s = 1$ and $C(v) = \boldsymbol{t}$, or $s = -1$ and $C(v) = \boldsymbol{f}$. Notably, if $C(v) = \boldsymbol{t}$ then $F(v) = \boldsymbol{t}$, whereas if $C(v) = \boldsymbol{f}$ then $F(v) = \boldsymbol{f}$. Therefore, all

grounded instantiations of the first rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,s)$ with $v \in dom(C)$ and $s \in \{1,-1\}$ cannot be a model of Π^X. Next, consider the second rule in $\Pi_{A.4}$. Its grounded instances belong to Π^X only if $clamped(v,s) \notin X$ with $v \in dom(\phi), s \in \{1,-1\}$. Since $free(v, s_{\phi(v)}) \in X$ for all $v \in dom(\phi) \setminus dom(C)$, all such rules are satisfied by X, and any set excluding from X at least one atom over predicate *free*/2 cannot be a model of Π^X. Next, consider the third rule in $\Pi_{A.4}$. Notably, all its grounded instantiations belong to Π^X. Recall that $eval\_clause(s_\psi, -1) \in X$ only if $v \in dom(\phi)$ and $F(v) = \boldsymbol{f}$ for all $\psi \in \phi(v)$. But, $F(v) = \boldsymbol{f}$ iff for all $\psi \in \phi(v)$ there exists w such that, either $F(w) = \boldsymbol{f}$ with $w \in \psi \cap V$, or $F(w) = \boldsymbol{t}$ with $\neg w \in \psi$. Hence, the set X includes either

$$\{clause(s_\psi, w, 1), eval(w, -1)\}$$

or

$$\{clause(s_\psi, w, -1), eval(w, 1)\}\,.$$

Therefore, any set excluding from X at least one atom over the predicate *eval_clause*/2 cannot be a model of Π^X. Now, let us consider the fourth rule in $\Pi_{A.4}$. Note that all its grounded instantiations belong to Π^X. Furthermore, if $v \in dom(\phi) \setminus dom(C)$ then, $F(v) = \boldsymbol{t}$ iff there exists $\psi \in \phi(v)$ such that $F(\psi) = \boldsymbol{t}$. Then, $F(w) = \boldsymbol{t}$ for all $w \in \psi \cap V$, and $F(w) = \boldsymbol{f}$ for all $\neg w \in \psi$. Hence, the set of atoms

$$\begin{aligned}&\{free(v, s_{\phi(v)}), dnf(s_{\phi(v)}, s_\psi)\}\\ \cup &\{clause(s_\psi, w, 1), eval(w, 1) \mid w \in \psi \cap V\} \cup \{clause(s_\psi, w, -1), eval(w, -1) \mid \neg w \in \psi\}\end{aligned}$$

is included in X. Therefore, all grounded instantiations of the fourth rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,1)$ with $v \in dom(\phi) \setminus dom(C)$ cannot be a model of Π^X. Finally, for the fifth rule all grounded instantiations belong to Π^X. If $v \in dom(\phi) \setminus dom(C)$ and $F(v) = \boldsymbol{f}$, then the set of atoms

$$\{free(v, s_{\phi(v)})\} \cup \{eval\_clause(s_\psi, -1), dnf(s_{\phi(v)}, s_\psi) \mid \psi \in \phi(v)\}$$

is included in X. Therefore, all grounded instantiations of the fifth rule are satisfied by X, and any set excluding from X at least one atom of the form $eval(v,-1)$ with $v \in dom(\phi) \setminus dom(C)$ cannot be a model of Π^X.

Hence, we have investigated all rules in Π and shown that their ground instances in Π^X are satisfied by X. Moreover, we have checked that any set excluding from X at least one atom is not a model of Π^X. Hence, X is a $\subseteq$-minimal model of Π^X and thus an answer set of Π. □

Soundness. Let X be an answer set of $\Pi = \tau((V,\phi), C) \cup \Pi_{A.4}$. Let us briefly describe the reduct Π^X. Notably, the set of facts $\tau((V,\phi), C)$ is included, i.e., $\tau((V,\phi), C) \subseteq \Pi^X$. Ground instances of all rules in $\Pi_{A.4}$ except the second one are also included since they are positive. For the second rule, only ground instantiations such $clamped(v,s) \notin X$ with $v \in dom(\phi), s \in \{1,-1\}$ are included. Next, by definition, we know that X is a $\subseteq$-minimal model of Π^X. Furthermore,

$Cn(\Pi^X) = \bigcup_{i \geq 0} T^i_{\Pi^X}\emptyset = X$.

Next, for all $v \in V$ and $j \geq 2$ we show by induction that if $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$ then, either $s = 1$ and $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{t}$, or $s = -1$ and $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{f}$. Otherwise, $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{u}$. Note that $T^0_{\Pi^X}\emptyset = \emptyset$ and $T^1_{\Pi^X}\emptyset = \tau((V,\phi),C)$.

Case ($j = 2$)**.** *By definition,* $T^2_{\Pi^X}\emptyset = T_{\Pi^X}T^1_{\Pi^X}\emptyset = T_{\Pi^X}\tau((V,\phi),C)$. *Then, it is easy to see that* $\mathit{eval}(v,1) \in T^2_{\Pi^X}\emptyset$ *iff* $\mathit{clamped}(v,1) \in \tau((V,\phi),C)$. *Furthermore,* $\mathit{clamped}(v,1) \in \tau((V,\phi),C)$ *also implies* $v \in dom(C)$ *and* $C(v) = \boldsymbol{t}$. *Also, we can see that* $\mathit{eval}(v,-1) \in T^2_{\Pi^X}\emptyset$ *iff* $\mathit{clamped}(v,-1) \in \tau((V,\phi),C)$ *Moreover, in this case* $\mathit{clamped}(v,-1) \in \tau((V,\phi),C)$ *also implies* $v \in dom(C)$ *and* $C(v) = \boldsymbol{f}$.

Next, if we assume $\mathit{eval}(v,1) \in T^2_{\Pi^X}\emptyset$, *then we have* $\phi|_C(v) = \top$. *Therefore, it is easy to see that* $\Omega^1_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{t}$. *If we assume* $\mathit{eval}(v,-1) \in T^2_{\Pi^X}\emptyset$, *then we have* $\phi|_C(v) = \bot$. *Therefore, again it is easy to see that* $\Omega^1_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{f}$. *Notably, if* $\mathit{clamped}(v,s) \notin \tau((V,\phi),C)$ *for neither* $s = 1$ *nor* $s = -1$, *we have* $v \notin dom(C)$. *Therefore, we can see that* $\Omega^1_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{u}$.

Case ($j > 2$)**.** *By definition,* $T^{j+1}_{\Pi^X}\emptyset = T_{\Pi^X}T^j_{\Pi^X}\emptyset$. *Thus, if* $\mathit{eval}(v,s) \in T^{j+1}_{\Pi^X}\emptyset$, *either* $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$, *or* $\mathit{eval}(v,s)$ *is a consequence of* $T^j_{\Pi^X}\emptyset$ *by applying rules in* Π^X. *If* $\mathit{eval}(v,s) \in T^j_{\Pi^X}\emptyset$, *by inductive hypothesis and monotonicity of* $\Omega_{(V,\phi|_C)}$ *over three-valued logics we have, either* $s = 1$ *and* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{t}$, *or* $s = -1$ *and* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{f}$. *Otherwise,* $\mathit{free}(v, s_{\phi(v)}) \in T^j_{\Pi^X}\emptyset$ *and thus,* $v \in dom(\phi) \setminus dom(C)$.

If $s = 1$, *there must exist* $\psi \in \phi(v)$ *such that* $Y \subseteq T^j_{\Pi^X}\emptyset$ *with*

$$Y = \{\mathit{free}(v, s_{\phi(v)}), \mathit{dnf}(s_{\phi(v)}, s_\psi)\} \cup \{\mathit{clause}(s_\psi, w, 1), \mathit{eval}(w,1) \mid w \in \psi \cap V\} \cup \{\mathit{clause}(s_\psi, w, -1), \mathit{eval}(w,-1) \mid \neg w \in \psi\}\,.$$

Next, by inductive hypothesis we have $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(w) = \boldsymbol{t}$ *for all* $w \in \psi \cap V$, *and* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(w) = \boldsymbol{f}$ *for all* $\neg w \in \psi$. *Then, we also have* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(\psi) = \boldsymbol{t}$. *Therefore, provided that* $\phi|_C(v)$ *is in disjunctive normal form, we have* $\Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(\phi|_C(v)) = \Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{t}$.

If $s = -1$, *for all* $\psi \in \phi(v)$ *we must have* $\mathit{eval\_clause}(s_\psi, -1) \in T^j_{\Pi^X}\emptyset$. *Hence, we also have that there exists* w *such that, either* $\{\mathit{clause}(s_\psi, w, 1), \mathit{eval}(w,-1)\} \subseteq T^j_{\Pi^X}\emptyset \subseteq X$ *with* $w \in \psi \cap V$, *or* $\{\mathit{clause}(s_\psi, w, -1), \mathit{eval}(w,1)\} \subseteq T^j_{\Pi^X}\emptyset \subseteq X$ *with* $\neg w \in \psi$. *Next, by inductive hypothesis we have* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(w) = \boldsymbol{f}$ *for all* $w \in \psi \cap V$, *and* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(w) = \boldsymbol{t}$ *for all* $\neg w \in \psi$. *Then, we also have* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(\psi) = \boldsymbol{f}$. *Therefore, provided that* $\phi|_C(v)$ *is in disjunctive normal form, we have* $\Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(\phi|_C(v)) = \Omega^j_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{f}$.

Finally, if $\mathit{eval}(v,s) \notin T^{j+1}_{\Pi^X}\emptyset$ *with* $s \in \{1,-1\}$, *by inductive hypothesis and monotonicity of* $\Omega_{(V,\phi|_C)}$ *over three-valued logics we have* $\Omega^{j-1}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{u}$. *Furthermore, since* $\mathit{eval}(v,s)$ *is neither a consequence of* $T^j_{\Pi^X}\emptyset$ *by applying rules in* Π^X, *based on the previous cases we can see*

that $\Omega^{j}_{(V,\phi|_C)}(A_{\boldsymbol{u}})(v) = \boldsymbol{u}$.

Hence, it follows that $F = \{v \mapsto \boldsymbol{t} \mid eval(v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(v,-1) \in X\} \cup \{v \mapsto \boldsymbol{u} \mid eval(v,1) \notin X, eval(v,-1) \notin X\}$ is the unique fixpoint of $\Omega_{(V,\phi|_C)}$ reachable from $A_{\boldsymbol{u}}$. □

A.2.2 Extended ASP representations

In what follows we provide informal arguments to show the correctness of each ASP encoding relying on the proofs given above. In each case, our application-specific encodings are straightforward elaborations of the rules in $\Pi_{A.3}$ or $\Pi_{A.4}$, plus certain rules or optimization statements describing the exact corresponding mathematical formulation.

Learning Boolean logic models of immediate-early response

Let (V,E,σ) be a PKN. Let $\xi = (C_i, P_{C_i})$ be an experimental dataset over (V,E,σ) and let k define the discretization scheme.
Then, there is an answer set X of $\tau((V,E,\sigma),\xi,k) \cup \Pi_{3.2}$ such that

$$\phi_{opt} = \left\{ v \mapsto \bigvee_{p \in \mathscr{P}(v)} \left(\bigwedge_{(w,1)\in p} w \right) \wedge \left(\bigwedge_{(w,-1)\in p} \neg w \right) \;\middle|\; dnf(s_{\mathscr{P}(v)}, s_p) \in X, v \in V \right\}$$

if and only if $(V,\phi_{opt}) \in \arg\min_{(V,\phi)\in\mathbb{M}_{(V,E,\sigma)}} (\Theta_{rss_k}((V,\phi),\xi), \Theta_{size}((V,\phi)))$.

Proof. The proof follows from the Proposition 2.2.1. First, we note that rules 1-10 in $\Pi_{3.2}$ generate the representation of all logical networks $(V,\phi) \in \mathbb{M}_{(V,E,\sigma)}$. Next, it is easy to extend Proposition 2.2.1 to consider several clamping assignments C_i. More precisely, let us denote with Π' the set of rules 1-19 in $\Pi_{3.2}$. Then, we have that for each $(V,\phi) \in \mathbb{M}_{(V,E,\sigma)}$ there is an answer set X of $\tau((V,E,\sigma),\xi,k) \cup \Pi'$ such that $F_i = \{v \mapsto \boldsymbol{t} \mid eval(i,v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(i,v,-1) \in X\}$ if and only if F_i is the unique fixpoint of $\Omega_{(V,\phi|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$. Also, one can verify that considering the complete set of rules in $\Pi_{3.2}$, X must include atoms of the form $rss(o,v,1,(f-d)^2)$ and $rss(o,v,-1,d^2)$ with $f = 10^k$ and $d = 10^k \delta_k(P_{C_i}(v))$ for all $i = 1,\ldots,n$ and $v \in dom(P_{C_i})$. Therefore, we have that rule 25 in $\Pi_{3.2}$ enforces the minimization of the residual sum of squares between the Boolean predictions and discretized experimental observations (Eq. (3.5)). Furthermore, with lower priority rule 24, enforces the minimization over the size of logic models (Eq. (3.3)). □

Finding logical input-output behaviors

Let $(V,\phi_j),(V,\phi_{j'})$ be two Boolean logic models.
Then, there is an answer set of $\tau((V,\phi_j),(V,\phi_{j'})) \cup \Pi_{3.5}$ if and only if there exist $C_i \in \mathscr{C}$, $v \in V_R$ such that $F^j_i(v) \neq F^{j'}_i(v)$.

Proof. The proof follows from the Proposition 2.2.1. First, we note that rules 1-3 in $\Pi_{3.5}$ generate the representation of all clamping assignments $C_i \in \mathscr{C}$. Next, it is easy to extend Proposition 2.2.1 to consider several logical networks, in this only two. More precisely, we have that for each $C_i \in \mathscr{C}$ there is an answer set X of $\Pi = \tau((V,\phi_j),(V,\phi_{j'})) \cup \Pi_{3.5}$ such that $F_i^j = \{v \mapsto \boldsymbol{t} \mid eval(j,v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(j,v,-1) \in X\}$ if and only if F_i^j is the unique fixpoint of $\Omega_{(V,\phi_j|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$, and the same holds with j' instead of j. Therefore, it follows that X is an answer set of Π (in particular, the atom *diff* is included) iff there exists $v \in V_R$ such that $F_i^j(v) \neq F_i^{j'}(v)$. □

Experimental design

Let $\mathscr{B}$ be a finite set of input-output behaviors represented by Boolean logic models $((V,\phi_j))_{j\in J}$. Let ε, s, and k be three positive integers with $s \leq |V_S|$ and $k \leq |V_K|$.
Then, there is an answer set X of $\tau(\mathscr{B},s,k) \cup \Pi_{4.2}$ such that $C_i = \{v \mapsto \boldsymbol{t} \mid v \in V_S, clamped(i,v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid v \in V_K, clamped(i,v,-1) \in X\}$ with $i = 1,\ldots,\varepsilon$ if and only if,

1. ε is the least number of clamping assignments for which (4.1) holds,
2. and $(C_1,\ldots,C_\varepsilon) \in \arg\min_{(C_1,\ldots,C_\varepsilon)\in\Delta}\left(\Theta_{V_S}((C_1,\ldots,C_\varepsilon)),\Theta_{V_K}((C_1,\ldots,C_\varepsilon))\right)$ with $\Delta = \arg\max_{(C_1,\ldots,C_\varepsilon)\in\mathscr{C}^\varepsilon(s,k)}\left(\Theta_{diff}(\mathscr{B},(C_1,\ldots,C_\varepsilon))\right)$ and $\mathscr{C}^\varepsilon(s,k)$ the set of all ε-tuples $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}_1(s,k) \times \cdots \times \mathscr{C}_\varepsilon(s,k)$ satisfying (4.1).

Proof. The proof follows from the Proposition 2.2.1. First, we note that rules 8-10 in $\Pi_{4.2}$ generate the representation of all clamping assignments $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}_1(s,k) \times \cdots \times \mathscr{C}_\varepsilon(s,k)$ for increasing values of $\varepsilon \geq 1$. Next, it is easy to extend Proposition 2.2.1 to consider several logical networks and clamping assignments. More precisely, let us denote with Π' the set of rules 1-17 in $\Pi_{4.2}$. Then, we have that for each $(C_1,\ldots,C_\varepsilon) \in \mathscr{C}_1(s,k) \times \cdots \times \mathscr{C}_\varepsilon(s,k)$ there is an answer set X of $\tau(\mathscr{B},s,k) \cup \Pi'$ such that $F_i^j = \{v \mapsto \boldsymbol{t} \mid eval(i,j,v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(i,j,v,-1) \in X\}$ if and only if F_i^j is the unique fixpoint of $\Omega_{(V,\phi_j|_{C_i})}$ reachable from $A_{\boldsymbol{f}}$. Also, one can verify that considering the complete set of rules in $\Pi_{4.2}$, such an answer set X exists iff ε is the least integer such that for all pairs of networks $(V,\phi_j),(V,\phi_{j'})$ with $j < j'$, there are $i \in \{1,\ldots,\varepsilon\}$, $v \in V_R$ such that $\{diff(i,j,j',v), diff(i,j,j')\} \subseteq X$. Therefore, the first condition above is satisfied, whereas the three optimization statements in rules 23-25 enforce the desired optimality criteria in the second condition. □

Minimal intervention strategies

Let $(V,\phi_i)_{i\in N}$ be a finite family of logical networks. Let $(G_j,C_j)_{j\in J}$ be a finite family of intervention scenarios and k some positive integer.
Then, there is an answer set X of $\tau((V,\phi_i)_{i\in N},(G_j,C_j)_{j\in J},k) \cup \Pi_{5.2}$ such that $I = \{v \mapsto \boldsymbol{t} \mid intervention(v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid intervention(v,-1) \in X\}$ if and only if I is a bounded intervention strategy for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ and k.

Proof. The proof follows from the Proposition 2.2.2. First, we note that rules 1-14 in $\Pi_{5.2}$ generate the representation of all "candidate" intervention sets I. Next, it is easy to extend Proposition 2.2.2 to consider several logical networks $(V,\phi_i)_{i\in N}$ and clamping assignments $(C_j \circ I)_{j\in J}$. More precisely, let us denote with Π' the set of rules 1-26 in $\Pi_{5.2}$. Then, we have that for each "candidate" intervention set I, there is an answer set X of $\tau((V,\phi_i)_{i\in N},(G_j,C_j)_{j\in J},k)\cup \Pi'$ such that $F_j^i = \{v \mapsto \boldsymbol{t} \mid eval(i,j,v,1) \in X\} \cup \{v \mapsto \boldsymbol{f} \mid eval(i,j,v,-1) \in X\}$ if and only if F_j^i is the unique fixpoint of $\Omega_{(V,\phi_i|_{C_j\circ I})}$ reachable from $A_{\boldsymbol{u}}$. Also, one can verify that considering the complete set of rules in $\Pi_{5.2}$, such an answer set X exists iff for all $(G_j)_{j\in J}$ and $i \in N$, it hold $G_j \subseteq F_j^i$ and $|I| \leq k$. Therefore, I is a bounded intervention strategy for $(G_j,C_j)_{j\in J}$ wrt $(V,\phi_i)_{i\in N}$ and k. Moreover, if $k \leq 0$ the integrity constraint in rule 31 is not applied and thus, I is an unbounded intervention strategy. □

Bibliography

A. Abdi, M. B. Tahoori, and E. S. Emamian. Fault diagnosis engineering of digital circuits can identify vulnerable molecules in complex cellular pathways. *Science Signaling*, 1(42), 2008.

T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, July 2000.

T. Akutsu, S. Kuhara, O. Maruyama, and Satoru S. Miyano. Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model. *Theoretical Computer Science*, 298(1):235–251, March 2003.

I. Albert, J. Thakar, S. Li, R. Zhang, and R. Albert. Boolean network simulations for life scientists. *Source Code for Biology and Medicine*, 3(16), December 2007.

B. B. Aldridge, J. M. Burke, D. A. Lauffenburger, and P. Sorger. Physicochemical modelling of cell signalling pathways. *Nature cell biology*, 8(11):1195–1203, 2006.

L. G. Alexopoulos, J. Saez-Rodriguez, B. Cosgrove, D. A. Lauffenburger, and P. Sorger. Networks inferred from biochemical data reveal profound differences in toll-like receptor and inflammatory signaling between normal and transformed hepatocytes. *Molecular & Cellular Proteomics*, 9(9):1849–1865, 2010.

K. R. Apt and M. H. Van Emden. Contributions to the theory of logic programming. *ACM*, 29 (3):841–862, July 1982.

J. Banga. Optimization in computational systems biology. *BMC systems biology*, 2(1):47, 2008.

C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

C. Baral, K. Chancellor, N. Tran, N. Tran, A. Joy, and M. Berens. A knowledge based approach for representing and reasoning about signaling networks. In *Proceedings of the Twelfth International Conference on Intelligent Systems for Molecular Biology/Third European Conference on Computational Biology (ISMB'04/ECCB'04)*, pages 15–22, 2004.

C. L. Barrett and B. Ø. Palsson. Iterative reconstruction of transcriptional regulatory networks: an algorithmic approach. *PLoS Computational Biology*, 2(5), April 2006.

G. Batt, D. Ropers, de 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, May 2005.

G. Batt, B. Besson, P. Ciron, H. de Jong, E. Dumas, J. Geiselmann, R. Monte, P. T. Monteiro, M. Page, F. Rechenmann, and D. Ropers. Genetic network analyzer: a tool for the qualitative modeling and simulation of bacterial regulatory networks. *Methods in Molecular Biology*, 804:439–462, 2012.

N. Berestovsky and L. Nakhleh. An evaluation of methods for inferring boolean networks from time-series data. *PLoS ONE*, 8(6):e66031, 2013.

S. Bornholdt. Boolean network models of cellular regulation: prospects and limitations. *Journal of the Royal Society Interface*, 5:S85–S94, 2008.

A. G. Busetto, A. Hauser, G. Krummenacher, M. Sunnåker, S. Dimopoulos, C. S. Ong, J. Stelling, and J. M. Buhmann. Near-optimal experimental design for model selection in systems biology. *Bioinformatics*, 29(20):2625–2632, October 2013.

L. Calzone, F. Fages, and S. Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, July 2006.

L. Calzone, L. Tournier, S. Fourquet, D. Thieffry, B. Zhivotovsky, E. Barillot, and A. Zinovyev. Mathematical modelling of cell-fate decision in response to death receptor engagement. *PLoS Computational Biology*, 6(3), February 2010.

A. Di Cara, A. Garg, G. De Micheli, I. Xenarios, and L. Mendoza. Dynamic simulation of regulatory networks using SQUAD. *BMC Bioinformatics*, 8(462), December 2006.

E. G. Cerami, B. E. Gross, E. Demir, I. Rodchenkov, Ö. Babur, N. Anwar, N. Schultz, G. D. Bader, and C. Sander. Pathway commons, a web resource for biological pathway data. *Nucleic Acids Research*, 39(Database issue):D685–D690, 2011.

N. Chabrier and F. Fages. Symbolic model checking of biochemical networks. In C. Pirami, editor, *Computational Methods in Systems Biology*, pages 149–162. Springer-Verlag, February 2003.

C. Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4): 210–219, July 2007.

C. Chaouiya, D. Bérenguier, S. M. Keating, A. Naldi, M. P. van Iersel, N. Rodriguez, A. Dräger, F. Büchel, T. Cokelaer, B. Kowal, B. Wicks, E. Gonçalves, J. Dorier, M. Page, P. T. Monteiro, A. Von Kamp, I. Xenarios, H. de Jong, M. Hucka, S. Klamt, D. Thieffry, N. Le Novère, J. Saez-Rodriguez, and T. Helikar. SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools. *BMC systems biology*, 7(135), December 2013.

W. W. Chen, B. Schoeberl, P. J. Jasper, M. Niepel, U. B. Nielsen, D. A. Lauffenburger, and P. Sorger. Input-output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data. *Molecular Systems Biology*, 5(1), January 2009.

T. S. Christensen, A. P. Oliveira, and J. Nielsen. Reconstruction and logical modeling of glucose repression signaling pathways in saccharomyces cerevisiae. *BMC systems biology*, 3(7), 2009.

A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma and K. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, pages 359–364. Springer-Verlag, 2002.

G. Collet, D. Eveillard, M. Gebser, S. Prigent, T. Schaub, A. Siegel, and S. Thiele. Extending the metabolic network of Ectocarpus siliculosus using answer set programming. In P. Cabalar and T. Son, editors, *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, volume 8148 of *Lecture Notes in Artificial Intelligence*, pages 245–256. Springer-Verlag, 2013.

P. Csermely, T. Korcsmáros, H. J. M. Kiss, F. London, and R. Nussinov. Structure and dynamics of molecular networks: a novel paradigm of drug discovery: a comprehensive review. *Pharmacology & therapeutics*, 138(3):333–408, 2013.

V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1): 69–110, December 2003.

H. de Jong, J. Gouzé, C. Hernandez, M. Page, T. Sari, and J. Geiselmann. Qualitative simulation of genetic regulatory networks using piecewise-linear models. *Bulletin of mathematical biology*, 66(2):301–340, March 2004.

E. Dubrova and M. Teslenko. A SAT-based algorithm for finding attractors in synchronous Boolean networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 8 (5):1393–1399, August 2011.

M. Durzinsky, W. Marwan, M. Ostrowski, T. Schaub, and A. Wagler. Automatic network reconstruction using ASP. *Theory and Practice of Logic Programming*, 11(4-5):749–766, 2011.

T. Fayruzov, J. Janssen, D. Vermeir, C. Cornelis, and M. De Cock. Modelling gene and protein regulatory networks with answer set programming. *International Journal of Data Mining and Bioinformatics*, 5(2):209–229, 2011.

J. Fisher and T. A. Henzinger. Executable cell biology. *Nature biotechnology*, 25(11):1239–1249, November 2007.

M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4): 295–312, 1985.

A. Freitas. A critical review of multi-objective optimization in data mining. *ACM SIGKDD Explorations Newsletter*, 6(2):77, December 2004.

G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, 1993.

S. Gay, F. Fages, F. Santini, and S. Soliman. Solving subgraph epimorphism problems using CLP and SAT. In *Proceedings of seventh Workshop on Constraint Based Methods for Bioinformatics WCB'11*, pages 59–66, 2011.

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. Schlipf, editors, *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*, pages 260–265. Springer-Verlag, 2007.

M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In F. Lin and U. Sattler, editors, *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 497–507. AAAI Press, 2010.

M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011a.

M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2-3): 323–360, 2011b.

M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012a.

M. Gebser, B. Kaufmann, and T. Schaub. Multi-threaded ASP solving with clasp. *Theory and Practice of Logic Programming*, 12(4-5):525–545, 2012b.

M. Gebser, B. Kaufmann, R. Otero, J. Romero, T. Schaub, and P. Wanko. Domain-specific heuristics in answer set programming. In M. desJardins and M. Littman, editors, *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13)*, pages 350–356. AAAI Press, 2013.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

B. D. Gomperts, I. M. Kramer, and P. E. R. Tatham. *Signal Transduction*. Academic Press, San Diego, California, 2009.

W. Guo, G. Yang, W. Wu, L. He, and M. Sun. A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks. *PLoS ONE*, 9(4), April 2014.

C. Guziolowski, A. Kittas, F. Dittmann, and N. Grabe. Automatic generation of causal networks linking growth factor stimuli to functional cell state changes. *FEBS Journal*, 279(18):3462–3474, 2012. ISSN 1742-4658.

C. Guziolowski, S. Videla, F. Eduati, S. Thiele, T. Cokelaer, A. Siegel, and J. Saez-Rodriguez. Exhaustively characterizing feasible logic models of a signaling network using answer set programming. *Bioinformatics*, 29(18):2320–2326, 2013.

P. Hainaut and A. Plymoth. Targeting the hallmarks of cancer: towards a rational approach to next-generation cancer therapy. *Current Opinion in Oncology*, 25(1):50–51, 2012.

D. Hanahan and R. A. Weinberg. Hallmarks of Cancer: The Next Generation. *Cell*, 144(5): 646–674, 2011.

M. Heiner and D. Gilbert. How might Petri nets enhance your systems biology toolkit. In L. M. Kristensen and L. Petrucci, editors, *Applications and Theory of Petri nets*, volume 6709 of *Lecture Notes in Computer Science*, pages 17–37. Springer-Verlag, 2011.

T. Helikar and J. A. Rogers. ChemChains: a platform for simulation and analysis of biochemical networks aimed to laboratory scientists. *BMC systems biology*, 3(58), December 2008.

T. Helikar, B. Kowal, S. McClenathan, M. Bruckner, T. Rowley, A. Madrahimov, W. Ben, M. Shrestha, K. Limbu, and J.A. Rogers. The Cell Collective: toward an open and collaborative approach to systems biology. *BMC systems biology*, 6:96–96, December 2011.

T. A. Henzinger. The theory of hybrid automata. In *Proceedings 11th IEEE Symposium on Logic in Computer Science*, pages 278–292, 1996.

F. Hinkelmann, M. Brandon, B. Guang, R. McNeill, G. Blekherman, A. Veliz-Cuba, and R. Laubenbacher. ADAM: analysis of discrete models of biological systems using computer algebra. *BMC Bioinformatics*, 12(295), December 2010.

W. S. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, and W. Fontana. Rules for modeling signal-transduction systems. *Sci STKE*, re6(345), July 2006.

H. H. Hoos and T. Stützle. *Stochastic Local Search*. Foundations and Applications. Morgan Kaufmann, January 2005.

M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, G. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D.

Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, February 2003.

T. Ideker, T. Galitski, and L. Hood. A new approach to decoding life: systems biology. *Annual review of genomics and human genetics*, 2:343–372, 2001.

T. E. Ideker, V. Thorsson, and R. M. Karp. Discovery of regulatory interactions through perturbation: inference and experimental design. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 5, pages 305–316, 2000.

K. Inoue. Logic programming for boolean networks. In T. Walsh, editor, *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 924–930. IJCAI/AAAI, 2011.

R. Kaminski, T. Schaub, A. Siegel, and S. Videla. Minimal intervention strategies in logical signaling networks with answer set programming. *Theory and Practice of Logic Programming*, 13(4-5):675–690, 2013.

M. Kanehisa, S. Goto, M. Furumichi, M. Tanabe, and Mika M. Hirakawa. Kegg for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Research*, 38(Database issue), January 2010.

S. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, February 1969.

H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

S. Klamt. Generalized concept of minimal cut sets in biochemical networks. *Biosystems*, 83 (2-3):233–247, January 2006.

S. Klamt, J. Saez-Rodriguez, and E. Gilles. Structural and functional analysis of cellular networks with CellNetAnalyzer. *BMC systems biology*, 1(2), December 2006a.

S. Klamt, J. Saez-Rodriguez, J. Lindquist, L. Simeoni, and E. Gilles. A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics*, 7(1):56, 2006b.

S. C. Kleene. *Introduction to metamathematics*. Princeton, NJ, 1950.

P. Kohl, E. J. Crampin, T. A. Quinn, and D. Noble. Systems biology: An approach. *Clinical Pharmacology & Therapeutics*, 88(1):25–33, June 2010.

A. Kremling, S. Fischer, K. Gadkar, F. Doyle, T. Sauter, E. Bullinger, F. Allgöwer, and E. Gilles. A benchmark for methods in reverse engineering and model discrimination: problem formulation and solutions. *Genome Research*, 14(9):1773–1785, September 2004.

C. Kreutz and J. Timmer. Systems biology: experimental design. *FEBS Journal*, 276(4):923–942, January 2009.

L. Kuepfer, M. Peter, U. Sauer, and J. Stelling. Ensemble modeling for analysis of cell signaling dynamics. *Nature biotechnology*, 25(9):1001–1006, September 2007.

H. Lähdesmäki, I. Shmulevich, and O. Yli-Harja. On learning gene regulatory networks under the Boolean network model. *Machine learning*, 52(1-2):147–167, 2003.

R. Layek, A. Datta, M. Bittner, and E. R. Dougherty. Cancer therapy design based on pathway logic. *Bioinformatics*, 27(4):548–555, February 2011.

S. Li, S. M. Assmann, and R. Albert. Predicting essential components of signal transduction networks: A dynamic model of guard cell abscisic acid signaling. *PLoS Biology*, 4(10), 2006.

S. Liang, S. Fuhrman, and R. Somogyi. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 3, pages 18–29, 1998.

A. Macnamara, C. Terfve, D. Henriques, B. P. Bernabé, and J. Saez-Rodriguez. State-time spectrum of signal transduction logic models. *Physical biology*, 9(4), August 2012.

D. Marbach, T. Schaffter, C. Mattiussi, and D. Floreano. Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of Computational Biology*, 16(2):229–239, February 2009.

D. Marbach, J. Costello, R. Küffner, N. Vega, R. Prill, D. Camacho, K. Allison, M. Kellis, J. Collins, and G. Stolovitzky. Wisdom of crowds for robust gene network inference. *Nature Methods*, 9 (8):796–804, July 2012.

R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, April 2004.

E. J. McCluskey. Minimization of Boolean functions. *Bell System Technical Journal*, 1956.

T. Melham. Modelling, abstraction, and computation in systems biology: A view from computer science. *Progress in Biophysics and Molecular Biology*, 111(2-3):129–136, 2013.

B. Mélykúti, E. August, A. Papachristodoulou, and H. El-Samad. Discriminating between rival biochemical network models: three approaches to optimal experiment design. *BMC systems biology*, 4, 2010.

L. Mendoza, D. Thieffry, and E. R. Alvarez-Buylla. Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis. *Bioinformatics*, 15(7-8):593–606, June 1999.

R. Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, New York, NY, USA, 1999.

A. Mitsos, I. Melas, P. Siminelakis, A. Chairakaki, J. Saez-Rodriguez, and L. G. Alexopoulos. Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data. *PLoS Computational Biology*, 5(12): e1000591, September 2009.

M. Morris, J. Saez-Rodriguez, P. Sorger, and D. A. Lauffenburger. Logic-based models for the analysis of cell signaling networks. *Biochemistry*, 49(15):3216–3224, 2010.

M. K. Morris, J. Saez-Rodriguez, D. C. Clarke, P. Sorger, and D. A. Lauffenburger. Training signaling pathway maps to biochemical data with constrained fuzzy logic: quantitative analysis of liver cell responses to inflammatory stimuli. *PLoS Computational Biology*, 7(3), March 2011.

C. Mussel, M. Hopfensitz, and H. A. Kestler. BoolNet–an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, 26(10):1378–1380, May 2010.

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, December 2008.

C. J. Needham, J. R. Bradford, A. J. Bulpitt, and D. R. Westhead. A primer on learning in Bayesian networks for computational biology. *PLoS Computational Biology*, 3(8), 2007.

D. Noble. Biophysics and systems biology. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 368(1914):1125–1139, March 2010.

I. Papatheodorou, M. Ziehm, D. Wieser, N. Alic, L. Partridge, and J. M. Thornton. Using Answer Set Programming to integrate RNA expression with signalling pathway information to infer how mutations affect ageing. *PLoS ONE*, 7(12), December 2012.

J. A. Papin, T. Hunter, B. Ø. Palsson, and S. Subramaniam. Reconstruction of cellular signalling networks and analysis of their properties. *Nature Reviews Molecular Cell Biology*, 6(2): 99–111, February 2005.

L. Paulevé and A. Richard. Static analysis of boolean networks based on interaction graphs: A survey. *Electronic Notes in Theoretical Computer Science*, 284:93–104, June 2012.

C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, December 2000.

R. J. Prill, J. Saez-Rodriguez, L. G. Alexopoulos, P. K. Sorger, and G. Stolovitzky. Crowdsourcing network inference: the DREAM predictive signaling network challenge. *Sci Signal*, 4(189): mr7, September 2011.

O. Ray, K. Whelan, and R. King. Logic-based steady-state analysis and revision of metabolic networks with inhibition. In L. Barolli, F. Xhafa, S. Vitabile, and H. Hsu, editors, *Proceedings of the Fourth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'10)*, pages 661–666. IEEE Computer Society, 2010.

O. Ray, T. Soh, and K. Inoue. Analyzing pathways using asp-based approaches. In K. Horimoto, M. Nakatsui, and N. Popov, editors, *Proceedings of the Fourth International Conference on Algebraic and Numeric Biology (ANB'10)*, volume 6479 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 2012.

A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470. World Scientific Press, 2001.

A. Réka and R. Wang. Discrete dynamic modeling of cellular signaling networks. *Methods in Enzymology*, 467:281–306, December 2008.

E. Remy, P. Ruet, and D. Thieffry. Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335–350, 2008.

A. Saadatpour and A. Réka. Boolean modeling of biological regulatory networks: A methodology tutorial. *Methods*, 62(1):3–12, 2013.

A. Saadatpour, R. Wang, A. Liao, X. Liu, T. P. Loughran, I. Albert, and R. Albert. Dynamical and structural analysis of a t cell survival network identifies novel candidate therapeutic targets for large granular lymphocyte leukemia. *PLoS Computational Biology*, 7(11), October 2011.

K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005.

J. Saez-Rodriguez, L. Simeoni, J. Lindquist, R. Hemenway, U. Bommhardt, B. Arndt, U. Haus, R. Weismantel, E. Gilles, S. Klamt, and B. Schraven. A logical model provides insights into t cell receptor signaling. *PLOS Computational Biology*, 3(8), August 2007.

J. Saez-Rodriguez, A. Goldsipe, J. Muhlich, L. G. Alexopoulos, B. Millard, D. A. Lauffenburger, and P. Sorger. Flexible informatics for linking experimental data to mathematical models via DataRail. *Bioinformatics*, 24(6):840–847, March 2008.

J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. Sorger. Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. *Molecular Systems Biology*, 5(331), 2009.

J. Saez-Rodriguez, L. G. Alexopoulos, M. Zhang, M. Morris, D. A. Lauffenburger, and P. Sorger. Comparing signaling networks between normal and transformed hepatocytes using discrete logical models. *Cancer Research*, 71(16), 2011.

R. Samaga and S. Klamt. Modeling approaches for qualitative and semi-quantitative analysis of cellular signaling networks. *Cell communication and signaling*, 11(1):43, 2013.

R. Samaga, J. Saez-Rodriguez, L. G. Alexopoulos, P. Sorger, and S. Klamt. The logic of EGFR/ErbB signaling: theoretical properties and analysis of high-throughput data. *PLoS Computational Biology*, 5(8), August 2009.

R. Samaga, A. Von Kamp, and S. Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. *Journal of Computational Biology*, 17(1):39–53, January 2010.

L. Sánchez, C. Chaouiya, and D. Thieffry. Segmenting the fly embryo: logical analysis of the role of the segment polarity cross-regulatory module. *International journal of developmental biology*, 52(8):1059–1075, 2008.

C. F. Schaefer, K. Anthony, S. Krupa, J. Buchoff, M. Day, T. Hannay, and K. H. Buetow. Pid: the pathway interaction database. *Nucleic Acids Research*, 37(Database issue):D674–D679, 2009.

T. Schaub and S. Thiele. Metabolic network expansion with ASP. In P. Hill and D. Warren, editors, *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*, pages 312–326. Springer-Verlag, 2009.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

J. A. Shapiro. Revisiting the central dogma in the 21st century. *Annals of the New York Academy of Sciences*, 1178:6–28, October 2009.

R. Sharan and R. M. Karp. Reconstructing Boolean models of signaling. *Journal of Computational Biology*, 20(3):249–257, March 2013.

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, January 2002a.

I. Shmulevich, E. R. Dougherty, and W. Zhang. Gene perturbation and intervention in probabilistic Boolean networks. *Bioinformatics*, 18(10):1319–1331, October 2002b.

S. Soliman. Invariants and other structural properties of biochemical models as a constraint satisfaction problem. *Algorithms for Molecular Biology*, 7(15), 2011.

A. Sparkes, W. Aubrey, E. Byrne, A. Clare, M. N. Khan, M. Liakata, M. Markham, J. Rowland, L. N. Soldatova, K. E. Whelan, M. Young, and R. D. King. Towards robot scientists for autonomous scientific discovery. *Automated Experimentation*, 2, January 2010.

J. Stegmaier, D. Skanda, and D. Lebiedz. Robust optimal design of experiments for model discrimination using an interactive software tool. *PLoS ONE*, 8(2), 2013.

J. Stelling, U. Sauer, Z. Szallasi, F. Doyle, and J. Doyle. Robustness of cellular functions. *Cell*, 118(6):675–685, 2004.

G. Stolovitzky, D. Monroe, and A. Califano. Dialogue on reverse-engineering assessment and methods: the dream of high-throughput pathway inference. *Annals of the New York Academy of Sciences*, 1115:1–22, December 2007.

E. Szczurek, I. Gat-Viks, J. Tiuryn, and M. Vingron. Elucidating regulatory mechanisms downstream of a signaling pathway using informative experiments. *Molecular Systems Biology*, 5: 287–287, December 2008.

C. D. A. Terfve, T. Cokelaer, D. Henriques, A. Macnamara, E. Gonçalves, M. Morris, M. van Iersel, D. A. Lauffenburger, and J. Saez-Rodriguez. CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. *BMC systems biology*, 6 (1), October 2012.

R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563–585, November 1973.

R. Thomas. Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*, 153(1):1–23, 1991.

I. Vatcheva, H. de Jong, O. Bernard, and N. J. I. Mars. Experiment selection for the discrimination of semi-quantitative models of dynamical systems. *Artificial Intelligence*, 170(4): 472–506, December 2005.

S. Videla, C. Guziolowski, F. Eduati, S. Thiele, N. Grabe, J. Saez-Rodriguez, and A. Siegel. Revisiting the training of logic models of protein signaling networks with ASP. In D. Gilbert and M. Heiner, editors, *Proceedings of the Tenth International Conference on Computational Methods in Systems Biology (CMSB'12)*, volume 7605 of *Lecture Notes in Computer Science*, pages 342–361. Springer-Verlag, 2012.

R. Wang and R. Albert. Elementary signaling modes predict the essentiality of signal transduction network components. *BMC systems biology*, 5:44, 2011.

R. Wang, A. Saadatpour, and R. Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Physical biology*, 9(5), September 2012.

D. M. Wittmann, J. Krumsiek, J. Saez-Rodriguez, D. A. Lauffenburger, S. Klamt, and F. J. Theis. Transforming Boolean models to continuous models: methodology and application to T-cell receptor signaling. *BMC systems biology*, 3:98, 2009.

C. Yeang, H. C. Mak, S. McCuine, C. Workman, T. Jaakkola, and T. E. Ideker. Validation and refinement of gene-regulatory pathways on a network of physical interactions. *Genome biology*, 6(7), 2005.

J. Zheng, D. Zhang, P.F. Przytycki, R. Zielinski, J. Capala, and T. M. Przytycka. SimBoolNet–a Cytoscape plugin for dynamic simulation of signaling networks. *Bioinformatics*, 26(1): 141–142, January 2010.

Santiago Videla

PERSONAL DETAILS
date of birth: June 14th 1983
place of birth: Buenos Aires, Argentina
email: santiago.videla@gmail.com

EDUCATION

- *Licenciatura* en Ciencias de la Computación (equivalent to MSc) December 2010
Facultad de Matemática, Astronomía y Física - Univ. Nacional de Córdoba. Argentina.
GPA: 9.14 / 10
- *Analista* en Computación (equivalent to BSc) July 2008
Facultad de Matemática, Astronomía y Física - Univ. Nacional de Córdoba. Argentina
GPA: 9.08 / 10

RESEARCH EXPERIENCE

PhD thesis September 2011 - August 2014
University of Rennes 1, France & University of Potsdam, Germany
Reasoning on the response of logical signaling networks with Answer Set Programming.
Advisor: A. Siegel. Dyliss Team, CNRS - IRISA & Inria Rennes, France.
Co-Advisor: T. Schaub. Knowledge Processing and Information Systems, Potsdam University, Germany.

MSc thesis April 2010 - December 2010
Universidad Nacional de Córdoba. Argentina.
Design of attenuated vaccines with minor likelihood of revert to virulence.
Advisor: L. Alonso i Alemany. Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba. Argentina.

PUBLICATIONS

Schaub T., Siegel A., Videla S. (2014). *Reasoning on the response of logical signaling networks with Answer Set Programming.* Luis Fariñas del Cerro and Katsumi Inoue (Eds.): Logical Modeling of Biological Systems. John Wiley & Sons, Inc., 49-92. DOI: 10.1002/9781119005223.ch2

Videla S., Guziolowski C., Eduati F., Thiele S., Gebser M, Nicolas J., Saez-Rodriguez J., Schaub T., Siegel A. (2014). *Learning Boolean logic models of signaling networks with Answer Set Programming.* Theoretical Computer Science, in press. DOI: 10.1016/j.tcs.2014.06.022

Kaminski R., Schaub T, Siegel A and Videla S. (2013). *Minimal Intervention Strategies in Logical Signaling Networks with Answer Set Programming.* Theory and Practice of Logic Programming, 13(4-5), 675-690. DOI: 10.1017/S1471068413000422

Guziolowski C. and Videla S. and Eduati F. and Thiele S. and Cokelaer T. and Siegel A. and Saez-Rodriguez J. (2013). *Exhaustively characterizing feasible logic models of a signaling network using Answer Set Programming.* Bioinformatics, 29(18), 2320-2326. DOI: 10.1093/bioinformatics/btt393

Videla S., Guziolowski C., Eduati F., Thiele S., Grabe N., Saez-Rodriguez J. and Siegel A. (2012). *Revisiting the Training of Logic Models of Protein Signaling Networks with*

a Formal Approach Based on Answer Set Programming. D. Gilbert and M. Heiner (Eds.): 10th International Conference on Computational Methods in Systems Biology. LNCS 7605, pp. 342-361. Springer, Heidelberg. DOI: 10.1007/978-3-642-33636-2_20

AWARDS

Best Paper at "The 10th Conference on Computational Methods in Systems Biology, 3-5 October 2012, The Royal Society, London, UK". Videla S., Guziolowski C., Eduati F., Thiele S., Grabe N., Saez-Rodriguez J. and Siegel A. *Revisiting the Training of Logic Models of Protein Signaling Networks with a Formal Approach based on Answer Set Programming.*

TEACHING EXPERIENCE

Answer Set Programming for Systems Biology (graduate seminar) November 2013
Center of Mathematical Modeling, University of Chile, Chile.

Teacher Assistant March 2006 - March 2007 / March 2010 - March 2011
Facultad de Matemática, Astronomía y Física - Univ. Nacional de Córdoba. Argentina.

- Introduction to Algorithms
- Algorithms and Data Structures I

INDUSTRY EXPERIENCE

Software Engineer January 2011 - July 2011
Intel. Web applications development at the Argentina Software Development Center.

- HTML5, Javascript, CSS3

Entrepreneur June 2007 - July 2009
Development of web sites and applications using free software technologies:

- Plone CMS, Zope/Grok, Django, PostgreSQL, ExtJS

Junior Programmer March 2007 - June 2007
Santex America. Development and support of web sites.

- PHP, OsCommerce, MySQL, HTML, CSS, Javascript

ATTENDANCE TO SCHOOLS AND SEMINARS

- Summer School in Formal Modeling of Biological Regulatory Networks. June 2013. Île de Porquerolles, France.
- Latin American eScience Workshop 2013. May 2013. São Paulo, Brazil.
- The São Paulo School of Advanced Science on e-Science for Bioenergy Research. October 2012. Campinas, Brazil.
- 2012 International Summer School in Methods in Bioinformatics. August 2012. Tarragona, Spain.
- 30^{th} Scientific Annual Meeting of the Argentine Society of Virology. December 2010. Córdoba, Argentina.
- 4^{th} School of Biology and Mathematics. August 2010. Córdoba, Argentina.
- Testing on Web Applications. September 2009. INTI. Córdoba, Argentina.
- Testing as part of the Quality Assurance. May 2009. INTI. Córdoba, Argentina.

SOFTWARE CONTRIBUTIONS

Contributions to the Python community (main developer):

- caspo
- pyzcasp
- extdirect.django
- hurry.extjs
- megrok.jinja

REFERENCES

- Dr. Anne Siegel (France): anne.siegel@irisa.fr
- Dr. Torsten Schaub (Germany): torsten@cs.uni-potsdam.de
- Dr. Laura Alonso i Alemany (Argentina): alemany@famaf.unc.edu.ar

Abstract: Deciphering the functioning of biological networks is one of the central tasks in systems biology. In particular, signal transduction networks are crucial for the understanding of the cellular response to external and internal perturbations. Importantly, in order to cope with the complexity of these networks, mathematical and computational modeling is required. We propose a computational modeling framework in order to achieve more robust discoveries in the context of logical signaling networks. More precisely, we focus on modeling the response of logical signaling networks by means of automated reasoning using Answer Set Programming (ASP). ASP provides a declarative language for modeling various knowledge representation and reasoning problems. Moreover, available ASP solvers provide several reasoning modes for assessing the multitude of answer sets. Therefore, leveraging its rich modeling language and its highly efficient solving capacities, we use ASP to address three challenging problems in the context of logical signaling networks: learning of (Boolean) logical networks, experimental design, and identification of intervention strategies. Overall, the contribution of this thesis is three-fold. Firstly, we introduce a mathematical framework for characterizing and reasoning on the response of logical signaling networks. Secondly, we contribute to a growing list of successful applications of ASP in systems biology. Thirdly, we present a software providing a complete pipeline for automated reasoning on the response of logical signaling networks.

Key words: systems biology, logical signaling networks, answer set programming

---

Résumé : Décrypter le fonctionnement des réseaux biologiques est une des missions centrales de la biologie des systèmes. En particulier, les réseaux de transduction du signal sont essentiels pour la compréhension de la réponse cellulaire à des perturbations externes ou internes. Pour faire face à la complexité de ces réseaux, des modélisations aussi bien numériques que formelles sont nécessaires. Nous proposons un cadre de modélisation formelle, dans le cadre de réseaux logiques, afin d'obtenir des prédictions robustes sur le comportement et le contrôle des voies de signalisation. Nous modélisons la réponse des réseaux logiques de signalisation par du raisonnement automatique à l'aide de Programmation par Ensembles-Réponses (Answer Set Programming, ASP). ASP fournit un langage déclaratif pour la modélisation de divers problèmes de représentation des connaissances et de raisonnement. Des solveurs permettent plusieurs modes de raisonnement pour étudier la multitude d'ensembles réponses. En s'appuyant sur la richesse du langage de modélisation et ses capacités de résolution très efficaces, nous utilisons ASP pour modéliser et résoudre trois problèmes dans le contexte des réseaux logiques de signalisation : apprentissage de réseaux booléens, calculs de plan d'expériences, et identification des contrôleurs. Globalement, la contribution de cette thèse est de trois ordres. Premièrement, nous introduisons un cadre formel pour la caractérisation et le raisonnement sur la réponse des réseaux logiques de signalisation. Deuxièmement, nous contribuons à une liste croissante d'applications réussies d'ASP en biologie des systèmes. Troisièmement, nous présentons un logiciel fournissant un pipeline complet de raisonnement automatisé sur la réponse des réseaux logiques de signalisation.

Mots clefs : biologie des systèmes, réseaux de signalisation logiques, *answer set programming*