Dissertation

MODELING BIOLOGICAL SYSTEMS WITH ANSWER SET PROGRAMMING

vorgelegt von

Dipl.-Inf. Sven Thiele

zur Erlangung des akademischen Grades
“doctor rerum naturalium” (Dr. rer. nat.)

in der Wissenschaftsdisziplin
“Wissensverarabeitung und Informationssysteme”

Universität Potsdam

Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik
Professur für Wissensverarbeitung und Informationssysteme

This work is licensed under a Creative Commons License:
Attribution 3.0 Germany
To view a copy of this license visit
http://creativecommons.org/licenses/by/3.0/de/

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2012/5938/
URN urn:nbn:de:kobv:517-opus-59383
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-59383

Abstract

Biology has made great progress in identifying and measuring the building blocks of life. The availability of high-throughput methods in molecular biology has dramatically accelerated the growth of biological knowledge for various organisms. The advancements in genomic, proteomic and metabolomic technologies allow for constructing complex models of biological systems. An increasing number of biological repositories is available on the web, incorporating thousands of biochemical reactions and genetic regulations.

Systems Biology is a recent research trend in life science, which fosters a systemic view on biology. In Systems Biology one is interested in integrating the knowledge from all these different sources into models that capture the interaction of these entities. By studying these models one wants to understand the emerging properties of the whole system, such as robustness.

However, both measurements as well as biological networks are prone to considerable incompleteness, heterogeneity and mutual inconsistency, which makes it highly nontrivial to draw biologically meaningful conclusions in an automated way. Therefore, we want to promote Answer Set Programming (ASP) as a tool for discrete modeling in Systems Biology. ASP is a declarative problem solving paradigm, in which a problem is encoded as a logic program such that its answer sets represent solutions to the problem. ASP has intrinsic features to cope with incompleteness, offers a rich modeling language and highly efficient solving technology.

We present ASP solutions, for the analysis of genetic regulatory networks, determining consistency with observed measurements and identifying minimal causes for inconsistency. We extend this approach for computing minimal repairs on model and data that restore consistency. This method allows for predicting unobserved data even in case of inconsistency.

Further, we present an ASP approach to metabolic network expansion. This approach exploits the easy characterization of reachability in ASP and its various reasoning methods, to explore the biosynthetic capabilities of metabolic reaction networks and generate hypotheses for extending the network.

Finally, we present the BioASP library, a Python library which encapsulates our ASP solutions into the imperative programming paradigm. The library allows for an easy integration of ASP solution into system rich environments, as they exist in Systems Biology.

Acknowledgements

Time passes by so quickly, it has been almost four years since I started my doctoral studies. In this time I have met many people who have influenced and supported me. They helped me grow as person and researcher. Here is the place to thank them.

Many thanks go to Torsten Schaub, for his supervision, advice and guidance over the years. He mostly inspired me with his passion for science, the appreciation of quality work and his desire to contribute in the scientific community.

Moreover, I would like to thank all the fellow researchers, who supported me in my work. Without them this thesis would not be possible. I thank Martin Gebser, his genius often helped me when I felt stuck. Thank you to Benjamin Kaufman and Roland Kaminski, their coding skills created the tools that made my projects possible. Thanks to Max Ostrowski, who often helped me see problems from an different perspective. Thanks also to Pierre Blavy, Nils Christian, Zoran Nikoloski, Anne Siegel and Björn Usadel, for their support and insights to various biological issues. Many thanks in particular to Sylvain Blachon, Carito Guziolowski and Philippe Veber, who worked tightly with me and answered even my stupidest questions with the utmost patience.

Thanks to all my colleagues at the institute: Benjamin Andres, Steve Dworschak, Arne König, Orkunt Sabuncu and Marius Schneider, they created a pleasant and inspiring work environment. Special thanks go to the Coffee-Break-Gang, for all the intellectually demanding discussions. I will miss our walks.

Last but not least, I thank my family and friends. They are my foundation, the safe harbor where I could turn to in difficult times and replenish my strength. Thank you Dunja, Jule, Lars, Luise, Neo, Roman and Uli.

Certainly there are many more people to whom I owe thanks. Thank you all for accompanying me on my way.

The work in this thesis has been partially funded by the Federal Ministry for Education and Research (BMBF) within the GoForsys Project.

Contents

1 Introduction **1**
1.1 Systems Biology 1
1.2 Answer Set Programming 2
1.3 Scientific Contributions 3
1.4 Organisation of This Thesis 4

2 Regulatory Networks as Sign Consistency Models **7**
2.1 Biological Background 8
2.2 Mathematical Formalism 8
2.3 Checking Consistency 10
2.3.1 Problem Instance 11
2.3.2 Generating Solution Candidates 12
2.3.3 Testing Solution Candidates 12
2.3.4 Soundness and Completeness 13
2.3.5 Input Reduction 13
2.4 Identifying Minimal Inconsistent Cores 15
2.4.1 Generating MIC Candidates 16
2.4.2 Testing for Inconsistency 17
2.4.3 Testing for Minimality 18
2.4.4 Soundness and Completeness 18
2.4.5 Exploiting Strongly Connected Components for MIC Extraction . 19
2.5 Repair 21
2.5.1 Problem Instance 22
2.5.2 Repair Operations 23
2.5.3 Repair Encoding 24
2.5.4 Minimal Repairs 26
2.6 Prediction under Repairs 27
2.7 Empirical Evaluation and Application 27
2.7.1 Checking Consistency 27
2.7.2 Minimal Inconsistent Cores 29
2.7.3 Biological Case Study 29
2.7.4 Repair 32
2.7.5 Prediction under Repair 34
2.8 Discussion 37

3 Metabolic Network Expansion **39**
3.1 Biological Background 40
3.2 Mathematical Formalism 40
3.3 Metabolic network completion 43

3.3.1 Problem Instance . . . 43
3.3.2 Scope and Potential Scope . . . 44
3.3.3 Metabolic Network Completion. . . . 45
3.3.4 Refined Metabolic Network Completion. . . . 45
3.3.5 Reasoning Modes. . . . 47
3.4 Inverse Scope Problem . . . 47
3.4.1 Basic Setting . . . 48
3.4.2 Refined Setting . . . 48
3.4.3 Avoiding Side or Seed Metabolites . . . 49
3.4.4 Reasoning Modes . . . 49
3.5 Empirical Evaluation . . . 49
3.5.1 Metabolic network completion . . . 50
3.5.2 Inverse scope problem . . . 52
3.6 Discussion . . . 53

4 The BioASP Library 57
4.1 System Architecture . . . 58
4.1.1 The *asp* module . . . 58
4.1.2 The *data* module . . . 61
4.1.3 The *query* module . . . 63
4.2 Applications . . . 68
4.2.1 Diagnosing and Repairing on Gene Regulatory Networks . . . 68
4.2.2 Metabolic Network Expansion . . . 72
4.2.3 Web Service . . . 75
4.3 Discussion . . . 77

5 Conclusions 79

A Proofs 81
A.1 Proof of Theorem 2.1 and 2.2 . . . 81
A.2 Proof of Theorem 2.4 and 2.5 . . . 84
A.3 Proof of Theorem 3.1 . . . 91

List of Figures 95

List of Tables 97

Bibliography 99

1 Introduction

This thesis investigates the application of the declarative problem solving paradigm of Answer Set Programming (ASP) to the field of Systems Biology. In this chapter, we give a short introductions to both fields, Systems Biology as well as ASP. We present the scientific contribution of this thesis. Finally, an overview on the organization of the thesis is given.

1.1 Systems Biology

This section provides a brief introduction to Systems Biology [71, 76], a recent research trend in life science, which fosters a systemic view on biology. Biology has made great progress in identifying and measuring the building blocks of life. The advancements in genomic, proteomic and metabolomic technologies allow for constructing complex models of biological systems. Systems Biology is interested in integrating the knowledge from all these different sources into models that capture the interaction of these entities. By studying these models one wants to understand the emerging properties of the whole system, such as robustness. The research in Systems Biology is mainly hypothesis driven. It starts with a biological question and the creation of a model representing the phenomenon. The creation of the model can be done automatically by data driven methods or by hand. The model represents a set of assumptions and hypotheses that can be tested "in silico" via simulations, or by confrontation with experimental results. Inaccurate models will be revealed by inconsistent behavior that contradicts known biological facts or experimental results. Inconsistent models need to be modified and the consistent models can be used for predictions. These predictions can be used to design further "in vivo" experiments, which again are used for testing the the model. The cycle of modeling, *in silico* testing, prediction, *in vivo* verification and further refinement of the model leads gradually to biological models that are evermore correct.

The availability of high-throughput methods in molecular biology has dramatically accelerated the growth of biological knowledge for various organisms. These methods allow for gathering multiple orders of magnitude more measured data than was procurable before. Furthermore, there is an increasing number of biological repositories on the web, such as KEGG [75], Biomodels [80], Reactome [74], MetaCyc [17] and others, incorporating thousands of biochemical reactions and genetic regulations.

In Systems Biology Ordinary Differential Equations (ODEs) are a widely used technique for quantitative modeling. Here, a system is presented as a set of time-dependent functions whose values represent concentrations of the systems entities like genes, proteins and metabolites. Such a system of ODEs allows the smooth simulation of the system under a given parameter set. But ODE approaches rely on the availability of production rates and kinetic parameters, which for many biological systems are unknown. Measurements as well as biological models are prone to considerable incompleteness, hetero-

geneity and mutual inconsistency, which makes it highly non-trivial to draw biologically meaningful conclusions in an automated way. This is where qualitative modeling methods come into play, they have the advantage that they do not rely on detailed quantitative information, but still they allow to study qualitative properties of the system.

There exist several approaches to qualitative modeling in Systems Biology. *Petri Nets* are used to model biological pathways, genetic regulations, metabolic and signaling networks [68, 67, 8]. Petri Nets are a natural representation to describe networks that conveys the intuitive understanding of biochemical reactions. They allow the modeling of concurrent reactions, which collaborate in the synthesis of compounds or compete for resources. Petri Nets can be used for simulating the behavior of large networks, which allows to gain statistical results from the model. On smaller models, the Petri Net approach can be used to compute system invariant properties. For example, *t-invariants* can be used to identify sub networks, which describe basic systems behaviors.

Furthermore, there exists a variety of formal methods that are used for the discrete modeling of different aspects of biological systems. Flux Balance Analysis [14] allows to determine the flux rates in a metabolic network such that a biological relevant variable like the growth rate is maximized. Temporal logics [37] and Timed Automata [2] allow the modeling of temporal aspects in biological systems. With the Process Calculus [91] one can model systems of concurrently working agents that interact, communicate and synchronize with each other. These methods allow for high level descriptions to model specific problems. For computation these high level descriptions are often mapped into well established constraint solving techniques that offer highly efficient solving engines, like SAT [10], Constraint Programming [94], Integer Linear Programming [98] and ASP [4]. Other approaches use these techniques directly to encode of biological problems like haplotype inference [85, 35], protein structure prediction [3] and the analysis of gene regulatory as well as metabolic networks [22, 102]. While the direct encoding offers more flexibility for expressing system properties, it also requires a deeper insight into the used formalism. In this work, we will focus on ASP for the modeling of biological problems.

1.2 Answer Set Programming

This section provides a brief introduction to Answer Set Programming (ASP) [4], a declarative problem solving paradigm from the field of logic programming and knowledge representation. ASP offers a rich modeling language [107, 44] along with highly efficient inference engines [81, 114, 82, 48, 28] based on Boolean constraint solving technology [56, 46, 28]. ASP has been used in many application areas, such as planning [26, 84], model checking [69], hardware design [33], product configuration [101], music composition [12] and bio-informatics [109, 110, 108, 29, 30, 35].

The basic idea of ASP is to encode a problem as a logic program such that its answer sets represent solutions to the problem. In ASP, a problem encoding is a set of logic programming rules including universally quantified first order variables. In a first step, a grounder [59] transforms the logic program into an equivalent propositional logic program. In a second step, the propositional program is processed by an answer set solver, which searches for answer sets. ASP allows for solving search problems from the com-

plexity class NP and with the use of disjunctive logic programs from the class Σ_2^P. In view of our applications, we take advantage of the elevated expressiveness of disjunctive programs, capturing problems at the second level of the polynomial hierarchy [32]. A *disjunctive logic program* P is a finite set of *rules* of the form

$$a_1;\ldots;a_l \leftarrow a_{l+1},\ldots,a_m, not\ a_{m+1},\ldots, not\ a_n\ , \tag{1.1}$$

where a_i is an *atom* for $1 \leq i \leq n$. A rule r as in (1.1) is called a *fact* if $l = m = n = 1$, and an *integrity constraint* if $l = 0$. Let $head(r) = \{a_1,\ldots,a_l\}$ be the *head* of r, $body(r) = \{a_{l+1},\ldots,a_m, not\ a_{m+1},\ldots, not\ a_n\}$ be the *body* of r, as well let $body(r)^+ = \{a_{l+1},\ldots,a_m\}$ and $body(r)^- = \{a_{m+1},\ldots,a_n\}$.

An interpretation is represented by the set of atoms that are true in it. A *model* of a program P is an interpretation in which all rules of P are true according to the standard definition of truth in propositional logic. Apart from letting ';' and ',' stand for disjunction and conjunction, respectively, this implies treating rules and default negation 'not' as implications and classical negation, respectively. Note that the (empty) head of an integrity constraint is false in every interpretation, while the empty body is true in every interpretation. Answer sets of P are particular models of P satisfying an additional stability criterion. Roughly, a set X of atoms is an answer set, if for every rule of form (1.1), X contains a minimum of atoms among $a_1,\ldots,a_l$ whenever $a_{l+1},\ldots,a_m$ belong to X and no $a_{m+1},\ldots,a_n$ belongs to X. However, the disjunction in heads of rules, in general, is not exclusive. Formally, an *answer set* X of a program P is a $\subseteq$-minimal model of

$$\{head(r) \leftarrow body(r)^+ \mid r \in P, body(r)^- \cap X = \emptyset\}\ .$$

For example, program $\{a;b \leftarrow .\ c;d \leftarrow a, not\ b.\ \leftarrow b.\}$ has answer sets $\{a,c\}$ and $\{a,d\}$.

Although answer sets are usually defined on ground (i.e., variable-free) programs, ASP allows for non-ground problem encodings, where schematic rules stand for their ground instantiations. Grounders, such as *gringo* [44] and *lparse* [107], are capable of combining a problem encoding and an instance (typically a set of ground facts) into an equivalent ground program, which is then processed by an ASP solver. We follow this methodology and provide encodings for the problems considered in this thesis.

1.3 Scientific Contributions

We want to promote ASP with its intrinsic incompleteness-tolerating capacities and its rich modeling language as an appropriate tool to cope with these issues. Therefore, we present ASP approaches addressing problems from the field of Systems Biology, in particular the analysis of gene regulatory and metabolic networks. Systems Biology is a complex research field, where many different modeling approaches find its application. To foster the use of ASP within such an environment it must be easily integratable with existing applications. To this end, we present the BioASP library, which provides a generic way to encapsulate ASP solution into the imperative programming paradigm of Python [111] and allows for an easy integration with existing applications.

In summary, the main contributions of this thesis are:

1. We provide an ASP framework for the analysis of gene regulatory networks in-

cluding consistency checking, diagnosis, prediction, minimal repair and prediction under inconsistency.

2. We present an ASP approach to metabolic network expansion that represents the bio-synthetic capabilities of metabolic networks and allows for the hypothesis generation of possible extensions.

3. We provide the BioASP Library, a Python library that encapsulates the ASP tools and allows for an easy integration of ASP solution into standard Python functions.

The work on genetic regulatory networks has been conducted jointly with Philippe Veber, Martin Gebser and Carito Guiziolowski. Philippe introduced us to the topic of consistency checking on genetic regulatory networks and provided invaluable biological insight, Martin is responsible for the disjunctive encoding and the corresponding proofs and Carito provided us with the biological data and her view on the concept of repairs. The work on metabolic network expansion is inspired by the work of Nils Chistian who investigates a stochastic approach to metabolic network expansion and provided us with sample data. Finally, the BioASP library emerged out of a work started by Philippe Veber and is today hosted as an open source project [11]. Further contributions were made by Peter Schüller and Arne König, who wrote a web service based on the BioASP library.

Part of the results presented in this thesis have been published in [54, 51, 41, 96, 53], coauthored by the author of this thesis.

Further, the author has contributed to research beyond the work that is presented here. These contributions are mainly in the field of ASP including subjects, such as the ground instantiation of logic programs [52, 42, 45], incremental ASP solving [43], multithreaded ASP solving [57, 58] and belief revision [24, 23].

1.4 Organisation of This Thesis

In Chapter 2, we provide an ASP framework for the analysis of genetic regulatory networks. We begin with giving a brief introduction into the biological background of genetic regulation. Then, we give a mathematical formalization for representing genetic regulatory networks as influence graphs and define a consistency notion for influence graphs. We develop an ASP formulation for checking the consistency and extend this approach to identifying minimal representations of conflicts. Further, we extend the approach and propose a framework for repairing genetic regulation networks and corresponding measurements. We then use the repair framework to allow for predicting unobserved variations even from inconsistent models and data. Finally, we present results of an empirical evaluation of our approach along with a case study on yeast.

Chapter 3 presents an ASP approach to metabolic network expansion. To begin with, we give an brief introduction to the topic of metabolism and metabolic reactions continued by a mathematical formal representation of metabolic networks and the metabolic network completion problem. We then develop an ASP formulation for solving the metabolic network completion problem and extend this solution for solving the inverse scope problem. Finally, we present the results of our empirical evaluation and conclude the chapter with a discussion.

In Chapter 4, we present the BioASP library. We outline the system architecture of the BioASP library and give detailed descriptions of the functionalities provided by the application programming interface (API). Further, we then present applications that are built on top of the BioASP library. We show how the library is applied to solve the specific biological problems described in the previous chapters.

Finally, we conclude this thesis in Chapter 5. Proofs of formal results are given in the Appendix.

2 Regulatory Networks as Sign Consistency Models

In this chapter, we deal with the analysis of high-throughput measurements in molecular biology, like microarray data or metabolic profiles. Up to now, it is still common practice to use expression profiles merely for detecting over- or under-expressed genes under specific conditions, leaving the task of making biological sense out of a multitude of gene identifiers to human experts. However, many efforts have also been made to better utilize high-throughput data, in particular, by integrating them into large-scale models of transcriptional regulations or metabolic processes [38, 77].

One possible approach consists of investigating the compatibility between experimental measurements and knowledge available in reaction databases. This can be done by using formal frameworks, for instance, the ones developed in [61] and [99]. In what follows, we rely upon the so-called *Sign Consistency Model* (SCM) due to [99]. SCM imposes constraints between experimental measurements and a graph representation of cellular interactions, called an *influence graph* [103]. Such a graph provides an over-approximation of the actual biological model, where an “influence” is modeled by a causal rule. This methodology is particularly well-suited for dealing with incomplete knowledge (missing reactions, lacking kinetic details) or unreliable (noisy data) information.

Building on the SCM framework, we develop declarative techniques based on ASP to detect and explain inconsistencies in large data sets, propose repairs for models and data, and compute predictions under minimal repair.

The framework we develop is configurable, so that biological experts may selectively investigate critical parts of biological networks and/or measurements. Apart from modeling the aforementioned biological problems in ASP, our major concern lies with the scalability of the approach. To this end, we designed a suite of artificial yet biologically meaningful benchmarks indicating that an ASP-based approach scales well on the considered class of applications and applied our methods to the real-world gene-regulatory networks and experimentally derived data sets of yeast and *Escherichia coli*. Notably, to the best of our knowledge, the functionalities we provide go beyond the ones of the only comparable approach [63]. To begin with, we give a brief introduction into the biological background of genetic regulation, the transcription of genes, activator and repressors in Section 2.1. Section 2.2 gives the mathematical formalization for representation of genetic regulatory networks as influence graphs and defines consistency for such an influence graph. In Section 2.3, we develop an ASP formulation for checking the consistency between experimental profiles and influence graphs. We further extend this approach in Section 2.4 to identifying minimal representations of conflicts if the experimental data is inconsistent with an influence graph. Also, we describe a connectivity property that is used to further refine the presented encoding. Section 2.5, extends our basic approach and proposes a framework for repairing genetic regulation networks and corresponding measurements. We discuss the interest of different repair operations wrt several criteria:

biological meaning, minimality measures and computational cost. In Section 2.6 we use the repair framework to allow for predicting unobserved variations even from inconsistent models and data. Section 2.7 is dedicated to an empirical evaluation of our approach along with an exemplary case study on yeast. We evaluate the effect of different repair operations, both quantitatively and qualitatively, by considering the well-studied organism *Escherichia coli* along with published experimental data. Finally, Section 2.8 concludes the chapter.

2.1 Biological Background

Gene expression is the process by which information from a DNA sequence is used to synthesize functional gene products, the proteins. Regulation of gene expression gives a cell control over its structure and function. It is the basis for cellular differentiation, versatility and adaptabillity. The process of gene expression has two phases, transcription and translation. In the transcription phase, RNA-polymerase attaches to the DNA sequence to create a complementary mRNA copy of the gene. In the translation phase, this mRNA is decoded by a ribosome to synthesize functional structures, the proteins. The control which genes should be turned *on* or *off* is executed by *transcription factors* (TFs). One way to regulate a genes expression is the *initiation of transcription*, here a molecule RNA-polymerase attaches to the promotor region of the DNA sequence (gene) and begins transcription along all the DNA strand. The RNA-polymerase transcription can be regulated by TFs that are proteins or protein-complexes. They can be activators, which enhance the interaction between RNA and a particular promotor, encouraging the expression of the gene, or repressors, which bind to non-coding sequences of the DNA strand, impeding the progress of RNA-polymerase along the strand, thus, impeding the expression of the gene. Such regulatory processes can be presented as a *regulatory network*.

The technological revolution of the last years established high-throughput methods that allow to measure simultaneously the mRNA concentration of thousands of genes. Furthermore, an increasing number of web data bases allow access to thousands of biochemical reactions and genetic regulations in various organisms. This motivated the development of new methods to exploit these vast amounts of observations and confront it with the available knowledge, to gain deeper insight into the genetic controls of a cell.

2.2 Mathematical Formalism

A regulatory network is traditionally described as a system of differential equations of the form $\frac{dX}{dt} = F(X, P)$, where X is the state vector of concentrations for the constituents (mRNAs, proteins, or metabolites), and P denotes a set of control parameters, inputs of the system. The particular form of the vector function F is unknown as we have only incomplete information, and quantitative details like kinetic parameters for biochemical reactions are often unavailable. Following [62], an influence graph is a common representation for biochemical systems where arrows show activations or inhibitions. Influence graphs [103] are a common representation for a wide range of dynamical systems. In the field of genetic networks, they have been investigated for various classes of systems,

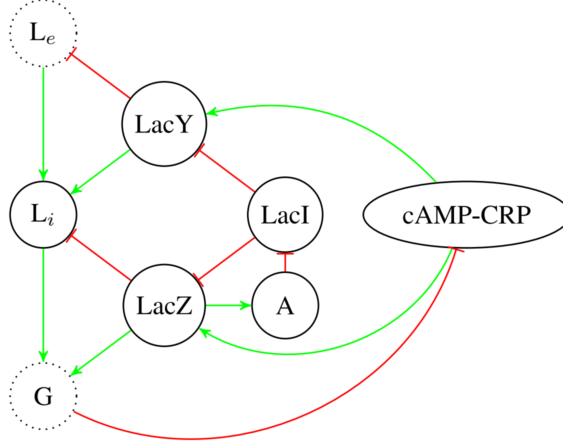

Figure 2.1: Simplified model of lactose operon in *Escherichia coli*, represented as an influence graph. The vertices represent either genes, metabolites, or proteins, while the edges indicate the regulations among them. Edges with an arrow stand for positive regulations (activations), while edges with a tee head stand for negative regulations (inhibitions). Vertices G and L_e are considered to be inputs of the system, that is, their signs are not constrained via their incoming edges.

ranging from ordinary differential equations [104] to synchronous [92] and asynchronous [93] Boolean networks. Influence graphs have also been introduced in the field of qualitative reasoning [79].

Definition 2.1 (Influence Graph) *An* influence graph *is a directed graph* (V, E, σ)*, where* V *is a set of vertices,* E *a set of edges, and* $\sigma : E \to \{+,-\}$ *a (partial) labeling of the edges.*

We represent a regulatory network as an *influence graph* whose vertices are the state and control variables of the system and whose edges express the effects of variables on each other. Therefore, there is an edge $j \to i \in E$ iff $\frac{\delta F_i}{\delta X_j} \neq 0$, which means the production rate of i depends on X_j. Furthermore, we assume that the *sign* of $\frac{\delta F_i}{\delta X_j}$ is constant, that is, the influence graph is independent of the state. The edges are labelled by $\sigma(j,i) = sign(\frac{\delta F_i}{\delta X_j})$. An example influence graph is given in Figure 2.1 it represents a simplified model of the lactose operon in *Escherichia coli*.

A *steady-state* X^{eq} of the system is a solution to the following equation: $F(X^{eq}, P) = 0$ for a fixed P. In SCM, experimental profiles are supposed to come from steady state shift experiments, where a system, initially in a steady state $eq1$, is perturbed using control parameters P, and eventually settles in a new steady state $eq2$. Measuring the changes of X_i, we represent the sign of the variations as a partial labeling $\mu : V \to \{+,-\}$ of the corresponding influence graph $\mu(i) = sign(X_i^{eq2} - X_i^{eq1})$. One can easily enhance this setting to also considering null (or more precisely, non-significant) variations, by exploiting the concept of sign algebra [79].

Given an influence graph (as a representation of a regulatory system) and a labeling of its vertices with signs (as a representation of experimental profiles), we now describe the constraints that relate both. Informally, for every non-input species i, its variation $\mu(i)$ ought to be explained by the influence of at least one predecessor j of i, $j \neq i$ in the

Species	L_e	L_i	G	LacY	LacZ	LacI	A	cAMP-CRP
μ_1	−	−	−	−	−	+	−	+
μ_2	+	+	−	+	−	+	−	−
μ_3	+	?	−	?	?	+	?	?
μ_4	?	?	?	−	+	?	?	+

Table 2.1: Some vertex labelings (reflecting measurements of two steady states) for the influence graph depicted in Figure 2.1; unobserved values indicated by question mark '**?**'.

influence graph. Thereby, the *influence* of j on i is given by the sign $\mu(j)\sigma(j,i) \in \{+,-\}$, where the multiplication of signs is derived from that of numbers. Sign consistency constraints can then be formalized as follows.

Definition 2.2 (Sign Consistency Constraints) *Let (V, E, σ) be an influence graph and $\mu : V \to \{+,-\}$ a (partial) vertex labeling.*

Then, (V, E, σ) and μ are consistent*, if there are some total extensions $\sigma' : E \to \{+,-\}$ of σ and $\mu' : V \to \{+,-\}$ of μ such that $\mu'(i)$ is consistent for each non-input vertex $i \in V$, where $\mu'(i)$ is consistent, if there is some edge $j \to i$ in E such that $\mu'(i) = \mu'(j)\sigma'(j,i)$.*

Note that labelings σ and μ of vertices and edges, respectively, are admitted to be partial. This occurs frequently in practice where the kind of an influence may depend on environmental factors or experimental data may not include all elements of a biological system. In order to decide whether a partially labeled influence graph and a partial experimental profile are mutually consistent, we thus consider the possible totalizations of them. If at least one total edge and one total vertex labeling (extending the given labelings) are such that the signs of all non-input vertices are explained, it is sufficient for mutual consistency.

Table 2.1 gives four vertex labelings for the influence graph in Figure 2.1. Total labeling μ_1 is consistent with the influence graph: the variation of each vertex (except for input vertex L_e) can be explained by the effect of one of its regulators. For instance, in μ_1, LacY receives a positive influence from cAMP-CRP as well as a negative influence from LacI, the latter accounting for the decrease of LacY. The second labeling, μ_2, is not consistent: LacY receives only negative influences from cAMP-CRP and LacI, and its increase cannot be explained. Partial vertex labeling μ_3 is consistent with the influence graph in Figure 2.1, as setting the signs of L_i, LacY, LacZ, A, and cAMP-CRP to +, −, −, −, and +, respectively, extends μ_3 to a consistent total labeling. In contrast, μ_4 cannot be extended consistently.

2.3 Checking Consistency

We now address how to check whether an experimental profile is consistent with a given influence graph. Note that, if the profile provides us with a sign for each vertex of the influence graph, the task can be accomplished simply by checking whether each non-input vertex receives at least one influence matching its variation. However, as soon as the experimental profile has missing values (which is very likely in practice), the problem becomes NP-hard [113]. In fact, a Boolean satisfiability problem over clauses $C_1, \dots, C_m$

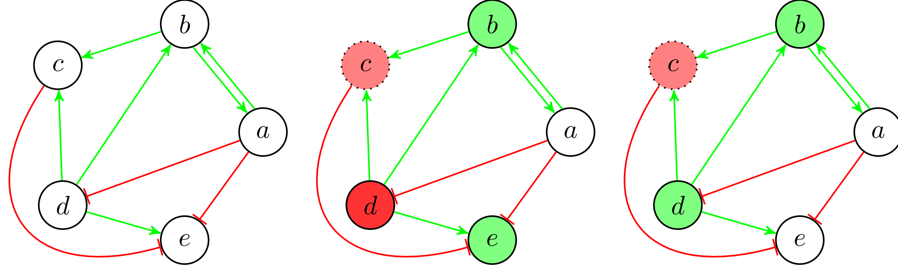

Figure 2.2: An influence graph (left) along with two experimental profiles (middle and right), in which increases (decreases) have been observed for vertices colored green (red), and vertex d is an input.

$$\Pi_g = \left\{\begin{array}{lllll} \mathit{vertex}(a). & \mathit{vertex}(b). & \mathit{vertex}(c). & \mathit{vertex}(d). & \mathit{vertex}(e). \\ \mathit{edge}(a,b). & \mathit{observedE}(a,b,+). & \mathit{edge}(c,e). & \mathit{observedE}(c,e,-). & \\ \mathit{edge}(a,d). & \mathit{observedE}(a,d,-). & \mathit{edge}(d,b). & \mathit{observedE}(d,b,+). & \\ \mathit{edge}(a,e). & \mathit{observedE}(a,e,-). & \mathit{edge}(d,c). & \mathit{observedE}(d,c,+). & \\ \mathit{edge}(b,a). & \mathit{observedE}(b,a,+). & \mathit{edge}(d,e). & \mathit{observedE}(d,e,+). & \\ \mathit{edge}(b,c). & \mathit{observedE}(b,c,+). & & & \end{array}\right\} \quad (2.1)$$

$$\Pi_{p_1} = \left\{\begin{array}{llll} \mathit{input}(c). & & & \\ \mathit{observedV}(b,+). & \mathit{observedV}(c,-). & \mathit{observedV}(d,-). & \mathit{observedV}(e,+). \end{array}\right\} \quad (2.2)$$

Figure 2.3: Facts representing the influence graph and experimental profile p_1 (middle) from Figure 2.2 in Π_g and Π_{p_1}, respectively .

and variables $x_1, \ldots, x_n$ can be reduced as follows: introduce unlabeled input vertices $x_1, \ldots, x_n$, non-input vertices $C_1, \ldots, C_m$ labeled +, and edges $x_j \rightarrow C_i$ labeled + (−) if x_j occurs positively (negatively) in C_i. It is not hard to check that the labeling of $C_1, \ldots, C_m$ by + is consistent with the obtained influence graph iff the conjunction of $C_1, \ldots, C_m$ is satisfiable.

We next provide a logic program such that each of its answer sets matches a consistent extension of vertex and edge labelings. Our encodings as well as instances are available at [11]. The program for consistency checking is composed of three parts, described in the following subsections.

2.3.1 Problem Instance

We now explain how we represent an influence graph as well as an experimental profile as a set of ground facts. For each species i, we introduce a fact *vertex*(i), and for each edge $j \rightarrow i$, a fact *edge*(j,i). If $s \in \{+,-\}$ is known the sign of an edge $j \rightarrow i$, it is expressed by a fact *observedE*(j,i,s). For an experiment the measured variation of a species i is expressed by a fact *observedV*(i,s). We assume that, for a given species i (or regulation $j \rightarrow i$) and an experimental profile p, an instance contains at most one of the facts *observedV*$(i,+)$ and *observedV*$(i,-)$ (or *observedE*$(j,i,+)$ and *observedE*$(j,i,-)$), but not both of them.

Example 2.1 *The facts describing the influence graph and the experimental profile p_1 shown in Figure 2.2 are provided in Figure 2.3. Note that experimental profile p_2 (cf. right in Figure 2.2) is inconsistent with the given influence graph. It necessitates labeling vertex a with – in order to explain the observed increase of d, while such a decrease of a cannot be explained. However, there were no such inherent inconsistencies, e.g., if increases of c had been observed in p_1 and p_2.* $\diamondsuit$

2.3.2 Generating Solution Candidates

As mentioned above, our goal is to check whether an experimental profile is consistent with an influence graph. If so, it is witnessed by total labelings of the vertices and edges, which are generated via the following rules:

$$\begin{array}{r} \textit{labelV}(V,+); \textit{labelV}(V,-) \leftarrow \textit{vertex}(V). \\ \textit{labelE}(U,V,+); \textit{labelE}(U,V,-) \leftarrow \textit{edge}(U,V). \end{array} \tag{2.3}$$

Moreover, the following rules ensure that known labels are respected by total labelings:

$$\begin{array}{r} \textit{labelV}(V,S) \leftarrow \textit{observedV}(V,S,P). \\ \textit{labelE}(U,V,S) \leftarrow \textit{observedE}(U,V,S). \end{array} \tag{2.4}$$

Note that the stability criterion for answer sets demands that a known label derived via a rule in (2.4) is also derived via (2.3), thus, excluding the opposite label. In fact, the disjunctive rules used in this section could actually be replaced with non-disjunctive rules via "shifting" [55],[1] given that our first encoding results in a so-called *head-cycle-free* (HCF) [9] ground program. However, similar disjunctive rules are also used in Section 2.4 where they cannot be compiled away. Also note that HCF programs, for which deciding answer set existence stays in NP, are recognized as such by disjunctive ASP solvers [81, 28]. Hence, the purely syntactic use of disjunction, as done here, is not harmful to efficiency.

Given the facts in (2.1) and(2.2) combined with the rules in (2.3) and (2.4), the resulting program admits two answer sets. The first one, including $\textit{labelV}(a,+)$ and the second one including $\textit{labelV}(a,-)$. On the remaining atoms, both answer sets coincide by containing the atoms in (2.1) along with $\textit{labelV}(b,+)$, $\textit{labelV}(c,-)$, $\textit{labelV}(d,-)$, $\textit{labelV}(e,+)$, $\textit{labelE}(a,b,+)$, $\textit{labelE}(a,d,-)$, $\textit{labelE}(a,e,-)$, $\textit{labelE}(b,c,+)$, $\textit{labelE}(c,e,-)$, $\textit{labelE}(d,b,+)$, $\textit{labelE}(d,c,+)$ and $\textit{labelE}(d,e,+)$.

2.3.3 Testing Solution Candidates

We now check whether generated total labelings satisfy the sign consistency constraints stated in Definition 2.2, requiring an influence of sign s for each non-input vertex i with variation s. We thus define $\textit{receive}(i,s)$ to indicate that i receives an influence of sign s:

$$\begin{array}{l} \textit{receive}(V,+) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,S). \\ \textit{receive}(V,-) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,T), S \neq T. \end{array} \tag{2.5}$$

[1] Alternatively, one could also use cardinality constraints (cf. [107]), which would however preclude a comparison with *dlv* in Section 2.7.

Inconsistent labelings, where a non-input vertex does not receive any influence matching its variation, are then ruled out by integrity constraints of the following form:

$$\leftarrow \mathit{labelV}(V,S), \mathit{not}\ \mathit{receive}(V,S), \mathit{not}\ \mathit{input}(V). \tag{2.6}$$

Starting from the answer sets described in the previous subsection, the included atoms *labelE*$(b,a,+)$ and *labelV*$(b,+)$ allow us to derive *receive*$(a,+)$ via a ground instance of the second rule in (2.5), while *receive*$(a,-)$ is not derivable. If we include the rules in 2.5 and 2.6 to the program, only the solution candidate containing *labelV*$(a,+)$ remains an answer set. In contrast, since *receive*$(a,-)$ is underivable, the solution candidate containing *labelV*$(a,-)$ violates the following ground instance of (2.6):

$$\leftarrow \mathit{labelV}(a,-), \mathit{not}\ \mathit{receive}(a,-), \mathit{not}\ \mathit{input}(a).$$

That is, the solution candidate with *labelV*$(a,-)$ does not pass the consistency test.

2.3.4 Soundness and Completeness

By letting $\tau((V,E,\sigma),\mu)$ denote the set of facts representing the problem instance induced by an influence graph (V,E,σ) and a vertex labeling μ, and P_C the logic program consisting of the rules given in (2.3), (2.4), (2.5), and (2.6), respectively, we can show the following soundness and completeness results.

Theorem 2.1 (Soundness) *Let* (V,E,σ) *be an influence graph and* $\mu : V \rightarrow \{+,-\}$ *a (partial) vertex labeling.*
If there is an answer set of $P_C \cup \tau((V,E,\sigma),\mu)$*, then* (V,E,σ) *and* μ *are consistent.*

Theorem 2.2 (Completeness) *Let* (V,E,σ) *be an influence graph and* $\mu : V \rightarrow \{+,-\}$ *a (partial) vertex labeling.*
If (V,E,σ) *and* μ *are consistent, then there is an answer set of* $P_C \cup \tau((V,E,\sigma),\mu)$.

The following correspondence result is immediately obtained from Theorem 2.1 and 2.2.

Corollary 2.3 (Soundness and Completeness) *Let* (V,E,σ) *be an influence graph and* $\mu : V \rightarrow \{+,-\}$ *a (partial) vertex labeling.*
Then, (V,E,σ) *and* μ *are consistent iff there is an answer set of* $P_C \cup \tau((V,E,\sigma),\mu)$.

2.3.5 Input Reduction

It is likely in practice that biological networks include simple tractable substructures or that parts of experimental observations are easily explained. Dealing with such particular cases before doing complex computations like checking consistency is therefore advisable. Given an influence graph (V,E,σ) and a partial vertex labeling μ capturing experimental data, we below describe conditions to identify vertices that can always be labeled consistently. Such vertices can then be marked as (additional) inputs to exclude their sign consistency constraints from consistency checking and to make explicit that they cannot cause any inconsistency. Any of the following conditions is sufficient to identify a vertex i as effectively unconstrained:

1. There is a regulation $i \rightarrow i$ in E such that $\sigma(i,i) = +$, that is, i supports its variation.

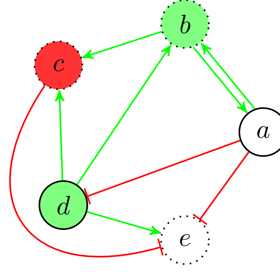

Figure 2.4: A partially labeled influence graph with uncritical vertices surrounded by dots.

2. There is a regulation $j \to i$ in E such that $\sigma(j,i)$ is undefined. In fact, undetermined regulations are used in practice to model influences that vary, e.g., relative to environmental conditions. Any variation of the target i of such a regulation can be explained by assigning the appropriate label to $j \to i$ (w.r.t. the label of j).

3. There are regulations $j \to i, k \to i$ in E such that $\mu(j)\sigma(j,i) = \mathbf{+}$ and $\mu(k)\sigma(k,i) = \mathbf{-}$. That is, any variation of i is already explained by the given observations.

4. An observed variation $\mu(i)$ of i is explained if there is some regulation $j \to i$ in E such that $\mu(j)\sigma(j,i) = \mu(i)$. Any further regulations targeting i can be ignored.

5. If for all regulations $i \to k$ in E, we have that k is an input, then the variation of i is insignificant for its targets. In this case, if i is unobserved ($\mu(i)$ is undefined) and target of at least one regulation $j \to i$ in E, we can assign an appropriate label to i (w.r.t. the labels of j and $j \to i$) without any further conditions.

6. There is a regulation $j \to i$ in E such that j is unobserved ($\mu(j)$ is undefined), an input, and all targets $k \neq i$ of j ($j \to k$ belongs to E) are inputs. Without any further conditions, we can assign an appropriate label to j for explaining the variation of i.

The reduction idea is to mark a vertex i as additional input, if it meets one of the above conditions. Since the two last conditions inspect inputs, they may become applicable to further vertices once inputs are added. Hence, checking the conditions and adding inputs needs to be done exhaustively. As we see below, this can easily be encoded in ASP.

Reconsidering the influence graph and partial observations in Figure 2.5, we see that vertex b receives an influence from d matching its observed increase. Thus, the fourth condition applies to already explained vertex b. Moreover, vertex e is unobserved and does not regulate anything. That is, the fifth condition applies to e, and its variation can simply be picked from influences it receives from a, c, and d. After establishing that e can be labeled consistently, we find that d does not regulate any critically constrained vertex. Applying again the fifth condition, we notice that the variation of d is actually insignificant.

Figure 2.4 shows the situation resulting from the identification of uncritical vertices by iteratively applying the above conditions. The fact that only a and d are critically constrained tells us that only they can be the cause of inconsistency.

The aforementioned idea to mark uncritical vertices as *input* can be encoded as follows:

$$
\begin{aligned}
\mathit{obs}(V) &\leftarrow \mathit{observedV}(V,S).\\
\mathit{get}(V,\mathbf{+}) &\leftarrow \mathit{observedE}(U,V,S), \mathit{observedV}(U,S).\\
\mathit{get}(V,\mathbf{-}) &\leftarrow \mathit{observedE}(U,V,S), \mathit{observedV}(U,T), S \neq T.\\
\mathit{input}(V) &\leftarrow \mathit{observedE}(V,V,\mathbf{+}).\\
\mathit{input}(V) &\leftarrow \mathit{edge}(U,V), \mathit{not}\ \mathit{observedE}(U,V,\mathbf{+}), \mathit{not}\ \mathit{observedE}(U,V,\mathbf{-}).\\
\mathit{input}(V) &\leftarrow \mathit{get}(V,\mathbf{+}), \mathit{get}(V,\mathbf{-}).\\
\mathit{input}(V) &\leftarrow \mathit{observedV}(V,S), \mathit{get}(V,S).\\
\mathit{input}(V) &\leftarrow \mathit{edge}(U,V), \mathit{input}(W) : \mathit{edge}(V,W), \mathit{not}\ \mathit{obs}(V).\\
\mathit{input}(V) &\leftarrow \mathit{edge}(U,V), \mathit{input}(W) : \mathit{edge}(U,W) : W \neq V, \mathit{input}(U), \mathit{not}\ \mathit{obs}(U).
\end{aligned}
$$

Auxiliary predicates *obs* and *get* are used to exhibit whether either variation has been observed for a vertex and whether a particular influence is received for certain, respectively. The last six rules check the described conditions (in the same order) and mark a vertex as *input* if one of them applies. Importantly, the above rules are stratified and thus yield a unique set of derived input vertices. This allows us to perform the reduction efficiently within grounding, without deferring to any procedural implementation external to ASP.

The situation shown in Figure 2.4 is reflected by the reduction encoding deriving atoms $\mathit{input}(b)$, $\mathit{input}(c)$, and $\mathit{input}(e)$ from an instance (cf. Section 2.3.1) corresponding to the depicted influence graph and observed variations. Consistency checking can then focus on the remaining non-input vertices a and d.

2.4 Identifying Minimal Inconsistent Cores

In view of the usually large amount of data, it is crucial to provide concise explanations whenever an experimental profile is inconsistent with an influence graph (i.e., if the logic program given in the previous section has no answer set). To this end, we adopt a strategy that was successfully applied on real biological data [64]. The basic idea is to isolate minimal subgraphs of an influence graph such that the vertices and edges cannot be labeled consistently. This task is closely related to extracting Minimal Unsatisfiable Cores (MUCs) [25] in the context of Boolean satisfiability (SAT). In allusion, we call a minimal subgraph of an influence graph whose vertices and edges cannot be labeled consistently a *Minimal Inconsistent Core* (MIC), whose formal definition is as follows. We note that verifying a MUC is D^P-complete [25, 89], and the same applies to MICs in view of the reduction of SAT described in Section 2.3. However, solving a decision problem is not sufficient for our application because we also need to provide MIC candidates to verify. As regards checking inconsistency of an (a priori unknown) MIC candidate, we are unaware of ways to accomplish such a co-NP test in non-disjunctive ASP without destroying the candidate at hand.

Definition 2.3 (Minimal Inconsistent Core) *Let (V, E, σ) be an influence graph and $\mu : V \to \{\mathbf{+},\mathbf{-}\}$ a (partial) vertex labeling.*
Then, a subset W of V is a Minimal Inconsistent Core *(MIC), if*

1. *for all total extensions $\sigma' : E \to \{\mathbf{+},\mathbf{-}\}$ of σ and $\mu' : V \to \{\mathbf{+},\mathbf{-}\}$ of μ, there is some non-input vertex $i \in W$ such that $\mu'(i)$ is inconsistent, and*

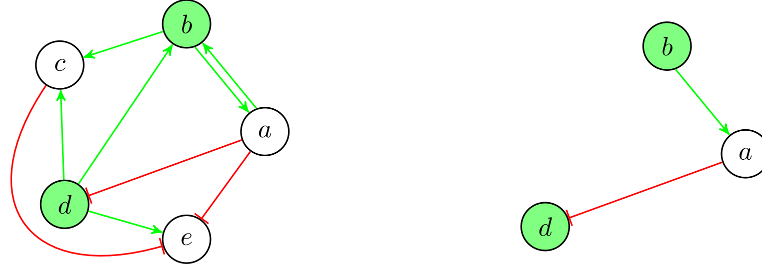

Figure 2.5: A partially labeled influence graph and a MIC consisting of a and d.

2. *for every* $W' \subset W$*, there are some total extensions* $\sigma' : E \rightarrow \{+, -\}$ *of* σ *and* $\mu' : V \rightarrow \{+, -\}$ *of* μ *such that* $\mu'(i)$ *is consistent for each non-input vertex* $i \in W'$.

To encode MICs, we make use of three important observations made on Definition 2.3. First, the inherent inconsistency of a MICs vertices stipulated in the first condition must be implied by the MIC and its external regulators, while vertices not connected to the MIC cannot contribute anything. Moreover, the second condition on proper subsets prohibits the inclusion of an input vertex in a MIC, as it could always be removed without affecting inherent (in)consistency of the remaining vertices variations. Finally, for establishing consistency of all proper subsets of a MIC, it is sufficient to consider subsets excluding a single vertex of the MIC, given that their consistency carries forward to all smaller subsets.

For illustration, consider the influence graph and the MIC in Figure 2.5. One can check that the observed simultaneous increase of b and d is not consistent with the influence graph, but the reason for this might not be apparent at first glance. However, once the MIC consisting of a and d is extracted, we see that the increase of b implies an increase of a, so that the observed increase of d cannot be explained. Note that the elucidation of inherent inconsistency provided by a MIC takes its vertices along with their regulators into account, the latter being incapable of jointly explaining the variations of all vertices in the MIC.

We next provide an encoding for identifying MICs, where a problem instance, that is, an influence graph along with an experimental profile, is represented by facts as specified in Section 2.3.1. The encoding then consists of three parts: the first generating MIC candidates, the second asserting inconsistency, and the third verifying minimality.

2.4.1 Generating MIC Candidates

The generating part comprises rules in (2.4) for deriving known vertex and edge labels. In addition, it includes the following rules:

$$
\begin{aligned}
\mathit{active}(V); \mathit{inactive}(V) &\leftarrow \mathit{vertex}(V), \mathit{not}\ \mathit{input}(V). \\
\mathit{edgeMIC}(U,V) &\leftarrow \mathit{edge}(U,V), \mathit{active}(V). \\
\mathit{vertexMIC}(U) &\leftarrow \mathit{edgeMIC}(U,V). \\
\mathit{vertexMIC}(V) &\leftarrow \mathit{active}(V). \\
\mathit{labelV}(V,+); \mathit{labelV}(V,-) &\leftarrow \mathit{vertexMIC}(V). \\
\mathit{labelE}(U,V,+); \mathit{labelE}(U,V,-) &\leftarrow \mathit{edgeMIC}(U,V).
\end{aligned}
\tag{2.7}
$$

The first rule permits guessing non-input vertices forming a MIC candidate. Such vertices are marked as *active*. The subgraph of the influence graph consisting of the active vertices, their regulators, and the connecting edges provides the context of the MIC candidate.In Definition 2.3, (in)consistency is checked only for the (non-input) vertices in a MIC, while other vertices variations do not need to be explained. Hence, guessing unobserved vertex (and edge) labels can be restricted to vertices belonging to or connected to the MIC, which reduces combinatorics. The vertices and edges contributing to this subgraph are identified via *vertexMIC* and *edgeMIC*. The guessing of (unobserved) vertex and edge labels is restricted to them in the last two rules of (2.7). Finally, note that the rules in (2.4) propagate known labels also for vertices and edges not correlated to the MIC candidate, viz., to the active vertices. This does not incur additional combinatorics; rather, it reduces derivations depending on MIC candidates.

2.4.2 Testing for Inconsistency

By adapting the methodology used in [32], the following subprogram makes sure that the active vertices cannot be labeled consistently, taking (implicitly) into account all possible labelings for them, their regulators, and connecting edges:[2]

$$\begin{aligned}
\mathit{opposite}(U,V) &\leftarrow \mathit{labelE}(U,V,\mathbf{-}), \mathit{labelV}(U,S), \mathit{labelV}(V,S).\\
\mathit{opposite}(U,V) &\leftarrow \mathit{labelE}(U,V,\mathbf{+}), \mathit{labelV}(U,S), \mathit{labelV}(V,T), S \neq T.\\
\mathit{bottom} &\leftarrow \mathit{active}(V), \mathit{opposite}(U,V) : \mathit{edge}(U,V).\\
&\leftarrow \mathit{not}\ \mathit{bottom}.\\
\mathit{labelV}(V,\mathbf{+}) &\leftarrow \mathit{bottom}, \mathit{vertex}(V).\\
\mathit{labelV}(V,\mathbf{-}) &\leftarrow \mathit{bottom}, \mathit{vertex}(V).\\
\mathit{labelE}(U,V,\mathbf{+}) &\leftarrow \mathit{bottom}, \mathit{edge}(U,V).\\
\mathit{labelE}(U,V,\mathbf{-}) &\leftarrow \mathit{bottom}, \mathit{edge}(U,V).
\end{aligned} \tag{2.8}$$

In this (part of the) encoding, $\mathit{opposite}(U,V)$ indicates that the influence of regulator U on V is opposite to the variation of V. If all regulators of an active vertex V have such an opposite influence, the sign consistency constraint for V is violated, in which case atom *bottom* along with all labels for vertices and edges are derived. Note that the stability criterion for an answer set X imposes that *bottom* and all labels belong to X only if the active vertices cannot be labeled consistently. Finally, integrity constraint $\leftarrow \mathit{not}\ \mathit{bottom}$ necessitates the inclusion of *bottom* in any answer set, thus, stipulating an inevitable sign consistency constraint violation for some active vertex.

Reconsidering our example in Figure 2.5, the ground instances of (2.7) permit guessing $\mathit{active}(a)$ and $\mathit{active}(d)$. When labeling a with **+** (or assuming $\mathit{labelV}(a,\mathbf{+})$ to be true), we derive $\mathit{opposite}(a,d)$ and *bottom*, producing in turn all labels for vertices and edges. Furthermore, setting the sign of a to **–** (or $\mathit{labelV}(a,\mathbf{-})$ to true) makes us derive $\mathit{opposite}(b,a)$, which again gives *bottom* and all labels for vertices and edges. We have thus verified that the sign consistency constraints for a and d cannot jointly be satisfied, given the observed increases of b and d. That is, active vertices a and d are sufficient to explain the inconsistency between the observations and the influence graph.

[2] In the language of *gringo* (and *lparse*), the expression $\mathit{opposite}(U,V) : \mathit{edge}(U,V)$ used below refers to the conjunction of all ground atoms $\mathit{opposite}(j,i)$ for which $\mathit{edge}(j,i)$ holds.

2.4.3 Testing for Minimality

It remains to be verified whether the sign consistency constraints for all active vertices are necessary to identify an inherent inconsistency. This test is based on the idea that, excluding any single active vertex, the sign consistency constraints for the other active vertices should be satisfied by appropriate labelings, which can be implemented as follows:

$$\begin{aligned}
\textit{labelV'}(W,V,\mathbf{+});\textit{labelV'}(W,V,\mathbf{-}) &\leftarrow \textit{active}(W),\textit{vertexMIC}(V).\\
\textit{labelE'}(W,U,V,\mathbf{+});\textit{labelE'}(W,U,V,\mathbf{-}) &\leftarrow \textit{active}(W),\textit{edgeMIC}(U,V).\\
\textit{labelV'}(W,V,S) &\leftarrow \textit{active}(W),\textit{observedV}(V,S).\\
\textit{labelE'}(W,U,V,S) &\leftarrow \textit{active}(W),\textit{observedE}(U,V,S).\\
\textit{receive'}(W,V,\mathbf{+}) &\leftarrow \textit{labelE'}(W,U,V,S),\textit{labelV'}(W,U,S),V\neq W.\\
\textit{receive'}(W,V,\mathbf{-}) &\leftarrow \textit{labelE'}(W,U,V,S),\textit{labelV'}(W,U,T),V\neq W,S\neq T.\\
&\leftarrow \textit{labelV'}(W,V,S),\textit{active}(V),V\neq W,not\ \textit{receive'}(W,V,S).
\end{aligned} \tag{2.9}$$

This subprogram is similar to the consistency check encoded via the rules in (2.3), (2.4), (2.5), and (2.6). However, sign consistency constraints are only checked for active vertices, and they must be satisfiable for all but one arbitrary active vertex W. In fact, labelings such that the variations of all active vertices but W are explained witness the fact that W cannot be removed from a MIC candidate without re-establishing consistency. As W ranges over all (non-input) vertices of an influence graph, each active vertex is taken into consideration. Regarding computational complexity, recall from Section 2.3 that checking consistency is NP-complete. As a consequence, one cannot easily identify conditions to select a particular witness for consistency of a MIC candidate minus some vertex W, and so we do not encode any such conditions. This leads to the potential of multiple answer sets comprising the same MIC but different witnesses, in particular, if many vertices and edges belong to the context of the MIC.

For the influence graph in Figure 2.5, it is easy to see that the sign consistency constraint for a is satisfied by setting the sign of a to **+**, expressed by atom *labelV'*$(d,a,\mathbf{+})$ in the ground rules obtained from the above encoding part. In turn, the sign consistency constraint for d is satisfied by setting the sign of a to **−**. This is reflected by atom *labelV'*$(a,a,\mathbf{-})$, allowing us to derive *receive'*$(a,d,\mathbf{+})$. That is, the ground instance of the above integrity constraint containing *labelV'*$(a,d,\mathbf{+})$ is satisfied. The fact that atoms *labelV'*$(d,a,\mathbf{+})$ and *labelV'*$(a,a,\mathbf{-})$, used for explaining the variation of either a or d, respectively, disagree on the sign of a also shows that jointly considering a and d yields an inconsistency.

2.4.4 Soundness and Completeness

Similar to Section 2.3.4, we can show the soundness and completeness for our MIC extraction encoding P_D, consisting of the rules in (2.4), (2.7), (2.8), and (2.9), respectively.

Theorem 2.4 (Soundness) *Let* (V,E,σ) *be an influence graph and* $\mu : V \rightarrow \{\mathbf{+},\mathbf{-}\}$ *a (partial) vertex labeling.*

If X *is an answer set of* $P_D \cup \tau((V,E,\sigma),\mu)$*, then* $\{i \mid \textit{active}(i) \in X\}$ *is a MIC.*

Theorem 2.5 (Completeness) *Let* (V,E,σ) *be an influence graph and* $\mu : V \rightarrow \{\mathbf{+},\mathbf{-}\}$ *a (partial) vertex labeling.*

If $W \subseteq V$ *is a MIC, then there is an answer set* X *of* $P_D \cup \tau((V, E, \sigma), \mu)$ *such that* $\{i \mid active(i) \in X\} = W$.

The following correspondence result is immediately obtained from Theorem 2.4 and 2.5.

Corollary 2.6 (Soundness and Completeness) *Let* (V, E, σ) *be an influence graph and* $\mu : V \to \{+, -\}$ *a (partial) vertex labeling.*
Then, $W \subseteq V$ *is a MIC iff there is an answer set* X *of* $P_D \cup \tau((V, E, \sigma), \mu)$ *such that* $\{i \mid active(i) \in X\} = W$.

As mentioned above, several answer sets may represent the same MIC because witnesses needed for minimality testing are not necessarily unique.

2.4.5 Exploiting Strongly Connected Components for MIC Extraction

In what follows, we introduce a connectivity property of MICs that can be used to further refine the encoding presented in Section 2.4. Incorporating additional background knowledge into the problem encoding is straightforward (as soon as such knowledge is established). In practice, ancillary (and actually redundant) conditions may significantly narrow and thus speed up both the grounding and the solving process.

MIC Connectivity Property. For analyzing interactions within a MIC, we make use of a graph described in the following. Let (V, E, σ) be an influence graph and $\mu : V \to \{+, -\}$ be a (partial) vertex labeling, and let $D(\mu)$ denote the set of vertices labeled by μ. For a set $W \subseteq V$ of vertices, we define a graph $(V[W], E[W])$ by:

$$V[W] = W \cup \{j \mid (j \to i) \in E, i \in W\}$$
$$E[W] = \{(j \to i) \mid (j \to i) \in E, i \in W\} \cup \{(i \to j) \mid (j \to i) \in E, i \in W, j \notin D(\mu)\}.$$

The construction of $(V[W], E[W])$ is based on the idea that a regulator j of some $i \in W$ is connected to i via its sign consistency constraint, and a connection in the opposite direction applies if j is unlabeled by μ. In fact, given some total extensions $\sigma' : E \to \{+, -\}$ of σ and $\mu' : V \to \{+, -\}$ of μ, we can check a matching influence of j on i by $\mu'(i) = \mu'(j)\sigma'(j, i)$ or equivalently by $\mu'(j) = \mu'(i)\sigma'(j, i)$. That is, provided that $\mu(j)$ is undefined, $\mu'(i)$ constrains $\mu'(j)$ by contraposition whenever i does not receive a matching influence from any other regulator than j. This observation motivates the inclusion of inverse edges from vertices in W to regulators unlabeled by μ in $E[W]$.

For illustration, the right-hand side of Figure 2.6 shows graph $(V[\{a, d\}], E[\{a, d\}])$ resulting from the partially labeled influence graph on the left-hand side. The single regulator b of a is labeled, and thus there is no inverse edge from a to b in $E[\{a, d\}]$. On the other hand, a is an unlabeled regulator of d, and so $E[\{a, d\}]$ includes an inverse edge from d to a. The addition of this edge turns the subgraph of $(V[\{a, d\}], E[\{a, d\}])$ induced by a and d into a strongly connected component. In view that a and d belong to a MIC (as discussed in Section 2.4), we below show that this connectivity is not by chance.

Theorem 2.7 (MIC Connectivity) *Let* (V, E, σ) *be an influence graph and* $\mu : V \to \{+, -\}$ *a (partial) vertex labeling.*
If $W \subseteq V$ *is a MIC, then all vertices in* W *belong to the same strongly connected component in* $(V[W], E[W])$.

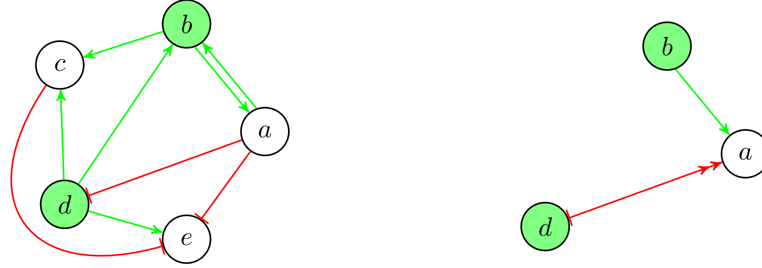

Figure 2.6: A partially labeled influence graph and the graph $(V[\{\mathbf{A},\mathbf{D}\}], E[\{\mathbf{A},\mathbf{D}\}])$.

Optimized MIC Encoding. We now apply Theorem 2.7 to improve the basic MIC extraction encoding (cf. Section 2.4) in two aspects: adding (redundant) constraints for search space pruning and adding positive body literals for reducing grounding efforts. The following rules pave the way by determining the (non-trivial) strongly connected components in $(V, E[V])$ as an over-approximation of the ones in $(V[W], E[W])$ for any $W \subseteq V$:

$$\begin{array}{r}
\mathit{edges}(U,V) \leftarrow \mathit{edge}(U,V), \mathit{not}\ \mathit{input}(V). \\
\mathit{edges}(V,U) \leftarrow \mathit{edge}(U,V), \mathit{not}\ \mathit{input}(V), \mathit{not}\ \mathit{observedV}(U,\mathbf{+}), \mathit{not}\ \mathit{observedV}(U,\mathbf{-}). \\
\mathit{reach}(U,V) \leftarrow \mathit{edges}(U,V). \\
\mathit{reach}(U,V) \leftarrow \mathit{edges}(U,W), \mathit{reach}(W,V), \mathit{vertex}(V). \\
\mathit{cycle}(U,V) \leftarrow \mathit{reach}(U,V), \mathit{reach}(V,U), U \neq V.
\end{array} \tag{2.10}$$

The first rule simply collects edges whose targets are not input, while the second rule adds edges in the inverse direction for unobserved regulators. Reachability w.r.t. the so obtained graph is determined via the third and the fourth rule. Finally, predicate *cycle* indicates whether two (distinct) vertices reach each other in $(V, E[V])$ relative to an influence graph (V, E, σ) and a (partial) vertex labeling μ. In fact, if two vertices belong to a MIC $W \subseteq V$, then mutual reachability in $(V[W], E[W])$ implies the same in $(V, E[V])$, in view that $V[W] \subseteq V$ and $E[W] \subseteq E[V]$. Conversely, if two vertices do not reach each other in $(V, E[V])$, then they cannot jointly belong to any MIC.

The over-approximation of potential MICs provides an easy means to prune the search space by adding the following integrity constraint:

$$\leftarrow \mathit{active}(U), \mathit{active}(V), U < V, \mathit{not}\ \mathit{cycle}(U,V). \tag{2.11}$$

The constraint makes the fact explicit that distinct vertices of a MIC must reach each other in $(V, E[V])$, and it immediately refutes MIC candidates that do not satisfy this condition.

After making use of Theorem 2.7 to narrow search, we now shift the focus to grounding. As a matter of fact, the quadratic space complexity of the minimality test's ground instantiation, as encoded in (2.9), is a major bottleneck in scaling. The knowledge about potential pairwisely connected vertices in MICs, represented by integrity constraint (2.11), also allows us to include positive body literals in order to restrict the scope of minimality tests:

$$
\begin{aligned}
\textit{labelV'}(W,V,\mathbf{+}); \textit{labelV'}(W,V,\mathbf{-}) &\leftarrow \textit{active}(W), \textit{active}(V), \textit{cycle}(V,W).\\
\textit{labelV'}(W,U,\mathbf{+}); \textit{labelV'}(W,U,\mathbf{-}) &\leftarrow \textit{active}(W), \textit{edgeMIC}(U,V), \textit{cycle}(V,W).\\
\textit{labelE'}(W,U,V,\mathbf{+}); \textit{labelE'}(W,U,V,\mathbf{-}) &\leftarrow \textit{active}(W), \textit{edgeMIC}(U,V), \textit{cycle}(V,W).\\
\textit{labelV'}(W,V,S) &\leftarrow \textit{active}(W), \textit{observedV}(V,S), \textit{cycle}(V,W).\\
\textit{labelV'}(W,U,S) &\leftarrow \textit{active}(W), \textit{observedV}(U,S), \textit{edge}(U,V), \textit{cycle}(V,W).\\
\textit{labelE'}(W,U,V,S) &\leftarrow \textit{active}(W), \textit{observedE}(U,V,S), \textit{cycle}(V,W).\\
\textit{receive'}(W,V,\mathbf{+}) &\leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,S).\\
\textit{receive'}(W,V,\mathbf{-}) &\leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,T), S \neq T.\\
&\leftarrow \textit{labelV'}(W,V,S), \textit{active}(V), \textit{cycle}(V,W), \mathit{not}\ \textit{receive'}(W,V,S).
\end{aligned}
\tag{2.12}
$$

In comparison to (2.9), the extra condition *cycle*(V,W) in the bodies of the first three rules establishes that labels used for testing minimality are guessed only for pairs W and V of vertices that can potentially jointly belong to a MIC. The same restriction is used in the next three rules forwarding observed vertex and edge labels, but now limited to vertices that can jointly belong to a MIC and to their respective regulators. Finally, the last two rules and the integrity constraint perform the same test as in (2.9) for a restricted set of pairs W and V. (The fact that *cycle*(V,W) implies $V \neq W$ in *labelE'*(W,U,V,S) also allows us to drop this condition, used in (2.9), from the bodies of the rules defining *receive'*.)

The complete optimized MIC encoding consists of the original rules in (2.4), (2.7), and (2.8), (2.10) and (2.11) as add-ons, and (2.12) as a replacement for (2.9). As regards the computational impact, we note that the optimized encoding needs less than two seconds for grounding and finding all MICs on the case study in Section 2.7.3, which took more than a minute with the unoptimized encoding.

A second version of the optimized encoding is obtained by tightening the consideration of connected vertices in $(V[W],E[W])$ relative to a MIC candidate W. This can be achieved by adding condition *active*(V) to the rules in (2.10) defining the *edges* predicate. In this way, the static reachability information encoded in (2.10), which is completely evaluated by grounder *gringo*, is turned into a dynamic relation computed during search. As it turns out, there is no significant performance difference between these two versions of the optimized MIC extraction encoding on the case study in Section 2.7.3. Hence, more real examples are needed to reliably compare their grounding and solving efficiency.

2.5 Repair

The natural question arising now is how to *repair* networks and data that have been found to be inconsistent, that is, how to modify network and/or data in order to re-establish their mutual consistency. A major challenge lies in the range of possible repair operations, since an inconsistency can be explained by missing interactions or inaccurate information in a network as well as by aberrant data. However, once consistency is re-established, network and data can be used for predicting unobserved variations.

To this end, we extend our basic approach and propose a framework for repairing large-scale biological networks and corresponding measurements in order to allow for predict-

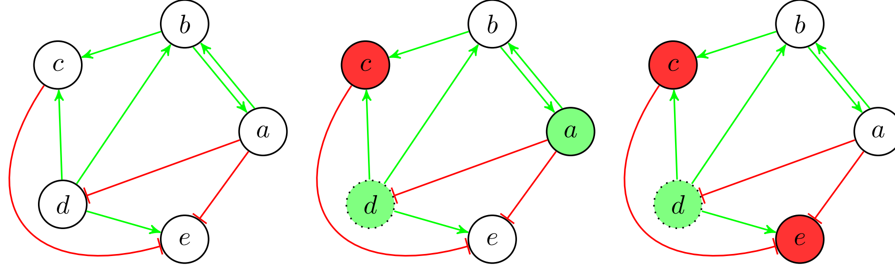

Figure 2.7: An influence graph (left) along with two experimental profiles (middle and right), in which increases (decreases) have been observed for vertices colored green (red), and vertex d is an input.

ing unobserved variations. We detail how reasoning modes can be used for efficient prediction under minimal repairs, complying with the objective of minimal change.

Our goal is to provide ASP solutions for reasoning over influence graphs and experimental profiles, in particular, if they are inconsistent with each other. To this end, we identify below several operations, called *repairs*, which can be applied to re-establish consistency.

In what follows, we provide logic program representations of repair in the input language of ASP grounder *gringo* [45]. After describing the format of instances, repair operations, and our repair encoding, we consider *minimal* repairs. Finally, we explain the usage of minimal repairs for *prediction* (under inconsistency).

2.5.1 Problem Instance

In order to compute correction that respect more than just a single experimental profile, we have to adjust the fact representation of experimental profiles given in Section 2.3.1. We need to uniquely identify experimental profiles, and relate each input parameters and each observations to a single experimental profile. Hence, each experimental profile is declared via a fact *exp*(p); and its observed variations and inputs are specified by facts *observedV*(p, i, s) with $s \in \{+, -\}$ and *input*(p, j), respectively. The fact representation of an influence graph is like described in Section 2.3.1. We assume that, for a given species i (or regulation $j \rightarrow i$) and an experimental profile p, an instance contains at most one of the facts *observedV*$(p, i, +)$ and *observedV*$(p, i, -)$ (or *observedE*$(j, i, +)$ and *observedE*$(j, i, -)$), but not both of them.

Example 2.2 *The facts describing the influence graph* (Π_g) *and experimental profiles* (Π_{p_1} *and* Π_{p_2}) *shown in Figure 2.7 are provided in Figure 2.8. Note that experimental profile* p_1 *(cf. middle in Figure 2.7) and* p_2 *(cf. right in Figure 2.7) are inconsistent with the given influence graph. Both necessitate labeling vertex* b *with* $-$ *in order to explain the observed decrease of* c*. With* p_1*, such a decrease of* b *is unexplained; with* p_2*, it can be explained by labeling* a *with* $-$*, which in turn leaves the observed decrease of* e *unexplained. However, there were no such inherent inconsistencies, e.g., if increases of* c *had been observed in* p_1 *and* p_2*.* $\diamondsuit$

$$\Pi_g = \left\{\begin{array}{lllll} vertex(a). & vertex(b). & vertex(c). & vertex(d). & vertex(e). \\ edge(a,b). & observedE(a,b,+). & edge(a,d). & observedE(a,d,-). & \\ edge(a,e). & observedE(a,e,-). & edge(b,a). & observedE(b,a,+). & \\ edge(b,c). & observedE(b,c,+). & edge(c,e). & observedE(c,e,-). & \\ edge(d,b). & observedE(d,b,+). & edge(d,c). & observedE(d,c,+). & \\ edge(d,e). & observedE(d,e,+). & & & \end{array}\right\} \quad (2.13)$$

$$\Pi_{p_1} = \left\{\begin{array}{lll} exp(p_1). & input(p_1,d). & \\ observedV(p_1,d,+). & observedV(p_1,c,-). & observedV(p_1,a,+). \end{array}\right\} \quad (2.14)$$

$$\Pi_{p_2} = \left\{\begin{array}{lll} exp(p_2). & input(p_2,d). & \\ observedV(p_2,d,+). & observedV(p_2,c,-). & observedV(p_2,e,-). \end{array}\right\} \quad (2.15)$$

Figure 2.8: Facts representing the influence graph and experimental profiles from Figure 2.7 in Π_g, Π_{p_1}, and Π_{p_2}, respectively.

2.5.2 Repair Operations

Repairs are operations on an influence graph (*model*) or experimental profiles (*data*) that can be applied to make model and data mutually consistent. Consistency of the repaired model/data is then witnessed by consistent total labelings of vertices and edges.

To begin with, we use the following rules to define admissible repair operations:

$$\begin{array}{ll} rep(add_e(U,V)) & \leftarrow rep_a, vertex(U), vertex(V), U \neq V, not\ edge(U,V). \\ rep(flip_e(U,V,S)) & \leftarrow rep_e, edge(U,V), observedE(U,V,S). \\ rep(inp_v(V)) & \leftarrow rep_g, vertex(V), exp(P), not\ input(P,V). \\ rep(inp_v(P,V)) & \leftarrow rep_i, vertex(V), exp(P), not\ input(P,V). \\ rep(flip_v(P,V,S)) & \leftarrow rep_v, vertex(V), exp(P), observedV(P,V,S). \end{array} \quad (2.16)$$

Note that particular operations are identified with *function terms* inside of predicate *rep*, which enables us to deal with repairs in a general way whenever knowing particular repair types is unnecessary. Otherwise, the meanings of the function terms are as follows:

Term	Target	Meaning
$add_e(U,V)$	model	Introduce an edge from U to V
$flip_e(U,V,S)$	model	Flip the sign S of the existing edge from U to V
$inp_v(V)$	model	Treat vertex V as an input in all experimental profiles
$inp_v(P,V)$	data	Treat vertex V as an input in experimental profile P
$flip_v(P,V,S)$	data	Flip the sign S of vertex V in experimental profile P

These repair operations are inspired by existing biological use cases. To repair a model by adding new edges makes sense when the model is incomplete (which is often the case in practice). Flipping the sign of an edge is a way to curate the model; it means that in some experiment the effect of a regulator (activator or inhibitor) should be corrected. Turning a vertex into an input can be used to indicate missing (unknown) regulations or oscillations of regulators. Revising experimental observations puts the dataset into question and may help to identify aberrant measurements (frequent in microarray data).

Which repair operations ought to be permitted or omitted requires background knowledge about the model and data at hand. By offering a variety of operations, our framework is flexible and may be adjusted to particular situations. In (2.16 the declaration of

admissible repair operations is governed by atoms $rep_a, \ldots, rep_v$. Depending on the requested repair types, such atoms are to be provided as facts. It would also be possible to restrict repair operations to particular edges or vertices, respectively, based on the availability of biological expert knowledge.

Finally, note that the rules (2.16) filter some redundant repairs. An edge between distinct vertices can be introduced only if there is none in the model. Flipping the sign of an edge or vertex is possible only if a sign is provided in the model or data, respectively. Making a vertex input, globally or in a particular experimental profile, requires it to not already be input in an arbitrary or the considered profile.

2.5.3 Repair Encoding

With admissible repairs at hand, the next rules encode the choice of operations to apply:

$$\begin{aligned} \mathit{app}(R) &\leftarrow \mathit{rep}(R), \mathit{not}\ \overline{\mathit{app}}(R). \\ \overline{\mathit{app}}(R) &\leftarrow \mathit{rep}(R), \mathit{not}\ \mathit{app}(R). \\ &\leftarrow \mathit{app}(\mathit{inp_v}(V)), \mathit{app}(\mathit{inp_v}(P, V)). \end{aligned} \tag{2.17}$$

Note that the integrity constraint above denies repair applications where a vertex is made input both globally and also in a particular experimantal profile. In such a case, the latter operation would be redundant. In general, the question of declaring a vertex as input either globally or local to an experiment depends on the intention whether to repair the model or data; however, simultaneously applying similar operations is futile.

The rest of the repair encoding is about identifying witnesses for the consistency of the repaired model/data. To this end, we first declare available signs and their complement relation[3]:

$$\mathit{sig}(+). \quad \mathit{sig}(-). \qquad\qquad \mathit{opp}(S, -S) \leftarrow \mathit{sig}(S). \tag{2.18}$$

The next rules take care of labeling edges and also incorporate repairs on them:

$$\begin{aligned} \mathit{labelE}(U,V,S) &\leftarrow \mathit{edge}(U,V), \mathit{observedE}(U,V,S), \mathit{not}\ \mathit{app}(\mathit{flip_e}(U,V,S)). \\ \mathit{labelE}(U,V,T) &\leftarrow \mathit{app}(\mathit{flip_e}(U,V,S)), \mathit{opp}(S,T). \\ \mathit{labelE}(U,V,S) &\leftarrow \mathit{app}(\mathit{add_e}(U,V)), \mathit{opp}(S,T), \mathit{not}\ \mathit{labelE}(U,V,T). \\ \mathit{labelE}(U,V,S) &\leftarrow \mathit{edge}(U,V), \mathit{opp}(S,T), \mathit{not}\ \mathit{labelE}(U,V,T). \end{aligned} \tag{2.19}$$

The first rule is to preserve (known) signs of edges if not flipped by a repair; otherwise, the second rule is used to derive the opposite sign instead. For edges introduced by repairs and unlabeled edges in the model, respectively, the last two rules encode the choice of a sign, making sure that any answer set comprises a total edge labeling given by ground atoms over predicate *labelE*.

Using the same methodology as with edges, but now relative to experimental profiles, the following rules deal with vertex labels and repairs on them:

$$\begin{aligned} \mathit{labelV}(P,V,S) &\leftarrow \mathit{vertex}(V), \mathit{exp}(P), \mathit{observedV}(P,V,S), \mathit{not}\ \mathit{app}(\mathit{flip_v}(P,V,S)). \\ \mathit{labelV}(P,V,T) &\leftarrow \mathit{app}(\mathit{flip_v}(P,V,S)), \mathit{opp}(S,T). \\ \mathit{labelV}(P,V,S) &\leftarrow \mathit{vertex}(V), \mathit{exp}(P), \mathit{opp}(S,T), \mathit{not}\ \mathit{labelV}(P,V,T). \end{aligned} \tag{2.20}$$

[3] Note that *gringo* interprets arithmetic functions like '–'. For instance, it evaluates –– to +.

In analogy to the rules in (2.19), the first rule maintains signs given in experimental profiles, while the second applies repairs flipping such signs. Ground instances of the third rule direct choosing signs of unobserved vertices, not yet handled by either the first or the second rule. As a consequence, the instances of *labelV*in an answer set provide a total vertex labeling.

Finally, we need to check whether the variations of all non-input vertices are explained by the influences of their regulators. This is accomplished as follows:

$$\begin{array}{rcl} \textit{receive}(P,V,S{*}T) & \leftarrow & \textit{labelE}(U,V,S), \textit{labelV}(P,U,T), \textit{not input}(P,V). \\ & \leftarrow & \textit{labelV}(P,V,S), \textit{not receive}(P,V,S), \\ & & \textit{not input}(P,V), \textit{not app}(\textit{inp_v}(V)), \textit{not app}(\textit{inp_v}(P,V)). \end{array} \quad (2.21)$$

First, observe that the influence of a regulator U on V is simply the product of the signs of the edge and of U. Based on this, the integrity constraint denies cases where a non-input vertex V, neither a given input of a profile P nor made input globally or in P by any repair, receives no influence matching its variation S. That is, a non-input vertex must not be unexplained in a profile. Conversely, any answer set comprises consistent total vertex and edge labelings wrt the repaired model/data.

Example 2.3 *Reconsider the influence graph, described by Π_g in (2.13), and experimental profiles, represented as Π_{p_1} and Π_{p_2} in (2.14) and (2.15), shown in Figure 2.7. Let Π be the encoding consisting of the rules in (2.16)–(2.21). Then, $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \{\textit{rep_v}.\}$ admits two answer sets comprising $\textit{app}(\textit{flip_v}(p_1, c, -))$ as single repair operation to apply, viz., the sign of c is flipped to +. Edge labels are as determined in Π_g, and the consistent total vertex labelings given by ground atoms over predicate labelVare shown in Row I:*

	a	b	c	d	e
p_1	+	+	+	+	+|−

So witnessing labelings require increases of a, b, c, and d, while e may either increase or decrease.

Similarly, there are two answer sets of for $\Pi \cup \Pi_g \cup \Pi_{p_2} \cup \{\textit{rep_v}.\}$ flipping c to +:

	a	b	c	d	e
p_2	+|−	+|−	+	+	−

Here, c as well as d must increase and e decrease, and the variations of a and b are variable but must comply with each other.

When looking at model repairs using program $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \Pi_{p_2} \cup \{\textit{rep_g}.\}$, where we may globally make vertices input, we get two answer sets applying only the repair operation expressed by $\textit{app}(\textit{inp_v}(c))$. The witnesses for p_1 and p_2 are shown in Row III.

	a	b	c	d	e
p_1	+	+	−	+	+|−
p_2	+	+	−	+	−

We have that p_1 and p_2 necessitate the same signs for a, b, c, and d wrt the repaired model. Moreover, with p_1, e can either increase or decrease, while it must decrease with p_2. $\Diamond$

2.5.4 Minimal Repairs

Typically, plenty of repairs are possible, in particular, if several repair operations are admitted by adding multiple control atoms *rep_a*, ..., *rep_v* as facts. However, one usually is only interested in repairs that make few changes on the model and/or data.

Repairs that re-establish consistency by applying a minimum number of operations can easily be selected among candidate repairs by using the *#minimize* directive available in *lparse*'s and *gringo*'s input languages [107, 45]. The respective statement is as follows:

$$\textit{\#minimize}\{\textit{app}(R) : \textit{rep}(R)\}. \tag{2.22}$$

It means that the number of instances of predicate *app* in answer sets, with argument R ranging over the ground instances of "domain predicate" *rep*, is subject to minimization. Note that (2.22) does not explicitly refer to the types of repair operations whose application is to be minimized.

Example 2.4 *As discussed in Example 2.2, the experimental profiles in Π_{p_1} and Π_{p_2} are inconsistent with the influence graph represented by Π_g. When augmenting program $\Pi \cup \Pi_g \cup \Pi_{p_1} \cup \Pi_{p_2} \cup \{rep_g.\}$ from Example 2.3 with the statement in (2.22), the answer sets comprising app$(inp_v(c))$ as the only repair operation to apply, along with the corresponding witnesses given in Example 2.3, yield a cardinality-minimal and thus optimal repair.* ♢

Although we do not detail them here, we note that alternative minimality criteria, such as weighted sum or subset inclusion, can also be used with the repair encoding in (2.16)–(2.21). While augmenting the *#minimize* statement in (2.22) with weights for atoms is straightforward, encoding subset-based minimization is more sophisticated. An encoding of the subset-minimality test is thus deferred to the appendix.

In fact, cardinality-minimal repairs may sometimes be too coarse and suppress further structurally interesting repairs being subset-minimal.

Example 2.5 *The previous example resulted in cardinality-minimal answer sets comprising app$(inp_v(c))$ as the single repair operation to apply. As a consequence, answer sets containing both app$(inp_v(a))$ and app$(inp_v(b))$ as applied repair operations are ignored because they fail to be cardinality-minimal. However, such answer sets exist, and the following vertex labelings are their witnesses:*

	a	b	c	d	e
p_1	+	−	−	+	+|−
p_2	+	−	−	+	−

Note that the increase of a and the decrease of b are both unexplained by these witnesses. Hence, neither app$(inp_v(a))$ nor app$(inp_v(b))$ can be dropped without losing consistency, so that the repair at hand is subset-minimal. ♢

Example 2.5 shows that minimizing cardinality can miss subset-minimal repairs. In fact, any cardinality-minimal repair is also subset-minimal, while the converse does not hold in general. Regarding computational complexity, we have that cardinality-minimization is usually accomplished with algorithms devised for problems in Δ_2^P (see, e.g., [100]), while algorithms usable for subset-minimization typically still handle Σ_2^P-hard problems

(cf. [81, 28]). The different complexities of solving methods suggest that cardinality-minimization is presumably more efficient in practice than subset-minimization. This is also confirmed by our experiments, whose results based on cardinality-minimization are presented below, while subset-minimization turned out to be too costly for an extensive empirical investigation.

2.6 Prediction under Repairs

For large real-world biological networks and measurements, there can be plenty witnessing vertex and edge labelings after re-establishing consistency via repairs. To not get lost in manifold scenarios, it is thus reasonable or even mandatory to focus on their common consequences. We therefore present the identification of consequences shared by all consistent vertex and edge labelings under minimal repairs as the ultimate application of our method, and we call this task *prediction*. Due to the capability of repairing, our approach enables prediction even if model and data are mutually inconsistent, which is often the case in practice. Importantly, enumerating all consistent total labelings is unnecessary. In fact, cautious reasoning [49], supported by ASP solver *clasp* [48], allows for computing the intersection of all (optimal) answer sets while investigating only linearly many of them.

For prediction, an input program Π is composed of an instance (cf. Figure 2.8), a definition of admissible repair operations ($rep_a, \ldots, rep_v$), the repair encoding in (2.16)–(2.21), and the *#minimize* statement in (2.22) (or alternatively the subset-minimality test in the appendix). Predicted signs for edges and vertices are then simply read off from instances of predicates *labelE* and *labelV* in the intersection of all optimal answer sets of Π, that is, answer sets comprising a minimum number of instances of predicate *app*. Though we mainly target at prediction wrt mutually inconsistent model and data, we note that prediction under consistency, where the unique minimal set of repair operations to apply is empty, is merely a particular case of prediction under repairs.

2.7 Empirical Evaluation and Application

For assessing the feasibility of our approach, we performed various benchmarks including artificially created and real biological datasets. We first evaluate our approach to consistency checking and diagnosis on a parameterized suite of randomly generated instances, aiming at structures similar to those found in biological applications. Then we tested our approach on the real-world data of genetic regulations in yeast. For validating our approach to repair, we used the regualatory network of *Escherichia coli* and confront this model with datasets of corresponding experiments.

2.7.1 Checking Consistency

We first evaluate our approach to consistency checking on randomly generated instances, aiming at structures similar to those found in biological applications. Instances are composed of an influence graph, a complete labeling of its edges, and a partial labeling of its vertices. Our random generator takes three parameters: (i) the number α of vertices

α	*claspD* *Berkmin*	*claspD* *VMTF*	*claspD* *VSIDS*	*cmodels*	*dlv*	*gnt*
500	0.14	0.11	0.11	0.16	0.46	0.71
1000	0.41	0.25	0.25	0.35	1.92	3.34
1500	0.79	0.38	0.38	0.53	4.35	7.50
2000	1.33	0.51	0.51	0.71	8.15	13.23
2500	2.10	0.66	0.66	0.89	13.51	21.88
3000	3.03	0.80	0.79	1.07	20.37	31.77
3500	3.22	0.93	0.92	1.15	21.54	34.39
4000	4.35	1.06	1.06	1.36	30.06	46.14

Table 2.2: Run-times for consistency checking with *claspD*, *cmodels*, *dlv*, and *gnt*.

in the influence graph, (ii) the average degree β of the graph, and (iii) the proportion γ of observed variations for vertices. To generate an instance, we compute a random graph with α vertices (the value of α varying from 500 to 4000) under the model by Erdős-Rényi [36]. Each pair of vertices has equal probability to be connected via an edge, whose label is chosen independently with probability 0.5 for both signs. We fix the average degree β to 2.5, which is considered to be a typical value for biological networks [73]. Finally, $\lfloor \gamma\alpha \rfloor$ vertices are chosen with uniform probability and assigned a label with probability 0.5 for both signs. For each number α of vertices, we generated 50 instances using five different values for γ, viz., 0.01, 0.02, 0.033, 0.05, and 0.1. All instances are available at [11].

We used *gringo* (2.0.0) [44] for combining the generated instances and the encoding given in Section 2.3 into equivalent ground programs. For checking consistency by computing an answer set (if it exists), we ran disjunctive ASP solvers *claspD* (1.1) [28] with "Berkmin", "VMTF", and "VSIDS" heuristics, *cmodels* (3.75) [56] using *zchaff*, *dlv* (BEN/Oct 11) [81], and *gnt* (2.1) [72]. All runs were performed on a Linux machine equipped with an AMD Opteron 2 GHz processor and a memory limit of 2GB RAM.

Table 2.2 shows average run-times in seconds over 50 instances per number α of vertices, including grounding times of *gringo* and solving times. We checked that grounding times of *gringo* increase linearly with the number α of vertices, and they do not vary significantly over γ. For all solvers, run-times also increase linearly in α.[4] For fixed α values, we found two clusters of instances: consistent ones where total labelings were easy to compute, and inconsistent ones where inconsistency was detected from preassigned labels. This tells us that the influence graphs generated as described above are usually (too) easy to label consistently, and inconsistency only occurs if it is explicitly introduced via fixed labels. However, such constellations are not unlikely in practice (cf. Section 2.7.3), and isolating MICs from them, as done in the next subsection, turned out to be hard for most solvers. Finally, greater values for γ led to an increased proportion of inconsistent instances, without making them much harder.

[4] Longer run-times of *claspD* with "Berkmin" in comparison to the other heuristics are due to a more expensive computation of heuristic values in the absence of conflict information. Furthermore, the time needed for performing "Lookahead" slows down *dlv* as well as *gnt*.

α	*gringo*	*claspD* *Berkmin*	*claspD* *VMTF*	*claspD* *VSIDS*
50	0.24	1.16 (0)	0.65 (0)	0.97 (0)
75	0.55	39.11 (1)	1.65 (0)	3.99 (0)
100	0.87	41.98 (1)	3.40 (0)	4.80 (0)
125	1.37	15.47 (0)	47.56 (1)	10.73 (0)
150	2.02	54.13 (0)	48.05 (0)	15.89 (0)
175	2.77	30.98 (0)	116.37 (2)	23.07 (0)
200	3.82	42.81 (0)	52.28 (1)	24.03 (0)
225	4.94	99.64 (1)	30.71 (0)	41.17 (0)
250	5.98	194.29 (3)	228.42 (5)	110.90 (1)
275	7.62	178.28 (2)	193.03 (4)	51.11 (0)
300	9.45	241.81 (2)	307.15 (7)	124.31 (0)

Table 2.3: Run-times for grounding with *gringo* and solving with *claspD*.

2.7.2 Minimal Inconsistent Cores

We now investigate the problem of finding a MIC within the same setting as in the previous subsection. Because of the elevated size of ground instantiations and problem difficulty, we varied the number α of vertices from 50 to 300, thus, using considerably smaller influence graphs than before. We again use *gringo* for grounding, now taking the encoding given in Section 2.4. As regards solving, we restrict our attention to *claspD* because all three of the other solvers showed drastic performance declines.

Table 2.3 shows average run-times in seconds over 50 instances per number α of vertices. Timeouts, indicated in parentheses, are taken as maximum time of 1800 seconds. We observe a quadratic increase in grounding times of *gringo*, which is in line with the fact that ground instantiations for our MIC encoding grow quadratically with the size of influence graphs. In fact, the schematic rules in Section 2.4.3 give rise to α copies of an influence graph. Considering solving times spent by *claspD* for finding one MIC (if it exists), we observe that they are relatively stable, in the sense that they are tightly correlated to grounding times. This regularity again confirms that, though it is random, the applied generation pattern tends to produce rather uniform influence graphs. Finally, we observed that unsatisfiable instances, i.e., consistent instances without any MIC, were easier to solve than the ones admitting answer sets. We conjecture that this is because consistent total labelings provide a disproof of inconsistency as encoded in Section 2.4.2.

As our experimental results demonstrate, computing MICs is computationally harder than just checking consistency. This is not surprising because the related (yet simpler) decision problem of verifying a MUC is D^P-complete [25, 89] and thus more complex than just deciding satisfiability. With our declarative technique, we spot the quadratic space blow-up incurred by the MIC encoding in Section 2.4 as a bottleneck. However, there are approaches aiming at a reduction of grounding efforts, and some of them have been presented in Sections 2.3.5 and 2.4.5.

2.7.3 Biological Case Study

In the following, we present the results of applying our consistency checking and diagnosis approach to real-world data of genetic regulations in yeast. We tested the gene-

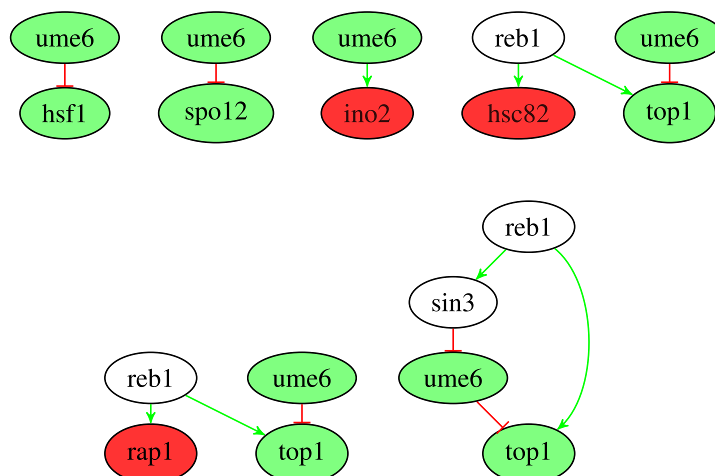

Figure 2.9: Some MICs obtained by comparing the regulatory network of yeast with a genetic profile.

regulatory network of yeast provided in [60] against genetic profile data of *snf2* knock-outs [106] from the Saccharomyces Genome Database[5]. The regulatory network of yeast contains 909 genetic or biochemical regulations, all of which have been established experimentally, among 491 genes.

Comparing the yeast regulatory network with the genetic profile of *snf2*, we found the data to be inconsistent with the network, which was easily detected using the approach of Section 2.3. Applying our diagnosis technique from Section 2.4, we obtained a total of 19 MICs. While computing the first MIC took less than a second using *gringo* and *claspD* (regardless of the heuristic used), the computation of all MICs was considerably harder. Using "VMTF" as search heuristic on top of the enumeration algorithm [47] inherited from *clasp* [46], *claspD* had found all 19 MICs in about 30 seconds, while another 40 seconds were needed to decide that there is no further MIC. With "VSIDS", finding the 19 MICs took about the same time as with "VMTF", but another 80 seconds were used to verify that all MICs had been found. Finally, using "Berkmin" heuristic, 12 MICs had been found before aborting after 30 minutes. The observation that search heuristics matter tells us that investigations into the structure of biological problems and particular methods to solve them efficiently can earn considerable benefits. Notably, by exploiting additional background knowledge, the optimized encoding presented in Section 2.4.5 requires less than two seconds (regardless of heuristics) for grounding and finding all 19 MICs. In fact, its ground instantiation contains only 8481 atoms and 10843 rules, compared to 47260 atoms and 56522 rules with the basic encoding in Section 2.4. In addition to problem size, also the difficulty drops dramatically: from 23345 conflicts down to 270 conflicts, encountered with "VMTF" heuristic during search for all answer sets. Furthermore, we note that the potential existence of multiple answer sets encompassing the same MIC did not emerge on the yeast network and *snf2* knock-out data. That is, we obtained 19 answer sets, each one corresponding one-to-one to a MIC.

Six of the computed MICs are exemplarily shown in Figure 2.9. While the first three of them are pretty obvious, we also identified more complex topologies. However, our example demonstrates that the MICs obtained in practice are still small enough to be

[5] `http://www.yeastgenome.org`

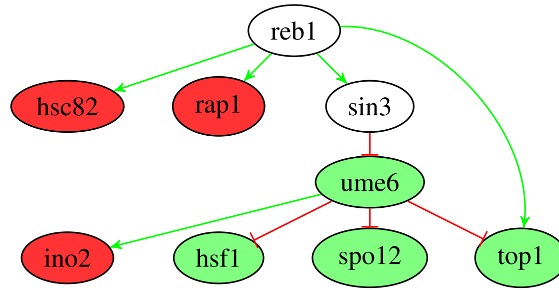

Figure 2.10: Subgraph obtained by connecting the six MICs given in Figure 2.9.

understood easily. For finding suitable corrections to the inconsistencies, it is often even more helpful to display the connections between several overlapping MICs. Observe that all six MICs in Figure 2.9 are related to gene *ume6*. Connecting them yields the subgraph of the yeast regulatory network in Figure 2.10.

The most obvious problem in Figure 2.10 is that the observed increase of *ume6* is incompatible with its four targets. This suggests that either the observation on *ume6* is incorrect or that some regulations are missing or wrongly modeled. In the first hypothesis though, one should note that the current model cannot explain a decrease of *ume6*: this would imply an increase of *sin3* and in turn an increase of *reb1*, but then there would be no explanation left for the variation of *hsc82* and *rap1*. So, in either case, our model should be revised. This is not a great surprise: our literature-based network, although very reliable, was presumably far from being complete.

Regarding the biological background, note that *ume6* is a known regulator of sporulation in yeast: in case of nutritional stress, yeast cells stop dividing and produce spores by meiosis. These spores are reproductive structures better adapted to extreme conditions. *ume6* is known as a key inhibitor of early meiotic genes: upon entry in meiosis, this inhibitory effect is released and the target genes are expressed. Notably, a knock-out of *ume6* causes the expression of meiotic genes during vegetative growth (hence its name, *Unscheduled Meiotic Expression*) as well as almost complete failure of sporulation [115]. *ume6* seems to have activation capabilities as well, though in that case the effect is believed to be indirect [18].

In the current view, *ume6* switches from inhibitor to (indirect) activator at the beginning of meiosis: Ume6p (the protein corresponding to the gene *ume6*) has a repressive effect when it forms a complex with Sin3p (note that *sin3* is in our network) and Rdp3p, which is degraded upon entry in meiosis [86]. This molecular mechanism can be interpreted in our model and one possible result is given in Figure 2.11. At least for negative targets, we now have a plausible explanation: the real effector of the inhibition on *hsf1*, *spo12*, *top1*, and *ume6* itself is the complex Ume6p-Sin3p, whose variation is unobserved but depends on the variation of *ume6* and *sin3*. The variation of the targets can be explained if the protein complex decreases, which is in turn possible if *sin3* decreases. Regretfully *sin3* is not observed in our data, but we note that a decrease of this gene is fully compatible with the rest of the network, that is, if we suppose a decrease of *reb1*. Now concerning *ino2*, our network should be updated with more recent evidence: as reviewed in [18], *ino2* has several additional regulators, such as *opi1* and *pah1* (see Figure 2.11). The observed variation of *pah1* is not useful to explain that of *ino2*, but *opi1* is definitely a plausible

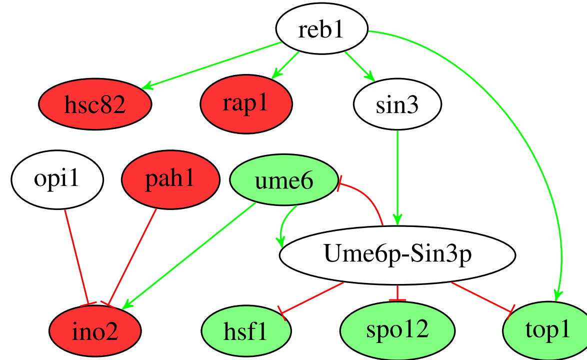

Figure 2.11: Local correction of the network based on our diagnosis method and literature research.

candidate.

Here we illustrated one main usage of our diagnosis technique: identifying poorly modeled regions of a regulatory network that are incompatible with a given data set. This is definitely a key asset if one wants to build a large-scale regulatory database and check its coherence with newly produced data on a regular basis. Given new data, our diagnosis method produces human-understandable representations of possible incompatibilities with the current model, which serve as the basis for a targeted literature research. With this data-driven approach, a network can then be improved with considerably less effort than with a random traversal of publications, for a much more coherent result.

2.7.4 Repair

For validating our approach, we use the transcriptional network of *Escherichia coli*, obtained from RegulonDB [39].

In the corresponding influence graph, the label of an edge depends on whether the interaction was determined as activation, inhibition, dual, or complex in RegulonDB. Overall, the influence graph consists of 5150 interactions between 1914 genes. We confront this model with datasets corresponding to the Exponential-Stationary growth shift study in [16] and the Heatshock experiment in [1]. For each of them, the extracted data yields about 850 significant variations (**+** or **–**) of *Escherichia coli* genes. Since the data is highly noisy, not surprisingly, it is inconsistent with the RegulonDB model. We generated data samples by randomly selecting 3%, 6%, 9%, 12%, or 15% of the whole data (about 850 variations with either experiment). We use these samples for testing both our repair modes as well as prediction (of omitted experimental data). All experiments were run with grounder *gringo* (2.0.3) and solver *clasp* (1.2.1) on a Linux PC equipped with AthMP+1900 processor and 4GB main memory, imposing a maximum time of 600 seconds per run. Below, we report runtime results for (cardinality-minimal) repair.

We first tested the feasibility of our repair modes on consistent as well as inconsistent samples (depending on the random selection). Table 2.4 provides average runtimes and numbers of timeouts in parentheses over 200 samples per percentage of selected measurements; timeouts are included as 600s in average runtimes. We ran experiments admitting the following repair operations and combinations thereof: flipping edge labels denoted by `e` (*flip_e*), making vertices input denoted by `i` (*inp_v*), and flipping preassigned varia-

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
e	6.58 (0)	8.44 (0)	11.60 (0)	14.88 (0)	26.20 (0)
i	2.18 (0)	2.15 (0)	2.21 (0)	2.23 (0)	2.21 (0)
v	1.41 (0)	1.40 (0)	1.40 (0)	1.41 (0)	1.37 (0)
e i	73.16 (6)	202.66 (23)	392.97 (87)	518.50(143)	574.85(179)
e v	28.53 (0)	85.17 (0)	189.27 (12)	327.98 (33)	470.48 (88)
i v	2.09 (0)	2.14 (0)	2.45 (0)	3.08 (0)	6.06 (0)
e i v	133.84 (8)	391.60 (76)	538.93(151)	593.33(193)	600.00(200)

Heatshock					
Repair	3%	6%	9%	12%	15%
e	25.54 (4)	42.76 (8)	50.46 (5)	69.23 (6)	84.77 (6)
i	2.10 (0)	2.13 (0)	2.13 (0)	2.05 (0)	2.08 (0)
v	1.41 (0)	1.47 (0)	1.42 (0)	1.37 (0)	1.39 (0)
e i	120.91(21)	374.69 (91)	553.00(169)	593.20(197)	595.99(198)
e v	67.92 (3)	236.05 (31)	465.92(107)	579.88(179)	596.17(197)
i v	2.27 (0)	4.94 (0)	60.63 (8)	257.68 (56)	418.93(123)
e i v	232.29(26)	542.48(152)	593.88(195)	600.00(200)	600.00(200)

Table 2.4: Repair Times.

tions denoted by v (*flip_v*). Note that the two modes to make vertices input (globally or locally) fall together here, and we used only the local repair operation in i while skipping tests with the other, equivalent one. Moreover, we do not include results on the adding edges repair (*add_e*), where the bottleneck is grounding since the potential addition of arbitrary edges turns the influence graph into a huge clique at the encoding level. To avoid this, edges that can possibly be added by repairs should be restricted to a (smaller) set of reasonable ones, which requires biological knowledge, e.g., regulations known for organisms related to the investigated one.

In view that *clasp* applies a branch-and-bound approach to search for a cardinality-minimal repair, we observe that the average runtimes shown in Table 2.4 are directly correlated to the number of admissible repair operations. The fewest repairs are admitted with v, given that only about 25–130 observed variations are included in the samples of varying percentage. All vertices of the influence graph can potentially be made input with i, but when run in isolation or together with v, it still performs relatively well. Finally, as signs are available for almost all edges of the influence graph, permitting to flip each of them in e explains long runtimes and many timeouts obtained with it, in particular, on its combinations with i. However, the tight correlation between number of admitted repairs and runtime suggests that our method could significantly benefit from the inclusion of biological knowledge to restrict scopes of repairs.

In addition, we tested our approach also by selecting the 100% of the samples (containing all 850 observations). The test on the 100% of the samples were run on a faster machine than the tests for the other samples, therefore these times only show the relation between the different repair modes. The results show a continuation of the trend. The application of single repair modes e, i, v return within 10 minutes, the combination of repair modes always take longer than 10 minutes. Repairing the data by flipping observation with v performs the best taking 0.6 seconds with minimal 40 repairs

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
`e`	13.27(0)	12.19(0)	14.76(0)	15.34 (0)	25.90 (1)
`i`	6.18(0)	5.26(0)	4.77(0)	4.60 (0)	4.42 (0)
`v`	4.64(0)	4.45(0)	4.39(0)	4.40 (0)	4.30 (0)
`e i`	35.25(0)	97.66(1)	293.80(3)	456.55 (3)	550.33 (1)
`e v`	14.35(0)	26.17(0)	90.17(3)	200.25(13)	363.36(16)
`i v`	6.43(0)	5.75(0)	6.27(0)	6.69 (0)	8.61 (0)
`e i v`	42.51(0)	248.30(1)	468.71(2)	579.58 (0)	— (0)

Heatshock					
Repair	3%	6%	9%	12%	15%
`e`	25.77(0)	37.18(0)	29.09(0)	36.23 (0)	41.88 (0)
`i`	6.57(0)	5.93(0)	5.17(0)	4.86 (0)	4.54 (0)
`v`	4.86(0)	5.06(0)	5.34(0)	5.42 (0)	5.52 (0)
`e i`	85.47(0)	293.28(1)	524.19(3)	591.81 (0)	594.74 (0)
`e v`	23.32(0)	111.99(0)	338.95(0)	545.56 (2)	591.23 (0)
`i v`	6.91(0)	6.63(0)	30.33(0)	176.14 (1)	371.95 (0)
`e i v`	101.82(1)	466.91(0)	585.64(0)	— (0)	— (0)

Table 2.5: Prediction Times.

on the Exponential-Stationary growth shift study and 0.5 seconds and 34 repairs on the Heatshock experiment. Repairing the influence graph by making vertices input with `i` performs second best with 1 second for 42 repairs on the Exponential-Stationary growth shift study and 1 second for 94 repairs on the Heatshock experiment. The reparation by flipping signs of the edges remains last with 27.3 second and also 42 repairs on the Exponential-Stationary growth shift study. On the Heatshock experiment the runtime for reparation by flipping edges exceeded 10 minutes.

2.7.5 Prediction under Repair

In the second step, done after computing the minimum number of repairs needed, we performed prediction by computing the intersection of all answer sets comprising a cardinality-minimal (and sometimes empty) repair. To this end, we used the cautious reasoning capacities of *clasp* (option `--cautious`) along with options `--opt-value` and `--opt-all` for initializing the objective function with the minimum number of repairs and enumerating all optimal answer sets, respectively. Runtime results are presented in Table 2.5, using the same notations for repair operations as in Section 2.7.4, but taking average runtimes only over those of the 200 samples per percentage where a cardinality-minimal repair was computed before the timeout (as the optimum is not known otherwise). We observe that the runtimes for prediction are in line with the ones for computing a cardinality-minimal repair, and maximum time is only rarely exceeded on the samples with known optimum. This shows that prediction is successfully applicable if computing a cardinality-minimal repair is feasible.

In what follows, we analyze quantity and quality of the predictions we obtained. To this end, we determined the following numbers for each run: N vertices without variation given in the sample, P newly predicted vertices (variation not given in the sample), V

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
`e`	15.00	18.51	20.93	22.79	23.94
`i`	15.00	18.51	20.93	22.79	23.93
`v`	14.90	18.37	20.86	22.73	23.77
`e i`	14.92	18.61	20.55	21.96	22.80
`e v`	14.89	18.33	21.07	22.52	23.74
`i v`	14.89	18.33	20.79	22.59	23.66
`e i v`	14.58	19.00	20.29	21.13	—

Heatshock					
Repair	3%	6%	9%	12%	15%
`e`	15.47	19.54	21.87	23.17	24.78
`i`	15.48	19.62	21.89	23.20	24.80
`v`	15.32	19.59	21.37	22.13	23.79
`e i`	15.37	19.62	22.83	23.44	24.05
`e v`	15.33	19.21	21.00	22.65	24.90
`i v`	15.41	19.47	21.36	21.81	23.55
`e i v`	15.01	19.11	22.52	—	—

Table 2.6: Prediction Rate.

newly predicted vertices having the same variation as in the whole dataset, and W newly predicted vertices having the opposite variation in the whole dataset. Based on this, the *prediction rate* is obtained via the formula $(P{*}100)/N$, and the *prediction accuracy* is given by $(V{*}100)/(V{+}W)$. That is, the prediction rate reflects the amount of newly predicted vertices, while the prediction accuracy measures in how many cases variations available in the whole dataset (but not in the sample) have been recovered. Average prediction rates over samples where both repair and prediction terminated are shown in Table 2.6, the accuracies of the computed predictions are shown in Table 2.7. Note that some averages result from few instances only (many timeouts reported in Table 2.5); such results should be interpreted with care.

We first notice that in both experiments, Exponential-Stationary growth shift and Heatshock, the prediction rates are significant, varying from about 15 to 24 percent. As it can be expected, prediction rates increase with the size of samples, in particular, for the transition from 3% to 6% of preassigned vertices. However, while the size of the samples increase linearly, the prediction rates do not. This may mean that the prediction rate may reach a plateau related to the topology of the network as it was also observed in [112]. Interestingly, we do not observe any significant decrease of prediction rates when admitting multiple repair operations simultaneously. This suggests that predicted variations are rather insensitive to the repair operations used for re-establishing consistency, given that the application of repairs is governed by cardinality-minimality. Comparing individual operations `e`, `i`, and `v` with each other, we observe that `v` (flipping variations) yields a slightly lower prediction rate than the others, in particular, on the larger samples of Heatshock.

As regards prediction accuracies, they are consistently higher than 90 percent, meaning that predicted variations and experimental observations correspond in most cases. As with prediction rates, accuracies increase with sample size, while the choice of admissi-

Exponential-Stationary growth shift					
Repair	3%	6%	9%	12%	15%
e	90.93	91.98	92.42	92.70	92.81
i	90.93	91.98	92.42	92.70	92.81
v	90.99	92.05	92.44	92.73	92.89
e i	91.09	91.90	92.57	93.03	93.19
e v	90.99	92.03	92.50	92.82	92.94
i v	90.99	92.03	92.42	92.71	92.87
e i v	91.35	92.29	92.52	93.04	—

Heatshock					
Repair	3%	6%	9%	12%	15%
e	91.87	92.93	92.92	92.83	92.71
i	91.93	92.90	92.94	92.87	92.76
v	92.29	93.27	93.88	94.27	94.36
e i	91.99	92.49	91.16	93.62	94.44
e v	92.30	93.37	93.66	94.36	94.35
i v	92.24	93.34	93.90	94.26	94.38
e i v	92.26	93.04	91.78	—	—

Table 2.7: Prediction Accuracy.

ble repair operations does not exhibit much impact. This indicates that filtering repairs by cardinality-minimality makes the qualitative results largely independent of repair types, at least on the datasets we consider here. Also note that edge labels (activation or inhibition) are well-curated in RegulonDB, which both enables and explains the obtained high accuracies. Despite of this, we still observe that individual operation v yields higher accuracy than e and i. Interestingly, this gap is greatest for the larger samples of Heatshock, where v also has a lower prediction rate. This demonstrates that prediction quantity and quality can diverge, and an appropriate compromise certainly depends on the application at hand.

The observation that repair operation v yields better prediction accuracy than e (flipping edge labels) or i (making vertices input), particularly with Heatshock, suggests that repairing the data is more appropriate than repairing the model wrt the datasets we consider.

Finally, we compared prediction accuracies to the ones obtained when using a different kind of repair method: iteratively removing inconsistent subnetworks (cf. [54]) until the remaining network is found to be consistent. Repair results, that is, networks, obtained in such a way depend on the order of removing inconsistent subnetworks, while the technique presented here is fully declarative. The alternative repair method achieved prediction accuracies between 65 and 73 percent on Exponential-Stationary growth shift and from 76 to 80 percent on Heatshock data. The higher accuracies of our declarative technique show that a firm repair concept pays off in prediction quality.

2.8 Discussion

We have provided an approach based on ASP to investigate the consistency between experimental profile and influence graphs. In case of inconsistency, the concept of a MIC can be exploited for identifying concise explanations, pointing to unreliable data and/or missing reactions.

Further, we have introduced repair-based reasoning techniques for computing minimal modifications of biological networks and experimental profiles to make them mutually consistent. Finally, we provided an approach to predict unobserved data even in case of inconsistency under the Sign Consistency Model. We evaluated our approach on real biological examples and showed that predictions on the basis of minimal repairs were, for one, feasible and, for another, highly accurate. This is of practical relevance because genetic profiles from DNA microarrays tend to be noisy and available biological networks far from being complete.

Given that the framework is configurable, it can be adjusted to varying tasks that occur in practice. As illustrated in Section 2.7, it enables a meaningful analysis of partially unreliable experimental data, and reasonable conclusions from it can be drawn automatically. Our repair method could be exploited to indicate and correct spurious parts of such generated models.

3 Metabolic Network Expansion

In this chapter, we investigate the reconstruction of metabolic networks. The reconstruction of a metabolic network for an organism is an iterative process that usually starts with the genome sequence and the identifcation of the protein-coding genes. The correlation of genome and metabolism is then done by searching gene databases, such as KEGG (`http://www.genome.jp/kegg`) and GeneDB (`http://www.genedb.org`), for particular genes by inputting enzyme or protein names. The result is a first draft network that usually suffers from substantial incompleteness with lots of missing reactions. Usually the draft network is further refined by confronting it with gas chromatography–mass spectrometry (GC-MS) data from experiments. These metabolite profiles give hints to the biosynthetic capabilities of the metabolic networks, unfortnately these measured datasets are far from complete and often contain unreliable data. Therefore, many networks are only partially defined and only few metabolites can be identified without ambiguity. Moreover, traditional formal approaches to biosynthesis (cf. [95, 78, 13, 97, 116]) require kinetic information, which is rarely available.

We address this problem and propose a qualitative approach based on ASP. This approach benefits from the intrinsic incompleteness-tolerating capacities of ASP and allows us to take advantage of its rich modelling language and highly efficient implementations. Our approach is endorsed by recent complexity results, showing that the reconstruction of metabolic networks and related problems are NP-hard [87, 88].

Our approach builds upon a formal method for analyzing large-scale metabolic networks developed in [31, 65]. The basic idea is that a reaction operates only if its reactants are either available as nutrients or can be provided by other metabolic reactions. Starting from some nutrients, referred to as *seeds*, this allows for expanding a metabolic network by successively adding operable reactions and their products. The set of metabolites in the resulting network is called the *scope* of the seeds and represents all metabolites that can principally be synthesized from the seeds by the analyzed metabolic network.

Mapping the principles of this approach into ASP allows us to address various biologically relevant problems. A primary problem deals with the completion of genome-scale metabolic networks. When building a metabolic network, as for the recently sequenced green alga *Chlamydomonas reinhardtii* (`http://www.goforsys.de`), the initial core draft is done by appeal to genomic information. Then, experimental data, in particular, measured metabolites, are taken to define the functionality of the overall network. The above methodology can then be used to check whether a drafted network provides the synthesis routes to comply with the required functionality. If this fails, the draft network can be completed by importing reactions from metabolic reference network stemming from other organisms until the obtained network provides the measured functionality (cf. [20]). Another important problem concerns the determination of seed compounds needed for the synthesis of certain other compounds. As demonstrated in [66], solving this problem is important for indicating (minimal) nutritional requirements for

sustaining maintenance or growth of an organism.

Both problems have a combinatorial nature and thus give rise to a multitude of solutions. We address this problem by taking advantage of the various reasoning modes provided by ASP. On the one hand, we use ASPs optimization techniques for finding cardinality or subset minimal solutions, respectively. On the other hand, we exploit consequence aggregation for finding metabolites common to all (optimal) solutions or at least one of them, respectively. Moreover, the aforementioned problems are often subject to additional constraint, aiming at the avoidance of side products or producing target products by staying clear from certain seeds, respectively. Finally, the elaboration tolerance of ASP greatly supports the process of drafting metabolic networks involving continuous validation and increasing functionalities stemming from measured data.

To begin with, we give a brief introduction into the biological background of metabolism, enzymes and chemical reactions in Section 3.1. Section 3.2 gives a mathematical formal representation of metabolic reaction networks and defines the notion of a *scope* as well as *completion*. In Section 3.3, we develop an ASP formulation for solving the metabolic network completion problem. Section 3.4 extends this approach for solving the *inverse scope problem*. In Section 3.5, we show results of our empirical evaluation of our approach on the metabolic networks of *Escherichia coli*. Finally, we conclude this chapter with a discussion in Section 3.6.

3.1 Biological Background

Metabolism is the entirety of chemical processes that happen in a living oganism, from the uptake and transport, to the chemical transformation and disposal of substances. The metabolism transforms chemicals in order to build structures and maintain vital functionalities of the organism. Metabolism can be divided into two categories, catabolism which breaks down chemical components to harvest energy, and anabolism which uses energy to construct vital structures of the organism. The metabolism determines which substances are nutritious for an organism, and which are poisonous. Metabolic reactions that constitute a specific biological function or process are grouped into metabolic pathways. Figure 3.1 shows the example of the *Glycolysis* pathway.

An important role in the metabolism play the enzymes, they are special proteins that catalyse chemical reactions that require energy and would not occure by themselves. Enzymes allow the regulation of the metabolism and to adjust to changes in the environment. The synthesis of enzymes are mostly regulated on the genome level here is the connection between genetic regulation and metabolic responses. Therefore, by identifying protein coding sequences in the genome of an organism, one can infer possible reactions of its metabolism. Only few metabolites can be measured directly through methods like GC-MS, and kinetic details like reactions speeds are rarely known. This is why we need intelligent methods to fill the gaps in our knowledge on metabolism.

3.2 Mathematical Formalism

A common way to reason about the metabolism of an organism is in terms of a metabolic network. Metabolic networks are composed of a set of chemical reactions, they lack ki-

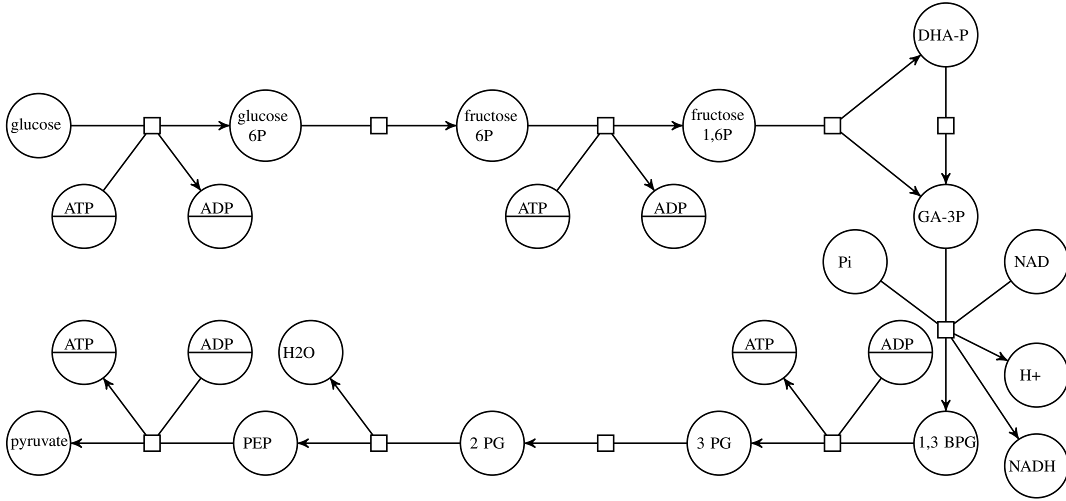

Figure 3.1: Glycolysis. A typical illustration of a metabolic pathway.

netic information like reaction speed, and are often simplified such that information on ubiquitous chemicals, co-factors and catalysing enzymes have been removed. The chemical reactions are directed such that one set of metabolites called *reactants* are transformed into another set of metabolites, the *products* of the reaction. Reversible reactions are usually represented by two reactions with opposite directions. Following [88], *metabolic networks* are commonly represented as a directed bipartite graph.

Definition 3.1 (Metabolic Network) *A metabolic network is a directed bipartite $G = (R \cup M, E)$, where R and M are sets of nodes standing for* reactions *and* metabolites*, respectively.*

Given such a metabolic network G, we sometimes refer to its components by $R(G)$, $M(G)$, and $E(G)$. Whenever $(m, r) \in E$ for $m \in M$ and $r \in R$, the metabolite m is called a *reactant* of reaction *r*; for $(r, m) \in E$, metabolite m is called a *product* of r. More formally, for $(R \cup M, E)$ and $r \in R$ define $reac(r) = \{m \in M \mid (m, r) \in E\}$ and $prod(r) = \{m \in M \mid (r, m) \in E\}$.

The biosynthetic capabilities of the underlying metabolism are approximated by the *scope*. The concept of a *scope* can be expressed in terms of reachability. Given a metabolic network $(R \cup M, E)$ and a set $M' \subseteq M$ of *seed* metabolites, a reaction $r \in R$ is *reachable* from M', if $reac(r) \subseteq M'$, that is, if all its reactants are reachable. Moreover, a metabolite $m \in M$ is *reachable* from M', either if $m \in M'$ or if $m \in prod(r)$ for some reaction $r \in R$ being *reachable* from M'. Therefore, we can define the *scope* of a set of seed compounds as follows.

Definition 3.2 (Scope) *Given a metabolic network $(R \cup M, E)$ and a set $M' \subseteq M$ of* seed *metabolites, the* scope *of M', written $\Sigma_{(R \cup M, E)}(M')$ or simply $\Sigma(M')$, is the set of all metabolite nodes reachable from M'.*

Figure 3.2 illustrates the scope of a metabolic network given a set of seed metabolites. Note that the scope of a set of metabolites can be computed in polynomial time.

In the *metabolic network expansion*, we are given a metabolic network along with two sets of metabolites (seeds and target), and a reference network. The goal is to extend

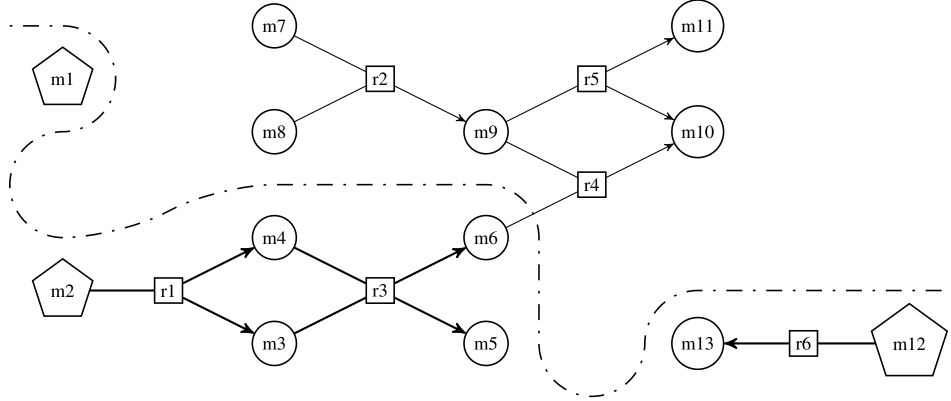

Figure 3.2: Given the seed metabolites m1, m2 and m12, the scope of this network is {m1, m2, m3, m4, m5, m6, m12, m13}.

the draft network with reactions from the reference network such that all targets metabolites can be reached from the seeds. Given the definition of *scope*, we can render more precisely the aforementioned biological problem.

Definition 3.3 (Completion) *Given a metabolic network* $(R \cup M, E)$ *along with two sets* $S, T \subseteq M$ *of (seed and target) metabolites, and a reference network* $(R' \cup M', E')$. *A* completion *is a set of reactions* $R'' \subseteq R' \setminus R$ *such that* $T \subseteq \Sigma_G(S)$ *where*

$$\begin{aligned}
G &= ((R \cup R'') \cup (M \cup M''), E \cup E'') \;, \\
M'' &= \{m \in M' \mid r \in R'', m \in \mathit{reac}(r) \cup \mathit{prod}(r)\} \;, \text{ and} \\
E'' &= \{(m, r) \in E' \mid r \in R'', m \in \mathit{reac}(r)\} \cup \{(r, m) \in E' \mid r \in R'', m \in \mathit{prod}(r)\} \;.
\end{aligned}$$

Two optimization variants of this problem are obtained by finding cardinality or subset minimal sets of reactions. Further refinements may also optimize on the distance between seeds and targets or minimize forbidden side products.

Example 3.1 (Completion) *Consider the metabolic network* $G_d = (R \cup M, E)$ *as shown in Figure 3.2, a set* $S = \{m1, m2, m12\}$ *of seeds, a set* $T = \{m10, m5\}$ *of target metabolites. The scope* $T \subseteq \Sigma_{G_d}(S) = \{m1, m2, m3, m4, m5, m6, m13\}$ *contains the target* $m5$ *but not the target* $m10$.

Given the reference network $G_r = (R' \cup M', E')$ *where*

$$\begin{aligned}
R' &= \{r7, r8, r9, r10\;\} \;, \\
M' &= \{m1, m5, m7, m8, m10, m12, m13, m15\;\} \text{ and} \\
E' &= \{(m1, r10), (r10, m7), (r10, m8), (m13, r7), (r7, m5), \\
&\quad (m12, r8), (r8, m15), (m15, r9), (r9, m10)\;\}.
\end{aligned}$$

There exist 2 subset minimal completions $R'_1 = \{r10\}$ *and* $R'_2 = \{r8, r9\}$ *such that all targets are in the scope of the expanded network. The completions ar shown in Figure 3.3. The reaction* $r7$ *is in no minimal completion as it does not expand the scope of the network, and completions* R'_1 *is the only cardinality minimal completion.*

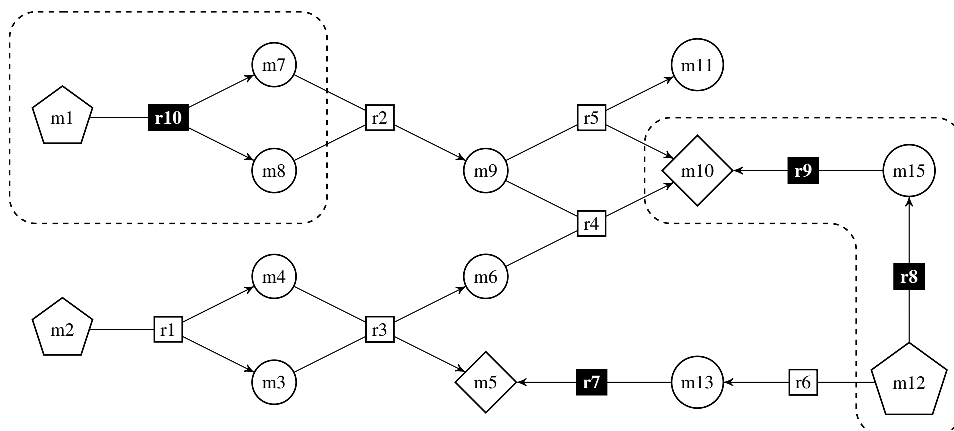

Figure 3.3: Given the seed metabolites m1, m2 and m12, the completions {r10} and {r8, r9} can repair the production pathway for target m10.

3.3 Metabolic network completion

We now address how to solve the metabolic network completion problem by means of ASP. Given a metabolic draft network, sets of seed and target metabolites, and a reference network, we compute minimal completions for the draft network, that restore its capability to produce the target metabolites from the given seeds. We next provide a logic program such that each of its answer sets matches a minimal completion of the metabolic network. The logic program is composed of three parts, described in the following subsections.

3.3.1 Problem Instance

We now explain how we represent a metabolic network as well as seed and target metabolites as a set of ground facts. A complete instance of a metabolic network completion problem consists of a metabolic draft network G_d along with a set S of seeds, a set T of targets, and a reference network G_r.

Given a metabolic network $G_n = (R \cup M, E)$ where n is the name of the network, we introduce for each reaction $r \in R$ a fact *reaction*(r, n), for each reactant $m = reac(r)$ an fact *reactant*(m, r), and for each product $p = prod(r)$ a fact *product*(p, r). If the network is our current draft network we also introduce the fact *draft*(n) For each seed metabolite $m \in S$ we introduce a fact *seed*(m), and for each target $m \in T$ a fact *target*(m) respectively. In the following, we let $\tau(G_d, G_r, T, S)$ denote the set of facts representing the problem instance induced by metabolic networks G_d, G_r, seeds S and targets T.

Example 3.2 *The metabolic network problem from Example 3.1, which is illustrated in Figure 3.3 consitute the following logic programming instance* $\tau(G_n, G_r, T, S)$.

$$\tau(G_n, G_r, T, S) = \begin{cases} \textit{draft}(d). \\ \textit{reaction}(r1,d). & \textit{reaction}(r2,d). & \textit{reaction}(r3,d). & \textit{reaction}(r4,d). & \textit{reaction}(r5,d). \\ \textit{reaction}(r6,d). \\ \textit{reactant}(m2,r1). & \textit{reactant}(m7,r2). & \textit{reactant}(m8,r2). & \textit{reactant}(m3,r3). & \textit{reactant}(m4,r3). \\ \textit{reactant}(m6,r4). & \textit{reactant}(m9,r4). & \textit{reactant}(m9,r5). & \textit{reactant}(m12,r6). \\ \textit{product}(m4,r1). & \textit{product}(m3,r1). & \textit{product}(m9,r2). & \textit{product}(m5,r3). & \textit{product}(m6,r3). \\ \textit{product}(m10,r4). & \textit{product}(m10,r5). & \textit{product}(m11,r5). & \textit{product}(m13,r6). \\ \textit{reaction}(r7,r). & \textit{reaction}(r8,r). & \textit{reaction}(r9,r). & \textit{reaction}(r10,r) \\ \textit{reactant}(m13,r7). & \textit{reactant}(m12,r8). & \textit{reactant}(m15,r9). & \textit{reactant}(m1,r10). \\ \textit{product}(m5,r7). & \textit{product}(m15,r8). & \textit{product}(m10,r9). & \textit{product}(m7,r10). & \textit{product}(m8,r10). \\ \textit{seed}(m1). & \textit{seed}(m2). & \textit{seed}(m12). \\ \textit{target}(m5). & \textit{target}(m10). \end{cases}$$

3.3.2 Scope and Potential Scope

The scope of the seed metabolites in the draft network G_d can be determined by the following rules.

$$\begin{array}{lcl} \textit{dscope}(M) & \leftarrow & \textit{seed}(M). \\ \textit{dscope}(M) & \leftarrow & \textit{product}(M,R), \textit{reaction}(R,N), \textit{draft}(N), \\ & & \textit{dscope}(M') : \textit{reactant}(M',R). \end{array} \tag{3.1}$$

The first rule declares all seed metabolites $M \in S$ as producible. The second rule defines recursively that a product M of a reaction R is producible, whenever all reactants M' of R are available. Together with the encoding of draft network G_d and seeds S as described in Section 3.3.1, the set of rules in (3.1) results in a single answer set X such that $\textit{dscope}(m) \in X$ iff $m \in \Sigma_{G_d}(S)$ for $m \in M(G_d)$.

While drafting a metabolic network of an organism biologists are regularly confronted with experiments that show that a certain metabolite can be measured, although it is not producible by the current draft network. To this end, they incorporate metabolic reactions known from metabolic networks of other organisms. In analogy to the rules in (3.1), the (potential) scope of the seed metabolites in the draft network G_d augmented by the reference network G_r can be determined as follows.

$$\begin{array}{lcl} \textit{pscope}(M) & \leftarrow & \textit{seed}(M). \\ \textit{pscope}(M) & \leftarrow & \textit{product}(M,R), \textit{reaction}(R,N), \\ & & \textit{pscope}(M') : \textit{reactant}(M',R). \end{array} \tag{3.2}$$

Note that dropping the qualification $\textit{draft}(N)$ from (3.1) makes us use all available reactions. As before, given the logic encoding of the problem instance, the set of rules in (3.2) induces a single answer set X such that $\textit{pscope}(m) \in X$ iff $m \in \Sigma_{G_d \cup G_r}(S)$ for $m \in M$ where $G_d \cup G_r$ stands for the pairwise union of G_d and G_r.

While the scope of the draft network in (3.1) gives a lower limit on the metabolites producible from the seeds by the draft network, the potential scope obtained from the augmented network in (3.2) constitutes an upper limit. Note that targets outside the potential scope cannot be explained.

3.3.3 Metabolic Network Completion.

The goal of metabolic network completion is to extend the draft network with reactions from the reference network, so that the target metabolites can be synthesized by the augmented network from the seeds. The reactions of interest belong to the reference network but not the draft network. The following choice rule captures all candidate reactions.

$$\{\mathit{xreaction}(R) : \mathit{reaction}(R,N) : \mathit{not}\ \mathit{draft}(N)\}. \tag{3.3}$$

The condition $\mathit{reaction}(R,N) : \mathit{not}\ \mathit{draft}(N)$ guarantees that all chosen reactions belong to $R(G_r) \setminus R(G_d)$. In fact, the instance encoding from Section 3.3.1 and the choice rule in (3.3) give a set of answer sets being in a one-to-one correspondence to the subsets of $R(G_r) \setminus R(G_d)$.

The (extended) scope of the seed metabolites in the draft network G_d extended by reactions from G_r is defined as follows.

$$\begin{array}{lcl} \mathit{xscope}(M) & \leftarrow & \mathit{seed}(M). \\ \mathit{xscope}(M) & \leftarrow & \mathit{product}(M,R), \mathit{reaction}(R,N), \mathit{draft}(N), \\ & & \mathit{xscope}(M') : \mathit{reactant}(M',R). \\ \mathit{xscope}(M) & \leftarrow & \mathit{product}(M,R), \mathit{xreaction}(R), \\ & & \mathit{xscope}(M') : \mathit{reactant}(M',R). \end{array} \tag{3.4}$$

Finally, we have to make sure that an extended scope is able to produce all target metabolites. This is addressed by the following integrity constraint. [1]

$$\leftarrow \mathit{target}(M), \mathit{not}\ \mathit{xscope}(M). \tag{3.5}$$

Given the above rules, each of its answer set corresponds to a completion of the draft network and vice versa.

Theorem 3.1 *Let G_d and G_r be metabolic networks and let S and T be sets of metabolites.*

If X is an answer set of logic program[2] $\tau(G_d, G_r, S, T) \cup \{(3.3),(3.4),(3.5)\}$, then $\{r \mid \mathit{xreaction}(r) \in X\}$ is a completion of G_d from G_r wrt (S,T) and vice versa.

3.3.4 Refined Metabolic Network Completion.

Although the above encoding is formally adequate, it suffers from too many uninteresting completions that makes it fail to scale on large metabolic networks comprising several thousand metabolites. We address this problem by some refinements reducing the set of candidate reactions.

At first, we restrict the choice in (3.3) to "interesting" reactions.

$$\leftarrow \quad \mathit{xreaction}(R), \mathit{not}\ \mathit{ireaction}(R). \tag{3.6}$$

The qualification expressed by $\mathit{ireaction}(R)$ requires that a reaction of interest must lead

[1] In practice, this constraint is extended by $\mathit{pscope}(M)$ to ignore non-producible targets.

[2] Recall that a rule with variables stands for the set of all its ground instantiations.

to some target metabolites.

$$\begin{array}{rcl} \mathit{ireaction}(R) & \leftarrow & \mathit{interesting}(M), \\ & & \mathit{product}(M,R), \mathit{reaction}(R,N). \\ \mathit{interesting}(M) & \leftarrow & \mathit{target}(M), \mathit{not}\ \mathit{dscope}(M). \\ \mathit{interesting}(M) & \leftarrow & \mathit{reactant}(M,R), \mathit{ireaction}(R), \\ & & \mathit{not}\ \mathit{dscope}(M). \end{array} \tag{3.7}$$

With the first rule we declare a reaction as interesting if it produces interesting metabolites. The second rule defines all target metabolites that cannot be produced by the draft network as interesting, and the third rule states that metabolites needed by interesting reactions and not producible by the draft network are interesting. This concept provides a significant reduction of the set of candidate reactions in view of the given target metabolites.

Second, we further restrict the choice in (3.3) to "operable" reactions.

$$\begin{array}{rcl} & \leftarrow & \mathit{xreaction}(R), \mathit{not}\ \mathit{oreaction}(R). \\ \mathit{oreaction}(R) & \leftarrow & \mathit{xscope}(M) : \mathit{reactant}(M,R), \\ & & \mathit{reaction}(R,N), \mathit{not}\ \mathit{draft}(N). \end{array} \tag{3.8}$$

The integrity constraint enforces that each extending reaction is operable, that is, satisfies $\mathit{oreaction}(R)$. The following rule defines a (candidate) reaction as operable, if all its reactants are producible by the current network extension.

The next result shows that the above refinements preserve soundness.

Proposition 3.1 *Let G_d and G_r be metabolic networks and let S and T be sets of metabolites.*

If X is an answer set of $\tau(G_d, G_r, S, T) \cup \{(3.1), (3.3), (3.4), (3.5), (3.6), (3.7), (3.8)\}$, then
$\{r \mid \mathit{xreaction}(r) \in X\}$ is a completion of G_d from G_r wrt (S, T).

A further natural way to reduce the number of solutions is to concentrate on network completions containing the fewest number of reactions. In ASP, this can be accomplished by the following minimize statement.

$$\mathit{minimize}\ \{\mathit{xreaction}(R) : \mathit{ireaction}(R) : \mathit{not}\ \mathit{reaction}(R,N)\}. \tag{3.9}$$

Interestingly, our refinements are satisfied by such minimal completions, so that we get a soundness and completeness result under optimization.

Proposition 3.2 *Let G_d and G_r be metabolic networks and let S and T be sets of metabolites.*

If X is an answer set of $\tau(G_d, G_r, S, T) \cup \{(3.1), (3.3), (3.4), (3.5), (3.6), (3.7), (3.8)\} \cup \{(3.9)\}$, then $\{r \mid \mathit{xreaction}(r) \in X\}$ is a minimum completion of G_d from G_r wrt (S, T) and vice versa.

Sometimes reactions can be associated with confidence levels, for instance, obtained from the proximity of their host organism to the organism addressed by the draft network. This allows us to prefer among the minimum completions those composed of reactions

with higher confidence levels; this is accomplished by adding the following statement.

$$maximize\{\textit{\textbf{xreaction}}(R)\!=\!L\!:\!\textit{\textbf{ireaction}}(R)\!:\!not\ \textit{\textbf{reaction}}(R,N)\!:\!confidence(R,L)\}.$$

3.3.5 Reasoning Modes.

Given the above ensemble of rules, the reasoning modes of ASP solvers allow us to answer a variety of additional biologically relevant questions. What target metabolites are producible by the draft network? What new metabolites can be produced by adding reactions from other pathways? What is the minimal number of reactions that must be added to explain a target metabolite? What are the minimum or minimal extended scopes? Which reactions belong to all extended scopes, or even all minimum extended scopes? The latter are accomplished by a combination of optimization and cautious reasoning. We return to these question in Section 3.5 and show how they are realized. The next section shows how certain seeds or side-products can either be avoided or minimized.

3.4 Inverse Scope Problem

In this section, we address a problem similar to metabolic network completion, the *inverse scope problem*. In the *inverse scope problem*, a metabolic network is given and a the goal is to determine *nutrition sets* (seed metabolites) such that a set of desired (target) metabolites can be produced.

Definition 3.4 (Nutrition Set) *Given a metabolic network $(R \cup M, E)$ along with a set $T \subseteq M$ of target metabolites, a* nutrition set *is a set of metabolites $S \subseteq M$ such that $T \subseteq \Sigma_{(R \cup M,E)}(S)$*

Again, two optimization variants of this problem are aiming at finding a cardinality or subset minimal solution.

Example 3.3 *Consider the metabolic network $G = (R \cup M, E)$ as shown in Figure 3.4 and a set $T = \{m10, m5\}$ of target metabolites. There exist 4 subset minimal nutrition sets, $S_1 = \{m12\}$, $S_2 = \{m11, m13\}$, $S_3 = \{m1, m9, m13\}$, and $S_4 = \{m7, m9, m13\}$, with S_1 being the only cardinality minimal nutrition set.*

Furthermore, two variations of the *inverse scope problem* can be distinguished [88]. The first variant restricts the domain of the available seed metabolites. In addition to $(R \cup M, E)$ and $T \subseteq M$, we are given a set of (forbidden) metabolites $F \subseteq M$. Then, the goal is to find a set of (seed) metabolites $S \subseteq (M \setminus F)$ such that $T \subseteq \Sigma(S)$. Apart from optimizing the required seed metabolites, one may also minimize undesired metabolites rather then excluding them. The second variant adds an additional constraint on the avoidance of side products. In addition to $(R \cup M, E)$ and $T, F \subseteq M$, we are given another set of (forbidden) metabolites $E \subseteq M$. Then, the goal is to find a set of (seed) metabolites $S \subseteq (M \setminus F)$ such that $T \subseteq \Sigma(S)$ and $\Sigma(S) \cap E = \emptyset$. As above, the optimization variants can also take side products into account.

We next provide a logic program such that each of its answer sets matches a minimal nutrition set of the metabolic network for the given target metabolites.

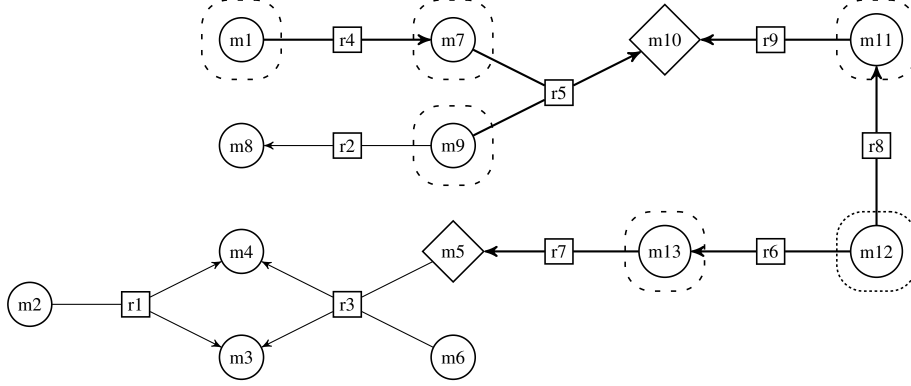

Figure 3.4: Given the target metabolites m10 and m5, the minimal nutrition sets are {m12},{m11,m13}, {m1,m9,m13},{m7,m9,m13}.

3.4.1 Basic Setting

An instance of an *inverse scope problem* consists of a metabolic draft network G_d along with a set T of target metabolites. Reactions, targets, and seeds are represented as shown in Section 3.3.1. In the following, we let $\tau(G_d, T)$ denote the set of facts representing the problem instance. By appeal to the encoding of the basic scope in (3.1), we can then express our task similar to the completion problem by exchanging the roles of reactions and seed metabolites.

$$\{\mathit{seed}(M) : \mathit{not}\ \mathit{target}(M), \mathit{reactant}(M, R)\}. \tag{3.10}$$

$$\leftarrow \mathit{target}(M), \mathit{not}\ \mathit{pscope}(M). \tag{3.11}$$

Similar to (3.3), the choice construct in (3.10) captures the seed candidates, while the integrity constraint in (3.11) together with the rules in (3.2) makes sure that all target metabolites can by synthesized from the seeds chosen in (3.10).

The next proposition shows that our encoding is sound and complete.

Proposition 3.3 *Let G_n be a metabolic network and let S and T be sets of metabolites.*

If X is an answer set of logic program $\tau(G_d, T) \cup \{(3.1), (3.10), (3.11)\}$, then $T \subseteq \Sigma(\{m \mid \mathit{seed}(m) \in X\})$ and vice versa.

3.4.2 Refined Setting

As above, some refinements lend themselves for reducing the putative seed metabolites.

$$\leftarrow \quad \mathit{seed}(M), \mathit{not}\ \mathit{imetabolite}(M). \tag{3.12}$$

A metabolite of interest, viz *imetabolite*(M), must lead to at least one target metabolite.

$$\begin{aligned} \mathit{imetabolite}(M) &\leftarrow \mathit{target}(M). \\ \mathit{imetabolite}(M) &\leftarrow \mathit{reactant}(M, R), \mathit{ireaction}(R). \\ \mathit{ireaction}(R) &\leftarrow \mathit{imetabolite}(M), \mathit{product}(M, R), \mathit{reaction}(R, N). \end{aligned} \tag{3.13}$$

The first rule defines target metabolites as interesting. The second one extends this to metabolites being reactants of interesting reactions. Similar to (3.7), the last rule states

that interesting reactions are those that produce interesting metabolites.

Although the last refinement eliminates (uninteresting) solutions, it preserves minimum ones. Hence, cardinality minimum solutions to the inverse seed problem are obtained by simply adding the following optimization statement.

$$minimize\{\textit{seed}(M)\}.$$

3.4.3 Avoiding Side or Seed Metabolites

The elaboration of biosynthetic capacities is often subject to further restrictions, for instance, avoiding seed metabolites or certain side products. This has led to the definition of the two variants of the inverse scope problem defined in Section 3.4.

Both problems are easily addressed in ASP, once a metabolite, m, is declared as being forbidden, viz. $\textit{forbidden}(m)$:

$$\begin{array}{l} \leftarrow \ \textit{seed}(M), \textit{forbidden}(M). \\ \leftarrow \ \textit{pscope}(M), \textit{forbidden}(M). \end{array}$$

While the first constraint eliminates forbidden metabolites from the seeds, the second rules out unwanted side products.

The complete exclusion of certain metabolites is sometimes to restrictive. To this end, one may replace one or both of the previous integrity constraint by appropriate minimization statements:

$$\begin{array}{r} minimize\{\textit{seed}(M) : \textit{forbidden}(M)\}. \\ minimize\{\textit{pscope}(M) : \textit{forbidden}(M)\}. \end{array}$$

Recall that the order of the two statements determines their precedence.

3.4.4 Reasoning Modes

The inverse scope problem usually leads to numerous solutions. Cautious reasoning allows us to compute the ultimately essential seeds belonging to all solutions. Also, brave reasoning is of interest because often solutions are similar, so that the union of all seeds in a solution form a pool of potentially relevant nutrients. Finally, in view of the numerous, often unrelated combinatorial sources, an important role is played by projective solution enumeration [50] for eliminating redundant solutions.

3.5 Empirical Evaluation

For validating our approach, we investigate the metabolic network of *Escherichia coli* (E.coli). This choice is motivated by the fact that E.coli is a well studied organism, whose metabolic network is of moderate size, consisting of 3645 reactions and 1556 metabolites. Our experiments consider furthermore 94 seed metabolites and 28 target metabolites. The targets and seeds were chosen by our biological partners in view of the fact that E.coli is able to grow when glucose is the only carbon source. Hence, its metabolic network must be able to synthesize all necessary precursors for high-level processes, from glucose and

inorganic material [19]. That is why the targets contain all 20 amino acids, the nucleotide phosphates ATP, CTP, GTP and UTP as well as the deoxy forms dATP, dGTP, dUTP and dTTP; and the seeds are only glucose and inorganic metabolites. In fact, all considered targets could be produced by the original E.coli network. This setup allows us to control and vary our experiments by producing draft networks through eliminating reactions from E.coli's original network.

All experiments[3] were run with ASP grounder *gringo* (2.0.2) and ASP solver *clasp* (1.2.0) on a Linux PC with a Core2DuoE6400 processor and 2GB memory. The computation time was limited to 600 seconds, timeouts are shown throughout as "-".

3.5.1 Metabolic network completion

For our experiments on network completion, biologist provided us with draft networks. The draft networks have been created with biological background knowledge, by removing 50, 100 and 200 reactions from the original E.coli network. Also, derived reactions have been removed by the biologists. This means, for example, that for reversible reactions also the inverse reactions were removed, and for reactions that are generalizations, all subsumed special cases were removed as well. The resulting networks failed to produce 7, 10, and 20 targets, respectively. As reference network, we have chosen the entire MetaCyc[4] database containing 13882 reactions. This set of reactions spans the search space specified in (3.3) for metabolic network completion.

In the first set of experiments, we proceed in two steps. First, we compute for each draft network and each target, the minimum number of reactions that need to be added to complete the network. Then, we compute all solutions satisfying this optimality criterion. In fact, in view of the large set of candidate reactions in the reference network, this approach turned out to be superior to a single step approach, enumerating all optimal solution through *clasp*'s branch and bound algorithm. Rather, we invoke *clasp* with the option `--restart-on-model` that restarts after each minimum solution. This makes *clasp* converge much faster to an optimum solution. Once this is found, *clasp* is invoked again for enumerating all solutions satisfying the optimality criterion.

Table 3.1 summarizes our first set of experiments. The columns headed by E.coli-50, E.coli-100, and E.coli-200, respectively, provide results obtained on the aforementioned draft networks obtained by removing 50, 100 and 200, respectively, reactions from the original E.coli network. The first column identifies the chosen target metabolite. Then, for each draft network, the columns labeled t_{opt} show the time in seconds for computing the minimum number of reactions that need to be added to produce the target. The columns labeled *opt* provide the minimum number of reactions. The columns t_{all} show the time in seconds for computing all optimal solutions and the column *#opt* gives the number of optimal solutions.

For targets that could not be produced by the draft network, the results are either shown in boldface or are timeouts. For target metabolites whose production pathways are not disturbed, the computation time is insignificant. We observe six timeouts, while searching for an optimal completion on the E.coli-50 network. These six target metabolites could not be produced by the draft network in general. Interestingly, those metabolites

[3] Instances and encodings are available at: `http://www.cs.uni-potsdam.de/bioasp`
[4] `http://metacyc.org`

	E.coli-50				E.coli-100				E.coli-200			
T	t_{opt}	*opt*	t_{all}	*#opt*	t_{opt}	*opt*	t_{all}	*#opt*	t_{opt}	*opt*	t_{all}	*#opt*
1	0.14	0	0.17	1	0.19	0	0.16	1	**368.72**	**1**	**2.17**	**7**
2	0.18	0	0.17	1	0.18	0	0.16	1	**368.52**	**1**	**2.22**	**7**
3	0.16	0	0.16	1	0.17	0	0.18	1	**195.34**	**7**	**304.35**	**135**
4	0.20	0	0.17	1	0.21	0	0.18	1	**42.89**	**3**	**20.40**	**35**
5	0.18	0	0.16	1	0.18	0	0.14	1	0.15	0	0.15	1
6	0.16	0	0.16	1	**159.07**	**2**	**2.53**	**6**	**226.41**	**7**	-	-
7	0.16	0	0.18	1	0.21	0	0.16	1	-	-	-	-
8	0.15	0	0.14	1	0.17	0	0.15	1	**46.39**	**3**	**29.59**	**35**
9	0.16	0	0.19	1	0.14	0	0.15	1	0.15	0	0.14	1
10	0.18	0	0.17	1	0.14	0	0.16	1	0.14	0	0.17	1
11	0.18	0	0.18	1	0.15	0	0.15	1	**26.58**	1	**2.18**	**7**
12	0.14	0	0.16	1	0.15	0	0.16	1	-	-	-	-
13	-	-	-	-	**105.15**	**4**	**12.35**	**1**	-	-	-	-
14	0.17	0	0.18	1	0.15	0	0.17	1	0.16	0	0.14	1
15	0.13	0	0.19	1	0.16	0	0.17	1	0.18	0	0.16	1
16	0.15	0	0.16	1	0.16	0	0.18	1	**367.10**	**1**	**2.20**	**7**
17	0.20	0	0.16	1	-	-	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-
19	-	-	-	-	**80.63**	**2**	**5.18**	**3**	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-
21	0.18	0	0.17	1	0.15	0	0.15	1	0.16	0	0.15	1
22	0.16	0	0.17	1	0.19	0	0.16	1	0.14	0	0.15	1
23	0.17	0	0.14	1	0.16	0	0.15	1	**353.70**	**1**	**2.17**	**1**
24	**37.87**	**3**	**21.28**	**4**	**3.92**	**6**	**29.78**	**5**	-	-	-	-
25	-	-	-	-	-	-	-	-	-	-	-	-
26	-	-	-	-	-	-	-	-	-	-	-	-
27	0.15	0	0.17	1	0.14	0	0.18	1	**46.07**	**3**	**37.08**	**35**
28	0.16	0	0.19	1	-	-	-	-	0.16	0	0.13	1

Table 3.1: computing optimal completions for E. coli networks

cannot be produced by all three draft networks, giving us the hint that the pathways for this metabolites are very fragile. Comparing the results for E.coli-50 and E.coli-100, we see that for two targets, the experiments on E.coli-50 timeout, while they could be solved in time on E.coli-100. For E.coli-200, we see 10 experiments timeout, 10 computing the optimal value in time, and for 9 experiments *clasp* finishes computing all optimal solutions in time. This suggests that pathways, which can be disturbed by removing few reactions, are very fragile and hard to reconstruct, while more robust pathways, which are only disturbed when removing lots of reactions, are more easily repaired. For target metabolites whose production pathways are not disturbed, the computation time is insignificant

In our second experiment, we investigate the scalability of our approach in view of the size of the reference network, taking into account the entire set of target metabolites. We created subsets of the MetaCyc network, choosing 10 random samples of 5000, 6000, 7000, 8000, and 9000 reactions. We fixed the draft network by removing 200 reactions from the E.coli network and tried to complete its completion relative to the differently large reference networks. Note that the joint explanation of all 28 targets is much more

difficult than just explaining a single target. This is because the restrictions to interesting reactions introduced in Section 3.3 become less effective when aiming at multiple targets. On the other hand, the identification of a minimum completion producing a maximum set of target is a highly significant question in synthetic biology.

As above, our experiments use a multi-step process. In a first step, we use *clasp* to compute for each reference network the minimum number of reactions needed to complete the network. Once we have computed the optimal value, we continue by computing the reactions essential to all 28 targets, that is, the reactions contained in every answer set satisfying the optimality criterion. This is accomplished by computing the cautious consequences using the option `--cautious` of *clasp*. These reactions are essential for the joint production of all target metabolites. Finally, we use *clasp* as before to enumerate all optimal solutions.

The first line gives the size of the investigated reference network. The columns labeled with i identify the instance of the reference network. The column t_{opt} gives the computation time for computing the minimum number of reactions needed for a completion. Column t_c shows the time needed to compute the essential reactions, that is, all reactions that are in all minimum completion. The columns labeled with t_{all} show the time needed to compute all optimal solutions. The columns labeled with *#opt* show how many optimal solution have been found. All times are given in seconds.

We observe that the problem is easily handled up to a size of 6000 reactions; all such problems can be solved under a second. Starting with 7000 reactions, we start to obtain computational more demanding problems, and finally a lot of timeouts at size 9000. Notably, our experiments are restricted by a timeout of 10 minutes; existing approaches to network completion usually run simulations over the period of a day. Of course, we have to extend the timeout in a production mode as well. Interestingly, the successful runs show that finding the optimal number of solutions takes most of the computation time; an issue we want to address in the future by biological domain-specific heuristics.

3.5.2 Inverse scope problem

Finaly, we evaluate our approach to the inverse scope problem. As above, we consider the complete E.coli network and try to compute for every target the minimum number of seeds needed to produce it. Once accomplished, we enumerate all minimum sets of seeds.

Again, we first solve the optimality problem and use *clasp* to compute the minimum number of seeds needed to produce the target metabolite; and in a second step we relaunch *clasp* to compute all optimal solutions.

The first column denotes the target metabolite for whose production the seeds are computed. The second column shows the time in seconds for computing the minimum number of seeds. The third one gives the minimum number of seeds. The fourth column shows the time in seconds for computing all optimal solutions, and the fifth one shows the number of optimal solutions.

The results show that most of the targets can be produced by providing one or two seeds only. Interestingly, we found that only groups of three seeds are needed to produce all 28 targets. We also checked with the cautious reasoning mode for essential seeds, belonging to all minimum solution but none were found. We further used *clasp* with

option `--brave` to compute the union of all reactions occurring in optimal solutions and found a set of 136 different metabolites, from which all minimum sets of seeds are taken. Since we are only discriminating the targets among the seeds in (3.10), we were surprised to find many seeds among the reactants of the reactions producing the targets. However, for more meaningful results, we need more biological knowledge, to exclude more metabolites as seeds.

3.6 Discussion

The easy characterization of reachability is one of the key features of ASP. We have exploited this to provide a simple yet powerful account of metabolic network synthesis, a crucial application in the elaboration and design of bioprocesses. The distinguishing feature of our ASP-based approach lies in the unique combination of ease of modelling and powerful reasoning modes, supported by efficient solver technology. In fact, existing qualitative approaches to network synthesis are based on stochastic simulations based on hidden Markov models (cf. [20]), taking several hours to obtain results from the relative frequencies of compounds in the simulations. Unlike this, our approach is complete and thus allows for proving rather than estimating the production of metabolites. Moreover, the various reasoning modes, including the enumeration of optimal solutions as well as cautious and brave reasoning with respect to all or optimal solutions only, respectively, are indispensable in a biological application due to the large number of possible solutions. For instance, cautious reasoning relative to optimal solutions makes us discover the essential nutritions for producing a target metabolite. These reasoning modes together with the high-level specification of metabolic networks make our approach attractive to biologists, given that they can easily elaborate and explore their model “in silico“ by means of ASP.

From the perspective of ASP, our application fostered the development of new reasoning modes that were implemented within the ASP solver *clasp*. For one thing, *clasp* allows for optimization techniques not available in any other ASP solver. Of particular interest is the `--restart-on-model` option that restarts after finding a solution (instead of backtracking). This led to a significant increase in converging to an optimal solution. To a turn, we then exploit the options `--opt-all` and `--opt-value` for enumerating all optimal models. Even though the latter can also be addressed by adding an appropriate constraint to the underlying ASP program, the options allow us to leave the underlying program untouched. For another thing, *clasp* allows for computing all brave and cautious consequences by means of a linear number of calls to a solver (internally computing one answer set) rather then enumerating the entire set of answer sets. This is accomplished by consecutive refinements of an internal constraint by appeal to the incremental solving techniques introduced in [43]. This feature is also unique to *clasp*, although a-priori given brave and cautious queries can be decided by other ASP solvers, like dlv [81], as well. Although, to the best of our knowledge, our application is novel in the field of ASP in particular and declarative programming in general, there has been an increasing interest in using ASP and/or LP technology for addressing biological problems over the last years. Among them, we find [6, 29, 35] as well as [90] building upon abductive logic programming.

5000					
i	t_{opt}	*opt*	t_c	t_{all}	*#opt*
1	0.25	5	0.19	0.21	72
2	0.23	8	0.16	0.19	36
3	0.20	5	0.16	0.20	3
4	0.14	11	0.20	0.17	12
5	0.18	3	0.16	0.14	24
6	0.39	11	0.26	0.26	24
7	0.18	4	0.18	0.15	2
8	0.18	9	0.22	0.23	60
9	0.27	15	0.19	0.20	16
10	0.15	3	0.16	0.14	6

6000					
i	t_{opt}	*opt*	t_c	t_{all}	*#opt*
1	0.24	4	0.20	0.20	6
2	0.28	4	0.23	0.19	3
3	3.46	16	0.46	0.43	50
4	0.17	7	0.21	0.22	6
5	0.19	2	0.21	0.23	4
6	0.54	17	0.26	0.28	28
7	0.23	5	0.21	0.21	8
8	0.29	19	0.18	0.26	55
9	0.43	9	0.23	0.27	24
10	0.18	3	0.16	0.18	30

7000					
i	t_{opt}	*opt*	t_c	t_{all}	*#opt*
1	105.16	27	3.24	3.33	160
2	-				
3	10.82	19	2.15	1.86	48
4	0.38	5	0.36	0.38	168
5	0.83	14	0.46	0.44	27
6	0.58	7	0.42	1.15	10
7	0.30	2	0.27	0.23	3
8	16.12	14	0.38	0.54	88
9	58.00	17	1.39	0.89	300
10	11.20	18	9.40	8.28	80

8000					
i	t_{opt}	*opt*	t_c	t_{all}	*#opt*
1	265.14	15	274.56	251.34	672
2	1.07	7	0.25	0.30	5
3	5.16	13	1.23	1.13	4
4	1.50	8	0.36	0.35	10
5	0.68	13	0.87	0.98	12
6	78.49	20	48.91	49.67	288
7	195.66	8	1.77	1.58	40
8	5.98	15	3.44	3.63	24
9	9.08	11	0.53	0.59	8
10	0.89	11	0.48	0.42	12

9000					
i	t_{opt}	*opt*	t_c	t_{all}	*#opt*
1	12.34	17	7.45	8.95	18
2	-				
3	28.05	12	11.32	13.99	88
4	-				
5	-				
6	410.76	30	3.88	3.79	14
7	271.02	16	11.13	28.61	2976
8	-				
9	-				
10	-				

Table 3.2: Completion with 5000,...,9000 reactions.

T	t_{opt}	opt	t_{all}	$\#opt$
1	2.40	1	14.82	6
2	0.45	1	35.76	12
3	0.38	1	16.02	6
4	28.21	1	25.42	4
5	19.41	2	-	-
6	4.30	2	187.06	50
7	1.29	2	166.73	63
8	15.79	1	17.24	4
9	13.45	1	13.98	4
10	0.89	1	17.00	5
11	0.53	1	25.92	9
12	7.28	1	14.78	4
13	4.78	1	9.88	4
14	13.23	1	7.67	4
15	0.32	1	29.58	11
16	0.32	1	31.16	11
17	14.05	1	24.46	1
18	0.28	1	19.66	3
19	10.44	2	-	-
20	23.33	1	27.58	5
21	14.23	1	6.90	4
22	0.37	1	49.79	11
23	-	-	-	-
24	-	-	-	-
25	17.19	1	21.36	4
26	0.55	1	33.24	5
27	19.85	1	15.22	4
28	-	-	-	-

Table 3.3: Computing minimal seeds for E.coli targets.

4 The BioASP Library

In the previous chapters, we have shown some of the challenges that modern biology poses to today's bioinformatics. By means of ASP, we invented methods coping with incompleteness, heterogeneity, and mutual inconsistency of data and models. We have shown that ASP is an excellent tool for solving a variety of biological questions. In this chapter, we present the BioASP library, which implements the methods for analyzing metabolic and gene regulatory networks, consistency checking, diagnosing, and repairing biological data and models. In particular, it allows for computing predictions and generating hypotheses about required expansions of biological models. To accomplish this, the library combines expert knowledge of both the biological application and the ASP paradigm. With the BioASP library, we offer our practical experience via easy-to-use Python functions, thus enabling ASP non-experts to solve biological questions with ASP.

Solving biological questions often requires combining several computational steps and thus integrating ASP with traditional programming paradigms. Raw input data in various formats needs to be parsed and preprocessed. The optimal solution of a problem needs to be determined, and eventually one needs to compute the intersection of all optimal solutions. Sometimes, also different ASP solvers have to be combined in a chain of computations. For example, one first checks the consistency of a biological model using the solver *clasp* [48] and then computes minimal diagnoses with the solver *claspD* [28]. Here, the different computational complexities of the involved tasks (NP versus NP^{NP}) lead to distinct adequate solving approaches. Moreover, the solution to one subproblem may be needed as input to solve a second subproblem. Last but not least, obtained results must be visualized in a user-friendly way. To accommodate such complex application scenarios, we created the library BioASP, written in Python. It provides functionalities for parsing inputs in several formats and transforming them into ASP facts. It encapsulates the grounder *gringo* [44, 45] as well as the solvers *clasp* and *claspD* into Python objects. In particular, these objects can be fed with logic programs describing different tasks, be launched with dedicated parameter settings, and return results for further processing. Thus, the BioASP library provides a framework for the convenient use of ASP within the imperative programming paradigm of Python.

We will outline the system architecture of the BioASP library in Section 4.1, and give detailed descriptions of the functionalities provided by the application programming interface (API). In Section 4.2, we then present some applications built on top of the BioASP library and show how the library is applied to solve the specific biological problems described in the previous chapters. Finally, we conclude with a discussion in Section 4.3.

4.1 System Architecture

The BioASP library originates from our research in Systems Biology, where ASP has proven to be an effective tool for modeling and solving a variety of questions. However, to produce solutions based on ASP, it is often necessary to integrate it with existing environments and traditional programming paradigms. Python is a flexible and extensible scientific programming language used in various applications. For example, the BioPython project [21] provides solutions for transforming biological inputs into Python-utilizable data structures and offers interfaces to common bioinformatics programs. It implements tools to work with sequence data, performs standard machine learning tasks, and integrates with BioSQL, a sequence database schema also supported by the BioPerl [105] and BioJava [70] projects.

In order to make the power of ASP accessible within an existing rich system environment, BioASP provides classes encapsulating ASP tools: the grounder *gringo* as well as the solvers *clasp* and *claspD*. For the library to work, it is required that binaries of these systems are installed. For correct functioning of the library, we have to make sure that the right versions of *gringo*, *clasp* and *claspD* are used such that all required features are provided and that the command line switches work as expected. Therefore, we ship the BioASP Library containing its own set of binaries of *gringo*, *clasp* and *claspD*.

In our biological applications, we are confronted with data in different formats, such as the Systems Biology Markup Language (SBML), and the *BioQuali* [63] format. The BioASP library provides functionalities to parse and transform these formats into ASP facts. For their implementation, BioASP utilizes the library *libSBML* [15] as well as the tool *PLY*, an implementation of lex and yacc parsing tools for Python.

As illustrated in Figure 4.1, the BioASP library consists of three main modules: the *data*, *asp* and the *query* module. In the following, we will describe these modules and the functionalities they provide.

4.1.1 The *asp* module

The *asp* module constitutes the core component of the BioASP library. It provides the classes *Term* and *TermSet*. These classes allow us to represent ASP atoms and sets of atoms. They grant easy access to the arguments of an atom including nested function symbols. We can join sets of atoms and write them to a file in a *gringo* readable format. Further, the *asp* module provides the classes *GringoClasp*, *GringoClaspD* and *GringoClaspOpt* that encapsulate the grounder *gringo* as well as the solvers *clasp* and *claspD*. These classes can be instantiated with dedicated parameter settings, and their objects can be used to solve logic programs in the input format of *gringo*. When such a solving process is finished, the result is returned as *TermSet*s containing the atoms of the answer sets. For optimization problems a tuple containing the found optima is returned.

In the following, we give descriptions for the most important classes provided by the *asp* module. These functions and classes can be used by a Python program after importing the *asp* module like:

```
from bioasp import asp.
```

The class *Term* constitutes a basic element of the *asp* module, the representation of an

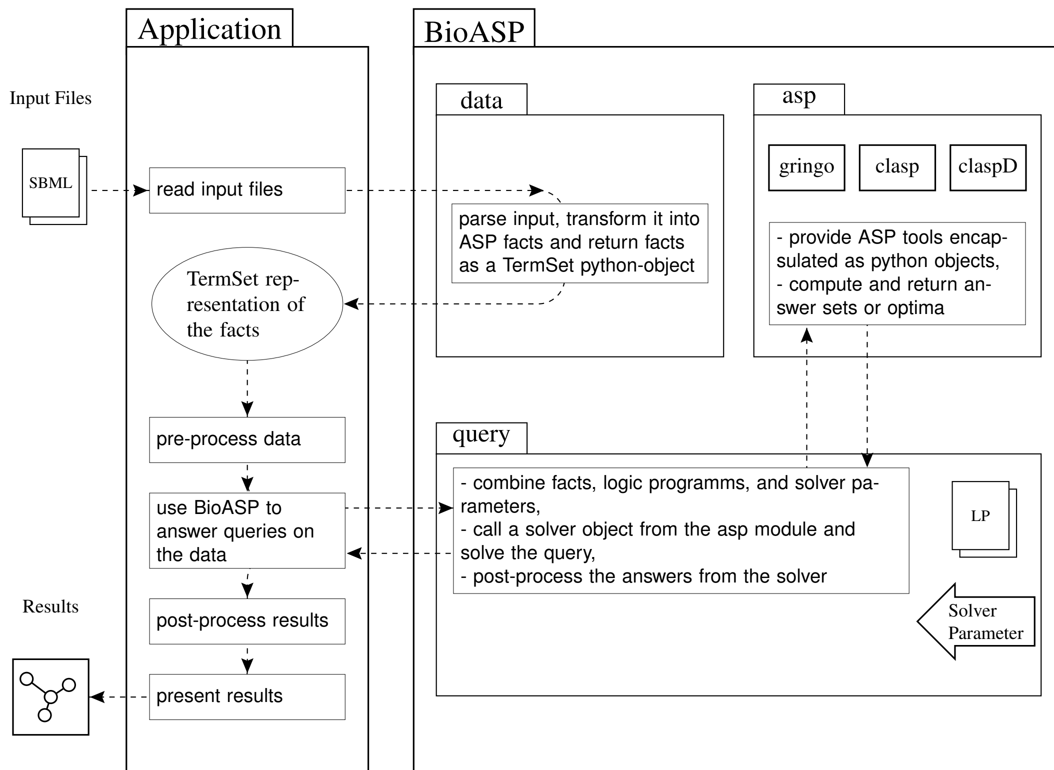

Figure 4.1: Architecture of the BioASP Library.

ASP atom. Instances of *Term* are used everywhere throughout the BioASP library from the representation of input facts to the resulting atoms of an answer set.

class `asp.`**`Term(`**`predicate,arguments=[]`**`)`**

Constructs a *Term* object, the basic representation of an ASP atom. The argument *predicate* sets the predicate name and the argument *arguments* sets the argument list of the atom.

Instances of *Term* provide the following methods.

`Term.`**`args()`**

Returns a list of all arguments of the *Term*.

`Term.`**`pred()`**

Returns the predicate name of the *Term*.

*Term*s usually appear as elements of a *TermSet*. A *TermSet* is a simple Python data structure which allows for easy pre-processing of the data, it can be joined with other *TermSet*s and finally be written to a file, suitable as input for the grounder *gringo*.

class `asp.`**`TermSet()`**

Constructs a *TermSet* object. A *TermSet* is an iterable collection of ASP atoms, it provides the common operations of the Python class *Set* like *union*, *intersection*, *difference* etc. Figure 4.2 shows exemplarily the textual representation of a *TermSet*.

Instances of *TermSet* also provide the following methods.

TermSet.**to_list()**

Returns a list of all *Term*s in the *TermSet*.

TermSet.**to_file(**fn=None**)**

Writes the *TermSet* to a file, as a set of facts readable by the grounder *gringo*. The argument *fn* sets the file name to which the facts are written. If no file name is given a temporary file is created. The method returns the name of the file to which the *TermSet* was written.

TermSet.**exclude_rule()**

Returns a string representation of an ASP constraint that states that a solution is not allowed to contain all the atoms in the *TermSet*.

```
TermSet([
  reaction("rea03981","ecoli"),
  reactant("com01126","rea03981"),
  product("com05702","rea03981"),
  reaction("rea09430","ecoli"),
  reactant("com05702","rea09430"),
  reactant("com00462","rea09430"),
  product("com03190","rea09430"),

  seed("com05702"),
  target("com03190"),
  target("com04283"),

  reaction("rea04982","metacyc"),
  reactant("com03190","rea04982"),
  reactant("com05702","rea04982"),
  product("com04283","rea04982"),
])
```

Figure 4.2: A *TermSet* of ASP facts representing metabolic networks, seeds and targets

class asp.**String2TermSet(**s**)**

Constructs a *TermSet* object from a string. The argument *s* is a string of comma separated atoms in gringo syntax. The string *s* is parsed and a *TermSet* representation is returned.

The classes *GringoClasp*, *GringoClaspD* and *GringoClaspOpt* encapsulate the grounder *gringo* as well as the solvers *clasp* and *claspD*. They can create solver objects, which enable the user to solve ASP problems and return the resulting answer sets as *TermSet*s.

class asp.**GringoClasp(**clasp_options='', gringo_options=''**)**

Constructs a *GringoClasp* object. The argument *gringo_options* sets the command line parameters for the grounder *gringo*, such as constant definitions like `'--const depth=10'`. The argument *clasp_options* sets the command line parameters for the *clasp* process, such as `'--heu=vsids'` for setting the search heuristic.

Instances of *GringoClasp* provide the following method:

`GringoClasp.`**`run`**`(programs, nmodels=1)`
The *run()* method launches the grounder *gringo* to instantiate the logic programs and pipes *gringo*'s output into the solver *clasp* to compute answer sets. Parameters, command line switches and logic program files are passed to the processes, the output of *clasp* is parsed, and finally the computed answer sets are returned as a list of *TermSet*s. The argument *programs* sets the list of logic program files, which are passed to the *gringo* process. The argument *nmodels* is passed to the *clasp* process, it determines how many models will be enumerated at most.

class `asp.`**`GringoClaspOpt`**`(clasp_options='', gringo_options='')`
Constructs a *GringoClaspOpt* object. The argument *gringo_options* sets the command line parameters for the grounder *gringo*, and the argument *clasp_options* sets the command line parameters for the *clasp* process. *GringoClaspOpt* is designed for dealing with optimization problems. Like *GringoClasp*, it encapsulates *gringo* and *clasp*, but it aims at logic programs containing optimization statements [44, 107]. Hence, *clasp* is here used to compute optimal solutions and the associated optima are returned by the `run` function.

Instances of *GringoClaspOpt* provide the following method:

`GringoClaspOpt.`**`run`**`(programs, nmodels=1)`
The *GringoClasp* object launches the grounder *gringo* and the solver *clasp* to compute optimal answer sets. Parameters, command line switches and logic program files are passed to the processes and the computed optima as well answer sets are returned. The return type is a pair where the first element is a tupel containing the computed optima and the second element is a list of *TermSet*s containing the optimal models. The argument *programs* sets the list of logic program files which are passed to the *gringo* process. The argument *nmodels* is passed to the *clasp* process, it determines how many optimal models will be enumerated at most.

class `asp.`**`GringoClaspD`**`(clasp_options='', gringo_options='')`
Constructs a *GringoClaspD* object. The argument *gringo_options* sets the command line parameters for the grounder *gringo* and the argument *clasp_options* sets the command line parameters for the *claspD* process.

Due to their computational complexity, some problems are encoded by disjunctive logic programs. Since *GringoClasp* cannot handle such programs, the class *GringoClaspD* encapsulates *gringo* and the solver *claspD*. The class works similarly to *GringoClasp*, using *gringo* to instantiate logic programs but *claspD* instead of *clasp* for computing their answer sets. Therefore, *GringoClaspD* objects can handle disjunctive logic programs.

The generic classes of the *asp* module can be utilized to solve a variety of problems, depending on the logic programs passed to the `run()` method of their objects.

4.1.2 The *data* module

The *data* module implements functions to read and write the various biological formats supported by the BioASP library such as *SBML* and the *BioQuali* format. These functions

mainly parse input files and transform its data into ASP facts, representing the problem instance at hand. Further, they create a Python data structure *TermSet* containing these ASP facts.

The parsing functions provided by the data module are application-specific, as the ASP facts produced by the parsing function must match the atoms used in the logic programs that encode the biological questions on the input. For example, for our application in metabolic network expansion the data module provides multiple functions to parse SBML files, extract information and return corresponding ASP facts. One function parses an SBML file and interprets the extracted information as a metabolic reaction network. A second function interprets the SBML file as data on measured target metabolites while a third function interprets it as data on seed metabolites. If new formats or new questions are to be addressed, new functions may need to be added to the data module for generating appropriate facts. In the following, we give a description of the most important functions provided by the *data* module.

The *data* module defines the following functions that parse SBML files. These functions can be called from a Python program after importing the module *sbml* like:

```
from bioasp.data import sbml
```

`sbml.`**`readSBMLnetwork(`**`filename, name`**`)`**
: Read an SBML file and return a *TermSet* containing the ASP fact representation of the metabolic network. The *filename* argument is the SBML file and the *name* argument is a string used in the ASP facts to identify reactions from this network.

`sbml.`**`readSBMLtargets(`**`filename`**`)`**
: Read an SBML file and return a *TermSet* containing the ASP fact representations of a set of target metabolites. The *filename* argument is the SBML file.

`sbml.`**`readSBMLseeds(`**`filename`**`)`**
: Read an SBML file and return a *TermSet* containing the ASP fact representations of a set of seed metabolites. The *filename* argument is the SBML file.

An exemplary *TermSet* obtained by these functions, representing metabolic networks, seeds and targets is shown in Figure 4.2.

The BioQuali format has been designed for the electronic representation and exchange of gene regulatory networks and variation data of gene expressions. The *data* module defines the following functions to parse influence graphs and observations in BioQuali format. These functions can be called from a Python program after importing the module *bioquali* like:

```
from bioasp.data import bioquali
```

`bioquali.`**`readGraph(`**`filename`**`)`**
: Read a BioQuali graph file and return a *TermSet* containing the ASP fact representation of the influence graph. The *filename* argument is the BioQuali graph file. Figure 4.3 shows the BioQuali graph representation of the influence graph in Figure 2.10 as well as the obtained ASP facts.

```
                                TermSet([
reb1 -> hsc82 +                   obs_elabel(gen("reb1"),gen("hsc82"),1),
reb1 -> rap1 +                    obs_elabel(gen("reb1"),gen("rap1"),1),
reb1 -> sin3 +                    obs_elabel(gen("reb1"),gen("sin3"),1),
reb1 -> top1 +                    obs_elabel(gen("reb1"),gen("top1"),1),
sin3 -> ume6 -                    obs_elabel(gen("sin3"),gen("ume6"),-1),
ume6 -> ino2 +                    obs_elabel(gen("ume6"),gen("ino2"),1),
ume6 -> hsf1 -                    obs_elabel(gen("ume6"),gen("hsf1"),-1),
ume6 -> spo12 -                   obs_elabel(gen("ume6"),gen("spo12"),-1),
ume6 -> top1 -                    obs_elabel(gen("ume6"),gen("top1"),-1)
                                ])
```

Figure 4.3: BioQuali format for networks (left) and the *TermSet* containing the ASP facts (right)

`bioquali.`**`readSIFGraph(`**`filename`**`)`**

Read a file in SIF (Simple Interaction File) format and return a *TermSet* containing the ASP fact representation of the influence graph. The *filename* argument is the file in SIF format.

`bioquali.`**`readProfile(`**`filename`**`)`**

Read a BioQuali observation file and return a *TermSet* containing the ASP fact representations of observed variations. The *filename* argument is the BioQuali observation file. Figure 4.4 shows the BioQuali representation of observations as shown in Figure 2.10 as well as the obtained ASP facts (The name `"exp1"` associated with experimental observations is obtained from the name of the *filename* argument).

```
                        TermSet([
hsc82 = -                 obs_vlabel("exp1",gen("hsc82"),-1),
rap1 = -                  obs_vlabel("exp1",gen("rap1"),-1),
ume6 = +                  obs_vlabel("exp1",gen("ume6"),1),
ino2 = -                  obs_vlabel("exp1",gen("ino2"),-1),
hsf1 = +                  obs_vlabel("exp1",gen("hsf1"),1),
spo12 = +                 obs_vlabel("exp1",gen("spo12"),1),
top1 = +                  obs_vlabel("exp1",gen("top1"),1)
                        ])
```

Figure 4.4: BioQuali format for observations (left) and the corresponding *TermSet* (right).

4.1.3 The *query* module

The *query* module implements functionalities particular to the biological questions addressed in Chapters 2 and 3. The provided functions work in a generic way on inputs given as *TermSet*s. The facts, contained in the *TermSet*s, are combined with logic programs encoding the problem to be solved. Depending on the concrete problem that is to be solved, an appropriate solver object either of *GringoClasp*, *GringoClaspD*, or *GringoClaspOpt* is created. Notably, the addressed biological question is taken into account for picking the parameter setting of such an object. The solver object is then run and parameters, logic programs and facts are passed to the solver. Once the solver has computed a

solution, answer sets can be further processed, e.g., by filtering out atoms derived from facts and finally the result is returned as a *TermSet*.

One of the areas for which the *query* module provides functions is the analysis of gene regulatory networks and data on the variation of gene expressions from steady state shift experiments. Here, we rely upon the *Sign Consistency Model* as described in Chapter 2, In Chapter 2, we developed the logic programs to detect and explain inconsistencies between a network and experimental observations and to compute minimal repairs allowing for prediction under inconsistency. In the following, we give brief descriptions for most important functions that encapsulated these logic programs and provide their functionalities. These functions can be used by a Python program after importing the module *influence_graphs* like:

```
from bioasp.query import influence_graphs as ig
```

`ig.`**`is_consistent(`** `instance` **`)`**

Returns *True* if the given instance is consistent and *False* otherwise. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem like the one shown in Figure 4.3 and Figure 4.4.

The function combines the problem instance with the logic program encoding the consistency check and uses a *GringoClasp* object for solving. If this object returns some answer set, network and observations are mutually consistent, and inconsistent otherwise.

`ig.`**`get_consistent_labelings(`** `instance, nmodels=0` **`)`**

Returns a list of *TermSet*s containing an ASP representation of labelings consistent with the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem and the argument *nmodels* determines how many labelings will be enumerated at most.

`ig.`**`get_minimal_inconsistent_cores(`** `instance, nmodels=0` **`)`**

Returns a list of *TermSet*s each containing a minimal inconsistent core for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem and the argument *nmodels* determines how many minimal inconsistent cores will be enumerated at most.

The function combines the problem instance with a logic program encoding diagnosis and uses a *GringoClaspD* object for solving. The obtained answer sets, representing minimal inconsistent subnetworks, are returned as a list of *TermSet*s. For example, Figure 4.5 exemplarily shows the first part of such a list.

```
[TermSet([active("exp1",gen("hsf1"))]),
 TermSet([active("exp1",gen("ino2"))]),
 TermSet([active("exp1",gen("rap1")),
          active("exp1",gen("top1"))]),
 ...]
```

Figure 4.5: Minimal diagnoses as a list of *TermSet*s.

`ig.`**`guess_inputs(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of input nodes for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

The BioASP library supports several modes of repairing inconsistent networks and observations. A *repair* is understood as a set of modifications on a network and observations that makes them mutually consistent. Such modifications can be the addition of regulations, flipping the polarity (activation or inhibition) or the observed variation (increase or decrease) of regulations or species, respectively, and allowing for species with unexplained variation. Since the adequacy of these modifications is application-specific, BioASP provides separate functions for different repair modes.

`ig.`**`get_repair_options_flip_obs(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of possible repairs by flipping the sign of an observation for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

`ig.`**`get_repair_options_add_edge(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of possible repairs by adding edges to an influence graph for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

`ig.`**`get_repair_options_flip_edge(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of possible repairs by flipping edges of an influence graph for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

`ig.`**`get_repair_options_make_node_input(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of possible repairs of the given instance by declaring nodes of the influence graph as input. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

`ig.`**`get_repair_options_make_obs_input(`** `instance` **`)`**
Returns a *TermSet* containing the fact representation of possible repairs of the given instance by observed variations in an experiment as input. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

Figure 4.6, shows exemplarily a set of possible repair operations.

```
TermSet([
  pos(vflip("exp1",gen("hsc82"),-1),
  pos(vflip("exp1",gen("rap1"),-1),
  pos(vflip("exp1",gen("ume6"),1),
  pos(vflip("exp1",gen("ino2"),-1),
  pos(vflip("exp1",gen("hsf1"),1),
  pos(vflip("exp1",gen("spo12"),1),
  pos(vflip("exp1",gen("top1"),1)
])
```

Figure 4.6: A *TermSet* of ASP facts representing possible repair operations.

ig.**get_minimum_of_repairs(** instance, repair_options **)**
Returns the minimal number of repair operations that are needed to restore consistency for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem and the argument *repair_options* is a *TermSet* containing the fact representation of possible repair operations.

The function combines the problem instance with the possible repair operations and runs a *GringoClaspOpt* object to determine the minimum number of repair operations for making the problem instance consistent.

ig.**get_minimal_repair_sets(** instance, repair_options, optimum, nmodels=0 **)**
Returns a list of *TermSet*s containing an ASP representation of minimal repair sets that restore consistency for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem, the argument *repair_options* is a *TermSet* containing the fact representation of possible repair operations, the argument *optimum* determines the size of a minimal repair set and the argument *nmodels* determines how many minimal repair set will be enumerated at most.

This function computes all minimal repair sets by running a *GringoClasp* object with the options `--opt-val=Opt` and `--opt-all` of *clasp* being set. It returns a list of *TermSet*s containing the computed minimal repairs, such as the (singleton) list shown in Figure 4.7.

```
[TermSet([
  repair(vflip("exp1",gen("rap1"),-1)),
  repair(vflip("exp1",gen("ume6"),1)),
  repair(vflip("exp1",gen("hsc82"),-1))])
]
```

Figure 4.7: A list containing a minimal repair set as *TermSet*.

ig.**get_predictions_under_consistency(** instance **)**
Returns a *TermSet* containing an ASP representation of predicted labels and for the given instance. The argument *instance* is a *TermSet* containing the fact representation of a sign consistency problem.

The function combines the problem instance with a logic program that computes consistent labelings and runs a *GringoClasp* object on the input but with the option `--cautious` of *clasp* being set. This lets *clasp* compute the intersection of all answer sets, which corresponds to the predicted system behavior under the given observations. The atoms in the intersection, which are not already appear as facts in the problem instance are predicted and returned as a *TermSet* like the one shown in Figure 4.8.

ig.**get_predictions_under_minimal_repair(** instance, repair_options, optimum **)**
Returns a *TermSet* containing an ASP representation of labels for the given inconsistent instance that can be predicted under all minimal repair sets. The argument

```
TermSet([
  vlabel("exp1",gen("reb1"),1),
  vlabel("exp1",gen("hsc82"),1),
  vlabel("exp1",gen("rap1"),1),
  vlabel("exp1",gen("sin3"),1),
  vlabel("exp1",gen("ume6"),-1)
])
```

Figure 4.8: Predicted variations as a *TermSet*

instance is a *TermSet* containing the fact representation of a sign consistency problem and the argument *repair_options* is a *TermSet* containing the fact representation of possible repair operations and the argument *optimum* determines the size of a minimal repair set.

The function runs a *GringoClasp* object on the same input as used for computing minimal repairs, but with the option `--cautious` of *clasp* being set in addition to `--opt-val=Opt` and `--opt-all`. As in the case of consistency, the atoms in the intersection of all answer sets (comprising a minimal repair) that appear not as facts in the problem instance are predicted and returned as a *TermSet*, such as the one shown in Figure 4.8. Since the returned atoms hold under all minimal repairs relative to a repair mode, they describe system behavior that can sensibly be predicted even though the system is not globally inconsistent.

A second biological applications for which the *query* module provides functions is metabolic network expansion as discussed in Chapter 3. In the following, we give brief description of the most important of these functions. These functions can be used after importing the module *network_expansion* like:
`from bioasp.query import network_expansion as ne.`

`ne.`**`get_unproducible(`**` draft, targets, seeds `**`)`**
: Returns a *TermSet* containing the target metabolites that cannot be produced by the draft network given the seed metabolites. The arguments *draft*, *targets* and *seeds* are *TermSet*s containing fact representations of the draft network, target and seed metabolites.

`ne.`**`get_mimimal_completion_size(`**` draft, repairnet, targets, seeds`**`)`**
: Returns the minimal size for a completion of a draft network, such that the most target metabolites can be produced from the given set of seed metabolites. The arguments *draft*, *repairnet*, *targets* and *seeds* are *TermSet*s containing fact representations of the draft network, the repair network, target and seed metabolites.

`ne.`**`get_intersection_of_completions(`**` draft, repairnet, targets, seeds`**`)`**
: Returns a *TermSet* containing the fact representation of the reactions that are contained in every completion of the draft network, such that the most target metabolites can be produced from the given set of seed metabolites. The arguments *draft*, *repairnet*, *targets* and *seeds* are *TermSet*s containing fact representations of the draft network, the repair network, target and seed metabolites.

`ne.`**`get_intersection_of_optimal_completions(`** `draft, repairnet, targets, seeds, optimum)`

Returns a *TermSet* containing the fact representations of the reactions that are contained in every minimal completion of the draft network, such that the most target metabolites can be produced from the given set of seed metabolites. The arguments *draft*, *repairnet*, *targets* and *seeds* are *TermSet*s containing fact representations of the draft network, the repair network, target and seed metabolites. The argument *optimum* determines the size of an optimal completion.

In summary, the functions provided by the BioASP library implement a variety of functions that allow reasoning on gene regulatory networks and metabolic networks. Given that some functionalities build on top of others, composite tasks can be accomplished by chaining several function calls. In the following section, we will show how the different functions provided by the BioASP library are used and work in our applications.

4.2 Applications

In the Chapters 2 and 3, we described biological problems related to gene regulatory networks and metabolic networks. We developed logic programs to answer various questions on the biological models and data. In Section 4.1, we then presented the BioASP library that provides functions to solve the biological questions. We described how the library encapsulates these logic programs, and the ASP solver technology to provide a simple API. In this section, we describe two applications that are implemented using the BioASP library. These applications solve the biological question presented in Chapters 2 and 3. In the following, we will further detail the workflow of these applications.

4.2.1 Diagnosing and Repairing on Gene Regulatory Networks

In this section, we describe an application that is implemented using the BioASP library, which analysis genetic regulative models and observation data of gene expressions. In the context of gene regulatory networks, we are interested in checking whether behaviors observed in experiments can be explained. If experimental observations are inconsistent with a network, i.e., if they cannot be explained, minimal diagnoses can help to identify unreliable data or regulations missing in the network. Further, we are interested in predicting behaviors of unobserved species. Moreover, on the basis of minimal repairs, behaviors of unobserved species can be predicted even in the case of mutual inconsistency between network and data. Beyond analyzing available experimental data, the provided functionalities can be used for experiment planning. In this case, the input describes the desired behavior of a biological system, and predictions indicate conditions needed to achieve such behavior. In the following, we will describe the workflow of this application as shown in Figure 4.9. We will show how this application utilizes the BioASP library to systematically analyze influence graphs and observed variation data.

For the electronic representation and exchange of gene regulatory networks and variation data of gene expressions, the BioQuali format has been designed. The first step for the application is to read such a network file and transform it into ASP facts. Therefore, the *data* module of the BioASP library provides the function

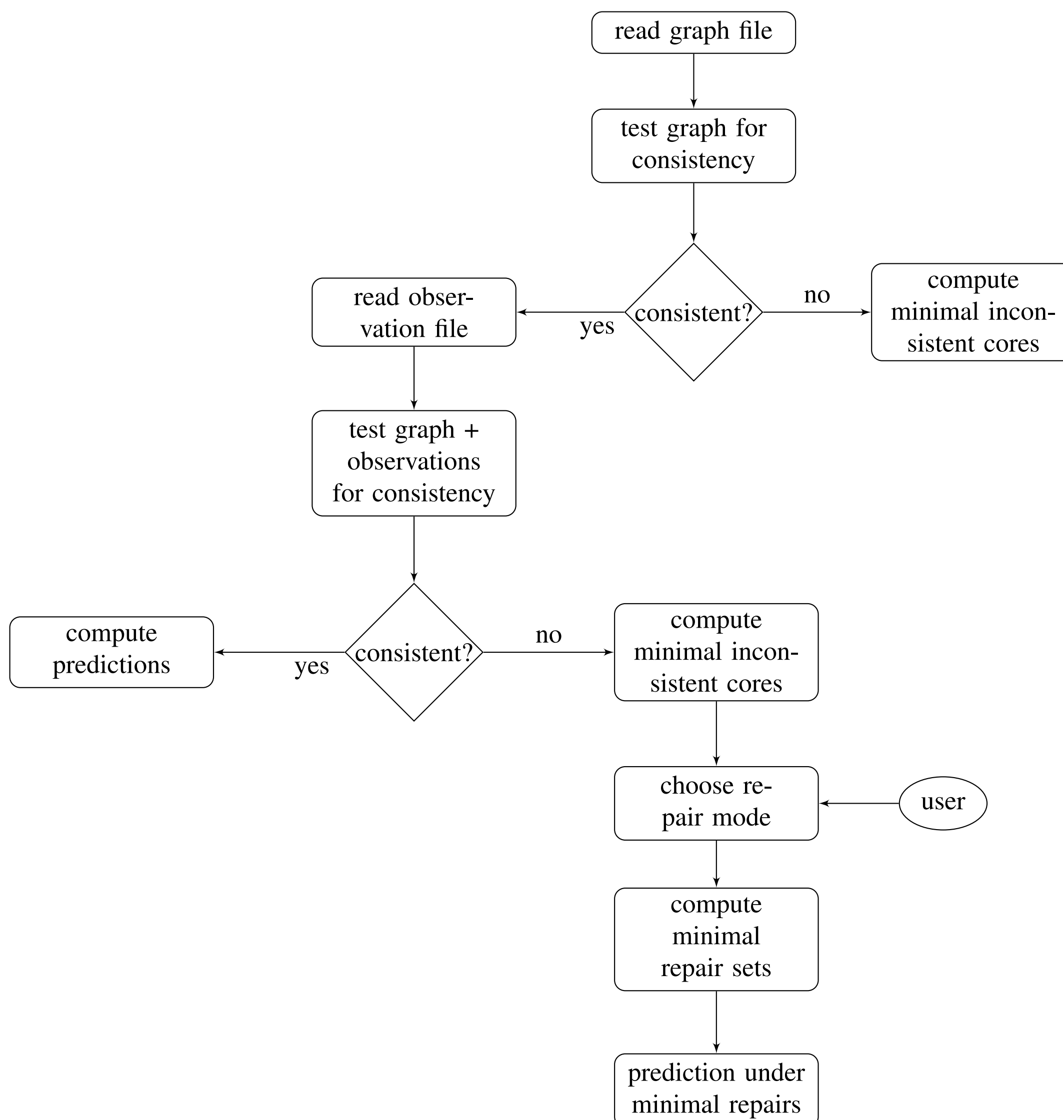

Figure 4.9: Workflow of the "Influence Graph Analyzer" application.

`bioquali.readGraph()` to parse a network and transform it into ASP facts. Figure 4.3 shows the BioQuali representation of the influence graph in Figure 2.10.

The next step after reading the input data is to check whether the network is consistent. Note, that the application first checks only the network without any observation data for consistency. If the network in itself is inconsistent, it will be inconsistent with any observation data, but we can narrow down the cause of inconsistency to errors in the network. For *consistency checking*, the BioASP *query* module provides the function `is_consistent()`. The function combines a problem instance like the one shown in Figure 4.3 with the logic program encoding the consistency check and uses a *Gringo-Clasp* object for solving. If this object returns some answer set, network and observations are mutually consistent, and inconsistent otherwise.

If the network in itself is inconsistent, we know that the cause of the inconsistency are errors in the network. Therefore, it makes sense to diagnose the inconsistent network

before doing any other analysis. To this end, our application uses the BioASP function `get_minimal_inconsistent_cores()` to compute the minimal inconsistent sub networks. Figure 4.10 exemplarily shows a textual representation of the diagnosis results as produced by the application.

```
mic 1:
   gen("ume6") -> gen("hsf1") -
   gen("ume6") = +
   gen("hsf1") = +
mic 2:
   gen("ume6") -> gen("ino2") +
   gen("ume6") = +
   gen("ino2") = -
mic 3:
   gen("reb1") -> gen("top1") +
   gen("reb1") -> gen("rap1") +
   gen("ume6") -> gen("top1") -
   gen("ume6") = +
   gen("rap1") = -
   gen("top1") = +
...
```

Figure 4.10: Textual representation of diagnosis results.

If the network itself is consistent, we continue by adding observation data. Therefore, our application has to read a file containing observation data in BioQuali format and transform it into ASP facts. The *data* module of the BioASP library provides the function `bioquali.readProfile()` to parse observations and transform it into ASP facts. Figure 4.4 shows the BioQuali representation of observations as shown in Figure 2.10.

As the network is consistent, the application now combines the network data with the data of the observed variations and tests whether network and observations are mutual consistent. The application joins the *TermSet* containing the facts representing the graph as in Figure 4.3 with the *TermSet* containing the facts for the observation data as shown in Figure 4.4 and uses again the function `is_consistent()` from the BioASP *query* module for consistency checking.

In the case of consistency, it is possible to compute the predicted variation for unobserved nodes in the graph. The application utilizes the BioASP function `get_predictions_under_consistency()` to compute predictions. Figure 4.11 shows an example of the textual representation of the predictions.

```
5 predictions found:
Experiment "exp1":
   gen("reb1") = +
   gen("hsc82 ") = +
   gen("rap1") = +
   gen("sin3") = +
   gen("ume6") = -
```

Figure 4.11: Textual representation of the predicted variations.

Otherwise, if network and observations are inconsistent, we are interested in what

causes the inconsistency. As explained before, our application uses the BioASP function `get_minimal_inconsistent_cores()`. The difference here is that we know that the cause for the inconsistency is not inherent to the network, but appears as an interplay between network and observations. Figure 4.10 shows textual representation of the diagnosis results as produced by the application.

After diagnosing the inconsistent network and observations, the application uses the BioASP functionalities to repair inconsistencies. BioASP supports several modes of repairing inconsistent networks and observations. Such modifications can be the addition of regulations, flipping the polarity (activation or inhibition) or the observed variation (increase or decrease) of regulations or species, respectively, and allowing for species with unexplained variation. Since the adequacy of these modifications is application-specific, and BioASP provides separate functions for different repair modes, the applications allows the user to choose interactively which repair mode should be applied. Figure 4.12 shows the dialog for choosing the repair mode.

```
Which repair mode do you want to perform ?
[1] flip observed variations
[2] flip influences
[3] define nodes as input
[4] define observed variations as input
[5] add influences

Choose repair mode 1-5: _
```

Figure 4.12: Dialog for choosing the repair mode.

Depending on the users choice, the application uses a different function of the BioASP *query* module to compute a set of possible repair operations. This function is either `get_repair_options_flip_obs()` to get options to repair the data by flipping the observed variations, `get_repair_options_flip_edge()`, which returns possible repairs that flip the sign of influences, `get_repair_options_make_node_input()` returning the possible repairs that declare some nodes in the graph as input, `get_repair_options_make_obs_input()`, which returns all options for repairs by declaring observed variations input, or `get_repair_options_add_edge()` to get the possible operations for repairs by adding influences.

Once the set of possible repair operations is computed, the application uses these to compute sets of repair operations that restore consistency of the network and observations. As the number of candidate repair sets can be huge, one is usually not interested in all of them, but only in minimal repair sets, which modify network and observations as little as possible. Thus, the application calls the `get_minimum_of_repairs()` function to determine the minimum number of required repairs relative to the set of possible repair operations. Then the application feeds the resulting optimum into the function `get_minimal_repair_sets()`. Figure 4.13 shows the textual representation of a minimal set of repair operations that restores consistency.

Finally, the application can use the repair functionalities of the BioASP library to compute predictions from the inconsistent data. Here, a prediction is a system behavior that holds in all consistent states comprising a minimal repair. Therefore, the application uses the function `prediction_under_minimal_repair()` from the BioASP

```
Computing all repair sets with size 3 ... done.
  repair 1 :
vflip("exp1",gen("rap1"),-1),
vflip("exp1",gen("ume6"),1),
vflip("exp1",gen("hsc82"),-1)
```

Figure 4.13: Textual representation of a minimal repair set.

query module. Figure 4.11 shows the textual representation of the predictions under minimal repair.

In summary, the application chains several function provided by the BioASP library to perform consistency checks, diagnosis, repair and prediction on gene regulatory networks modeled by influence graphs and experimental data that describes observed variations.

4.2.2 Metabolic Network Expansion

The second application built on the BioASP library explores the biosynthetic capabilities of metabolic networks. In metabolic network expansion, we confront a metabolic draft network of an investigated organism with experimentally established data to determine gaps in the network and create hypotheses for possible completions.

Given the metabolic draft network, we confront it with two sets of metabolites, called seeds and targets. In an experiment the investigated organism was able to synthesize the target metabolites while feeding only on the seed metabolites. Our application checks whether the measured targets metabolites can be produced by the investigated draft network. For target metabolites that cannot be produced by reactions in a network, the goal is to generate hypotheses about required expansions with additional reactions that complete the production pathways for these targets. Candidate reactions that can potentially be added are obtained from related networks available in web repositories, such as KEGG and MetaCyc. In this section, we will describe the workflow of this application and how this application utilizes the BioASP library to analyze metabolic networks.

The metabolic networks, seeds, and target metabolites used in our application are described in SBML format. The first task of the application is to parse and transform such inputs into ASP facts. Therefore, it uses the functions `ReadSBMLnetwork()`, `ReadSBMLseeds()`, and `ReadSBMLtargets()` provided by the *data* module of the BioASP library.

The next step in our application is the identification of unproducible target metabolites. We want to know which of the measured target metabolites cannot be produced by the draft network. For this, the BioASP function `get_unproducible()` is used. Figure 4.14 shows the textual representation of the unproducible target metabolites.

```
Checking draftnet for unproducible targets ... done.
  38 unproducible targets:
  "_5Z8Z11Z14Z17Z__45__EICOSAPENTAENOATE_CCO__45__CYTOSOL"
  "STEARIC_ACID_CCO__45__CYTOSOL"
  "LYS_CCO__45__CYTOSOL"
  ...
```

Figure 4.14: Textual representation of unproducible target metabolites.

For those targets that cannot be produced by the draft network, we want to know whether their production pathway can be restored by adding reactions from the repair network. Therefore, the application combines draft and repair network and recomputes the unproducible targets for the combined network. The difference to the unproducible targets of the draft network are the targets for which we can restore the production pathways with reactions from the repair network. Figure 4.15 exemplarily shows the text output of the application.

```
Checking draftnet + repairnet for unproducible targets ... done.
 still 11 unproducible targets:
 "HIS_CCO__45__CYTOSOL"
 "PHE_CCO__45__CYTOSOL"
 "ILE_CCO__45__CYTOSOL"
 ...

 27 targets to reconstruct:
 "LINOLENIC_ACID_CCO__45__CYTOSOL"
 "OLEATE__45__CPD_CCO__45__CYTOSOL"
 "CPD__45__9247_CCO__45__CYTOSOL"
 ...
```

Figure 4.15: Textual representation of target metabolites whose production pathways can be restored.

For targets that can only be produced by the draft network when adding the repair network, the minimal completions that restore production pathways are of interest. As it is usually a very hard problem to find these minimal completions, the application first solves some subproblems, whose solutions simplify the main problem.

As next step, the applications computes for each target the reactions, which are essential for the restoration of its production pathway. In other words, those reactions that are in every completion, which restores the production pathway of the metabolite. A reaction that is essential for the production of one target must also be part of a solution that restores the production pathways of all targets. Hence, the essential reactions can be used to simplify the main problem. The application uses the function `get_intersection_of_completions()` from the *query* module to compute the essential reactions. Figure 4.16, shows exemplarily the textual representation of essential reactions.

```
Computing essential reactions for "LINOLENIC_ACID_CCO__45" ... done.
  1 essential reactions found:
  "FATTY__45__ACID__45__SYNTHASE__45__RXN"

Computing essential reactions for "OLEATE__45__CPD_CCO__45" ... done.
  17 essential reactions found:
  "_4__46__2__46__1__46__58__45__RXN"
  "RXN__45__9536"
  "RXN__45__9537"
  ...
  Overall 34 essential reactions found.
```

Figure 4.16: Textual representation of essential reactions.

Further, the application computes for each target metabolite the minimal size of an completion that restores its production pathway. The maximum of these minima is a lower bound for the size of an completion, which restores the production of all target metabolites. While the sum of all minima approximates an upperbound for the size of such a completion. The application uses the function `get_mimimal_completion_size()` from the BioASP *query* module, to compute the minimal size of a completion for all targets.

```
Computing minimal score for a completion for "LINOLENIC_ACID_CCO__45"
  minimal score = 6

Computing minimal score for a completion for "OLEATE__45__CPD_CCO__45"
  minimal score = 25

Computing minimal score for a completion for "CPD__45__9247_CCO__45"
  minimal score = 20
  ...
  Lowerbound for minimal completion for all targets = 28
  Upperbound for minimal completion for all targets = 217
```

Figure 4.17: Text output for the computation of completion minima.

The application continues by including the information about the essential reactions, minimal as well as maximal size of a completion to compute the minimal size of a completion, which restores the production of all target metabolites. Therefore, it uses once more the function `get_mimimal_completion_size()`. The returned optima is then fed into the function `get_minimal_completions()` to compute the minimal completions, which restore the production of all targets. Figure 4.18 shows exemplarily the textual representation of a minimal completion.

```
Computing minimal score for a completion to produce all targets ... done.
  minimal score = 34

Computing minimal (score=34) completions for all targets ... done.
1: "GLYCERATE__45__DEHYDROGENASE__45__RXN"
   "HOMOCITRATE__45__SYNTHASE__45__RXN"
   "SERINE__45__GLYOXYLATE__45__AMINOTRANSFERASE__45__RXN"
   "HOMOACONITATE__45__HYDRATASE__45__RXN"
   "RXN__45__8", "RXN3O__45__1983"
   "ALARACECAT__45__RXN"
   "RIB5PISOM__45__RXN"
   "AMINOBUTDEHYDROG__45__RXN"
   "GLYCEROL__45__1__45__PHOSPHATASE__45__RXN"
   "RXN__45__961"
   "_2__46__6__46__1__46__35__45__RXN"
   ...
```

Figure 4.18: Textual representation of a minimal completion.

In summary, the application combines several functions provided by the BioASP library to perform analysis of metabolic networks, compute producible and unproducible metabolites and hypothetical completions for the network. Thereby, the application ex-

emplarily demonstrates, how to solve a biological problem with BioASP by solving subproblems, aggregating the results, and using them as input for consecutive computations.

4.2.3 Web Service

The last application we present is a web service based on the BioASP library. This web service was designed to make our methods easily accessible to a biological audience. The user of the web service[1] does not require any locally installed software on the user side except for a web browser. Also, the web service serves as an example, showing that the BioASP library eases the creation of integrated applications based on the underlying ASP technology. In the following, we will present the web service and explain its functionalities.

Figure 4.19 shows the interface of the web service. It provides the possibility to upload textual representations of biological networks and experimental profiles as shown in the Figures 4.3 and 4.4. Therefore, the service uses the functions `bioquali.readGraph()` and `bioquali.readProfile()` from the *data* module.

Figure 4.19: Web interface for consistency checking and diagnosis.

To allow the user to instantly experience the functionalities of the web service, a number of predefined examples are provided. Influence graphs representing biological networks usually contain vertices that are not subject to any regulation. Such entities are understood as controlled by external factors, like environmental or particular experimental conditions. To avoid trivial inconsistencies due to such unregulated and thus unexplainable vertices, the web interface provides an option "Guess input nodes" for automatically declaring all vertices without any predecessor as inputs. To compute these input nodes, the service uses the function `guess_inputs()` from the *query* module.

[1] `http://data.haiti.cs.uni-potsdam.de/wsgi/app`

Once the user has uploaded the network and observation files, the reasoning functionalities of the service can be used. These include consistency checking, diagnosis, i.e., finding minimal inconsistent cores (MICs), and prediction of variations.

While consistency checking simply results in a positive or negative answer, the service offers three diagnosis modes: “find one inconsistency”, “find all inconsistencies”, and “approximate all inconsistencies”. The first mode aims at finding a single MIC, and the second at finding all of them. Therefore, the service simply uses the function `get_minimal_inconsistent_cores()` from the *query* module. The third diagnosis mode, “approximate all inconsistencies”, works by repeatedly calling the function to find a single MIC, marking the vertices of a computed MIC as inputs before proceeding to look for further MICs, until no further solution exists. This approach has been used in previous work [63] and has been integrated into our framework for comparison. However, the results obtained with the third mode depend on the order in which MICs are found and their vertices declared to be inputs in future computations.

Once MICs have been computed, the service generates graphical and textual representations of the computed inconsistencies, as shown in Figure 4.20. If the result consists of several MICs, it is possible to view overlapping ones in a combined way, thus highlighting regions of inconsistency.

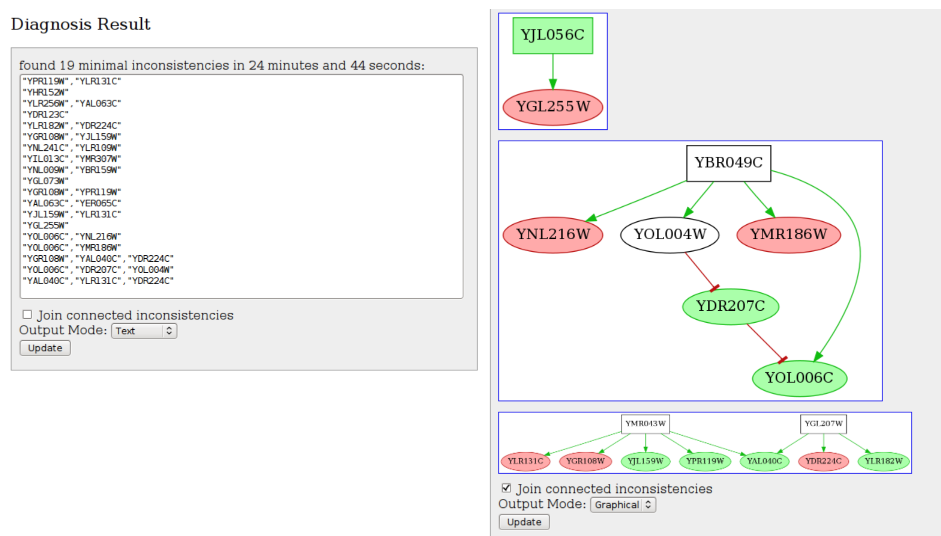

Figure 4.20: Representation of identified MICs in textual (left) and graphical (right) mode.

Finally, prediction under consistency and inconsistency are featured by the web service. While prediction under consistency only returns results when the consistency check above had a positive result, the web service allows the user to switch the reasoning mode to prediction under minimal repair. For the first mode the service simply calls the function `get_predictions_under_consistency()`, for the second mode the internal computations are more complex. First, the possible repair options are determined with `get_repair_options_make_node_input()`, then the minimal number of repair operations that restore consistency is computed via `get_minimum_of_repairs()` , and fed into the function `get_predictions_under_minimal_repair()` to finally predict unobserved variations of entities in the influence graph. Note, prediction under

minimal repair can also be applied to consistent samples, then the minimal number of repairs is 0.

The web service shows exemplarily how the BioASP library can be used to built complex applications that integrate ASP solutions with other state of the art technologies. It provides an easily accessible and visual appealing front-end to our solutions even for people that do not know ASP.

4.3 Discussion

We presented the BioASP library. We outlined the modular architecture of the library which facilitates extensibility. Especially the *query* module is designed for future extension with further functionalities, to provide access to further ASP solution that solve new problems. We hope that the library will serve in future as a repository for many ASP solutions solving a variety of problems. A detailed overview of the library API and the provided functionalities has been given. The library was initially built with a focus on our biological applications. In fact, the functionalities provided by the BioASP library exploit technical know-how of modeling biological problems in ASP and gearing ASP solvers' parameters to them. Therefore, the library provides many functions to solve the biological problems described in this thesis. On a more general level, the library facilitates the use of ASP solutions inside the imperative programming paradigm of Python [111]. With the *asp* module being the core of the library, we encapsulate the ASP tools: the grounder *gringo* as well as the solvers *clasp* and *claspD*, into Python classes. These classes allow for the integration of ASP solutions from various domains into existing frameworks. Hence, the application of the library is not limited to the field of biology. Nowadays, it is also used in the context of natural language processing [83] by the AspCcgTk toolkit[2].

Further, we presented applications that are built on top of the BioASP library. These applications range from terminal applications to complex web services. These applications present the practical outcome of our theoretical work. The applications are tools for biologists, ready to be used to analyze their data and solve their problems. But, they also serve as examples on how to use the BioASP library to built applications that solve complex problems with ASP, and that can easily be integrated with existing solutions.

[2]`http://www.kr.tuwien.ac.at/staff/ps/aspccgtk`

5 Conclusions

This thesis was focused on ASP as modeling tool for problems from the field of Systems Biology. Therefore, we provided ASP solutions to several biologically motivated problems. In Chapter 2, we provided an ASP based framework to analyze genetic regulatory networks represented as influence graphs. This approach allows for consistency checking between experimental profiles and influence graphs and for identifying the causes of the inconsistency as MICs. MIC identification is a valuable tool for biologist, pointing to missing regulations and questionable data and guiding the refinement of the investigated model. Further, we introduced repair-based reasoning techniques for computing minimal modifications of biological networks and experimental profiles to make them mutually consistent. We provided and discussed different repair options for repairing models and data. The choice of the applied repair operations is a highly domain dependent question. Therefore, the presented framework is very flexible to adjust to different repair methods. However, the definition of minimal repairs the approach allows for predicting unobserved data even in case of inconsistency. This is of practical relevance as it enables a meaningful analysis of partially unreliable experimental data, and allows to draw reasonable conclusions automatically. We evaluated the approach on real biological data and showed that predictions on the basis of minimal repairs were feasible and also highly accurate.

In Chapter 3, we then presented an ASP approach to analyze biosynthetic capabilities of metabolic networks. Here, we provided a simple yet powerful account of metabolic network synthesis, a crucial application in the elaboration and design of bioprocesses. In this approach we combined the ease of modeling in ASP with the powerful reasoning modes, supported by efficient solver technology. Unlike existing approaches that are based on stochastic simulations, our approach is complete and thus allows for proving rather than estimating the production of metabolites. The reasoning modes that were used by our approach, including the enumeration of optimal solutions as well as cautious and brave reasoning with respect to all or optimal solutions only, respectively, have shown to be indispensable in a biological application due to the large number of possible solutions. For instance, cautious reasoning relative to optimal solutions makes us discover the essential nutrition for producing a target metabolite. The high-level specification of metabolic networks combined with the different reasoning modes make our approach attractive to biologists, that can easily elaborate and explore their models "in silico".

As an important aspect, we want to stress the advantages of using ASP as paradigm for realizing our applications. On the knowledge representation side, ASP fosters a uniform problem specifications in terms of a separate encoding and instances given as facts. This accelerates the development of a problem solution and it keeps solutions comprehensible in view of the fact that encodings are usually compact. Notably, the task of prediction benefits from the cautious reasoning capacities of clasp, intersecting all answer sets by enumerating only linearly many of them. This illustrates the second major advantage of

ASP, namely, the availability of powerful off-the-shelf inference engines.

As elegance and flexibility in modeling are major advantages of ASP, our current application makes it attractive also for related biological questions, beyond the ones addressed in this thesis. For instance, ongoing work deals with the data driven reconstruction of genetic regulatory networks and the identification of *quantitative trait loci*, genes that are involved in the emergence of certain phenotypes. In turn, challenging applications like the ones presented here might contribute to the further improvement of ASP tools, as they might be geared towards efficiency in such domains.

Finally, we presented in Chapter 4 the BioASP library as a framework providing ASP solutions for applications in Systems Biology. BioASP integrates with traditional programming paradigms to make the power of ASP accessible within an existing, rich system environment. We presented its current functionalities, which includes our ASP solution for analyzing gene regulatory and metabolic networks. But, the library facilitates the use of ASP solutions inside the imperative programming paradigm in general. The easy access to encapsulated ASP tools allows for integrating ASP solution from various domains, not limited to biology. For instance, it has already found application in the field of natural language processing.

For the future, we envision the BioASP library as repository for many different ASP solutions, to address further biological applications and extend the BioASP library with new functionalities. Our goal is to utilize the advancements in the field of knowledge representation and ASP, to develop new methods for reconstructing and analyzing biological (genomic and metabolic) models. So far, our methods did not take into account multi-level observations and knowledge over the system, such as genome annotation and transcriptomic of metabolic information. It is our future challenge to create frameworks that intelligently combine these different levels and enable an automated reasoning process.

A Proofs

A.1 Proof of Theorem 2.1 and 2.2

We formalize the representation of instances, as described in Section 2.3.1, by defining a mapping τ of an influence graph (V, E, σ) and a (partial) vertex labeling $\mu : V \rightarrow \{+, -\}$:

$$\begin{aligned}\tau((V,E,\sigma),\mu) = \{&\mathit{vertex}(i). && \mid i \in V\}\\ \cup\ \{&\mathit{edge}(j,i). && \mid (j \rightarrow i) \in E\}\\ \cup\ \{&\mathit{observedE}(j,i,s). && \mid (j \rightarrow i) \in E, \sigma(j,i) = s\}\\ \cup\ \{&\mathit{observedV}(i,s). && \mid i \in V, \mu(i) = s\}\\ \cup\ \{&\mathit{input}(i). && \mid i \in V \text{ is an input}\}\ .\end{aligned} \tag{A.1}$$

By P_C, we denote the encoding containing the schematic rules in (2.3), (2.4), (2.5), and (2.6).

Proof A.1 (Proof of Theorem 2.1) *Assume that X is an answer set of $P_C \cup \tau((V,E,\sigma),\mu)$. Furthermore, let*

$$P^X = \{(head(r) \leftarrow body(r)^+)\theta \mid r \in P_C \cup \tau((V,E,\sigma),\mu), (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \rightarrow \mathcal{U}\}$$

where $var(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_C \cup \tau((V,E,\sigma),\mu)$, and θ is a ground substitution for the variables in r. Then, by the definition of an answer set, we know that X is a $\subseteq$-minimal model of P^X.

Given X, we define σ' and μ' as follows:

$$\begin{aligned}\sigma' &= \{(j \rightarrow i) \mapsto s \mid (j \rightarrow i) \in E, \mathit{labelE}(j,i,s) \in X\}\\ \mu' &= \{i \mapsto s \mid i \in V, \mathit{labelV}(i,s) \in X\}\ .\end{aligned}$$

We show that σ' and μ' are total labelings of edges and vertices, respectively, such that $\mu'(i) = \mu'(j)\sigma'(j,i)$ holds for every non-input vertex $i \in V$ and some edge $j \rightarrow i$ in E.

Regarding the uniqueness of labels assigned by σ' and μ', consider the following rules from (2.3) and (2.4) including predicates labelE and labelV in their heads:

$$\begin{aligned}\mathit{labelV}(V,+);\mathit{labelV}(V,-) &\leftarrow \mathit{vertex}(V).\\ \mathit{labelE}(U,V,+);\mathit{labelE}(U,V,-) &\leftarrow \mathit{edge}(U,V).\\ \mathit{labelV}(V,S) &\leftarrow \mathit{observedV}(V,S).\\ \mathit{labelE}(U,V,S) &\leftarrow \mathit{observedE}(U,V,S).\end{aligned} \tag{A.2}$$

Since the given (partial) labelings σ and μ assign unique labels to the elements of their domains, facts defining observedE and observedV are of the form $\mathit{observedE}(j,i,+)$. or $\mathit{observedE}(j,i,-)$. and $\mathit{observedV}(i,+)$. or $\mathit{observedV}(i,-)$., respectively, and at most

one of these facts is contained in $\tau((V,E,\sigma),\mu)$ for an edge $(j \rightarrow i) \in E$ or a vertex $i \in V$. Because X is a $\subseteq$-minimal model of P^X, the atoms in the heads of facts are in X, and all atoms in X over predicates observedE and observedV are derived from facts in $\tau((V,E,\sigma),\mu)$, in view that these predicates do not occur in the head of any rule in P_C. Hence, at most one of the atoms labelE$(j,i,+)$ and labelE$(j,i,-)$ or labelV$(i,+)$ and labelV$(i,-)$, respectively, is derivable for an edge $(j \rightarrow i) \in E$ or vertex $i \in V$ from a ground instance of the fourth or third rule in (A.2) and then included in X. Furthermore, the second and first rule in (A.2) impose that at least one of labelE$(j,i,+)$ or labelE$(j,i,-)$ and labelV$(i,+)$ or labelV$(i,-)$ belongs to X for every edge $(j \rightarrow i) \in E$ and vertex $i \in V$, respectively, while the atom containing the opposite label cannot belong to a $\subseteq$-minimal model of P^X. Hence, there is at most one term s such that labelE$(j,i,s) \in X$ or labelV$(i,s) \in X$ for an edge $(j \rightarrow i) \in E$ or vertex $i \in V$, respectively, and it holds that $s \in \{+,-\}$, which allows us to conclude that σ' and μ' are total labelings.

As regards extending σ and μ, we have that fact observedE(j,i,s). or observedV(i,s). belongs to $\tau((V,E,\sigma),\mu)$ if $\sigma(j,i) = s$ or $\mu(i) = s$, respectively, is given. This implies that labelE$(j,i,s) \in X$ or labelV$(i,s) \in X$, respectively, as the fourth or third rule in (A.2) would be unsatisfied otherwise. Thus, $\sigma'(j,i) = s$ if $\sigma(j,i) = s$, and $\mu'(i) = s$ if $\mu(i) = s$.

It remains to be shown that $\mu'(i)$ is consistent for each non-input vertex $i \in V$. To this end, we note that the integrity constraint

$$\leftarrow \textit{labelV}(V,S), \textit{not receive}(V,S), \textit{not input}(V).$$

from (2.6) necessitates receive$(i,r) \in X$ if $\mu'(i) = r$ (that is, if labelV$(i,r) \in X$) for a non-input vertex $i \in V$. Otherwise, P^X would contain an unsatisfied ground instance in view that input$(i) \in X$ exactly if fact input(i). is included in $\tau((V,E,\sigma),\mu)$. However, any ground instances of the integrity constraint contributing to P^X do not contain atoms over predicate receive. Such atoms can only be derived using the following rules from (2.5):

$$\begin{aligned}&\textit{receive}(V,+) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,S).\\&\textit{receive}(V,-) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,T), S \neq T.\end{aligned}$$

Since X is a $\subseteq$-minimal model of P^X, receive$(i,+) \in X$ or receive$(i,-) \in X$ is possible only if labelE$(j,i,s) \in X$ and labelV$(j,t) \in X$ such that $s = t$ or $s \neq t$, that is, if $\sigma'(j,i) = s$ and $\mu'(j) = t$ such that $\mu'(j)\sigma'(j,i) = +$ or $\mu'(j)\sigma'(j,i) = -$, respectively. As labelV(i,r) is accompanied by receive(i,r) in X for each non-input vertex $i \in V$, this allows us to conclude that $\mu'(i) = r$ implies $\mu'(j)\sigma'(j,i) = r$ for some regulator j of i. Hence, we have that $\mu'(i)$ is consistent for each non-input vertex $i \in V$.

Proof A.2 (Proof of Theorem 2.2) *Assume that (V,E,σ) and μ are consistent. Then, there are total extensions $\sigma' : E \rightarrow \{+,-\}$ of σ and $\mu' : V \rightarrow \{+,-\}$ of μ such that, for each non-input vertex $i \in V$, we have $\mu'(i) = \mu'(j)\sigma'(j,i)$ for some edge $j \rightarrow i$ in E.*

We consider the following set X of atoms:

$$
\begin{array}{ll}
X = \{\textit{vertex}(i), \textit{labelV}(i,s) & \mid i \in V, \mu'(i) = s\} \\
\quad \cup \{\textit{edge}(j,i), \textit{labelE}(j,i,s) & \mid (j \rightarrow i) \in E, \sigma'(j,i) = s\} \\
\quad \cup \{\textit{receive}(i,ts) & \mid (j \rightarrow i) \in E, \sigma'(j,i) = s, \mu'(j) = t\} \\
\quad \cup \{\textit{observedE}(j,i,s) & \mid (j \rightarrow i) \in E, \sigma(j,i) = s\} \\
\quad \cup \{\textit{observedV}(i,s) & \mid i \in V, \mu(i) = s\} \\
\quad \cup \{\textit{input}(i) & \mid i \in V \textit{ is an input}\}\ .
\end{array}
$$

For showing that X is an answer set of $P_C \cup \tau((V,E,\sigma),\mu)$, we need to verify that X is a $\subseteq$-minimal model of

$$
\begin{array}{l}
P^X = \{(head(r) \leftarrow body(r)^+)\theta \mid \\
\qquad\qquad r \in P_C \cup \tau((V,E,\sigma),\mu), (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \rightarrow \mathcal{U}\}
\end{array}
$$

where $var(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_C \cup \tau((V,E,\sigma),\mu)$, and θ is a ground substitution for the variables in r.

To start with, we note that X includes an atom vertex(i), edge(j,i), observedE(j,i,s), observedV(i,s), and input(i), respectively, exactly if there is a fact with the atom in the head in $\tau((V,E,\sigma),\mu)$. Each of these facts belongs also to P^X, is satisfied by X, but not by any set Y of atoms excluding at least one of the head atoms. Furthermore, since σ' and μ' are total mappings, we have that $|\{\textit{labelE}(j,i,\mathbf{+}), \textit{labelE}(j,i,\mathbf{-})\} \cap X| = 1$ and $|\{\textit{labelV}(i,\mathbf{+}), \textit{labelV}(i,\mathbf{-})\} \cap X| = 1$ for every $(j \rightarrow i) \in E$ and $i \in V$, respectively. Hence, X, but no subset Y of X excluding at least one atom over predicates labelE and labelV, satisfies all ground instances of the following rules from (2.3) in P^X:

$$
\begin{array}{c}
\textit{labelV}(V,\mathbf{+}); \textit{labelV}(V,\mathbf{-}) \leftarrow \textit{vertex}(V). \\
\textit{labelE}(U,V,\mathbf{+}); \textit{labelE}(U,V,\mathbf{-}) \leftarrow \textit{edge}(U,V).
\end{array}
$$

In addition, since σ' and μ' extend σ and μ, respectively, all ground instances of the following rules from (2.4) in P^X are satisfied by X:

$$
\begin{array}{c}
\textit{labelV}(V,S) \leftarrow \textit{observedV}(V,S). \\
\textit{labelE}(U,V,S) \leftarrow \textit{observedE}(U,V,S).
\end{array}
$$

Since labelE$(j,i,s) \in X$ and labelV$(j,t) \in X$ if $\sigma'(j,i) = s$ and $\mu'(j) = t$, respectively, we have that receive$(i,ts) \in X$ exactly if there is a ground instance of the rules

$$
\begin{array}{l}
\textit{receive}(V,\mathbf{+}) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,S). \\
\textit{receive}(V,\mathbf{-}) \leftarrow \textit{labelE}(U,V,S), \textit{labelV}(U,T), S \neq T.
\end{array}
$$

from (2.5) in P^X such that labelE(j,i,s), labelV$(j,t) \in X$ occur in the body and receive(i,ts) in the head. Hence, no subset Y of X excluding any atom over predicate receive is a model of P^X. Finally, since $\mu'(i) = \mu'(j)\sigma'(j,i)$ for each non-input vertex $i \in V$ and some $j \rightarrow i$ in E, labelV$(i,r) \in X$ implies that receive$(i,r) \in X$. That is, the ground instances of the integrity constraint

$$
\leftarrow \textit{labelV}(V,S), not\ \textit{receive}(V,S), not\ \textit{input}(V).
$$

from (2.6) that contribute to P^X are satisfied by X.

We have now investigated all rules in $P_C \cup \tau((V, E, \sigma), \mu)$ and shown that their ground instances in P^X are satisfied by X. Furthermore, we have checked for all atoms in X that they cannot be excluded in any model $Y \subset X$ of P^X. That is, X is indeed a $\subseteq$-minimal model of P^X and thus an answer set of $P_C \cup \tau((V, E, \sigma), \mu)$.

A.2 Proof of Theorem 2.4 and 2.5

This appendix provides proofs for soundness and completeness of the MIC extraction encoding in Section 2.4. We use $\tau((V, E, \sigma), \mu)$ as defined in (A.1) to refer to the facts representing an influence graph (V, E, σ) and a (partial) vertex labeling $\mu : V \to \{+, -\}$. By P_D, we denote the encoding consisting of the schematic rules in (2.4), (2.7), (2.8), and (2.9).

As an auxiliary concept, for any subset $W \subseteq V$, we say that $\sigma' : E \to \{+, -\}$ and $\mu' : V \to \{+, -\}$ are *witnessing labelings* for W if the following conditions hold:

1. σ' and μ' are total,
2. if $\sigma(j, i)$ is defined, then $\sigma'(j, i) = \sigma(j, i)$,
3. if $\mu(i)$ is defined, then $\mu'(i) = \mu(i)$, and
4. $\mu'(i)$ is consistent (relative to σ') for each non-input vertex $i \in W$.

The above conditions make sure that σ' and μ' are total extensions of σ and μ, respectively, such that the variations of vertices in W are explained. Comparing Definition 2.3, the first condition requires the absence of witnessing labelings for a MIC W, while the second condition stipulates the existence of witnessing labelings for each $W' \subset W$.

Proof A.3 (Proof of Theorem 2.4) *Assume that X is an answer set of $P_D \cup \tau((V, E, \sigma), \mu)$. Furthermore, let*

$$P^X = \{(head(r) \leftarrow body(r)^+)\theta \mid r \in P_D \cup \tau((V, E, \sigma), \mu), (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \to \mathcal{U}\}$$

where $var(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_D \cup \tau((V, E, \sigma), \mu)$, and θ is a ground substitution for the variables in r. Then, by the definition of an answer set, we know that X is a $\subseteq$-minimal model of P^X.

Let $W = \{i \mid active(i) \in X\}$. We have to show that the following conditions hold:

1. *There are witnessing labelings for each $W' \subset W$.*
2. *There are no witnessing labelings for W.*

We below consider these conditions one after the other.

Condition 1. *Let $W' = W \setminus \{k\}$ for any $k \in W$. Furthermore, define σ' and μ' as follows:*

$$\begin{aligned}
\sigma' = {} & \{(j \rightarrow i) \mapsto s \mid (j \rightarrow i) \in E, \mathit{labelE'}(k, j, i, s) \in X\} \\
& \cup \{(j \rightarrow i) \mapsto \mathbf{+} \mid (j \rightarrow i) \in E, \mathit{labelE'}(k, j, i, \mathbf{+}) \notin X, \mathit{labelE'}(k, j, i, \mathbf{-}) \notin X\} \\
\mu' = {} & \{i \mapsto s \mid i \in V, \mathit{labelV'}(k, i, s) \in X\} \\
\cup {} & \{i \mapsto \mathbf{+} \mid i \in V, \mathit{labelV'}(k, i, \mathbf{+}) \notin X, \mathit{labelV'}(k, i, \mathbf{-}) \notin X\} .
\end{aligned}$$

We show that σ' and μ' are witnessing labelings for W'.

Regarding the uniqueness of labels assigned by σ' and μ', consider the following rules from (2.9) including predicates labelE' and labelV' in their heads:

$$\begin{aligned}
\mathit{labelV'}(W, V, \mathbf{+}); \mathit{labelV'}(W, V, \mathbf{-}) &\leftarrow \mathit{active}(W), \mathit{vertexMIC}(V). \\
\mathit{labelE'}(W, U, V, \mathbf{+}); \mathit{labelE'}(W, U, V, \mathbf{-}) &\leftarrow \mathit{active}(W), \mathit{edgeMIC}(U, V). \\
\mathit{labelV'}(W, V, S) &\leftarrow \mathit{active}(W), \mathit{observedV}(V, S). \\
\mathit{labelE'}(W, U, V, S) &\leftarrow \mathit{active}(W), \mathit{observedE}(U, V, S).
\end{aligned} \tag{A.3}$$

Since the given (partial) labelings σ and μ assign unique labels to the elements of their domains, facts defining observedE and observedV are of the form observedE$(j, i, \mathbf{+})$. or observedE$(j, i, \mathbf{-})$. and observedV$(i, \mathbf{+})$. or observedV$(i, \mathbf{-})$., respectively, and at most one of these facts is contained in $\tau((V, E, \sigma), \mu)$ for an edge $(j \rightarrow i) \in E$ or vertex $i \in V$. Because X is a $\subseteq$-minimal model of P^X, the atoms in the heads of facts are in X, and all atoms in X over predicates observedE and observedV are derived from facts in $\tau((V, E, \sigma), \mu)$, in view that these predicates do not occur in the head of any rule in P_D. Hence, at most one of the atoms labelE'$(k, j, i, \mathbf{+})$ and labelE'$(k, j, i, \mathbf{-})$ or labelV'$(k, i, \mathbf{+})$ and labelV'$(k, i, \mathbf{-})$, respectively, is derivable for an edge $(j \rightarrow i) \in E$ or vertex $i \in V$ from a ground instance of the fourth or third rule in (A.3) and then included in X. If either of labelE'$(k, j, i, \mathbf{+})$ and labelE'$(k, j, i, \mathbf{-})$ or labelV'$(k, i, \mathbf{+})$ and labelV'$(k, i, \mathbf{-})$, respectively, is included in X, then the ground instance of the second or first rule in (A.3) for k and an edge $(j \rightarrow i) \in E$ or vertex $i \in V$ is satisfied, so that the atom containing the opposite label cannot belong to a $\subseteq$-minimal model of P^X. Hence, there is at most one term s such that $\sigma'(j, i) = s$ or $\mu'(i) = s$ for an edge $(j \rightarrow i) \in E$ or vertex $i \in V$, respectively, and it holds that $s \in \{\mathbf{+}, \mathbf{-}\}$. Furthermore, looking at the definitions of σ' and μ', it is obvious that both are total, which allows us to conclude that σ' and μ' are total labelings.

As regards extending σ and μ, we have that fact observedE(j, i, s). or observedV(i, s). belongs to $\tau((V, E, \sigma), \mu)$ if $\sigma(j, i) = s$ or $\mu(i) = s$, respectively, is given. Along with the premise that active$(k) \in X$, this implies that labelE'$(k, j, i, s) \in X$ or labelV'$(k, i, s) \in X$, respectively, as the fourth or third rule in (A.3) would be unsatisfied otherwise. Hence, we have $\sigma'(j, i) = s$ if $\sigma(j, i) = s$, and $\mu'(i) = s$ if $\mu(i) = s$.

It remains to be shown that $\mu'(i)$ is consistent for each non-input vertex $i \in W'$. To establish this, we first consider the following rules from (2.7):

$$\begin{aligned}
\mathit{edgeMIC}(U, V) &\leftarrow \mathit{edge}(U, V), \mathit{active}(V). \\
\mathit{vertexMIC}(U) &\leftarrow \mathit{edgeMIC}(U, V). \\
\mathit{vertexMIC}(V) &\leftarrow \mathit{active}(V).
\end{aligned} \tag{A.4}$$

In view that fact edge(j,i). belongs to $\tau((V,E,\sigma),\mu)$ for every $(j \rightarrow i) \in E$, we conclude that edge$(j,i) \in X$. Along with active$(i) \in X$ for every $i \in W$, it follows that edgeMIC$(j,i) \in X$ for every $(j \rightarrow i) \in E$ such that $i \in W$, and vertexMIC$(i) \in X$ for every $i \in W$. The last observation and the first rule in (A.3) imply that labelV'$(k,i,+) \in X$ or labelV'$(k,i,-) \in X$ for every $i \in W$. For $i \in W'$, i.e., $i \neq k$, the integrity constraint

$$\leftarrow \textit{labelV'}(W,V,S), \textit{active}(V), V \neq W, \textit{not receive'}(W,V,S).$$

from (2.9) imposes receive'$(k,i,+) \in X$ if labelV'$(k,i,+) \in X$, and receive'$(k,i,-) \in X$ if labelV'$(k,i,-) \in X$, while any ground instances of the integrity constraint contributing to P^X do not contain atoms over predicate receive'. Such atoms can only be derived using the following rules from (2.9):

$$\textit{receive'}(W,V,+) \leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,S), V \neq W.$$
$$\textit{receive'}(W,V,-) \leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,T), V \neq W, S \neq T.$$

Since X is a $\subseteq$-minimal model of P^X, receive'$(k,i,+) \in X$ or receive'$(k,i,-) \in X$ is possible only if labelE'$(k,j,i,s) \in X$ and labelV'$(k,j,t) \in X$ such that $s = t$ or $s \neq t$, respectively. Comparing $\tau((V,E,\sigma),\mu)$ and the rules in (A.3), (A.4), as well as (A.5) reveals that $(j \rightarrow i) \in E$ is a necessary condition for labelE'$(k,j,i,s) \in X$, and the same applies to $j \in V$ and labelV'$(k,j,t) \in X$. By the construction of σ' and μ', labelE'$(k,j,i,s) \in X$ implies that $\sigma'(j,i) = s$ and labelV'$(k,j,t) \in X$ that $\mu'(j) = t$. We conclude that receive'$(k,i,+) \in X$ or receive'$(k,i,-) \in X$ necessitates $\mu'(j)\sigma'(j,i) = +$ or $\mu'(j)\sigma'(j,i) = -$, respectively, for some regulator j of i. Finally, we have $\mu'(i) = +$ if labelV'$(k,i,+) \in X$ (and receive'$(k,i,+) \in X$), and $\mu'(i) = -$ if labelV'$(k,i,-) \in X$ (and receive'$(k,i,-) \in X$). This shows that i receives some influence matching $\mu'(i)$, so that $\mu'(i)$ is consistent. Since $i \in W'$ is arbitrary, σ' and μ' are witnessing labelings for W'.

To conclude the proof of the first condition to verify, we note that witnessing labelings for W' are also witnessing labelings for all subsets of W'. Hence, it is sufficient to check the existence of witnessing labelings for sets $W' = W \setminus \{k\}$ for any $k \in W$. As shown above, an answer set X of $P_D \cup \tau((V,E,\sigma),\mu)$ yields witnessing labelings for them. Hence, the second condition in Definition 2.3 holds for $W = \{i \mid \textit{active}(i) \in X\}$.

Condition 2. *We now show by contradiction that there cannot be witnessing labelings for W. To establish this, we first note that vertices in W cannot be input because, if fact input(i). belongs to $\tau((V,E,\sigma),\mu)$, then input(i) must be included in X, so that the rule*

$$\textit{active}(V); \textit{inactive}(V) \leftarrow \textit{vertex}(V), \textit{not input}(V). \tag{A.5}$$

from (2.7) does not contribute a ground instance for i to P^X. Since active(i) cannot be derived from any other ground rule in P^X, the fact that X is a $\subseteq$-minimal model of P^X implies that active$(i) \notin X$ for any input vertex i. Furthermore, the integrity constraint

$$\leftarrow \textit{not bottom}. \tag{A.6}$$

from (2.8) necessitates bottom $\in X$ *because* X *cannot be a model of* P^X *otherwise. Then, we get labelV*$(i, +)$, *labelV*$(i, -) \in X$ *and labelE*$(j, i, +)$, *labelE*$(j, i, -) \in X$ *for all vertices* $i \in V$ *and edges* $(j \rightarrow i) \in E$*, respectively, due to the following rules from (2.8):*

$$\begin{aligned} \textit{labelV}(V, +) &\leftarrow \textit{bottom}, \textit{vertex}(V). \\ \textit{labelV}(V, -) &\leftarrow \textit{bottom}, \textit{vertex}(V). \\ \textit{labelE}(U, V, +) &\leftarrow \textit{bottom}, \textit{edge}(U, V). \\ \textit{labelE}(U, V, -) &\leftarrow \textit{bottom}, \textit{edge}(U, V). \end{aligned} \tag{A.7}$$

We now show that the existence of witnessing labelings for W *yields a contradiction to the fact that* X *is a* $\subseteq$*-minimal model of* P^X*. To this end, assume that* σ' *and* μ' *are witnessing labelings for* W*. Then, let*

$$\begin{aligned} Y = (X \setminus (&\{\textit{bottom}\} \\ &\cup \{\textit{labelV}(i, s) \mid \textit{labelV}(i, s) \in X\} \\ &\cup \{\textit{labelE}(j, i, s) \mid \textit{labelE}(j, i, s) \in X\} \\ &\cup \{\textit{opposite}(j, i) \mid \textit{opposite}(j, i) \in X\})) \\ \cup\ &\{\textit{labelV}(i, s) \mid i \in V, \mu'(i) = s\} \\ \cup\ &\{\textit{labelE}(j, i, s) \mid (j \rightarrow i) \in E, \sigma'(j, i) = s\} \\ \cup\ &\{\textit{opposite}(j, i) \mid (j \rightarrow i) \in E, \mu'(i) \neq \mu'(j)\sigma'(j, i)\}\,. \end{aligned}$$

Since bottom $\in X \setminus Y$ *and* X *contains a maximum amount of atoms over predicates labelV, labelE, and opposite (the atoms over opposite are consequences of the inclusion of atoms over labelV and labelE), we have that* $Y \subset X$*, and we show that* Y *is a model of* P^X*.*

Considering the contributions of the facts in $\tau((V, E, \sigma), \mu)$ *and the rules in (2.9) to* P^X*, we observe that the atoms over predicates occurring in them are interpreted the same in* X *and* Y*. Hence, such facts and rules stay satisfied by* Y *because they were already satisfied by* X*. The same applies to the rules from (2.7) repeated in (A.4) and (A.5). Furthermore, since* σ' *and* μ' *are total and extend* σ *and* μ*, respectively, the contributions of the following rules from (2.4) and (2.7) to* P^X *are satisfied by* Y*:*

$$\begin{aligned} \textit{labelV}(V, S) &\leftarrow \textit{observedV}(V, S). \\ \textit{labelE}(U, V, S) &\leftarrow \textit{observedE}(U, V, S). \\ \textit{labelV}(V, +); \textit{labelV}(V, -) &\leftarrow \textit{vertexMIC}(V). \\ \textit{labelE}(U, V, +); \textit{labelE}(U, V, -) &\leftarrow \textit{edgeMIC}(U, V). \end{aligned}$$

Since the integrity constraint in (A.6) does not belong to P^X *and the rules in (A.7) are satisfied by* Y *in view of bottom* $\notin Y$*, it remains to consider the following rules from (2.8):*

$$\begin{aligned} \textit{opposite}(U, V) &\leftarrow \textit{labelE}(U, V, -), \textit{labelV}(U, S), \textit{labelV}(V, S). \\ \textit{opposite}(U, V) &\leftarrow \textit{labelE}(U, V, +), \textit{labelV}(U, S), \textit{labelV}(V, T), S \neq T. \\ \textit{bottom} &\leftarrow \textit{active}(V), \textit{opposite}(U, V) : \textit{edge}(U, V). \end{aligned}$$

The rules defining predicate opposite are such that, in order to satisfy their ground instances in P^X, Y *must contain opposite*(j, i) *if labelE*(j, i, r)*, labelV*(j, s)*, and labelV*(i, t) *belong to* Y *such that* $t \neq sr$*. This matches the definition of* Y*, including labelE*(j, i, r) *if* $\sigma'(j, i) = r$*, labelV*(j, s) *if* $\mu'(j) = s$*, labelV*(i, t) *if* $\mu'(i) = t$*, and opposite*(j, i) *if*

$\mu'(i) \neq \mu'(j)\sigma'(j,i)$. *Hence, rules defining opposite in* P^X *are satisfied by* Y. *It remains to be shown that bottom is not derivable from any ground instance of the last rule. In this regard, recall that* $W = \{i \mid active(i) \in X\} = \{i \mid active(i) \in Y\}$, *and we have seen above that active*(i) *can only belong to* X *if* i *is not an input. As* σ' *and* μ' *are witnessing labelings for* W, *for every* $i \in W$, *there is an edge* $(j \rightarrow i) \in E$ *such that* $\mu'(i) = \mu'(j)\sigma'(j,i)$. *By the definition of* Y, *this implies opposite*$(j,i) \notin Y$, *while edge*(j,i) *belongs to* X *and* Y *because* X *and* Y *are models of* $\tau((V,E,\sigma),\mu)$. *As a consequence, for every* $i \in W$, *we have* $\{opposite(j,i) \mid edge(j,i) \in Y\} \not\subseteq Y$, *so that the ground instance for* i *in* P^X *of the rule with bottom in the head is satisfied by* Y. *We have thus established that* $Y \subset X$ *is indeed a model of* P^X, *a contradiction to the assumption that* X *is a* $\subseteq$*-minimal model of* P^X *and an answer set of* $P_D \cup \tau((V,E,\sigma),\mu)$.

The above contradiction shows that the second condition to verify, which is the first condition in Definition 2.3, holds for $W = \{i \mid active(i) \in X\}$. *The fact that the second condition in Definition 2.3 holds for* W *has been shown before. Hence,* W *is a MIC.*

Proof A.4 (Proof of Theorem 2.5) *Assume that* $W = \{k_1, \ldots, k_n\}$ *is a MIC. Then, the following conditions hold:*

1. *There are witnessing labelings* $\sigma_1, \mu_1, \ldots, \sigma_n, \mu_n$ *for* $W \setminus \{k_1\}, \ldots, W \setminus \{k_n\}$.
2. *There are no witnessing labelings for* W.

We consider the following set X *of atoms:*

$$
\begin{array}{lll}
X = & \{vertex(i) & \mid i \in V\} \\
& \cup \{edge(j,i) & \mid (j \rightarrow i) \in E\} \\
& \cup \{observedE(j,i,s) & \mid (j \rightarrow i) \in E, \sigma(j,i) = s\} \\
& \cup \{observedV(i,s) & \mid i \in V, \mu(i) = s\} \\
& \cup \{input(i) & \mid i \in V \text{ is an input}\} \\
& \cup \{active(i) & \mid i \in W\} \\
& \cup \{inactive(i) & \mid i \in V \setminus W \text{ is not an input}\} \\
& \cup \{edgeMIC(j,i) & \mid (j \rightarrow i) \in E, i \in W\} \\
& \cup \{vertexMIC(j) & \mid (j \rightarrow i) \in E, i \in W\} \\
& \cup \{vertexMIC(i) & \mid i \in W\} \\
& \cup \{labelE'(k_m,j,i,r) & \mid (j \rightarrow i) \in E, i \in W, \sigma_m(j,i) = r, 1 \leq m \leq n\} \\
& \cup \{labelE'(k_m,j,i,r) & \mid (j \rightarrow i) \in E, \sigma(j,i) = r, 1 \leq m \leq n\} \\
& \cup \{labelV'(k_m,j,s) & \mid (j \rightarrow i) \in E, i \in W, \mu_m(j) = s, 1 \leq m \leq n\} \\
& \cup \{labelV'(k_m,i,s) & \mid i \in W, \mu_m(i) = s, 1 \leq m \leq n\} \\
& \cup \{labelV'(k_m,i,s) & \mid i \in V, \mu(i) = s, 1 \leq m \leq n\} \\
& \cup \{receive'(k_m,i,sr) & \mid (j \rightarrow i) \in E, i \in W, \\
& & \quad \sigma_m(j,i) = r, \mu_m(j) = s, i \neq k_m, 1 \leq m \leq n\} \\
& \cup \{receive'(k_m,i,sr) & \mid (j \rightarrow i) \in E, j \in W \text{ or } (j \rightarrow k) \in E \text{ for } k \in W, \\
& & \quad \sigma(j,i) = r, \mu_m(j) = s, i \neq k_m, 1 \leq m \leq n\} \\
& \cup \{receive'(k_m,i,sr) & \mid (j \rightarrow i) \in E, \\
& & \quad \sigma(j,i) = r, \mu(j) = s, i \neq k_m, 1 \leq m \leq n\} \\
& \cup \{labelV(i,+), labelV(i,-) & \mid i \in V\} \\
& \cup \{labelE(j,i,+), labelE(j,i,-) & \mid (j \rightarrow i) \in E\} \\
& \cup \{opposite(j,i) & \mid (j \rightarrow i) \in E\} \\
& \cup \{bottom\}\,. &
\end{array}
$$

For showing that X is an answer set of $P_D \cup \tau((V,E,\sigma),\mu)$ (such that $\{i \mid \textit{active}(i) \in X\} = W$), we need to verify that X is a $\subseteq$-minimal model of

$$P^X = \{(\mathit{head}(r) \leftarrow \mathit{body}(r)^+)\theta \mid \\ r \in P_D \cup \tau((V,E,\sigma),\mu), (\mathit{body}(r)^-\theta) \cap X = \emptyset, \theta : \mathit{var}(r) \to \mathcal{U}\}$$

where $\mathit{var}(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_D \cup \tau((V,E,\sigma),\mu)$, and θ is a ground substitution for the variables in r.

To start with, we note that X includes an atom vertex(i), edge(j,i), observedE(j,i,s), observedV(i,s), and input(i), respectively, exactly if there is a fact with the atom in the head in $\tau((V,E,\sigma),\mu)$. Each of these facts belongs also to P^X, is satisfied by X, but not by any set Y of atoms excluding at least one of the head atoms.

In view that W cannot contain any input (otherwise, satisfaction of the second condition in Definition 2.3 would immediately imply violation of the first one), we have that either active(i) or inactive(i) belongs to X for every non-input vertex $i \in V$. Hence, X satisfies all ground instances of the rule

$$\textit{active}(V); \textit{inactive}(V) \leftarrow \textit{vertex}(V), \mathit{not}\ \textit{input}(V).$$

from (2.7) belonging to P^X, while no set Y of atoms excluding both active(i) and inactive(i) for any non-input vertex $i \in V$ satisfies all of these ground instances.

Considering ground instances of the rules

$$\begin{aligned} \textit{edgeMIC}(U,V) &\leftarrow \textit{edge}(U,V), \textit{active}(V). \\ \textit{vertexMIC}(U) &\leftarrow \textit{edgeMIC}(U,V). \\ \textit{vertexMIC}(V) &\leftarrow \textit{active}(V). \end{aligned}$$

from (2.7), all of them belong to P^X, are satisfied by X, but not by any set Y of atoms such that $\{\textit{edgeMIC}(j,i) \mid \textit{edgeMIC}(j,i) \in X\} \cup \{\textit{vertexMIC}(i) \mid \textit{vertexMIC}(i) \in X\} \not\subseteq Y$ and $\{\textit{active}(i) \mid \textit{active}(i) \in X\} \subseteq \{\textit{active}(i) \mid \textit{active}(i) \in Y\}$, while it has been shown above that $\{\textit{active}(i) \mid \textit{active}(i) \in X\} \not\subseteq \{\textit{active}(i) \mid \textit{active}(i) \in Y\}$ necessitates $\{\textit{inactive}(i) \mid \textit{inactive}(i) \in Y\} \not\subseteq \{\textit{inactive}(i) \mid \textit{inactive}(i) \in X\}$ for Y being a model of P^X. Hence, there cannot be any model $Y \subset X$ of P^X excluding some atom edgeMIC(j,i) or vertexMIC(i) that belongs to X.

Now turning our attention to atoms of form labelE'(k_m,j,i,r) and labelV'(k_m,j,s), we note that they are included in X if edgeMIC$(j,i) \in X$ and vertexMIC$(j) \in X$, respectively, and $\sigma_m(j,i) = r, \mu_m(j) = s$ in witnessing labelings σ_m and μ_m for $W \setminus \{k_m\}$, where $1 \leq m \leq n$, or if $\sigma(j,i) = r, \mu(j) = s$. Then, the fact that active$(k_m) \in X$ and labels assigned by σ_m and μ_m are unique and respect those assigned by σ and μ implies that none of the atoms can be removed from X without violating some ground instance of the rules

$$\begin{aligned} \textit{labelV'}(W,V,+); \textit{labelV'}(W,V,-) &\leftarrow \textit{active}(W), \textit{vertexMIC}(V). \\ \textit{labelE'}(W,U,V,+); \textit{labelE'}(W,U,V,-) &\leftarrow \textit{active}(W), \textit{edgeMIC}(U,V). \\ \textit{labelV'}(W,V,S) &\leftarrow \textit{active}(W), \textit{observedV}(V,S). \\ \textit{labelE'}(W,U,V,S) &\leftarrow \textit{active}(W), \textit{observedE}(U,V,S). \end{aligned}$$

from (2.9) that belongs to P^X. However, X satisfies all of these ground instances by its

construction. We further consider the following rules from (2.9):

$$\textit{receive'}(W,V,\textbf{+}) \leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,S), V \neq W.$$
$$\textit{receive'}(W,V,\textbf{-}) \leftarrow \textit{labelE'}(W,U,V,S), \textit{labelV'}(W,U,T), V \neq W, S \neq T.$$

As shown above, labelE'(k_m,j,i,r) *belongs to* X *if* $i \in W$ *and* $\sigma_m(j,i) = r$*, or if* $\sigma(j,i) = \sigma_m(j,i) = r$*. Furthermore, labelV'*$(k_m,j,s)$ *is included in* X *if* $j \in W$ *or* $(j \rightarrow k) \in E, k \in W$ *and* $\mu_m(j) = s$*, or if* $\mu(j) = \mu_m(j) = s$*. Comparing the cross product of these conditions to the definition of* X *yields that an atom receive'*(k_m,i,sr) *belongs to* X *exactly if labelE'*(k_m,j,i,r) *and labelV'*(k_m,j,s) *are in* X *and* $i \neq k_m$*. Hence, when excluding any of the atoms receive'*(k_m,i,sr) *from* X*, some ground instance of the above two rules belonging to* P^X *becomes unsatisfied, and so we have that such atoms cannot be removed from* X *in order to construct a model* $Y \subset X$ *of* P^X*. Moreover, the fact that* σ_m *and* μ_m *are witnessing labelings for* $W' = W \setminus \{k_m\}$ *implies that all ground instances of the integrity constraint*

$$\leftarrow \textit{labelV'}(W,V,S), \textit{active}(V), V \neq W, \textit{not receive'}(W,V,S).$$

from (2.9) that belong to P^X *are satisfied by* X*. In fact, for every* $i \in W'$*, there is some* $(j \rightarrow i) \in E$ *such that* $\mu_m(i) = \mu_m(j)\sigma_m(j,i)$*. Since labelE'*$(k_m,j,i,\sigma_m(j,i))$ *and labelV'*$(k_m,j,\mu_m(j))$ *belong to* X*, this implies that each atom labelV'*$(k_m,i,\mu_m(i))$ *for* $i \in W'$ *is accompanied by receive'*$(k_m,i,\mu_m(i))$ = *receive'*$(k_m,i,\mu_m(j)\sigma_m(j,i))$ *in* X*, so that the ground instance for* k_m*,* i*, and* $\mu_m(i)$ *of the integrity constraint is not in* P^X*.*

Finally, we consider atoms of the form labelV(i,s)*, labelE*(j,i,s)*, and opposite*(j,i) *that belong to* X *for all* $i \in V$ *and* $(j \rightarrow i) \in E$*, respectively, and* $s \in \{\textbf{+},\textbf{-}\}$*. Since bottom is also in* X*, it is clear that the ground instances of the following rules from (2.4), (2.7), and (2.8), all of which belong to* P^X*, are satisfied by* X*:*

$$\begin{aligned}
\textit{labelV}(V,S) &\leftarrow \textit{observedV}(V,S).\\
\textit{labelE}(U,V,S) &\leftarrow \textit{observedE}(U,V,S).\\
\textit{labelV}(V,\textbf{+}); \textit{labelV}(V,\textbf{-}) &\leftarrow \textit{vertexMIC}(V).\\
\textit{labelE}(U,V,\textbf{+}); \textit{labelE}(U,V,\textbf{-}) &\leftarrow \textit{edgeMIC}(U,V).\\
\textit{opposite}(U,V) &\leftarrow \textit{labelE}(U,V,\textbf{-}), \textit{labelV}(U,S), \textit{labelV}(V,S).\\
\textit{opposite}(U,V) &\leftarrow \textit{labelE}(U,V,\textbf{+}), \textit{labelV}(U,S), \textit{labelV}(V,T), S \neq T.\\
\textit{bottom} &\leftarrow \textit{active}(V), \textit{opposite}(U,V) : \textit{edge}(U,V).\\
\textit{labelV}(V,\textbf{+}) &\leftarrow \textit{bottom}, \textit{vertex}(V).\\
\textit{labelV}(V,\textbf{-}) &\leftarrow \textit{bottom}, \textit{vertex}(V).\\
\textit{labelE}(U,V,\textbf{+}) &\leftarrow \textit{bottom}, \textit{edge}(U,V).\\
\textit{labelE}(U,V,\textbf{-}) &\leftarrow \textit{bottom}, \textit{edge}(U,V).
\end{aligned}$$

As shown above, any model $Y \subseteq X$ *of* P^X *must necessarily include observedV*(i,s) *if* $\mu(i) = s$*, observedE*(j,i,s) *if* $\sigma(j,i) = s$*, vertexMIC*(i) *if* $i \in W$ *or* $(i \rightarrow k) \in E$ *for some* $k \in W$*, edgeMIC*(j,i) *if* $(j \rightarrow i) \in E$ *for some* $i \in W$*, and active*(i) *if* $i \in W$*. Proceeding by proof by contradiction, assume that there is a model* $Y \subset X$ *of* P^X *such that labelV*(i,s)*, labelE*(j,i,s)*, or opposite*(j,i) *is not in* Y *for some* $i \in V$ *or* $(j \rightarrow i) \in E$*, respectively, and* $s \in \{\textbf{+},\textbf{-}\}$*. From the previous considerations and the first two rules repeated above, we know that labelV*(i,s) *and labelE*(j,i,s) *must belong*

to Y if $\mu(i) = s$ or $\sigma(j,i) = s$, respectively. Furthermore, the third rule necessitates $\{labelV(i,+), labelV(i,-)\} \cap Y \neq \emptyset$ for every $i \in W$ or $i \in V$ such that $(i \rightarrow k) \in E$ for some $k \in W$, and the fourth rule implies $\{labelE(j,i,+), labelE(j,i,-)\} \cap Y \neq \emptyset$ for every $(j \rightarrow i) \in E$ such that $i \in W$. In view of the last four rules, we immediately conclude that bottom $\notin Y$, which in turn implies that, for every $i \in W$, there is some $(j \rightarrow i) \in E$ such that opposite(j,i) does not belong to Y. Comparing the rules defining opposite, the exclusion of opposite(j,i) is possible only if Y does not include labelE(j,i,r), labelV(j,s), and labelV(i,t) such that $t \neq sr$. As we have shown above that some atoms labelE(j,i,r), labelV(j,s), and labelV(i,t) for $r,s,t \in \{+,-\}$ must belong to Y, we can now conclude that $t = sr$ holds and that the atoms over predicates labelE and labelV in Y define (partial) labelings σ' and μ' by:

- *For every $i \in W$, pick some edge $(j \rightarrow i) \in E$ such that opposite(j,i) does not belong to Y, and let $\sigma'(j,i) = r$ if labelE$(j,i,r) \in Y$, $\mu'(j) = s$ if labelV$(j,s) \in Y$, and $\mu'(i) = t$ if labelV$(i,t) \in Y$.*

As we have seen above, such an edge $(j \rightarrow i) \in E$ exists for every $i \in W$, and the fact that $t \neq sr$ is not obtained for atoms labelE(j,i,r), labelV(j,s), and labelV(i,t) in Y implies that σ' and μ' assign unique labels to $(j \rightarrow i)$, j, and i, respectively. When we totalize σ' and μ' by setting $\sigma'(j,i) = \sigma(j,i)$ and $\mu'(i) = \mu(i)$ if $\sigma(j,i)$ or $\mu(i)$, respectively, is defined, and $\sigma'(j,i) = +$ as well as $\mu'(i) = +$ for all remaining edges in E and vertices in V, we obtain witnessing labelings for W. But this is a contradiction to the fact that W is a MIC, which allows us to conclude that there cannot be any model $Y \subset X$ of P^X that omits labelV(i,s), labelE(j,i,s), or opposite(j,i) for some $i \in V$ or $(j \rightarrow i) \in E$, respectively, and $s \in \{+,-\}$.

To conclude the proof that X is a $\subseteq$-minimal model of P^X, note that the integrity constraint

$$\leftarrow \mathit{not}\ \mathit{bottom}.$$

from (2.8) does not contribute any rule to P^X because bottom $\in X$. We have now investigated all rules in $P_D \cup \tau((V,E,\sigma),\mu)$ and shown that their ground instances in P^X are satisfied by X. Furthermore, we have checked for all atoms in X that they cannot be excluded in any model $Y \subset X$ of P^X. That is, X is indeed a $\subseteq$-minimal model of P^X and thus an answer set of $P_D \cup \tau((V,E,\sigma),\mu)$.

A.3 Proof of Theorem 3.1

We formalize the representation of instances, as described in Section 3.3.1, by defining a mapping τ of a metabolic network network completion problem (G_d, G_n, T, S) as

follows:

$$\tau(G_d, G_n, T, S) = \begin{cases} \textit{draft}(d). & \\ \textit{reaction}(r, d). & \mid r \in R(G_d) \\ \textit{reactant}(m, r). & \mid r \in R(G_d),\ m = \textit{reac}(r) \\ \textit{product}(m, r). & \mid r \in R(G_d),\ m = \textit{prod}(r) \\ \textit{reaction}(r, n). & \mid r \in R(G_n) \\ \textit{reactant}(m, r). & \mid r \in R(G_n),\ m = \textit{reac}(r) \\ \textit{product}(m, r). & \mid r \in R(G_n),\ m = \textit{prod}(r) \\ \textit{seed}(m). & \mid m \in S \\ \textit{target}(m). & \mid m \in T \end{cases} \tag{A.8}$$

By P_C, we denote the encoding containing the schematic rules in (3.1),(3.3),(3.4),(3.5).

Proof A.5 (Proof of Theorem 3.1, Soundness) *Assume that X is an answer set of $P_C \cup \tau(G_d, G_n, T, S)$. Furthermore, let*

$$P^X = \{(head(r) \leftarrow body(r)^+)\theta \mid r \in P_C \cup \tau(G_d, G_n, T, S), (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \rightarrow \mathcal{U}\}$$

where $var(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_C \cup \tau(G_d, G_n, T, S)$, and θ is a ground substitution for the variables in r. Then, by the definition of an answer set, we know that X is a $\subseteq$-minimal model of P^X.

Given X, we define R_C as follows:

$$R_C = \{r \mid \textit{xreaction}(r) \in X\}.$$

We show that R_C is a completion of G_d from G_n wrt (S, T), such that $R_C \subseteq R(G_n) \setminus R(G_d)$ and $T \subseteq \Sigma_G(S)$ where

$$\begin{aligned} G &= ((R(G_d) \cup R_C) \cup (M(G_d) \cup M_C), E(G_d) \cup E_C)\ , \\ M_C &= \{m \mid r \in C, m \in \textit{reac}(r) \cup \textit{prod}(r)\}\ , \textit{ and} \\ E_C &= \{(m, r) \mid r \in C, m \in \textit{reac}(r)\} \cup \{(r, m) \in E(G_n) \mid r \in C, m \in \textit{prod}(r)\}\ . \end{aligned}$$

Consider the following rules from (3.1):

$$\begin{aligned} \textit{dscope}(M) &\leftarrow \textit{seed}(M). \\ \textit{dscope}(M) &\leftarrow \textit{product}(M, R), \textit{reaction}(R, N), \textit{draft}(N), \\ & \qquad \textit{dscope}(M') : \textit{reactant}(M', R). \end{aligned} \tag{3.1}$$

The first rule declares all seed metabolites $M \in S$ as producible. The second rule defines recursively that a product M of a reaction R is producible, whenever all reactants M' of R are available. Together with the problem encoding $\tau(G_d, G_n, T, S)$, the set of rules in (3.1) enforce that the atoms in the head are in every $\subseteq$-minimal model such that dscope$(m) \in X$ iff $m \in \Sigma_{G_d}(S)$ for $m \in M(G_d)$.

Further, the rule (3.3):

$$\{\textit{xreaction}(R) : \textit{reaction}(R, N) : not\ \textit{draft}(N)\}. \tag{3.3}$$

combined with the problem instance $\tau(G_d, G_n, T, S)$, guarantees that all chosen xreaction(R) reactions belong to $R(G_n) \setminus R(G_d)$. Hence, for every $\subseteq$-minimal model X it holds that xreaction$(r) \in X$ iff $r \in R(G_n) \setminus R(G_d)$.

Further, consider the following rules from (3.4):

$$
\begin{array}{rcl}
\textit{xscope}(M) & \leftarrow & \textit{seed}(M). \\
\textit{xscope}(M) & \leftarrow & \textit{product}(M,R), \textit{reaction}(R,N), \textit{draft}(N), \\
 & & \textit{xscope}(M') : \textit{reactant}(M',R). \\
\textit{xscope}(M) & \leftarrow & \textit{product}(M,R), \textit{xreaction}(R), \\
 & & \textit{xscope}(M') : \textit{reactant}(M',R).
\end{array}
\tag{3.4}
$$

In analogy to the rules from (3.1), these rules define the (extended) scope of $G = ((R(G_d) \cup R_C) \cup (M(G_d) \cup M_C), E(G_d) \cup E_C)$. The first two rules declare all seed metabolites $M \in S$ and recursively all products M of a reaction $R \in R(G_d)$ as producible, whenever all reactants M' of R are available. The third rule extends the scope with the products M of xreactions $R \in R_C$, whenever all reactants M' of R are available. Hence, for every $\subseteq$-minimal model it must hold that xscope$(m) \in X$ iff $m \in \Sigma_G(S)$ for $m \in M(G_d) \cup M_C$.

Finally, it remains to be shown that the extended scope contains all targets from T. To this end, note that the integrity constraint from (3.5):

$$
\leftarrow \textit{target}(M), \textit{not } \textit{xscope}(M). \tag{3.5}
$$

necessitates that every target $M \in T$ is also in the extended scope $M \in \Sigma_G(S)$ and it holds that $T \subseteq \Sigma_G(S)$.

Hence, every answer set of a logic program $P_C \cup \tau(G_d, G_n, T, S)$ represents a solution to the network completion problem (G_d, G_n, T, S).

Proof A.6 (Proof of Theorem 3.1, Completeness) *Assume that R_C is a completion of G_d from G_r wrt. (S, T). We consider the following set X of atoms:*

$$
X = \left\{
\begin{array}{ll}
\textit{draft}(d) & \\
\textit{reaction}(r,d) & \mid r \in R(G_d) \\
\textit{reactant}(m,r) & \mid r \in R(G_d),\ m = reac(r) \\
\textit{product}(m,r) & \mid r \in R(G_d),\ m = prod(r) \\
\textit{reaction}(r,n) & \mid r \in R(G_n) \\
\textit{reactant}(m,r) & \mid r \in R(G_n),\ m = reac(r) \\
\textit{product}(m,r) & \mid r \in R(G_n),\ m = prod(r) \\
\textit{seed}(m) & \mid m \in S \\
\textit{target}(m) & \mid m \in T \\
\textit{dscope}(m) & \mid m \in \Sigma_{G_d} \\
\textit{xreaction}(r) & \mid r \in R_C \\
\textit{xscope}(m) & \mid m \in \Sigma_G
\end{array}
\right.
\tag{A.9}
$$

For showing that X is an answer set of $P_C \cup \tau(G_d, G_n, T, S)$, we need to verify that X is a $\subseteq$-minimal model of

$$
\begin{array}{l}
P^X = \{(head(r) \leftarrow body(r)^+)\theta \mid \\
\qquad r \in P_C \cup \tau(G_d, G_n, T, S), (body(r)^-\theta) \cap X = \emptyset, \theta : var(r) \rightarrow \mathcal{U}\}
\end{array}
$$

where $var(r)$ is the set of all variables that occur in a rule r, $\mathcal{U}$ is the set of all constants appearing in $P_C \cup \tau(G_d, G_n, T, S)$, and θ is a ground substitution for the variables in r.

To start with, we note that X includes an atom draft(d), reaction(r, n), reactant(m, r), product(m, r), seed(m), target(m) exactly if there is a fact with the atom in the head in $\tau(G_d, G_n, T, S)$. Each of these facts belongs also to P^X, is satisfied by X, but not by any set Y of atoms excluding at least one of the head atoms.

Furthermore, X satisfies the rules in (3.1) by including atoms dscope(m) for $m \in \Sigma_{G_d}$. X contains an atom dscope(m) if it contains a corresponding atom seed(m) or if it includes the atoms product(m, r), reaction(r, d), draft(d), and for every atom reactant$(m', r) \in X$ exists an atom dscope$(m) \in X$. Hence, no subset of X excluding atleast one atom over predicate dscope would satisfy all ground instances of (3.1) in P^X.

The ground instantiations of the choice rule (3.3) are trivaly satisfied by X, which includes atoms xreaction(r) such that $r \in R_C$ and $R_C \cap R(G_d) = \emptyset$.

In analogy to the rules in (3.1), the rules in (3.4) are satisfied by X including an atom xscope(m) if $m \in \Sigma_G$. X contains an atom xscope(m) if X contains a coreponding atom seed(m) or if X contains the atoms product(m, r), reaction(r, d), draft(d), and for every atom reactant$(m', r) \in X$ exists an atom xscope$(m) \in X$ or if X contains the atoms product(m, r), xreaction(r), and for every atom reactant$(m', r) \in X$ exists an atom xscope$(m) \in X$. Hence, no subset of X excluding atleast one atom over predicate xscope would satisfy all ground instances of (3.4) in P^X.

Finally, all ground instantiations of the integrity constraint (3.5) are satisfied by X. By the definition of completion in Section 3.2, it holds that $T \subseteq \Sigma_G(S)$. Hence, for each atom target$(m) \in X$ exists an atom xscope$(m) \in X, m \in \Sigma_G$.

We have now investigated all rules in $P_C \cup \tau(G_d, G_n, T, S)$ and shown that their ground instances in P^X are satisfied by X. Furthermore, we have checked for all atoms in X that they cannot be excluded in any model $Y \subset X$ of P^X. That is, X is indeed a $\subseteq$-minimal model of P^X and thus an answer set of $P_C \cup \tau(G_d, G_n, T, S)$.

List of Figures

2.1 Simplified model of lactose operon in *Escherichia coli*, represented as an influence graph. The vertices represent either genes, metabolites, or proteins, while the edges indicate the regulations among them. Edges with an arrow stand for positive regulations (activations), while edges with a tee head stand for negative regulations (inhibitions). Vertices G and L_e are considered to be inputs of the system, that is, their signs are not constrained via their incoming edges. 9
2.2 An influence graph (left) along with two experimental profiles (middle and right), in which increases (decreases) have been observed for vertices colored green (red), and vertex d is an input. 11
2.3 Facts representing the influence graph and experimental profile p_1 (middle) from Figure 2.2 in Π_g and Π_{p_1}, respectively 11
2.4 A partially labeled influence graph with uncritical vertices surrounded by dots. 14
2.5 A partially labeled influence graph and a MIC consisting of a and d. . . . 16
2.6 A partially labeled influence graph and the graph $(V[\{\mathbf{A},\mathbf{D}\}],E[\{\mathbf{A},\mathbf{D}\}])$. 20
2.7 An influence graph (left) along with two experimental profiles (middle and right), in which increases (decreases) have been observed for vertices colored green (red), and vertex d is an input. 22
2.8 Facts representing the influence graph and experimental profiles from Figure 2.7 in Π_g, Π_{p_1}, and Π_{p_2}, respectively. 23
2.9 Some MICs obtained by comparing the regulatory network of yeast with a genetic profile. 30
2.10 Subgraph obtained by connecting the six MICs given in Figure 2.9. 31
2.11 Local correction of the network based on our diagnosis method and literature research. 32

3.1 Glycolysis. A typical illustration of a metabolic pathway. 41
3.2 Given the seed metabolites m1, m2 and m12, the scope of this network is {m1, m2, m3, m4, m5, m6, m12, m13}. 42
3.3 Given the seed metabolites m1, m2 and m12, the completions {r10} and {r8, r9} can repair the production pathway for target m10. 43
3.4 Given the target metabolites m10 and m5, the minimal nutrition sets are {m12},{m11,m13}, {m1,m9,m13},{m7,m9,m13}. 48

4.1 Architecture of the BioASP Library. 59
4.2 A *TermSet* of ASP facts representing metabolic networks, seeds and targets 60
4.3 BioQuali format for networks (left) and the *TermSet* containing the ASP facts (right) 63

4.4 BioQuali format for observations (left) and the corresponding *TermSet* (right). 63
4.5 Minimal diagnoses as a list of *TermSet*s. 64
4.6 A *TermSet* of ASP facts representing possible repair operations. 65
4.7 A list containing a minimal repair set as *TermSet*. 66
4.8 Predicted variations as a *TermSet* 67
4.9 Workflow of the “Influence Graph Analyzer” application. 69
4.10 Textual representation of diagnosis results. 70
4.11 Textual representation of the predicted variations. 70
4.12 Dialog for choosing the repair mode. 71
4.13 Textual representation of a minimal repair set. 72
4.14 Textual representation of unproducible target metabolites. 72
4.15 Textual representation of target metabolites whose production pathways can be restored. 73
4.16 Textual representation of essential reactions. 73
4.17 Text output for the computation of completion minima. 74
4.18 Textual representation of a minimal completion. 74
4.19 Web interface for consistency checking and diagnosis. 75
4.20 Representation of identified MICs in textual (left) and graphical (right) mode. 76

List of Tables

2.1 Some vertex labelings (reflecting measurements of two steady states) for the influence graph depicted in Figure 2.1; unobserved values indicated by question mark '**?**'. . . . 10
2.2 Run-times for consistency checking with *claspD*, *cmodels*, *dlv*, and *gnt*. . 28
2.3 Run-times for grounding with *gringo* and solving with *claspD*. 29
2.4 Repair Times. . . . 33
2.5 Prediction Times. . . . 34
2.6 Prediction Rate. . . . 35
2.7 Prediction Accuracy. . . . 36

3.1 computing optimal completions for E. coli networks 51
3.2 Completion with 5000,...,9000 reactions. . . . 54
3.3 Computing minimal seeds for E.coli targets. . . . 55

Bibliography

[1] T. Allen, M. Herrgård, M. Liu, Y. Qiu, J. Glasner, F. Blattner, and B. Palsson. Genome-scale analysis of the uses of the Escherichia coli genome: Model-driven analysis of heterogeneous data sets. *Journal of Bacteriology*, 185(21):6392–6399, 2003.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Backofen, S. Will, and E. Bornberg-Bauer. Application of constraint programming techniques for structure prediction of lattice proteins with extended alphabets. *Bioinformatics*, 15(3):234–242, 1999.

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

[5] 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*. Springer-Verlag, 2007.

[6] 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.

[7] C. Baral, G. Greco, N. Leone, and G. Terracina, editors. *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662, 2005.

[8] E. G. Belau, F. Schreiber, M. Heiner, A. Sackmann, B. Junker, S. Grunwald, A. Speer, K. Winder, and I. Koch. Modularization of biochemical networks based on classification of petri net t-invariants. *BMC Bioinformatics*, 9(1):90+, 2008.

[9] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):53–87, 1994.

[10] A. Biere, M.J.H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[11] `http://www.cs.uni-potsdam.de/wv/bioasp`.

[12] G. Boenn, M. Brain, M. de Vos, and J. Fitch. Automatic composition of melodic and harmonic music by answer set programming. In Garcia de la Banda and Pontelli [40], pages 160–174.

[13] H. Bonarius, G. Schmid, and J. Tramper. Flux analysis of underdetermined metabolic networks: The quest for the missing constraints. *Trends Biotechnology*, 15:308–314, 1997.

[14] H. P. J. Bonarius, G. Schmid, and J. Tramper. Flux analysis of underdetermined metabolic networks: the quest for the missing constraints. *Trends in Biotechnology*, 15(8):308 – 314, 1997.

[15] B. J. Bornstein, S. M. Keating, A. Jouraku, and M. Hucka. Libsbml: an api library for sbml. *Bioinformatics (Oxford, England)*, 24(6):880–881, March 2008.

[16] M. Bradley, M. Beach, A. de Koning, T. Pratt, and R. Osuna. Effects of Fis on Escherichia coli gene expression during different growth stages. *Microbiology*, 153:2922–2940, 2007.

[17] R. Caspi, H. Foerster, C. A. Fulcher, P. Kaipa, M. Krummenacker, M. Latendresse, S. Paley, S. Y. Rhee, A. G. Shearer, C. Tissier, T. C. Walk, P. Zhang, and P. D. Karp. The metacyc database of metabolic pathways and enzymes and the biocyc collection of pathway/genome databases. *Nucleic Acids Res*, October 2007.

[18] M. Chen, L. Hancock, and J. Lopes. Transcriptional regulation of yeast phospholipid biosynthetic genes. *Biochimica et Biophysica Acta*, 1771(3):310–21, 2007.

[19] N. Christian, P. May, S. Kempa, T. Handorf, and O. Ebenhöh, 2008. Personal communication.

[20] N. Christian, P. May, S. Kempa, T. Handorf, and O. Ebenhöh. An integrative approach towards completing genome-scale metabolic networks. *Molecular BioSystems*, 5:1889–1903, 2009.

[21] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, June 2009.

[22] F. Corblin, L. Bordeaux, Y. Hamadi, E. Fanchon, and L. Trilling. A sat-based approach to decipher gene regulatory networks. In *Integrative Post-Genomics, RIAMS, Lyon*, 2007.

[23] J. Delgrande, D. Liu, T. Schaub, and S. Thiele. Coba 2.0: A consistency-based belief change system. In Dix and Hunter [27], pages 267–273.

[24] J. Delgrande, D. Liu, T. Schaub, and S. Thiele. Coba 2.0: A consistency-based belief change system. In K. Mellouli, editor, *Proceedings of the Ninth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'07)*, volume 4724 of *Lecture Notes in Artificial Intelligence*, pages 78–90. Springer-Verlag, 2007.

[25] N. Dershowitz, Z. Hanna, and A. Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In A. Biere and C. Gomes, editors, *Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 36–41. Springer-Verlag, 2006.

[26] Y. Dimopoulos, B. Nebel, and J. Köhler. Encoding planning problems in nonmonotonic logic programs. In S. Steel and R. Alami, editors, *Proceedings of the Fourth European Conference on Planning*, volume 1348 of *Lecture Notes in Artificial Intelligence*, pages 169–181. Springer-Verlag, 1997.

[27] J. Dix and A. Hunter, editors. *Proceedings of the International Workshop on Nonmonotonic Reasoning (NMR'06)*, number IFI-06-04 in Technical Report Series. Clausthal University of Technology, Institute for Informatics, 2006.

[28] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub. Conflict-driven disjunctive answer set solving. In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[29] S. Dworschak, S. Grell, V. Nikiforova, T. Schaub, and J. Selbig. Modeling biological networks by action languages via answer set programming. *Constraints*, 13(1-2):21–65, 2008.

[30] S. Dworschak, T. Grote, A. König, T. Schaub, and P. Veber. The system bioc for reasoning about biological models in action language c^*. In *Proceedings of the Twentieth International Conference on Tools with Artificial Intelligence (ICTAI'08)*, volume 1, pages 11–18. IEEE Computer Society Press, 2008.

[31] O. Ebenhöh, T. Handorf, and R. Heinrich. Structural analysis of expanding metabolic networks. *Genome Informatics*, 15(1):35–45, 2004.

[32] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[33] E. Erdem, V. Lifschitz, and M. Wong. Wire routing and satisfiability planning. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic (CL'00)*, volume 1861 of *Lecture Notes in Computer Science*, pages 822–836. Springer-Verlag, 2000.

[34] E. Erdem, F. Lin, and T. Schaub, editors. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2009.

[35] E. Erdem and F. Türe. Efficient haplotype inference with answer set programming. In D. Fox and C. Gomes, editors, *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, pages 436–441. AAAI Press, 2008.

[36] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[37] François Fages and Aurélien Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theoretical Computer Science*, 408:55–65, November 2008.

[38] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3-4):601–620, 2000.

[39] S. Gama-Castro, V. Jiménez-Jacinto, M. Peralta-Gil, A. Santos-Zavaleta, M. Peñaloza-Spinola, B. Contreras-Moreira, J. Segura-Salazar, L. Muñiz-Rascado, I. Martínez-Flores, H. Salgado, C. Bonavides-Martínez, C. Abreu-Goodger, C. Rodríguez-Penagos, J. Miranda-Ríos, E. Morett, E. Merino, A. Huerta, L. Treviño-Quintanilla, and J. Collado-Vides. RegulonDB (version 6.0): gene regulation model of Escherichia coli K-12 beyond transcription, active (experimental) annotated promoters and Textpresso navigation. *Nucleic Acids Research*, 36:D120–D124, 2008.

[40] 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*. Springer-Verlag, 2008.

[41] 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.

[42] M. Gebser, T. Janhunen, M. Ostrowski, T. Schaub, and S. Thiele. A versatile intermediate language for answer set programming. In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 150–159, 2008.

[43] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In Garcia de la Banda and Pontelli [40], pages 190–205.

[44] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. Available at `http://potassco.sourceforge.net`, 2010.

[45] M. Gebser, R. Kaminski, M. Ostrowski, T. Schaub, and S. Thiele. On the input language of asp grounder gringo. In Erdem et al. [34], pages 502–508.

[46] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In Baral et al. [5], pages 260–265.

[47] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set enumeration. In Baral et al. [5], pages 136–148.

[48] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press/The MIT Press, 2007.

[49] M. Gebser, B. Kaufmann, and T. Schaub. The conflict-driven answer set solver clasp: Progress report. In Erdem et al. [34], pages 509–514.

[50] M. Gebser, B. Kaufmann, and T. Schaub. Solution enumeration for projected Boolean search problems. In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[51] M. Gebser, A. König, T. Schaub, S. Thiele, and P. Veber. The BioASP library: ASP solutions for systems biology. In E. Grégoire, editor, *Proceedings of the Twenty-second IEEE International Conference on Tools with Artificial Intelligence (ICTAI'10)*, pages 383–389. IEEE Computer Society, 2010.

[52] M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In Baral et al. [5], pages 266–271.

[53] M. Gebser, T. Schaub, S. Thiele, B. Usadel, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. In Garcia de la Banda and Pontelli [40], pages 130–144.

[54] 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):1–38, 2011.

[55] M. Gelfond, V. Lifschitz, H. Przymusinska, and M. Truszczyński. Disjunctive defaults. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 230–237. Morgan Kaufmann Publishers, 2091.

[56] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[57] J. Gressmann, T. Janhunen, R. Mercer, T. Schaub, S. Thiele, and R. Tichy. Platypus: A platform for distributed answer set solving. In Baral et al. [7], pages 227–239.

[58] J. Gressmann, T. Janhunen, R. Mercer, T. Schaub, S. Thiele, and R. Tichy. On probing and multi-threading in platypus. In Dix and Hunter [27], pages 30–38.

[59] gringo. `http://www.cs.uni-potsdam.de/gringo`.

[60] N. Guelzim, S. Bottani, P. Bourgine, and F. Képès. Topological and causal structure of the yeast transcriptional regulatory network. *Nature Genetics*, 31:60–63, 2002.

[61] R. M. Gutierrez-Rios, D. A. Rosenblueth, J. A. Loza, A. M. Huerta, J. D. Glasner, F. R. Blattner, and J. Collado-Vides. Regulatory network of Escherichia coli: consistency between literature knowledge and microarray profiles. *Genome Res*, 13(11):2435–2443, Nov 2003. Evaluation Studies.

[62] C. Guziolowski. *Analysis of Large Scale biological Networks with Constraint-Based Approaches over Static Models*. Dissertation, Université de Rennes 1, 2010. PhD Thesis.

[63] C. Guziolowski, A. Bourde, F. Moreews, and A. Siegel. BioQuali Cytoscape plugin: analysing the global consistency of regulatory networks. *BMC Genomics*, 10:244+, 2009.

[64] C. Guziolowski, P. Veber, M. Le Borgne, O. Radulescu, and A. Siegel. Checking consistency between expression data and large scale regulatory networks: A case study. *Journal of Biological Physics and Chemistry*, 7(2):37–43, 2007.

[65] T. Handorf, O. Ebenhöh, and R. Heinrich. Expanding metabolic networks: Scopes of compounds, robustness, and evolution. *Journal of Molecular Evolution*, 61(4):498–512, 2005.

[66] T. Handorf, O. Ebenhöh, and R. Heinrich. An environmental perspective on metabolism. *Journal of Theoretical Biology*, 252(3):498–512, 2008.

[67] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In *Proceedings of the Formal methods for the design of computer, communication, and software systems 8th international conference on Formal methods for computational systems biology*, SFM'08, pages 215–264, Berlin, Heidelberg, 2008. Springer-Verlag.

[68] M. Heiner and I. Koch. Petri net based model validation in systems biology. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*, pages 216–237. Springer Berlin / Heidelberg, 2004.

[69] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.

[70] R. C. G. Holland, T. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Drager, A. Yates, M. Heuer, and M. J. Schreiber. Biojava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):btn397–2097, August 2008.

[71] T. Ideker. Systems biology 101—what you need to know. *Nature Biotechnology*, 22(4):473–475, April 2004.

[72] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[73] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabàsi. The large-scale organization of metabolic networks. *Nature*, 407:651–654, Oct 2000.

[74] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome: a knowledgebase of biological pathways. *Nucleic Acids Res*, 33(Database issue), January 2005.

[75] M. Kanehisa and S. Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, January 2000.

[76] H. Kitano. Systems biology: A brief overview. *Science*, 295(5560):1662–1664, March 2002.

[77] S. Klamt and J. Stelling. Stoichiometric and constraint-based modelling. In *System Modeling in Cellular Biology: From Concepts to Nuts and Bolts*, pages 73–96. MIT Press, 2006.

[78] D. Kompala, D. Ramkrishna, N. Jansen, and G. Tsao. Investigation of bacterial-growth on mixed substrates — experimental evaluation of cybernetic models. *Biotechnology and Bioengineering*, 28(7):1044–1055, 1986.

[79] B. Kuipers. *Qualitative reasoning. Modeling and simulation with incomplete knowledge*. MIT Press, 1994.

[80] N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34(Database issue), January 2006.

[81] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[82] Y. Lierler. cmodels - SAT-based disjunctive answer set solver. In Baral et al. [7], pages 447–451.

[83] Y. Lierler and P. Schüller. Parsing combinatory categorial grammar with answer set programming: Preliminary report. *CoRR*, abs/1108.5567, 2011.

[84] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[85] I. Lynce and J. Marques-Silva. Sat in bioinformatics: Making the case with haplotype inference. In *In International Conference on Theory and Applications of Satisfiability Testing (SAT*, pages 136–141. Springer-Verlag, 2006.

[86] M. Mallory, K. Cooper, and R. Strich. Meiosis-specific destruction of the ume6p repressor by the cdc20-directed apc/c. *Molecular Cell*, 27(6):951–961, 2007.

[87] Z. Nikoloski, S. Grimbs, P. May, and J. Selbig. Metabolic networks are np-hard to reconstruct. *Journal of Theoretical Biology*, 254:807–816, 2008.

[88] Z. Nikoloski, S. Grimbs, J. Selbig, and O. Ebenhöh. Hardness and approximability of the inverse scope problem. In K. Crandall and J. Lagergren, editors, *Proceedings of the Eighth International Workshop on Algorithms in Bioinformatics (WABI'08)*, volume 5251 of *Lecture Notes in Computer Science*, pages 99–112. Springer-Verlag, 2008.

[89] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82)*, pages 255–260. ACM Press, 1982.

[90] O. Ray, K. Whelan, and R. King. A nonmonotonic logical approach for modelling and revising metabolic networks. In *Third International Conference on Complex, Intelligent and Software Intensive Systems (from Second International Workshop on Intelligent Informatics in Biology and Medicine)*. IEEE Computer Society, 2009.

[91] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[92] É. 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.

[93] A. Richard, J. Comet, and G. Bernot. R. Thomas' modeling of biological regulatory networks: Introduction of singular states in the qualitative dynamics. *Fundamenta Informaticae*, 65(4):373–392, 2004.

[94] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[95] M. Savageau. *Biochemical system analysis: a study of function and design in molecular biology*. Addison-Wesley Educational Publishers Inc, 1976.

[96] 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.

[97] C. Schilling, S. Schuster, B. Palsson, and R. Heinrich. Metabolic pathway analysis: Basic concepts and scientific applications in the post-genomic era. *Biotechnology progress*, 15:296–303, 1999.

[98] A. Schrijver. Theory of integer and linear programming. In *Stanford University*. John Wiley & sons, 1986.

[99] A. Siegel, O. Radulescu, M. Le Borgne, P. Veber, J.n Ouy, and S. Lagarrigue. Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks. *Biosystems*, 84(2):153–174, May 2006.

[100] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[101] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In G. Gupta, editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL'99)*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 1999.

[102] T. C. Son and E. Pontelli. Planning for biochemical pathways: A case study of answer set planning in large planning problem instances. Technical report, Department of Computer Science, University of Bath, 2007.

[103] C. Soulé. Graphic requirements for multistationarity. *Complexus*, 1(3):123–133, 2003.

[104] C. Soulé. Mathematical approaches to differentiation and gene regulation. *Comptes Rendus Biologies*, 329:13–20, 2006.

[105] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. Gilbert, I. Korf, H. Lapp, H. Lehväslaiho, C. Matsalla, C. J. Mungall, B. I. Osborne, M. R. Pocock, P. Schattner, M. Senger, L. D. Stein, E. Stupka, M. D. Wilkinson, and E. Birney. The bioperl toolkit: Perl modules for the life sciences. *Genome research*, 12(10):1611–1618, October 2002.

[106] P. Sudarsanam, V. Iyer, P. Brown, and F. Winston. Whole-genome expression analysis of snf/swi mutants of Saccharomyces cerevisiae. *Proceedings of the National Academy of Sciences of the United States of America*, 97(7):3364–3369, 2000.

[107] T. Syrjänen. Lparse 1.0 user's manual. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.

[108] N. Tran. *Reasoning and hypothesing about signaling networks*. PhD thesis, Arizona State University, December 2006.

[109] N. Tran and C. Baral. Reasoning about triggered actions in AnsProlog and its application to molecular interactions in cells. In D. Dubois, C. Welty, and M. Williams, editors, *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pages 554–564. AAAI Press, 2004.

[110] N. Tran, C. Baral, and C. Shankland. Issues in reasoning about interaction networks in cells: Necessity of event ordering knowledge. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, pages 676–681. AAAI Press, 2005.

[111] G. van Rossum. Python reference manual. Report CS-R9525, April 1995.

[112] P. Veber, C. Guziolowski, M. Le Borgne, O. Radulescu, and A. Siegel. Inferring the role of transcription factors in regulatory networks. *BMC Bioinformatics*, 9(228), 2008.

[113] P. Veber, M. Le Borgne, A. Siegel, S. Lagarrigue, and O. Radulescu. Complex qualitative models in biology: A new approach. *Complexus*, 2(3-4):140–151, 2004.

[114] J. Ward and J. Schlipf. Answer set programming with clause learning. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, pages 302–313. Springer-Verlag, 2004.

[115] B. Washburn and R. Esposito. Identification of the Sin3-binding site in Ume6 defines a two-step process for conversion of Ume6 from a transcriptional repressor to an activator in yeast. *Molecular and Cellular Biology*, 21(6):2057–2069, 2006.

[116] M. Wildermuth. Metabolic control analysis: biological applications and insights. *Genome Biology*, 1(6):1031.1–1031.5, 2000.