Predicting Changes of Reaction Networks with Partial Kinetic Information

Joachim Niehren[b,c], Mathias John[a,c],
Cristian Versari[a,c], François Coutte[a,d], Philippe Jacques[a,d]

[a] *Université de Lille, France*
[b] *Inria, Lille, France*
[c] *BioComputing team of CRIStAL lab (CNRS UMR 9189), Lille, France*
[d] *Research Institute Charles Viollette, EA-7394-ICV, Lille, France*

Abstract

We wish to predict changes of reaction networks with partial kinetic information that lead to target changes of its steady states. The changes may be either influxes increases or decreases, reaction knockouts, or multiple changes of these two kinds. Our prime applications are knockout prediction tasks for metabolic and regulation networks.

In a first step, we propose a formal modeling language for reaction networks with partial kinetic information. The modeling language has a graphical syntax reminiscent to Petri nets. Each reaction in a model comes with a partial description of its kinetics, that is based on a similarity relation on kinetic functions that we introduce. Such partial descriptions are able to model the regulation of existing metabolic networks, for which precise kinetic knowledge is usually not available.

In a second step, we develop prediction algorithms that can be applied to any reaction network modeled in our language. These algorithms perform qualitative reasoning based on abstract interpretation, by which the kinetic unknowns are abstracted away. Given a reaction network, abstract interpretation produces a finite domain constraint in a novel class. We show how to solve these finite domain constraints with an existing finite domain constraint solver, and how to interpret the solution sets as predictions of multiple reaction kockouts, that lead to a desired change of the steady states. We have implemented the prediction algorithm and integrated it into a prediction tool.

This journal article extends the two conference papers [1, 2] while adding a new prediction algorithm for multiple gene knockouts. An application to single gene knockout prediction for surfactin overproduction was presented in [3]. It illustrates the adequacy of the model-based predictions made by our algorithm in the wet lab.

Keywords: Reaction networks, model based prediction, abstract interpretation, constraint solving, metabolic engineering, genetic engineering.

Contents

1. Introduction

Mathematical methods for analysing reaction networks [4, 5, 6, 7] often require full kinetic information for all reactions, while in systems biology practice often only partial information is available. Therefore, we study the problem of how to model reaction networks with partial kinetic information, and how to reason qualitatively about such models with methods from computer science. In particular, we wish to predict changes of reaction networks with partial kinetic information that may or must lead to a target change of the steady state.

When full kinetic information is not available, the existing model-based reasoning methods tend to ignore the kinetic information all over. Most typically, this holds for flux balance analysis [8, 9] when applied to metabolic networks [10, 11]. The missing information is then compensated heuristically by the adoption of ad hoc optimisation criteria. Alternatively, pathway analysis approaches [9] rely on the structure of reactions networks, but the combinatorial nature of the problem makes difficult their application to densely interconnected networks. Both above methods have extensions that deal with partial kinetic information about inhibitors. This is done by adding boolean constraints that state the conditions when an inhibitor does block a reaction [12]. However, blocking inhibitors are not really appropriate in deterministic semantics, where the average over blocked and unblocked situations is to be considered. Therefore, it remains open how to deal with nonblocking inhibitors, which only slow down reaction average.

In the first part of this article, we propose a new modeling language for reaction networks with partial kinetic information, of which a short version was presented at the CMSB'2015 conference [2]. Our language is parameterised by a similarity relation on kinetic functions, so that the rate laws of chemical reactions need only to be specified up to similarity. For instance, two kinetic functions could be considered as similar if they have the same monotonic behaviour. Then, the kinetic function mapping x to $42x$ or to $5x/(7+x)$ would be similar, since whenever x increases then the values of both terms increase, and whenever x decreases then they both decrease.

Reaction networks are applied to a chemical solution, which are typically placed within a context. This may be another chemical solution that is subjet to an adjacent reaction network or to an experimental environment. The situation is illustrated in Figure 1. Since we do not want to model the context, we equip reaction networks with an interface to possible contexts only. The interface specifies which species can inflow from the context and which species may outflow into the context. It should be noticed that the inflows are controlled by the context, while the outflows are controlled by the network.

We also assume that some of the reactions of the network may be candidates for knockout. The choice of whether a reaction is knocked out or not remains external to the network. For the prediction task we interested in network changes combining possibly multiple reaction knockouts and influx changes.

The models of reaction networks in our language have a graphical syntax that is reminiscent of Petri nets, and also an equivalent XML syntax. They

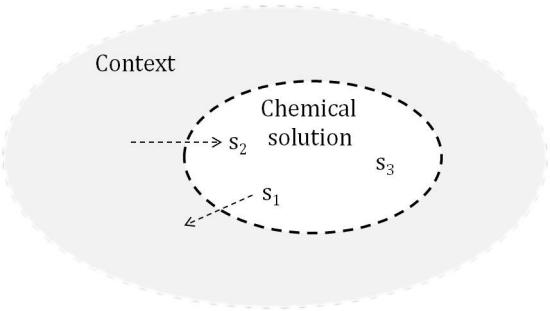


Figure 1: A reaction network describes a system of chemical reactions that acts on a chemical solution, the inflow of molecules from the context, and the outflow of molecules into the context.

are given a steady state semantics in terms of arithmetic equations, which defines the relation between influxes and outfluxes of the network in steady states, depending on which knockout candidates were knocked out. The steady state semantics subsumes the usual flux balance equations, but enhanced with equations on the rates of reactions based on the partial kinetic information. This information is expressed by variables for kinetic functions, that are subject to similarity constraints. It this way, the inhibitors of a reaction slow its rate down rather than shutting it down completely, while the activators of a reaction speed its rate up.

Our language can be used to model metabolic networks with complex regulation such as for *B. subtilis* in the Subtiwiki [13]. An example is the regulation network of the PIlv-Leu promoter of *B. subtilis*, which regulates the metabolism of the branched-chain amino acids Valine, Leucine, and Isoleucine. Previous models of these metabolic networks in the Subtiwiki were not given any formal semantics, so that they could not be used for directly for qualitative prediction algorithms. A detailed model of this network in the language presented here was published in [3] recently.

In the second part of this article, we develop a new prediction algorithm that can be applied to any reaction network modeled in our language. This algorithm does qualitative reasoning [14] based on abstract interpretation [15], by which the partial kinetic knowledge is discretised while the unknowns are abstracted away. A first version of the algorithm was presented at the VMCAI'2013 conference [1], but restricted to networks of reactions with mass action kinetics. The second version, presented the CMSB'2015 conference [2] and extended here, removes this limitation.

Given a reaction network, abstract interpretation can be applied to infer a difference constraint that relates network changes to changes in the steady states. This can be used to predict which network changes may or must lead to an expected change of the outfluxes. As already stated above, the network changes that we are interested in are reaction knockouts, influx changes or multiple combinations thereof. This becomes possible since the models of reaction

networks in our language do describe how network changes affect the steady state sematics.

Difference constraints are finite domain constraints from a novel class. The second step of our prediction algorithm consists in computing the set of all solutions of a difference constraint. Since difference constraints are finite domain constraints, their solution set is always finite. We build two constraint solvers for difference constraints based on finite domain constraint programming. The solver reported earlier in the conference paper preceding this article was developed from scratch in the programming language Scala [17]. Since then we developed a new solver by reduction to the minizinc solver [18] for finite domain constraints. While our Scala solver could enumerate only the n-best solutions where $n \leq 3000$ while consuming considerable time (more than 10 minutes), the minizinc solver can indeed return the complete set of all solutions sets in all our application, while enumerating more than 5000 solutions in less then a second.

Solutions of difference constraints can be interpreted as predictions of network changes that may or must lead to an overproduction target. When only seeing the n-best solutions, one can find solutions with few network changes that are compatible with the overproduction target. But since since we are now having access to the complete solutions sets of difference constraints obtained from reaction networks in practice, we can distinguish the solutions that are merely compatible with the overproduction target from those that safely entail it. As we will argue, safe solutions correspond to predictions that must satisfy the overproduction target, while compatible solutions yield predictions that may satisfy the overproduction target or not. It also turns out that multiple network changes may be necessary to obtain safe solutions, while single knockouts may not be enough.

We have implemented the prediction algorithm and integrated it into a prediction tool. We illustrate this tool and our prediction algorithm at the example of two simplified models of leucine production of *B. subtilis*, that focus on the regulation of the ilv-leu promoter. The target here is leucine overproduction. For the simpler of the two models, the prediction based on the graphical model can be done manually by humans. This illustrates that our algorithm formalizes a natural kind of qualitative reasoning.

In a follow up work [3], our algorithms were applied to single knockout prediction for leucine overproduction in a larger and more realistic model. The predictions obtained there were not safe, but still 6 out of 14 predictions could be verified successfully by gene knockouts in the wet lab.

Compared to the previous two conference papers [1, 2], we added extended the prediction algorithm to multiple change, rather than prediction of single reaction knockouts or single influx changes. We also introduce the notion of safe predictions, which requires to consider the set of all solutions of a difference constraints. While safe solutions could have been define in theory before, their practical relevance became apparent only with the new minizinc constraint solver for difference constraints, which permits to enumerate the set of *all* solutions of difference constraint inferred from reaction networks with up to 100 reactions very efficiently.

Part I
Modeling Language

We introduce reaction networks with complete kinetic information, in Section 2, the introduce a language for describing them modulo a similarity relation on kinetic functions on Section 3. In Section 4, we illustrate the language at two simplified models of leucine production, with present the intuition of how to use the steady state equations for change prediction. In Section 5, we define appropriate similarity relations on kinetics functions for the usage in our modeling language.

2. Reaction Networks

We define reaction networks with complete kinetic information, and show how to compute their steady state semantics. The notion of reaction networks and how to infer their ODEs is basically standard (see e.g. [4]), except for the addition of inflows, outflows, and knockout candidates.

We want to model biological systems that can interact with their context, i.e., their biological or experimental environment as illustrated in Figure 1. Therefore, we add inflows and outflows to our notion of reaction networks, that state how the context may add molecules into the system, and how molecules from the system can flow into its context. For instance, the chemical solution can model all molecules of a cell, and the context the cultivation medium of the cell for feeding or extracting molecules. Alternatively, the chemical solution can represent the subset of molecules affected by a selected pathway of a cell, and the context all molecules that are produced and consumed by connected pathways.

We also want to reason about a set of possible biological systems at the same time, which differ only in some reaction knockouts. Since knockouts do not make sense for arbitrary chemical reactions, each reaction network will distinguish a subset of reactions that are candidates for knockout. In the ODEs of reaction networks, we will then use a Boolean variable for each knockout candidate for selecting whether a candidate is knocked out or not.

We denote the set of non-negative real number by $\mathbb{R}_+$ and the subset of Booleans by $\mathbb{B} = \{0, 1\}$. A kinetic function of arity $k \geq 0$ is a function of type $\kappa : \mathbb{R}_+^k \to \mathbb{R}_+$. Kinetic functions will be used to define the rate laws of chemical reactions.

Let S a finite set of species. A chemical reaction r is a tuple of the form: $s_1, \ldots, s_k \xrightarrow{\kappa} s_{k+1}, \ldots, s_l$ where $0 \leq k \leq l$, $s_1, \ldots, s_l \in S$, and $\kappa : \mathbb{R}_+^k \to \mathbb{R}_+$ is a kinetic function. Any reaction has a tuple of reactants $s_1, \ldots, s_k$ and a tuple of products $s_{k+1}, \ldots, s_l$. In order to account for the stoichiometry of a reaction, we write $\mathsf{rct}_r(s)$ for number of occurrences of s in the tuple of reactants of r, and $\mathsf{prd}_r(s)$ for number of occurrences of s in the tuple of products of r. A modifier of a reaction is a species s with $\mathsf{rct}_r(s) = 1 = \mathsf{prd}_r(s)$. Whether a modifier

$$(\textsc{inflows})\ \frac{s \in S \setminus I}{x_s = 0} \qquad (\textsc{outflows})\ \frac{s \in S}{y_{\mathrm{s}} = O(s) \cdot z_{\mathrm{s}}}$$

$$(\textsc{cons})\ \frac{s \in S}{c_s = y_{\mathrm{s}} + \sum_{r \in R} \mathsf{rct}_r(s) \cdot v_r} \qquad (\textsc{prod})\ \frac{s \in S}{p_s = x_s + \sum_{r \in R} \mathsf{prd}_r(s) \cdot v_r}$$

$$(\textsc{rate})\ \frac{r \text{ is } s_1, \ldots, s_k \xrightarrow{\kappa} s_{k+1}, \ldots, s_l \qquad r \in R \setminus K}{v_r = \kappa(z_{\mathrm{s}_1}, \ldots, z_{\mathrm{s}_k})}$$

$$(\textsc{rate}_{\textsc{ko}})\ \frac{r \text{ is } s_1, \ldots, s_k \xrightarrow{\kappa} s_{k+1}, \ldots, s_l \qquad r \in K}{v_r = o_r \kappa(z_{\mathrm{s}_1}, \ldots, z_{\mathrm{s}_k})} \qquad (\textsc{ko}_{\textsc{var}})\ \frac{r \in K}{o_{\mathrm{r}} \in \{0, 1\}}$$

$$(\textsc{steady state})\ \frac{s \in S}{c_s = p_s}$$

Figure 2: Steady state equations of a reaction network $N = (S, R, I, O, K)$.

behaves as an activator or as an inhibitor depends on the choice of the rate law κ.

Definition 1. A reaction network is a tuple $N = (S, R, I, O, K)$ where S is a finite set, R is a finite set of chemical reactions with species in S, $K \subseteq R$ a subset of knockout candidates, $I \subseteq S$ a subset of inflow species, and $O : S \to \mathbb{R}_+$ a function for outflow species and their rates.

A reaction network defines the evolution of a chemical solution in a context. Each inflow species $s \in I$ specifies an inflow that adds s to the chemical solution, and is controlled by the context. An *outflow species* is an element of the following set:

$$S_O = \{s \in S \mid O(s) \neq 0\}$$

For any outflow species there is outflow into the context that consumes s from the chemical solution. The outflow kinetics for s follows the mass-action law with constant $O(s)$. Note that any species may have an inflow and an outflow at the same time.

Under the assumption of deterministic network behaviour, for any initial chemical solution a unique limit will be reached when time advances, which is called a steady state. Since we do not fix any initial chemical solution, many steady states may exist for the same reaction network. The rates of all inflows and and outflows are also assumed to be constant in any steady state, as well as the rates of all reactions and the concentrations of all species of the network.

The steady state semantics of a reaction network is given by a system of arithmetic equations. These equations are build over set of variables taking values in $\mathbb{R}_+$, that is totally ordered.

We assume pairwise distinct variables, three per species $s \in S$ and two per reaction.

- A variable z_{s} that denotes the concentration of s in a steady state.

- A variable x_{s} that denote the rate of the inflow of s if $s \in I$ and 0 otherwise. The rate of an inflow is also called an *influx*.
- A variable y_{s} that if $s \in S_O$ denotes the rate of the outflow of s and 0 otherwise. The rate of an outflow is also called its *outflux*.

Furthermore, for the reactions $r \in R$ we consider two further variables:

- A variable v_{r} for the rate of r in a steady state.
- And if $r \in K$ a Boolean variable o_{r} whose value will be 1 if r is knocked out and 0 otherwise.

The steady state equations of a network are inferred by the inference rules in Figure 2. They are mainly standard, except for the treatment of influxes, outfluxes, and knockout candidates. Each inference rule can be seen as an implication, whose condition is written above the line and whose conclusions is written below the line. Rule (INFLOW) states that the influx for any non-inflow species $s \notin I$ is zero. Rule (OUTFLOW) requires for any species $s \in S$ that its outflux is equal to $O(s) \cdot z_{\mathrm{s}}$ according to the mass-action law. The production rate p_s of a species s is defined by rule (PROD) and its consumption rate c_s by rules (CONS). Rules (RATE) and ($\mathrm{RATE}_{\mathrm{KO}}$) provide the rate of reaction r by applying its kinetic function to the concentrations of all its reactants: with respect to (RATE), the kinetic function given by rule ($\mathrm{RATE}_{\mathrm{KO}}$) is enriched with a variable o_r whose value in $\{0, 1\}$ allows the modeling of a potential gene knockout. The (STEADY STATE) states that consumption and production rates are balanced for all species.

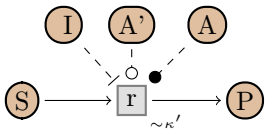
Definition 2. Any reaction network $N = (S, R, I, O, K)$ with $n = \#I$ inflow species, $m = \#S_O$ outflow species, and $o = \#K$ knockout candidates defines an exchange relation $E_N \subseteq \mathbb{R}_+^n \times \mathbb{R}_+^m \times \mathbb{B}^o$, which is obtained by projecting the solutions of the steady state equations for N to the following tuples of variables:

- the n-tuple of variables x_{s} for the inflow species $s \in I$,
- the m-tuple of variables $y_{\mathrm{s}'}$ for the outflow species $s' \in S_O$,
- the o-tuple of variables o_{r} for the knockout candidates $r \in K$.

The order in which the variables appear in these tuples is the total order on the variables that we assumed.

All those variables that are used by the exchange relation of a network are called *local*. The local variables are implicitly existentially quantified, since they are projected away from the solutions when used in the exchange relation.

Figure 3: An enriched reaction with a partially known rate law $\sim\kappa'$. It has substrate S, inhibitor I, accelerator A', activator A, and one product P beside of the modifiers I, A, and A'.



3. Modeling Language modulo Similarity

We now present a modeling language for reaction networks with partial kinetic information. As parameter of our language, we assume a similarity relation $\sim$ on kinetic functions. Rather than specifying rate laws of chemical reactions by kinetic functions, we will describe them only up to similarity: a rate law belongs to $\sim\kappa$ if it is similar to the kinetic function κ.

Enriched chemical reactions will be used to describe the chemical reactions of a reaction network. An example is given in Figure 3. The graph there represents an enriched chemical reaction r with substrate S, activator A, an accelerator A', and inhibitor I and a product P. Please note that the same species may play different roles even in the same reaction, and several times. Both, activators and accelerators speed up a reaction. Activators of reactions are like enzymes. The difference is that all activators of a reaction must be present for its application, while the accelerators need not to be there.

For graphical representation, we use conventions similarly to Petri nets. Species are represented by rounded nodes (S) containing the name s of the species, and enriched reactions are graphically represented by boxed nodes [r] containing the name r of the reaction. Reactions that belong to the set of knockout candidates K are coloured in light orange [r].

More generally, enriched chemical reactions have different kinds of reactants, that are fixed by a finite set of roles *Rol*, which is the second parameter of our language. In our example, there will be substrates – that are consumed – and three kinds of modifiers: inhibitors, activators, and accelerators, so we set *Rol* $= \{inh, subs, act, acc\}$. For our graphical syntax, we assign to each role an edge type, for edges pointing from the reactant to the reaction. We will use $\longrightarrow$ for *subs*, - - - -| for *inh*, - - -● for *act*, and - - -○ for *acc*. The products of a reaction – beside of the above modifiers – will be linked by arrows $\longrightarrow$ pointing from the reaction to the product.

Reactant roles serve to order the arguments of the rate law of a enriched chemical reaction. Such a rate law is given by an enriched kinetic function:

$$\kappa' : (Rol \times \mathbb{R}_+)^k \to \mathbb{R}_+$$

We assume that any enriched kinetic function is well-behaved, in that any permutation of arguments with the same role does not change its value. When fixing the order of the arguments, any enriched kinetic function κ' can be replaced by a standard kinetic function κ, for instance such that $\kappa(z_S, z_I, z_{A'}, z_A) = \kappa'(subs{:}\, z_S, inh{:}\, z_I, act{:}\, z_{A'}, acc{:}\, z_A)$. An enriched chemical reaction can then be replaced by a chemical reaction, in which the kinetic function is replaced by a

variable. With the same ordering as for obtaining κ from κ', we obtain for the example from Figure 3:

$$S, I, A', A \xrightarrow{\sim\kappa} P, I, A', A$$

Here, $\sim\kappa$ stands for a fresh variable for a standard kinetic function that is similar to κ. A model in our language is a tuple (S, R, I, O) where R is a set of enriched reactions and $I, O \subseteq S$. Note that we do not require to specify rate constants for outflows. Graphically, inflow species in I and outflow species in O are indicated respectively by ingoing and outgoing arrows $\dashrightarrow$. An example model in graphical syntax is given in Figure 4.

For any model in our language, we can generate a reaction network with variables for kinetic functions that are subject to similarity constraints. Therefore, we can define the steady state equations of any model in the language as before, except that kinetic functions will be represented by variables, as well as rate constants of outflows. An example is worked out in the next section.

Besides the graphical syntax, our language supports an XML syntax, which serves for writing the models, so that the graphs can be generated. We implemented tools for doing this in XSLT. These tools can also compute the steady state equations, and perform abstract interpretation.

4. Example: Regulation of Metabolism of *B. subtilis*

As an example, we model the leucine biosynthesis pathway of *B. subtilis* in our language. This is one of the complex regulation mechanisms of the metabolism of *B. subtilis*, for which informal models are given in the Subtiwiki [13]. The precise similarity relation of the model will be defined in Section 5.

4.1. Model Design

Two variations of the model are given in graphical syntax in Figure 4 and Figure 7. The first reaction network describes the base regulation of the *ilv-leu* promoter ($P_{Ilv-Leu}$), and has a single unregulated reaction for Leu production.[1] The second network refines the former, in that a regulation mechanism for Leu production is added, based on the *bkL-bcd* operon. Furthermore note that we keep the basic model simpler since we do not consider any knockout candidates there, in contrast to the refined model.

For clearer visualisation, species nodes have different colors depending on the type of the species: in this paper we will use proteins (P), metabolites (M), and promoters or binding sites (B). The variable z_B stands for the activity of

[1] The base network presented here is itself a refinement of the network from [2]. Here, CcpA does not have a direct acceleration effect on reaction r_2, but an indirect effect via r_9 which however is inhibited by BS_{CcpA}. The difference constraints obtained with or without this refinement are logically equivalent (up to existential quantification of the newly introduced variables).

the promoter or binding site B, while z_{P} and z_{M} stand for the concentrations of P and M.

We consider an acceleration function with $Acc(d) = 1 + d$ and an inhibition function with $Inh(d) = 1/Acc(d)$. We define the enriched kinetic functions *exp* such that for all tuples $t = (r_1 : d_1, \ldots, r_k : d_k) \in (Rol \times \mathbb{R}_+)^k$:

$$exp(t) = \prod_{r_i \in \{subs, act\}} d_i \;\cdot\; \prod_{r_j = acc} Acc(d_j) \;\cdot\; Inh(\sum_{r_l = inh} d_l)$$

Note that the order of arguments with the same role is not important, so that function *exp* is well-behaved. When a reaction has the *exp* kinetics, then its inhibitors slow down the reaction but do not block it. Accelerators and activators both speed up the reaction. Furthermore, if one of the activators is missing then the reaction is blocked. One might want to generalize *exp* with parameters defining the strength of respective accelerations and inhibitions. We do not do so, since these parameters are typically unknown, and since all such generalised expression kinetics will turn out to be similar. Generally, we are only interested in $\sim exp$, so similar definitions would do the job as well. The enriched mass-action kinetics is the special case $ma(t) = exp(t)$ for all $t \in (\{subs\} \times \mathbb{R}_+)^*$.

4.2. Basic Network

Leucine biosynthesis is realised by enzymes which are coded by the genes of the *ilv-leu* operon. This operon is under the regulation of the promoter $\mathrm{P_{Ilv-Leu}}$. For simplicity, we group the whole reaction network leading to the leucine biosynthesis into reaction $\mathrm{r_8}$.

The production of leucine depends of the activation of $\mathrm{P_{Ilv-Leu}}$, which is done by reaction $\mathrm{r_2}$ under the regulation of TnrA, CcpA, and CodY. Proteins TnrA and CodY are influx species added by the context and degraded by reactions $\mathrm{r_{16'}}$ and $\mathrm{r_{15'}}$ respectively. These two proteins play also a regulatory role in the degradation of the leucine by inhibiting the reaction $\mathrm{r_3}$. Protein CcpA is expressed by reaction $\mathrm{r_{14}}$ and degraded by reaction $\mathrm{r_{14'}}$.

Transcription at the *ilv-leu* promoter is well known to be inhibited by CodY through a binding of this latter on the promoter [16, 19, 20, 21, 22]. To model this action of CodY on the promoter $\mathrm{P_{Ilv-Leu}}$, we introduce the reaction $\mathrm{r_1}$ which activates the binding side $\mathrm{BS_{CodY}}$ of CodY at the promoter, which in turn slows down reaction $\mathrm{r_2}$ and thus reduces the promoter's activity. The binding of CodY to the promoter's binding site $\mathrm{BS_{CodY}}$ can be prohibited when CcpA is bound to the promoter. Therefore the presence of CcpA slows down reaction $\mathrm{r_1}$ [23, 21] but it does not block it on average in a steady state.

The promoter $\mathrm{P_{Ilv-Leu}}$ is also down-regulated by Leu in terms of a T-box [23, 24], which is captured by the negative control of the reaction $\mathrm{r_2}$ by Leu.

Protein TnrA forms a further inhibitor whose impact on the $\mathrm{P_{Ilv-Leu}}$ promoter is represented by the binding site $\mathrm{BS_{TnrA}}$ through the reaction $\mathrm{r_7}$, this latter is degraded by reactions $\mathrm{r_{7'}}$. Protein CcpA is also independently up-regulating the *ilv-leu* operon transcription by binding on it, and thus activating reaction $\mathrm{r_2}$.The binding of CcpA is captured by $\mathrm{BS_{CcpA}}$ through reaction $\mathrm{r_9}$. This latter is then degraded by $\mathrm{r_{9'}}$. Moreover, the acceleration of the reaction

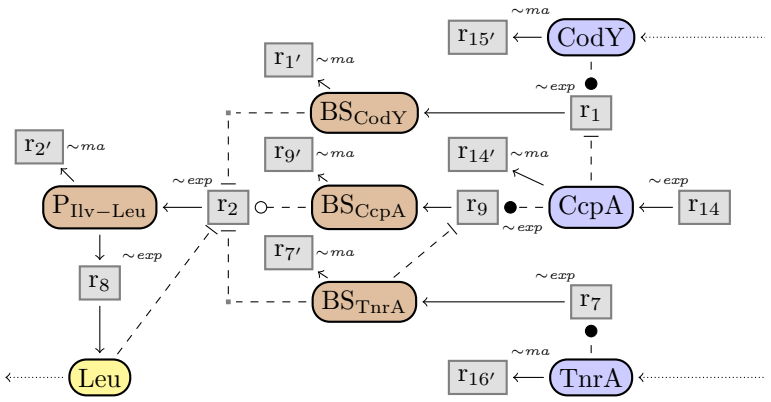


Figure 4: Basic reaction network of the regulation of promoter PIlv-Leu in *B. subtilis* without knockout candidates.

Flux balance equations:

(Leu) $v_{\mathrm{r_8}} = y_{\mathrm{Leu}}$
(CcpA) $v_{\mathrm{r_{14}}} = v_{\mathrm{r_{14'}}}$
(CodY) $x_{\mathrm{CodY}} = v_{\mathrm{r_{15'}}}$
(TnrA) $x_{\mathrm{TnrA}} = v_{\mathrm{r_{16'}}}$
(BS$_{\mathrm{CodY}}$) $v_{\mathrm{r_1}} = v_{\mathrm{r_{1'}}}$
(P$_{\mathrm{Ilv-Leu}}$) $v_{\mathrm{r_2}} = v_{\mathrm{r_{2'}}} + v_{\mathrm{r_8}}$
(BS$_{\mathrm{TnrA}}$) $v_{\mathrm{r_7}} = v_{\mathrm{r_{7'}}}$
(BS$_{\mathrm{CcpA}}$) $v_{\mathrm{r_9}} = v_{\mathrm{r_{9'}}}$

Outfluxes:

$y_{\mathrm{Leu}} = ma^{(0)}(subs\colon z_{\mathrm{Leu}})$

Reaction rates:

$v_{\mathrm{r_1}} = exp^{(1)}(inh\colon z_{\mathrm{CcpA}}, act\colon z_{\mathrm{CodY}})$
$v_{\mathrm{r_{1'}}} = ma^{(1)}(subs\colon z_{\mathrm{BS_{CodY}}})$
$v_{\mathrm{r_2}} = exp^{(2)}(inh\colon z_{\mathrm{BS_{CodY}}}, acc\colon z_{\mathrm{BS_{CcpA}}}, inh\colon z_{\mathrm{Leu}}, inh\colon z_{\mathrm{BS_{TnrA}}})$
$v_{\mathrm{r_{2'}}} = ma^{(2)}(subs\colon z_{\mathrm{P_{Ilv-Leu}}})$
$v_{\mathrm{r_7}} = exp^{(7)}(act\colon z_{\mathrm{TnrA}})$
$v_{\mathrm{r_{7'}}} = ma^{(7)}(subs\colon z_{\mathrm{BS_{TnrA}}})$
$v_{\mathrm{r_8}} = exp^{(8)}(subs\colon z_{\mathrm{P_{Ilv-Leu}}})$
$v_{\mathrm{r_9}} = exp^{(9)}(inh\colon z_{\mathrm{BS_{TnrA}}}, act\colon z_{\mathrm{CcpA}})$
$v_{\mathrm{r_{9'}}} = ma^{(9)}(subs\colon z_{\mathrm{BS_{CcpA}}})$
$v_{\mathrm{r_{14}}} = exp^{(14)}()$
$v_{\mathrm{r_{14'}}} = ma^{(14)}(subs\colon z_{\mathrm{CcpA}})$
$v_{\mathrm{r_{15'}}} = ma^{(15)}(subs\colon z_{\mathrm{CodY}})$
$v_{\mathrm{r_{16'}}} = ma^{(16)}(subs\colon z_{\mathrm{TnrA}})$

Figure 5: Steady state equations for the basic reaction network in Figure 4.

r$_2$ by BS$_{\mathrm{CcpA}}$ is inhibited when BS$_{\mathrm{TnrA}}$ is active. Indeed, the active binding sites BS$_{\mathrm{TnrA}}$ and BS$_{\mathrm{CcpA}}$ can bind to each other while forming a DNA loop. This phenomenon is captured by the inhibition of the reaction r$_9$ by BS$_{\mathrm{TnrA}}$.

From the model, the steady state equations in Figure 5 were inferred. These contain variables $exp^{(i)}$ for enriched kinetic functions similar to exp, and variables $ma^{(i)}$ for enriched kinetic functions similar to the mass-action law ma for any i. Note that $v_{\mathrm{r_{14}}} = exp^{(14)}()$, which means that the kinetic function of r$_{14}$ is applied without any arguments. This reflects that r$_{14}$ has no reactants or modifiers, so that its rate is constant.

Therefore the equations in Figure 5 can be simplified by eliminating local variables and replacing them by expressions with equal values. In particular, we can simplify the steady state equations in Figure 5 to the equations in Figure 6.

$$
\begin{aligned}
v_{\mathrm{r}_2} &= v_{\mathrm{r}_{2'}} + y_{\mathrm{Leu}} \\
y_{\mathrm{Leu}} &= ma^{(0)}(subs\colon z_{\mathrm{Leu}}) \\
v_{\mathrm{r}_1} &= exp^{(1)}(inh\colon z_{\mathrm{CcpA}}, act\colon z_{\mathrm{CodY}}) \\
v_{\mathrm{r}_1} &= ma^{(1)}(z_{\mathrm{BS_{CodY}}}) \\
v_{\mathrm{r}_2} &= exp^{(2)}(inh\colon z_{\mathrm{BS_{CodY}}}, acc\colon z_{\mathrm{BS_{CcpA}}}, \\
&\qquad\qquad inh\colon z_{\mathrm{Leu}}, inh\colon z_{\mathrm{BS_{TnrA}}}) \\
v_{\mathrm{r}_{2'}} &= ma^{(2)}(subs\colon z_{\mathrm{P_{Ilv-Leu}}}) \\
v_{\mathrm{r}_7} &= exp^{(7)}(act\colon z_{\mathrm{TnrA}})
\end{aligned}
\qquad
\begin{aligned}
v_{\mathrm{r}_7} &= ma^{(7)}(subs\colon z_{\mathrm{BS_{TnrA}}}) \\
y_{\mathrm{Leu}} &= ma^{(8)}(subs\colon z_{\mathrm{P_{Ilv-Leu}}}) \\
v_{\mathrm{r}_9} &= exp^{(9)}(inh\colon z_{\mathrm{BS_{TnrA}}}, act\colon z_{\mathrm{CcpA}}) \\
v_{\mathrm{r}_9} &= ma^{(9)}(subs\colon z_{\mathrm{BS_{CcpA}}}) \\
v_{\mathrm{r}_{14}} &= exp^{(14)}() \\
v_{\mathrm{r}_{14}} &= ma^{(14)}(subs\colon z_{\mathrm{CcpA}}) \\
x_{\mathrm{CodY}} &= ma^{(15)}(subs\colon z_{\mathrm{CodY}}) \\
x_{\mathrm{TnrA}} &= ma^{(16)}(subs\colon z_{\mathrm{TnrA}})
\end{aligned}
$$

Figure 6: Simplified steady state equations for the basic PIlv-Leu network.

4.3. Prediction of Input Changes

In order to illustrate the qualitative reasoning methods that we will develop, we consider the overproduction problem of Leu for the $\mathrm{P_{Ilv-Leu}}$ network. The question is which changes of the influxes may lead to an increase of the Leu outflux?

Informally, the problem can be solved as follows. Leu is produced only from $\mathrm{P_{Ilv-Leu}}$ via reaction $\mathrm{r_8}$. The speed of this reaction can be increased increasing the concentration of $\mathrm{P_{Ilv-Leu}}$. Since $\mathrm{P_{Ilv-Leu}}$ is solely produced by reaction $\mathrm{r_2}$, this requires to increase the speed of $\mathrm{r_2}$. This can be done by either decreasing one of its three inhibitors Leu, $\mathrm{BS_{CodY}}$, or $\mathrm{BS_{TnrA}}$, or by increasing its accelerator $\mathrm{BS_{CcpA}}$. Increasing $\mathrm{BS_{CcpA}}$ can either be reduced to decreasing it inhibitor $\mathrm{BS_{TnrA}}$, a choice that we were considering independently already, or by increasing its activator CcpA. The latter is not possible, since CcpA is not connected to any inflow, so it cannot be increased by changing the influxes. And inhibitor Leu cannot be decreased, when we want to increase its outflux. Hence, either $\mathrm{BS_{CodY}}$ or $\mathrm{BS_{TnrA}}$ must be decreased. This is possible only as follows:

1a. decrease the influx of CodY, or

1b. decrease the influx of TnrA.

Either of these single changes could be enough, but a double change is also compatible with the target of leucine overproduction.

4.4. Refined Network

One of the aspects missing in the basic network is that leucine is also degraded for its use in fatty acid biosynthesis. This is regulated by several enzyme captured by the *bkL-bcd* operon. This aspect is modeled in the refined network in Figure 7, where we added the species $\mathrm{OP_{BkL-Bcd}}$ to the basic network.

We need to keep the refined network as simple as possible, in order to be able to illustrate the relevance of multiple change prediction (see [3] for a detailed network). Therefore, we consider operon $\mathrm{OP_{BkL-Bcd}}$ as an inhibitor of the

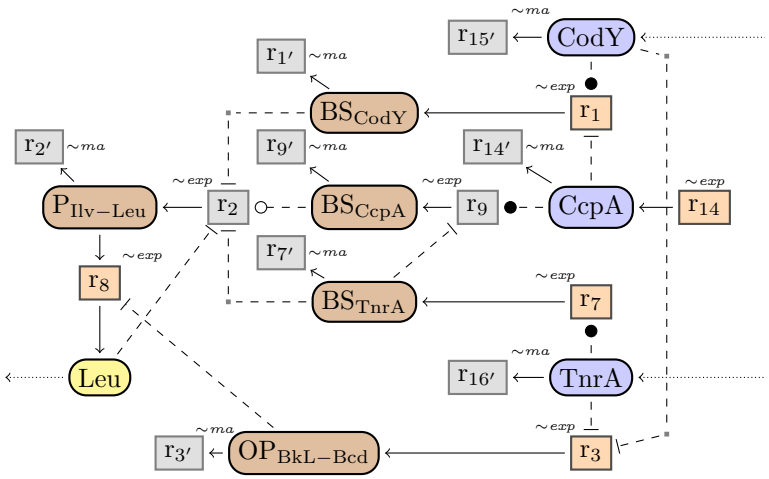


Figure 7: Refined PIlv-Leu network with regulation of Leu production by the bkL-bcd operon. As knockout candidates, we added the five reactions in light orange.

production reaction of Leu r_8. The activity of operon $OP_{BkL-Bcd}$ is increased by the new reaction r_3, which is inhibited by TnrA and CodY. Thereby, the presence of both CodY and TnrA now have a negative effect on the production of Leu, while in the basic network there were only positive effects. The second new reaction $r_{3'}$ degrades the activity of the operon so that it can become steady.

4.5. Prediction of Multiple Knockouts and Input Changes

It turns out that the predictions change with the refined network in Figure 7 by the addition of $OP_{BkL-Bcd}$. There are 2 more single influx changes without knockouts that may increase the Leu outflux. These may seem contradictory at the first view: Indeed, an increase or an decrease of CodY or TnrA may lead to leucine overproduction. For the basic network, only decreases of CodY or TnrA were compatible with this target. The difference with respect to the refined network is that these single inflow changes are no longer safe, in that the opposite of the target is also compatible with them.

When looking into multiple knockouts and input changes, the contradictory situation for single changes can be resolved. Indeed, it turns out that Leu production will be safely increased by either of the following four double changes:

2a. decrease the influx of CodY and knockout reaction r_3, or

2b. increase the influx of CodY and knockout reaction r_1, or

2c. decrease the influx of TnrA and knockout reaction r_3, or

2d. increase the influx of TnrA and knockout reaction r_1.

Another way to safely increase the Leu production is to perform a triple change:

$$\begin{array}{lccl} \text{increase} & < & = & \{(x,y) \in \mathbb{R}_+^2 \mid x < y\} \\ \text{decrease} & > & = & \{(x,y) \in \mathbb{R}_+^2 \mid x > y\} \\ \text{no change} & \doteq & = & \{(x,x) \mid x \in \mathbb{R}_+\} \end{array}$$

Figure 8: The difference relations of partition $\Delta_3 = \{<, >, \doteq\}$.

$$\begin{array}{lccl} \text{increase but not from zero} & \uparrow & = & \{(x,y) \in \mathbb{R}_+^2 \mid 0 < x < y\} \\ \text{increase from zero} & \Uparrow & = & \{(0,y) \in \mathbb{R}_+^2 \mid 0 < y\} \\ \text{decrease but not to zero} & \downarrow & = & \{(x,y) \in \mathbb{R}_+^2 \mid x > y > 0\} \\ \text{decrease to zero} & \Downarrow & = & \{(x,0) \in \mathbb{R}_+^2 \mid x > 0\} \\ \text{no change but not at zero} & \sim & = & \{(x,x) \mid 0 \neq x \in \mathbb{R}_+\} \\ \text{no change at zero} & \approx & = & \{(0,0)\}. \end{array}$$

Figure 9: The difference relations of partition $\Delta_6 = \{\uparrow, \Uparrow, \downarrow, \Downarrow, \sim, \approx\}$.

3. knockout reactions r_1, r_3, and r_7 at the same time.

The above results will be inferred by our prediction methods for multiple changes in Section 8. There we will also formalize what it means for a change to be safe.

5. Similarity by Difference Abstraction

We now define similarity relations on kinetic functions by abstracting from changes between real numbers.

We are interested in changes of the network raised for example by modification of influxes or outfluxes. A change of a concentration or a flow rate from one steady state to another is given by a pair of positive real numbers. We now want to abstract the space of all changes in $\mathbb{R}_+ \times \mathbb{R}_+$ into a finite set of *difference relations*. For this, we partition the set $\mathbb{R}_+ \times \mathbb{R}_+$ into a finite collection of subsets $\Delta \subseteq 2^{\mathbb{R}_+ \times \mathbb{R}_+}$, so that we can abstract any change in $\mathbb{R}_+ \times \mathbb{R}_+$ into a *difference relation* of Δ.

In the examples that follow, we will use two different partitions. When interested in inflow changes, it will often be sufficient to work with the ternary partition $\Delta = \Delta_3$ where $\Delta_3 = \{<, >, \doteq\}$. Its difference relations are defined and explained in Figure 8. We note that an influx increase or a reaction speed-up can be represented by $<$, and an influx decrease or a reaction slow-down corresponds to $>$.

When studying reaction knockouts, it is often better to use the senary partition $\Delta = \Delta_6$, where $\Delta_6 = \{\uparrow, \Uparrow, \downarrow, \Downarrow, \sim, \approx\}$ as defined in Figure 9, because its difference relations can distinguish a reaction slowdown $\downarrow$ from a reaction knockout $\Downarrow$. Note that Δ_6 refines the partition Δ_3 in that:

$$\begin{array}{lcll} < & = & \uparrow \uplus \Uparrow & , \\ > & = & \downarrow \uplus \Downarrow & , \\ \doteq & = & \sim \uplus \approx & . \end{array}$$

In general, we assume a relation $R \subseteq \mathbb{R}_+^p$, that may be either a kinetic function κ of arity $p-1$ or the relation R_N of a reaction network with p in- and outflows. We define the set of Δ-differences of the p-ary relation R as follows:

$$R^\Delta \quad = \quad \{(\delta_1, \ldots, \delta_p) \in \Delta^p \;\mid\; \forall i.\ (d_i, d_i') \in \delta_i, \\ (d_1, ..., d_p) \in R,\ (d_1', ..., d_p') \in R\}$$

So for instance, consider the exchange relation E_N for some reaction network N. Its difference abstraction ${E_N}^\Delta$ then expresses how the tuples in E_N may change when moving from one steady state of N to another.

Definition 3. Two kinetic functions $\kappa_1, \kappa_2 : (\mathbb{R}_+)^{p-1} \to \mathbb{R}_+$ are similar, written $\kappa_1 \sim_\Delta \kappa_2$, iff $\kappa_1^\Delta = \kappa_2^\Delta$.

Example 1. *Let $ma_k(subs{:}d, subs{:}d') = k \cdot d \cdot d'$ be the mass action law with constant k. We will freely identify binary functions such as ma_k with ternary relations. For any fixed Δ, the differences abstraction ${ma_k}^\Delta$ is then equal for all choices of parameter $k > 0$, For Δ_3, we identify the ternary relation ${ma_k}^{\Delta_3}$ with the binary set value function in the table on the right. The table for ${ma_k}^{\Delta_6}$ can be computed analogously.*

δ	δ'	${ma_k}^{\Delta_3}(\delta, \delta')$
$<$	$<$	$\{<\}$
$<$	$>$	$\{<, \doteq, >\}$
$<$	$\doteq$	$\{<\}$
$>$	$<$	$\{<, \doteq, >\}$
$>$	$>$	$\{>\}$
$>$	$\doteq$	$\{>\}$

Example 2. *We consider an enhanced Michaelis-Menten law with an additional activator: $mm_{k_1,k_2}(\text{subs}{:}d, \text{act}{:}d') = \frac{k_1 \cdot d \cdot d'}{k_2 + d'}$. Again, it can be shown that the abstractions $(mm_{k_1,k_2})^\Delta$ is independent of the choice of $k_1, k_2 \in \mathbb{R}_+$ for all Δ.*

The abstraction with respect to Δ_3 and Δ_6 cannot distinguish the mass-action law from the enhanced Michaelis-Menten, in that for all parameters $k, k_1, k_2 > 0$:

$$ma_k \sim_{\Delta_3} mm_{k_1,k_2} \qquad \text{and} \qquad ma_k \sim_{\Delta_6} mm_{k_1,k_2}$$

For other abstractions, the two families of kinetic functions can be distinguished though.

It should be noticed that R^Δ is always a finite relation, since Δ is chosen to be finite. The relation R in contrast, may contain infinitely many tuples. As a consequence, infinitely much information may be abstracted away, in particular the details about the parameters of kinetic functions. This is why the relations ${ma_k}^\Delta$ and $(mm_{k_1,k_2})^\Delta$ in the above examples could be computed independently of the parameters. The information that is preserved, however, is still able to distinguish inhibitors and activators.

Part II
Prediction Methods

We show in Section 6 how to derive difference constraints from steady state equations based on abstraction interpretation. In Section 4 we explain the main ideas of how difference constraints can be used for qualitative reasoning about reaction networks, as needed for our prediction algorithms. In section 8 we show how to infer from the solution set of a difference constraint the change predictions that it implies, and whether these predictions only compatible or even safe. Our tools for modeling reaction networks with partial kinetic information and prediction of network changes is presented in Section 9.

6. Abstract Interpretation to Difference Constraints

We next show how to interpret steady state equations abstractly as difference constraints, which will then be used for qualitative reasoning about reaction networks in our language in the next section.

The idea is to lift the difference abstraction $.^{\Delta}$ from relations over $\mathbb{R}_+$ to relations over Δ to the level of constraints defining such relations. For instance, the arithmetic equation $x_A = ma_k(z_{\mathrm{A}}, z_{\mathrm{B}})$ can be abstracted to a difference constraint that defines the relation ${ma_k}^{\Delta}$. We write this difference constraint as $x_A \in ma_k(z_{\mathrm{A}}, z_{\mathrm{B}})$, since now the variables are interpreted by values of Δ and ma_k is interpreted as the set valued function ${ma_k}^{\Delta}$. It should be noticed that the relation ${ma_k}^{\Delta}$ is finite and independent of the unknown parameter k, i.e., the unknown parameter has been abstracted away successfully.

6.1. Arithmetic Constraints

Arithmetic constraints were used to define the steady state semantics of reaction networks. More formally, an arithmetic constraint is a conjunctive logic formula with existential quantifiers with the following abstract syntax, where

$$\begin{array}{rcl} \phi & ::= & x{=}\kappa^{(i)}(x_1, \ldots, x_k) \mid x = x_1 + x_2 \mid x = x_1 \cdot x_2 \\ & \mid & x_1 = x_2 \mid x \in S \mid \phi \wedge \phi' \mid \exists x.\phi \end{array}$$

where $i \in \mathbb{N}$, $\kappa : \mathbb{R}_+^k \to \mathbb{R}_+$, x, x_1, x_2 are are variables, and $S \subseteq \mathbb{R}_+$ a finite subset. The expression $\kappa^{(i)}$ is a variable for a kinetic function that is similar to κ, i.e., an implicitly existentially quantified variable that is subject to the similarity constraint $\kappa^{(i)} \sim \kappa$.

A solution of an arithmetic constraint ϕ with n variables can be identified with a tuple in $\mathbb{R}_+^n$ since we assumed a total order on the variables. Therefore, the solution set $sol(\phi)$ of a formula ϕ satisfies $sol(\phi) \subseteq \mathbb{R}_+^n$.

Lemma 1. *The steady state equations of any reaction network N can be rewritten in linear time to an equivalent arithmetic constraint ϕ_N so that:*

$$sol(\phi_N) = E_N$$

PROOF. It is sufficient to flatten the terms in steady state equations, by introducing new existentially bound variables for all nested subterms. The resulting constraint ϕ_N is clearly equivalent, and thus it satisfies $sol(\phi_N) = E_N$ by Definition 2 of the exchange relation.

6.2. Difference Constraints

A difference constraint is a conjunctive logic formula with existential quantifiers with the following abstract syntax, where x is a variable, $\delta \in \Delta$, and $D \subseteq \Delta$:

difference relation	t	::=	$x \mid \delta$
set of difference relations	s	::=	$t \mid D \mid s + s' \mid s \cdot s' \mid \kappa(s_1, \ldots, s_k)$
difference constraints	ψ	::=	$t \in s \mid t{=}t' \mid \psi \wedge \psi' \mid \exists x.\psi$

In contrast to before, all arithmetic functions do now return sets of values in difference constraints. Therefore, we consider terms for difference relations t from terms for sets of difference relations s. When the term t appears in positions where an s is expected, then the difference relation denoted by t is implicitly converted into a singleton set.

Difference constraints are interpreted over Δ, so that variables x are assigned to elements of Δ (rather than elements of $\mathbb{R}_+$). Arithmetic functions such as $+$ are interpreted as set-valued functions on Δ such as $+^\Delta$, and similarly a kinetic function κ is interpreted as the set valued function κ^Δ. Since variables are totally ordered, a solution of a difference constraint can be identified with a tuple in Δ^n, so that the solution set $sol(\psi)$ of any difference constraint ψ satisfied $sol(\psi) \subseteq \Delta^n$.

6.3. Abstract Interpretation

We can now abstract from arithmetic constraints by interpreting them as difference constraints:

$$[\![x{=}\kappa^{(i)}(x_1, \ldots, x_k)]\!] = x \in \kappa(x_1, \ldots, x_k) \qquad [\![x = x_1 + x_2]\!] = x \in x_1 + x_2$$
$$[\![x \in S]\!] = x \in \{\delta \in \Delta \mid \exists s, s' \in S.\ (s, s') \in \delta\} \qquad [\![x = x_1 \cdot x_2]\!] = x \in x_1 \cdot x_2$$
$$[\![\phi \wedge \phi']\!] = [\![\phi]\!] \wedge [\![\phi']\!] \qquad [\![x_1{=}x_2]\!] = (x_1{=}x_2)$$
$$[\![\exists x.\phi]\!] = \exists x.[\![\phi]\!]$$

An important point here is that the variables $\kappa^{(i)}$ for the partially known kinetic functions are replaced by kinetic functions κ for which we have a definition, as for *exp* and *ma*. An application of these can thus be replaced by arithmetic expressions, in which the only remaining kinetic functions will be *Inh* and *Acc*. In this way, the simplified steady state equations in Figure 6 for the basic $P_{Ilv-Leu}$ network are abstracted to the difference constraints in Figure 10.

For instance, the abstraction of $x = ma^{(i)}(subs{:}x_1)$ then becomes the equation $x = x_1$, and the abstraction of $x = ma^{(i)}(subs{:}x_1, acc{:}x_2)$ the membership constraint $x \in x_1 \cdot Acc(x_2)$. For $x = exp^{(i)}(subs{:}x_1, inh{:}x_2, inh{:}x_3)$ the abstraction yields $x \in x_1 \cdot Inh(x_2 + x_3)$. The equation $x = exp^{(i)}()$ is abstracted to the

$$
\begin{array}{ll}
v_{\mathrm{r}_2} \in v_{\mathrm{r}_{2'}} + y_{\mathrm{Leu}} & v_{\mathrm{r}_7} = z_{\mathrm{BS}_{\mathrm{TnrA}}} \\
y_{\mathrm{Leu}} = z_{\mathrm{Leu}} & y_{\mathrm{Leu}} = z_{\mathrm{P}_{\mathrm{Ilv-Leu}}} \\
v_{\mathrm{r}_1} \in z_{\mathrm{CodY}} \cdot \mathit{Inh}(z_{\mathrm{CcpA}}) & v_{\mathrm{r}_9} \in z_{\mathrm{CcpA}} \cdot \mathit{Inh}(z_{\mathrm{BS}_{\mathrm{TnrA}}}) \\
v_{\mathrm{r}_1} = z_{\mathrm{BS}_{\mathrm{CodY}}} & v_{\mathrm{r}_9} = z_{\mathrm{BS}_{\mathrm{CcpA}}} \\
v_{\mathrm{r}_2} \in z_{\mathrm{BS}_{\mathrm{CcpA}}} \cdot \mathit{Inh}(z_{\mathrm{BS}_{\mathrm{CodY}}} + z_{\mathrm{Leu}} & v_{\mathrm{r}_{14}} = \delta_{=1} \\
\qquad + z_{\mathrm{BS}_{\mathrm{TnrA}}}) & v_{\mathrm{r}_{14}} = z_{\mathrm{CcpA}} \\
v_{\mathrm{r}_{2'}} = z_{\mathrm{P}_{\mathrm{Ilv-Leu}}} & x_{\mathrm{CodY}} = z_{\mathrm{CodY}} \\
v_{\mathrm{r}_7} = z_{\mathrm{TnrA}} & x_{\mathrm{TnrA}} = z_{\mathrm{TnrA}}
\end{array}
$$

Figure 10: Difference constraints for the basic PIlv-Leu network (without $\mathrm{OP}_{BkL-Bcd}$), inferred from the simplified steady state equations in Figure 6.

equation $x = \delta_{=1}$, where $\delta_{=1}$ is the unique element in Δ that contains $(1,1)$. For Δ_3, this is the difference relation $\doteq$ and for Δ_6 it is $\sim$.

It should also be noticed that $x \in \{0,1\}$ is abstracted over Δ_6 to $x \in \{\Uparrow, \Downarrow, \sim, \approx\}$, while over Δ_3 it is abstracted to $x \in \Delta_3$, which is always true.

Theorem 1 (Soundness of Abstract Interpretation). $sol(\phi)^{\Delta} \subseteq sol([\![\phi]\!])$.

Proof. We first note for any two relations $R_1, R_2 \subseteq \mathbb{R}_+^n$ that:

$$\text{(intersect)} \qquad (R_1 \cap R_2)^{\Delta} \subseteq {R_1}^{\Delta} \cap {R_2}^{\Delta}$$

This can be verified straightforwardly from the definition of the difference abstraction. It will turn out to be the reason why proper approximations may appear when abstracting conjunctions.

Then we proceed by induction on the structure of arithmetic constraints ϕ. For the variables in the proof, we always assume for simplicity that they are ordered $x_1 x_2 \ldots x_k x$.

Case ϕ is $x = \kappa^{(i)}(x_1, \ldots, x_k)$. By definition of solutions we have that $sol(\phi) = \{(d_1, \ldots, d_k, d) \in \mathbb{R}_+^{k+1} \mid d = \kappa^{(i)}(d_1, \ldots, d_k)\} = \kappa^{(i)}$, and thus $sol(\phi)^{\Delta} = {\kappa^{(i)}}^{\Delta} = \kappa^{\Delta}$. Furthermore, $[\![\phi]\!] = x{\in}\kappa(x_1, \ldots, x_k)$ so that $sol([\![\phi]\!]) = \kappa^{\Delta} = sol(\phi)^{\Delta}$.

Case ϕ is $x = x_1 + x_2$. This follows, since the interpretation of $+$ in Δ is equal to the difference abstraction of $sol(\phi)$, so that: $sol(\phi)^{\Delta} = +^{\Delta} = sol(x \in +(x_1, x_2)) = sol([\![\phi]\!])$.

Case ϕ is $x_1 = x_2$. We first note that $\{(d,d) \mid d \in \mathbb{R}_+\}^{\Delta} = \{(\delta, \delta) \mid \delta \in \Delta\}$ since Δ is a partition of $\mathbb{R}_+ \times \mathbb{R}_+$. We now can conclude as follows: $sol(\phi)^{\Delta} = \{(d,d) \mid d \in \mathbb{R}_+\}^{\Delta} = \{(\delta, \delta) \mid \delta \in \Delta\} = sol([\![\phi]\!])$.

Case ϕ is $\phi_1 \wedge \phi_2$. Hence $sol(\phi) = sol(\phi_1) \cap sol(\phi_2)$ and thus as claimed by (intersect) at the beginning $sol(\phi)^{\Delta} \subseteq sol(\phi_1)^{\Delta} \cap sol(\phi_2)^{\Delta}$. From the induction hypothesis applied to ϕ_1 and ϕ_2, it follows that $sol(\phi_i)^{\Delta} \subseteq$

$sol([\![\phi_i]\!])$ for $i = 1, 2$, and thus $sol(\phi)^\Delta \subseteq sol(\phi_1)^\Delta \cap sol(\phi_2)^\Delta \subseteq sol([\![\phi_1]\!]) \cap sol([\![\phi_2]\!]) = sol([\![\phi]\!])$.

Case ϕ is $\exists x.\phi'$. There case where x does not occur freely in ϕ' is easy, since we can drop the quantifier $\exists x$. Otherwise, we can assume that ϕ' has the free variables $x_1, ..., x_k$ and that $x = x_i$ where $1 \leq i \leq k$. For any set D and tuple set $S \subseteq D^k$ we define the projection $\Pi_x(S) = \{d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_k) \mid (d_1, \ldots, d_k) \in S\}$. We first note that:

$$\text{(project)} \qquad \Pi_x(sol(\phi'))^\Delta = (\Pi_x(sol(\phi')))^\Delta$$

Here we all element of $\mathbb{R}_+ \times \mathbb{R}_+$ belong to some difference in Δ. Based on the equation on projection, we can conclude as follows:

$$\begin{aligned} sol(\exists x.\phi')^\Delta &= (\Pi_x(sol(\phi')))^\Delta =_{project} \Pi_x(sol(\phi'))^\Delta \\ &\subseteq_{ind.hyp.} \Pi_x sol([\![\phi']\!]) = sol(\exists x.[\![\phi']\!]) = sol([\![\exists x.\phi']\!]). \end{aligned}$$

This theorem shows for any reaction network N with steady state equations equivalent to ϕ_N that the set of solutions of the abstract interpretation $[\![\phi_N]\!]$ is a correct over-approximation of the abstraction ${E_N}^\Delta$ of the exchange relation of N.

Corollary 1. *If $E_N = sol(\phi_N)$ then $(E_N)^\Delta \subseteq sol([\![\phi_N]\!])$.*

Proof. This follows immediately from Theorem 1.

7. Qualitative Reasoning with Difference Constraints

We next illustrate how to reason qualitatively about reaction networks by investigating solution sets of difference constraints. Since difference constraints have finite domains, we can compute all solutions of difference constraints by using finite domain constraint programming. In simple cases, we can use constraint simplification for this purpose, similarly to what is done by the propagation mechanisms for finite domain constraint solvers.

7.1. Constraint Solving

We illustrate how to use constraint solving to answer the question from Section 4.3: Which changes of the influxes of the basic $P_{Ilv-Leu}$ network may raise an increase the outflux of Leu?

In order to find the answer, we interpret the difference constraint inferred for the network in Figure 10 over Δ_3, which is good enough for this simple case.

We the study the set of all solutions of this difference constraint, which is given in Figure 11. It was enumerated by finite domain constraint solver. There are 13 solutions after projection to the relevant variables x_{CodY}, x_{TnrA}, and y_{Leu}.

Only 5 solutions do also satisfy the target $y_{Leu} = <$. From those, only the first three are safe with respect to the target, in that no other solution with the same values of x_{CodY} and x_{TnrA} violates the target. This shows that the target will be safely satisfied if we either decrease the influx of CodY, the influx of TnrA, or both of them.

	x_{CodY}	x_{TnrA}	y_{Leu}	target satisfied?	quality
1.	$>$	$\doteq$	$<$	yes	safe
2.	$\doteq$	$>$	$<$	yes	safe
3.	$>$	$>$	$<$	yes	safe
4.	$>$	$<$	$<$	yes	unsafe by 7. or 11.
5.	$<$	$>$	$<$	yes	unsafe by 8. or 12.
6.	$\doteq$	$\doteq$	$\doteq$	no	
7.	$>$	$<$	$\doteq$	no	
8.	$<$	$>$	$\doteq$	no	
9.	$\doteq$	$<$	$>$	no	
10.	$<$	$\doteq$	$>$	no	
11.	$>$	$<$	$>$	no	
12.	$<$	$>$	$>$	no	
13.	$<$	$<$	$>$	no	

Figure 11: The set of all solutions of the difference constraint in Figure 10, derived from the basic PIlv-Leu network (without $\text{OP}_{BkL-Bcd}$), and their interpretation with respect to the target $y_{\text{Leu}} = <$.

7.2. Constraint Simplification

Since the $\text{P}_{\text{Ilv-Leu}}$ network is quite simple, one can obtain the same predictions based on constraint simplification by term rewriting without any search. This kind of formal reasoning is very similar to the intuitive reasoning from Section 4.3.

Here we present a simplifcation algorithm that is correct only for specific choices of Δ, as for instance Δ_3 and Δ_6. The first restriction we need is that Acc^{Δ} and Inh^{Δ} are functional relations, so that we can rewrite $x \in Inh(t)$ to $x = Inh(t)$ and then eliminate x by substitution with $Inh(t)$.

In order to do so, we consider a variant of difference constraints in which applications $Acc(t)$ and $Inh(t)$ are used for defining a single difference relation, rather than a set thereof.

difference relation	t	$::=$	$x \mid \delta \mid Acc(t) \mid Inh(t)$
set of difference relations	s	$::=$	$t \mid s + s' \mid s \cdot s' \mid Acc(s) \mid Inh(s)$
difference constraints	ψ	$::=$	$t \in s \mid t{=}t' \mid \psi \wedge \psi' \mid \exists x.\psi$

In Figure 12 we present a collection of basic term rewrite rule for simplifying difference constraints. There correctness under the above assumptions on Δ is straightforward Rule (bv) replaces equal by equal while eliminating existentially bound variables (all variables z_{A} and v_{r_i} are implicitly existentially quantified). The simplification rules (no_i) remove the no-change-at-1 $\delta_{=1}$. The third rule (si) simplifies membership in singletons to equality. Rule (ip) expresses the idempotence of addition. The result of the basic simplification for the difference constraints for the basic $\text{P}_{\text{Ilv-Leu}}$ network with the rewrite rules in Figure 12 is

$$(\mathrm{no}_1)\quad Inh(\delta_{=1}) \;\Rightarrow\; \delta_{=1} \qquad (\mathrm{no}_3)\quad t\cdot\delta_{=1} \;\Rightarrow\; t \qquad (\mathrm{ip})\quad x+x \;\Rightarrow\; x$$
$$(\mathrm{no}_2)\quad Acc(\delta_{=1}) \;\Rightarrow\; \delta_{=1} \qquad (\mathrm{no}_4)\quad \delta_{=1}\cdot t \;\Rightarrow\; t \qquad (\mathrm{si})\quad t\in t' \;\Rightarrow\; t=t'$$
$$(\mathrm{bv})\quad \exists x.\,(x=t\wedge\psi) \;\Rightarrow\; \psi[t/x]$$
$$(\mathrm{inh})\quad t\in Inh(t+s) \;\Rightarrow\; t\in Inh(s)$$

Figure 12: Basic simplification rules

$$(\mathrm{inh}{+})\quad Inh(s)\cdot Inh(s') \;\Rightarrow\; Inh(s+s')$$

Figure 13: Special simplification rule

the following constraint:

$$y_{\mathrm{Leu}} \in Inh(x_{\mathrm{TnrA}})\cdot Inh(x_{\mathrm{CodY}}+y_{\mathrm{Leu}}+x_{\mathrm{TnrA}})\ .$$

The constraint can be simplified further by applying the rewrite rule (inh+) in Figure 13, whose correctness requires a further restriction on Δ.

$$y_{\mathrm{Leu}} \in Inh(x_{\mathrm{TnrA}}+x_{\mathrm{CodY}}+y_{\mathrm{Leu}}+x_{\mathrm{TnrA}})\ .$$

With a further application of simplification rule (ip), we obtain:

$$y_{\mathrm{Leu}} \in Inh(x_{\mathrm{CodY}}+y_{\mathrm{Leu}}+x_{\mathrm{TnrA}})\ .$$

When working over Δ_3 and assuming our target $y_{\mathrm{Leu}} = <$ then we can simplify the constraint further to: $< \in Inh(x_{\mathrm{CodY}}+x_{\mathrm{TnrA}})$ This is equivalent to that

$$x_{\mathrm{CodY}} => \vee\ x_{\mathrm{TnrA}} => .$$

This can be satisfied by decreasing the influx of either CodY or TnrA. Thus, we obtain the same safe solutions as before.

8. Predicting Multiple Changes

Some changes of influxes or reaction knockouts may have more than one possible outcome, while others may have only one. This observation was made already in Section 7 with the solutions that were safe or unsafe for leucine overproduction.

For example, reconsider Figure 11 with the solutions of the difference constraint for the basic $\mathrm{P}_{\mathrm{Ilv-Leu}}$ network. Consider the solution 5, 8, and 12. All three solutions require an increase of the CodY influx and a decrease of the TnrA influx at the same time. But the outcomes are different: solution 5 satisfies the target of leucine overproduction, while solutions 8 and 12 do not since they decrease the outflux of Leu or respectively leave it unchanged. Therefore, we said that solution 5 is not safe for the target. On the other hand, some changes may have only one possible outcome, as for safe solutions 1, 2 and 3.

It should be noticed that multiple outcomes are possible because of the lack of information on kinetic parameters. In the example, this does not let us determine whether the effect of the increase of CodY influx prevails on the decrease of TnrA, or the other way around, or whether they cancel each other out.

When looking for the overproduction of metabolites it is then important to distinguish the safe changes that *for sure* produce the desired effect because the overproduction happens for all the possible outcomes, from the compatible change that simply *may* produce the target effect, because for some other outcomes the effect does not happen. To formalize this, we say that a constraint ψ_c representing a generic change of the steady state of the model (for example CodY $= >$) is safe with respect to a target ψ_t (for example Leu $= <$) if ψ_c entails ψ_t. If this is not true but ψ_c is *compatible* with ψ_t (that is there exists at least one solution satisfying both), then ψ_c is unsafe.

Definition 4 (Safeness). Given a network N with difference constraints $[\![\phi_N]\!]$ interpreted over the abstract domain Δ and two other difference constraints ψ_c, ψ_t, we say that ψ_c is safe with respect to ψ_t if $[\![\phi_N]\!] \wedge \psi_c \models \psi_t$. We say that ψ_c is unsafe if $[\![\phi_N]\!] \wedge \psi_c \wedge \psi_t$ is satisfiable but $[\![\phi_N]\!] \wedge \psi_c \not\models \psi_t$.

Several changes are possible generally for a given network, involving one or more influx variables as for the basic $\mathrm{P}_{\mathrm{Ilv-Leu}}$ network, but possibly also one or more knockout variables as discussed later for the refined $\mathrm{P}_{\mathrm{Ilv-Leu}}$ network. Depending on the number of variables affected by the change, we talk about single, double, triple or in general multiple changes when one, two, three or many variables take a value that is not $\delta_{=1}$. We can then define a change as a difference constraint where some variables have value different than $\delta_{=1}$.

Definition 5 (Change). Given a network $N = (S, R, I, O, K)$ with difference constraint $[\![\phi_N]\!]$ over a given abstract domain Δ, with $\mathsf{ncvars}([\![\phi_N]\!]) = \{x_s \mid s \in S\} \cup \{o_r \mid r \in K\}$ the set of its control variables, a control assertion is a difference constraint

$$\psi \quad = \quad (x_1 = \delta_1 \quad \wedge \quad x_2 = \delta_2 \quad \wedge \quad \cdots \quad \wedge \quad x_p = \delta_p)$$

where $\{x_1, \ldots, x_p\} \subseteq \mathsf{ncvars}([\![\phi_N]\!])$, $\delta_i \in \Delta$ for $1 \leq i \leq p$ and $d_i \neq \delta_{=1}$ for some i. If $\{x_1, \ldots, x_p\} \subset \mathsf{ncvars}([\![\phi_N]\!])$, the control assertion is said partial. A n-tuple change is a control assertion where n variables have value different from $\delta_{=1}$.

Inflow changes and knockouts are the two most basic kinds of changes. Inflow changes in the Δ_6 domain are represented by $x_s = \uparrow$ or $x_s = \downarrow$ for one or more inflow variables x_s, while knockouts by $o_r = \Downarrow$ for one or more knockout variables o_r.

These basic changes can be combined to obtain more complex overproduction strategies. When inflow changes and knockouts are combined, we talk about *mixed changes*, as opposed to *strict changes*, when only knockouts or inflow changes are involved.

So for example there are two single safe changes in the basic $P_{Ilv-Leu}$ network (that are strict inflow changes, since no knockout is available): they correspond to solutions 1 and 2 in Figure 11, while solution 3 is a double safe strict inflow change. Remarkably, this double safe change is obtained as a combination of two single safe changes. Conversely, when a safe change is combined with an unsafe change or with a change which is not compatible with the target, the result is often unsafe as well, as for solution 4 obtained by the combination of 1 and 9. According to our experience, the combination of safe or unsafe changes with other unsafe changes or changes incompatible with the target gives usually rise to unsafe changes. Exceptions are however possible as discussed in Section 8.1.

Depending on the complexity of the model there may exist no safe change at all.

8.1. Application example

The refined $P_{Ilv-Leu}$ network in Figure 7 allows several changes, given by all the combinations of inflow changes discussed for the simple $P_{Ilv-Leu}$ network together with the knockouts of r_1, r_3, r_7, r_{14}. All the safe and unsafe changes are listed respectively in Appendix 12.1 and 12.2. The presence of the species $OP_{BkL-Bcd}$ influenced by CodY and TnrA modifies considerably the effect of the changes discussed in the simple network. There the two inflows acted as simple inhibitors of Leu, while in the refined network both the effects of inhibition and acceleration are present and make unpredictable the outcome of any strict inflow change.

In fact, all the strict inflow changes are compatible with leucine increase but unsafe, as shown by solutions 1–4 and 20–23 in Appendix 12.2. The simplest safe changes are instead represented by the knockouts of r_1, r_3 or r_7, corresponding to solutions 1–3 in Appendix 12.1. These can be combined at will to obtain more complex changes, that are still safe as shown by solutions 4–6 and 11. Intuitively, each of these knockouts blocks one of the three inhibitions acting on the production of leucine, either indirectly through r_2 or directly on r_8. Conversely, the knockout of r_{14} alone is not compatible with leucine increase, as one would expect since the only effect of this reaction is an acceleration of the production of the unique Leu precursor.

Further safe changes can be remarkably obtained by combining inflow changes (that are unsafe when applied alone, as previously discussed, or even incompatible with leucine outflow increase) with targeted knockouts. For example, the knockout of r_1 can block the inhibitory effect of CodY on Leu, so that an increase of CodY inflow also contributes to the overproduction of leucine. Intuitively, in Figure 7 we can see that once r_1 is blocked, an increase of CodY inflow has the only effect of inhibiting the inhibition of the production of Leu carried out by $OP_{BkL-Bcd}$, whose net effect is an acceleration of leucine outflow. In the same way r_7 can be knocked out and the inflow of TnrA can be increased to obtain the safe mixed double change 10 in Appendix 12.1.

Conversely, it is possible to block the positive effect of CodY and TnrA on Leu by knocking out r_3, getting back in practice the basic $P_{Ilv-Leu}$ network.

Indeed, if r_3 is knocked out we obtain solutions analogous to those listed in Figure 11, corresponding respectively to safe changes 8, 9, 20 in Appendix 12.1 and unsafe changes 47, 48 in Appendix 12.2. The presence of the knockout of r_3 allows however several other unsafe changes obtained by combination with CodY and TnrA inflow changes.

Safe double mixed changes can be further combined to obtain more complex safe changes, represented by solutions 12–19, 21–33 in Appendix 12.1.

8.2. Application to more complex networks

The basic and refined $P_{Ilv-Leu}$ networks are part of a more complex network module presented in [3], where single mixed knockouts were discussed. Again or the target of leucine overproduction, this complex network is an example of a model, for which no safe single or double knockout exist. Despite the complexity of the network, qualitative reasoning allowed us to exclude 7 over 21 knockout candidates because they turned out to be incompatible with leucine overproduction. This shows how qualitative reasoning can be useful even in the absence of safe solutions, when uncertainty allows reasoning only about compatibility or incompatibility of changes with respect to the desired target.

Work on improving the predictions for the complex network in [3] are in progress. Preliminary results show us the presence of safe changes appearing after a minimum of three knockouts, remarkably in agreement with the choice of the 6 knockouts previously tested in wet lab experimentation.

9. Modeling and Prediction Tool

We have implemented a tool for modeling reaction networks in our language and performing change predictions. The input of the tool are a reaction network in XML syntax and an overproduction target. From that, the steady state semantics is derived, from which the corresponding difference constraints are computed and solved. The set of all solutions is then analysed in order to filter out solutions that are compatible with the target or even safe.

XML Syntax. As its input, our tool uses an XML syntax for reaction networks that is then converted into an equivalent graphical syntax by an XSLT transformation, which produces latex graphics using the Tikz package. These graphics are included into the sources of the present article. For instance, the XML syntax for the basic $P_{Ilv-Leu}$ network from Figure 4 is given in Appendix 13.

Since both syntaxes are equivalent, the XML syntax could be generated from the graphical syntax. There exist tools for graphical input for Petri nets, but their sources are not freely available. And as it turned out for our purposes, the XML syntax was by far good enough as input syntax.

Constraint Solver. Difference constraint can be solved by finite domain constraint solving. The previous constraint solver from [1] was implemented in Scala [17] from scatch. This solver could enumerate the n-best solutions where $n \leq 3000$, while consuming considerable time with more than 10 minutes.

We now implemented a new solver by compiling difference constraint to minizinc [18]. This is much more efficient. Therefore it can indeed return the complete set of all solutions sets in all our application, while enumerating more than 5000 solutions in less then a second. An example for the minizinc constraint formulation of the difference constraint in Figure 10 is given in the Appendix 14.

Analysis of Solutions Sets. Much effort had also to be spend in a proper analysis of solutions sets. Most importantly, they must be presented in a readable output format, since they tend to be very large. For this, we sorted and grouped solution sets, eliminated dublicates, and removed irrelevant variables. We also compute the sets of safe solutions automatically, based on XPath queries with data joins. All these transformations were also done in Xslt.

10. Conclusions

We have presented a formal modeling language for chemical reaction networks with partial kinetic information, and shown how to abstract away from the unknowns thanks abstract interpretation. We have illustrated that this allows us to reason qualitatively about such networks, so that we can predict influx changes, reaction knockouts, or multiple change of these types. We have illustrate the relevance of the notion of safe changes, and how to compute them by inspecting the set of all solutions of a difference constraint.

An immediate question for future work is to find safe multiple changes for big reaction networks as the one in [3] and to validate them in the wet lab. An important question for future work on the longer scale is how to develop finer abstractions for quantitative predictions.

11. References

[1] M. John, M. Nebut, J. Niehren, Knockout Prediction for Reaction Networks with Partial Kinetic Information, in: 14th International Conference on Verification, Model Checking, and Abstract Interpretation, 2013, pp. 355–374.

[2] J. Niehren, M. John, C. Versari, F. Coutte, P. Jacques, Qualitative Reasoning about Reaction Networks with Partial Kinetic Information, in: Computational Methods for Systems Biology, Nantes, France, 2015, p. 12.

[3] F. Coutte, J. Niehren, D. Dhali, M. John, C. Versari, P. Jacques, Modeling Leucine's Metabolic Pathway and Knockout Prediction Improving the Production of Surfactin, a Biosurfactant from Bacillus Subtilis, Biotechnology Journal 10 (8) (2015) 1216–34.

[4] M. Feinberg, Chemical reaction network structure and the stability of complex isothermal reactors–I. the deficiency zero and deficiency one theorems, Chemical Engineering Science 42 (10) (1987) 2229 – 2268.

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

[6] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, U. Kummer, CopasiâĂŤa complex pathway simulator, Bioinformatics 22 (24) (2006) 3067–3074.

[7] F. Fages, S. Gay, S. Soliman, Inferring reaction systems from ordinary differential equations, Theor. Comput. Sci. 599 (2015) 64–78.

[8] J. D. Orth, I. Thiele, B. O. Palsson, What is flux balance analysis?, Nature biotechnology 28 (3) (2010) 245–248.

[9] J. A. Papin, J. Stelling, N. D. Price, S. Klamt, S. Schuster, B. O. Palsson, Comparison of network-based pathway analysis methods, Trends in biotechnology 22 (8) (2004) 400–405.

[10] J. M. Otero, J. Nielsen, Industrial systems biology, Industrial Biotechnology: Sustainable Growth and Economic Success.

[11] S. B. Sohn, T. Y. Kim, J. M. Park, S. Y. Lee, In silico genome-scale metabolic analysis of *Pseudomonas putida* kt2440 for polyhydroxyalkanoate synthesis, degradation of aromatics and anaerobic survival, Biotechnol. J. 5 (7) (2010) 739–750.

[12] C. Jungreuthmayer, J. Zanghellini, Designing optimal cell factories: integer programming couples elementary mode analysis with regulation, BMC systems biology 6 (1) (2012) 103.

[13] U. Mäder, A. G. Schmeisky, L. A. Flórez, J. Stülke, Subtiwikiğ014a comprehensive community resource for the model organism bacillus subtilis, Nucleic acids research (2011) gkr923.

[14] K. D. Forbus, Qualitative reasoning, in: A. B. Tucker (Ed.), The Computer Science and Engineering Handbook, CRC Press, 1997, pp. 715–733.

[15] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: POPL, 1979, pp. 269–282.

[16] U. Mäder, S. Hennig, M. Hecker, G. Homuth, Transcriptional organization and posttranscriptional regulation of the Bacillus subtilis branched-chain amino acid biosynthesis genes., Journal of bacteriology 186 (8) (2004) 2240–2252.

[17] M. Odersky, The evolution of scala: Ple'14 keynote, in: R. Urma, D. A. Orchard, A. Mycroft (Eds.), Proceedings of the 1st Workshop on Programming Language Evolution, PLE@ECOOP 2014, Uppsala, Sweden, July 28, 2014, ACM, 2014, p. 4.

[18] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, Minizinc: Towards a standard CP modelling language, in: C. Bessiere (Ed.), Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, 2007, Proceedings, Vol. 4741 of LNCS, Springer, 2007, pp. 529–543.

[19] V. Molle, Y. Nakaura, R. P. Shivers, H. Yamaguchi, R. Losick, Y. Fujita, A. L. Sonenshein, Additional targets of the Bacillus subtilis global regulator CodY identified by chromatin immunoprecipitation and genome-wide transcript analysis., Journal of bacteriology 185 (6) (2003) 1911–1922.

[20] R. P. Shivers, A. L. Sonenshein, Activation of the Bacillus subtilis global regulator CodY by direct interaction with branched-chain amino acids., Molecular microbiology 53 (2) (2004) 599–611.

[21] S. Tojo, T. Satomura, K. Morisaki, J. Deutscher, K. Hirooka, Y. Fujita, Elaborate transcription regulation of the Bacillus subtilis ilv-leu operon involved in the biosynthesis of branched-chain amino acids through global regulators of CcpA, CodY and TnrA, Molecular Microbiology 56 (6) (2005) 1560–1573.

[22] A. C. Villapakkam, L. D. Handke, B. R. Belitsky, V. M. Levdikov, A. J. Wilkinson, A. L. Sonenshein, Genetic and Biochemical Analysis of the Interaction of Bacillus subtilis CodY with Branched-Chain Amino Acids, J. Bacteriol. 191 (22) (2009) 6865–6876.

[23] S. R. Brinsmade, R. J. Kleijn, U. Sauer, A. L. Sonenshein, Regulation of CodY Activity through Modulation of Intracellular Branched-Chain Amino Acid Pools, J. Bacteriol. 192 (24) (2010) 6357–6368.

[24] J. A. Grandoni, S. A. Zahler, J. M. Calvo, Transcriptional regulation of the ilv-leu operon of Bacillus subtilis., Journal of bacteriology 174 (10) (1992) 3212–3219.

Part III
Appendix

12. Solutions for Leucine Increase

12.1. Safe Changes

	o_{r_1}	o_{r_3}	o_{r_7}
1.	$\Downarrow$	$\sim$	$\sim$
2.	$\sim$	$\Downarrow$	$\sim$
3.	$\sim$	$\sim$	$\Downarrow$

	o_{r_1}	o_{r_3}	o_{r_7}
4.	$\Downarrow$	$\Downarrow$	$\sim$
5.	$\Downarrow$	$\sim$	$\Downarrow$
6.	$\sim$	$\Downarrow$	$\Downarrow$

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
7.	$\Downarrow$	$\sim$	$\sim$	$\uparrow$	$\sim$
8.	$\sim$	$\Downarrow$	$\sim$	$\downarrow$	$\sim$
9.	$\sim$	$\Downarrow$	$\sim$	$\sim$	$\downarrow$
10.	$\sim$	$\sim$	$\Downarrow$	$\sim$	$\uparrow$

	o_{r_1}	o_{r_3}	o_{r_7}
11.	$\Downarrow$	$\Downarrow$	$\Downarrow$

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
12.	$\Downarrow$	$\Downarrow$	$\sim$	$\downarrow$	$\sim$
13.	$\Downarrow$	$\Downarrow$	$\sim$	$\sim$	$\downarrow$
14.	$\Downarrow$	$\Downarrow$	$\sim$	$\uparrow$	$\sim$
15.	$\Downarrow$	$\sim$	$\Downarrow$	$\sim$	$\uparrow$
16.	$\Downarrow$	$\sim$	$\Downarrow$	$\uparrow$	$\sim$
17.	$\sim$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\sim$
18.	$\sim$	$\Downarrow$	$\Downarrow$	$\sim$	$\downarrow$
19.	$\sim$	$\Downarrow$	$\Downarrow$	$\sim$	$\uparrow$

	o_{r_3}	x_{CodY}	x_{TnrA}
20.	$\Downarrow$	$\downarrow$	$\downarrow$

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
21.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\sim$
22.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\sim$	$\downarrow$
23.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\sim$	$\uparrow$
24.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\sim$

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
25.	⇓	⇓	∼	↓	↓
26.	⇓	⇓	∼	↑	↓
27.	⇓	∼	⇓	↑	↑
28.	∼	⇓	⇓	↓	↓
29.	∼	⇓	⇓	↓	↑

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
30.	⇓	⇓	⇓	↓	↓
31.	⇓	⇓	⇓	↓	↑
32.	⇓	⇓	⇓	↑	↓
33.	⇓	⇓	⇓	↑	↑

12.2. Unsafe Changes

	x_{CodY}	x_{TnrA}
1.	↓	∼
2.	∼	↓
3.	∼	↑
4.	↑	∼

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{r_{14}}$
5.	⇓	∼	∼	⇓
6.	∼	⇓	∼	⇓
7.	∼	∼	⇓	⇓

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{r_{14}}$	x_{CodY}	x_{TnrA}
8.	⇓	∼	∼	∼	↓	∼
9.	⇓	∼	∼	∼	∼	↓
10.	⇓	∼	∼	∼	∼	↑
11.	∼	⇓	∼	∼	∼	↑
12.	∼	⇓	∼	∼	↑	∼
13.	∼	∼	⇓	∼	↓	∼
14.	∼	∼	⇓	∼	∼	↓
15.	∼	∼	⇓	∼	↑	∼
16.	∼	∼	∼	⇓	↓	∼
17.	∼	∼	∼	⇓	∼	↓

	$o_{r_{14}}$	x_{CodY}	x_{TnrA}
18.	⇓	∼	↑
19.	⇓	↑	∼

	x_{CodY}	x_{TnrA}
20.	↓	↓
21.	↓	↑
22.	↑	↓
23.	↑	↑

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$
24.	$\Downarrow$	$\Downarrow$	$\sim$	$\Downarrow$
25.	$\Downarrow$	$\sim$	$\Downarrow$	$\Downarrow$
26.	$\sim$	$\Downarrow$	$\Downarrow$	$\Downarrow$

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
27.	$\Downarrow$	$\Downarrow$	$\sim$	$\sim$	$\sim$	$\uparrow$
28.	$\Downarrow$	$\sim$	$\Downarrow$	$\sim$	$\downarrow$	$\sim$
29.	$\Downarrow$	$\sim$	$\Downarrow$	$\sim$	$\sim$	$\downarrow$
30.	$\Downarrow$	$\sim$	$\sim$	$\Downarrow$	$\downarrow$	$\sim$
31.	$\Downarrow$	$\sim$	$\sim$	$\Downarrow$	$\sim$	$\downarrow$
32.	$\Downarrow$	$\sim$	$\sim$	$\Downarrow$	$\sim$	$\uparrow$
33.	$\Downarrow$	$\sim$	$\sim$	$\Downarrow$	$\uparrow$	$\sim$
34.	$\sim$	$\Downarrow$	$\Downarrow$	$\sim$	$\uparrow$	$\sim$
35.	$\sim$	$\Downarrow$	$\sim$	$\Downarrow$	$\downarrow$	$\sim$
36.	$\sim$	$\Downarrow$	$\sim$	$\Downarrow$	$\sim$	$\downarrow$

	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
37.	$\Downarrow$	$\sim$	$\Downarrow$	$\sim$	$\uparrow$
38.	$\Downarrow$	$\sim$	$\Downarrow$	$\uparrow$	$\sim$
39.	$\sim$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\sim$
40.	$\sim$	$\Downarrow$	$\Downarrow$	$\sim$	$\downarrow$
41.	$\sim$	$\Downarrow$	$\Downarrow$	$\sim$	$\uparrow$
42.	$\sim$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\sim$

	o_{r_1}	o_{r_3}	o_{r_7}	x_{CodY}	x_{TnrA}
43.	$\Downarrow$	$\sim$	$\sim$	$\downarrow$	$\downarrow$
44.	$\Downarrow$	$\sim$	$\sim$	$\downarrow$	$\uparrow$
45.	$\Downarrow$	$\sim$	$\sim$	$\uparrow$	$\downarrow$
46.	$\Downarrow$	$\sim$	$\sim$	$\uparrow$	$\uparrow$
47.	$\sim$	$\Downarrow$	$\sim$	$\downarrow$	$\uparrow$
48.	$\sim$	$\Downarrow$	$\sim$	$\uparrow$	$\downarrow$
49.	$\sim$	$\Downarrow$	$\sim$	$\uparrow$	$\uparrow$
50.	$\sim$	$\sim$	$\Downarrow$	$\downarrow$	$\downarrow$
51.	$\sim$	$\sim$	$\Downarrow$	$\downarrow$	$\uparrow$
52.	$\sim$	$\sim$	$\Downarrow$	$\uparrow$	$\downarrow$

	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
53.	$\Downarrow$	$\sim$	$\uparrow$	$\uparrow$
54.	$\sim$	$\Downarrow$	$\downarrow$	$\downarrow$
55.	$\sim$	$\Downarrow$	$\downarrow$	$\uparrow$
56.	$\sim$	$\Downarrow$	$\uparrow$	$\downarrow$
57.	$\sim$	$\Downarrow$	$\uparrow$	$\uparrow$

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
58.	⇓	⇓	∼	⇓	↓	∼
59.	⇓	⇓	∼	⇓	∼	↓
60.	⇓	⇓	∼	⇓	∼	↑
61.	⇓	⇓	∼	⇓	↑	∼
62.	⇓	∼	⇓	⇓	↓	∼
63.	⇓	∼	⇓	⇓	∼	↓
64.	⇓	∼	⇓	⇓	∼	↑
65.	⇓	∼	⇓	⇓	↑	∼
66.	∼	⇓	⇓	⇓	↓	∼
67.	∼	⇓	⇓	⇓	∼	↓

	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
68.	⇓	⇓	⇓	∼	↑
69.	⇓	⇓	⇓	↑	∼

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
70.	⇓	⇓	∼	∼	↓	↑
71.	⇓	⇓	∼	∼	↑	↑
72.	⇓	∼	⇓	∼	↓	↓
73.	⇓	∼	⇓	∼	↓	↑
74.	⇓	∼	⇓	∼	↑	↓
75.	⇓	∼	∼	⇓	↓	↓
76.	⇓	∼	∼	⇓	↓	↑
77.	⇓	∼	∼	⇓	↑	↓
78.	⇓	∼	∼	⇓	↑	↑
79.	∼	⇓	⇓	∼	↑	↓

	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
80.	⇓	⇓	∼	↑	↑
81.	⇓	∼	⇓	↓	↓
82.	⇓	∼	⇓	↓	↑
83.	⇓	∼	⇓	↑	↓
84.	⇓	∼	⇓	↑	↑
85.	∼	⇓	⇓	↓	↓
86.	∼	⇓	⇓	↓	↑
87.	∼	⇓	⇓	↑	↓
88.	∼	⇓	⇓	↑	↑

	o_{r_1}	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
89.	$\Downarrow$	$\Downarrow$	$\sim$	$\Downarrow$	$\downarrow$	$\downarrow$
90.	$\Downarrow$	$\Downarrow$	$\sim$	$\Downarrow$	$\downarrow$	$\uparrow$
91.	$\Downarrow$	$\Downarrow$	$\sim$	$\Downarrow$	$\uparrow$	$\downarrow$
92.	$\Downarrow$	$\Downarrow$	$\sim$	$\Downarrow$	$\uparrow$	$\uparrow$
93.	$\Downarrow$	$\sim$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\downarrow$
94.	$\Downarrow$	$\sim$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\uparrow$
95.	$\Downarrow$	$\sim$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\downarrow$
96.	$\Downarrow$	$\sim$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\uparrow$
97.	$\sim$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\downarrow$
98.	$\sim$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\downarrow$	$\uparrow$

	o_{r_3}	o_{r_7}	$o_{\mathrm{r}_{14}}$	x_{CodY}	x_{TnrA}
99.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\downarrow$
100.	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\uparrow$	$\uparrow$

13. XML Syntax of Basic Model

```
<?xml version="1.0" encoding="utf-8"?>

<network id="PIlv-Leu-Op-BkL-Bcd">
   <metabolite id="Leu" x="12.5" y="-6.25" comment="Leucine"/>
   <protein id="CcpA" x="19.00" y=" -4.00"
           comment="Carbon catabolite control protein A"/>
   <protein id="CodY" x="19.00" y=" -1.75"
           comment="Transcriptional pleiotropic regulator"/>
   <protein id="TnrA" x="19.00" y=" -6.25"
           comment="Nitrogen pleiotropic transcriptional
               regulator"/>
   <actor id="PIlv-Leu" x="12.5" y=" -4.00"
          comment="Activity of promoter of starting network
              producing of \M\Leu"/>
   <actor id="BSCodY" x="16.0" y=" -2.75"
          comment="Activity of \M\CodY binding to promotor \M\
              PIlvLeu"/>
   <actor id="BSTnrA" x="16.00" y=" -5.25"
          comment="Activity of \M\TnrA binding to promoter \M\
              PIlvLeu"/>
   <actor id="BSCcpA" x="16.0" y=" -4.0"
          comment="Activity of \M\CcpA binding to promotor \M\
              PIlvLeu without \M\BSTnrA loop"/>
   <actor id="OpBkLBcd" x="15.20" y="-7.25"
          comment="Activity of promoter of \M\BkL \M\Bcd operon"/>

   <!-- edge cluster points -->

  <edgecluster id="CodY2a" x="20.0" y=" -2.00" type="inhibitor">
    <source spec="CodY"/>
  </edgecluster>
  <edgecluster id="CodY2b" x="20.0" y=" -7.00" type="inhibitor">
    <source edgecluster="CodY2a"/>
  </edgecluster>
  <edgecluster id="BSCodYa" x="14.2" y=" -2.75" type="inhibitor">
    <source spec="BSCodY"/>
  </edgecluster>
  <edgecluster id="BSTnrAa" x="14.2" y=" -5.25" type="inhibitor">
    <source spec="BSTnrA"/>
  </edgecluster>

  <!-- context -->
```

```
 <context id="100" x="21.5" y="-1.75">
    <input spec="CodY"/>
 </context>
 <context id="101" x="21.5" y="-6.25">
    <input spec="TnrA"/>
 </context>
 <context id="102" x="11.05" y="-6.25">
    <output spec="Leu"/>
 </context>

<!-- reactions -->

 <reaction id="1" x="19.00" y="-2.75"
          comment="bind \M\CodY to \M\PIlvLeu for inhibition">
    <kinetics id="exp" angle="-120"/>
    <inhibitor spec="CcpA"/>
    <activator spec="CodY"/>
    <product spec="BSCodY"/>
 </reaction>
 <reaction id="1'" x="15.0" y="-2.05" comment="implicit
     degradation">
    <kinetics id="ma" angle="45"/>
    <reactant spec="BSCodY"/>
 </reaction>
 <reaction id="2" x="14.20" y="-4.00"
          comment="activate \M\PIlvLeu promoter">
    <kinetics id="exp" angle="30"/>
    <inhibitor edgecluster="BSCodYa"/>
    <accelerator spec="BSCcpA"/>
    <inhibitor spec="Leu"/>
    <inhibitor edgecluster="BSTnrAa"/>
    <product spec="PIlv-Leu"/>
 </reaction>
 <reaction id="2'" x="11.8" y="-3.3" comment="implicit
     degradation">
    <kinetics id="ma" angle="30"/>
    <reactant spec="PIlv-Leu"/>
 </reaction>
 <reaction id="3" x="19.0" y="-7.25"
          comment="bind \M\BkdR to \M\BkL \M\Bcd promoter">
   <kinetics id="exp" angle="-30"/>
   <inhibitor spec="TnrA"/>
   <inhibitor edgecluster="CodY2b"/>
```

```
  <product spec="OpBkLBcd"/>
 </reaction>
 <reaction id="3'" x="13.65" y="-7.25" comment="implicit
     degradation">
    <kinetics id="ma" angle="30"/>
    <reactant spec="OpBkLBcd"/>
 </reaction>
 <reaction id="7" x="19.00" y="-5.25"
          comment="bind \M\TnrA to \M\PIlvLeu promoter for
              inhibition">
    <kinetics id="exp" angle="120"/>
     <activator spec="TnrA"/>
    <product spec="BSTnrA"/>
 </reaction>
 <reaction id="7'" x="15.0" y=" -4.7" comment="implicit
     degradation">
    <kinetics id="ma" angle="30"/>
    <reactant spec="BSTnrA"/>
 </reaction>
 <reaction id="8" x="12.5" y="-5.00"
          comment="\M\PIlvLeu expression followed by \M\Leu
              production">
    <kinetics angle="45" id="exp"/>
    <inhibitor spec="OpBkLBcd"/>
    <reactant spec="PIlv-Leu"/>
    <product spec="Leu"/>
 </reaction>
<reaction id="9" x="17.5" y=" -4.0"
          comment="bind \M\CcpA to \M\PIlvLeu promoter without \
              M\BSTnrA loop">
  <kinetics id="exp" angle="-30"/>
  <activator spec="CcpA"/>
  <inhibitor spec="BSTnrA"/>
  <product spec="BSCcpA"/>
</reaction>
 <reaction id="9'" x="15.0" y="-3.3" comment="implicit
     degradation">
    <kinetics id="ma" angle="30"/>
    <reactant spec="BSCcpA"/>
 </reaction>
 <reaction id="14" x="20.5" y="-4.0" comment="expressions of
     CcpA">
    <kinetics id="exp" angle="30"/>
```

```
      <product spec="CcpA"/>
    </reaction>
    <reaction id="14'" x="17.5" y="-3.3" comment="implicit
        degradation">
      <kinetics id="ma" angle="30"/>
      <reactant spec="CcpA"/>
    </reaction>
    <reaction id="15'" x="17.5" y="-1.75" comment="\CodY
        deactivation">
      <kinetics id="ma" angle="30"/>
      <reactant spec="CodY"/>
    </reaction>
    <reaction id="16'" x="17.5" y="-6.25" comment="\TnrA
        deactivation">
      <kinetics id="ma" angle="30"/>
      <reactant spec="TnrA"/>
    </reaction>
  </network>
```

14. Minizinc Version of Difference Constraint

```
int: No = 1;
int: Up = 2;
int: Do = 3;
int: NoN = 4;
int: UpN = 5;
int: DoN = 6;
set of int: Directions = {No, Up, Do, NoN, UpN, DoN};
array[1..6] of string: dirStrings = ["No", "Up", "Do", "NoN", "
    UpN", "DoN"];

include "table.mzn";


% s1 (presumably a reaction flow) is accelerated by s2 (
    presumably a chemical species)
predicate abstractAccelerator(var int: s1, var int: s2, var int:
    finalRes) =
  abstractAcceleratorTable(s1, s2, finalRes);
  % abstractAcceleratorBackward(s1, s2, finalRes) /\
      abstractAcceleratorForward(s1, s2, finalRes);


array [1..44, 1..3] of Directions: acceleratorTable =
  [| No, No, No | No, Up, Up | No, Do, Do | No, NoN, No | No, UpN
      , Up | No, DoN, Do
   | Up, No, Up | Up, Up, Up | Up, Do, Up | Up, Do, No | Up, Do,
       Do | Up, NoN, Up | Up, UpN, Up | Up, DoN, Up | Up, DoN, No
       | Up, DoN, Do
   | Do, No, Do | Do, Up, Up | Do, Up, No | Do, Up, Do | Do, Do,
       Do | Do, NoN, Do | Do, UpN, Up | Do, UpN, No | Do, UpN, Do
       | Do, DoN, Do
   | NoN, No, NoN | NoN, Up, NoN | NoN, Do, NoN | NoN, NoN, NoN |
       NoN, UpN, NoN | NoN, DoN, NoN
   | UpN, No, UpN | UpN, Up, UpN | UpN, Do, UpN | UpN, NoN, UpN |
       UpN, UpN, UpN | UpN, DoN, UpN
   | DoN, No, DoN | DoN, Up, DoN | DoN, Do, DoN | DoN, NoN, DoN |
       DoN, UpN, DoN | DoN, DoN, DoN
   |];

% s1 (presumably a reaction flow) is accelerated by s2 (
    presumably a chemical species)
predicate abstractAcceleratorTable(var Directions: s1, var
    Directions: s2, var Directions: finalRes) =
   let {
```

```
      array [1..3] of var int: dd;
      constraint dd[1] = s1;
      constraint dd[2] = s2;
      constraint dd[3] = finalRes;
  } in table(dd, acceleratorTable);


% s1 (presumably a reaction flow) is inhibited by s2 (presumably
    a chemical species)
predicate abstractInhibitor(var int: s1, var int: s2, var int:
    finalRes) =
  abstractInhibitorTable(s1, s2, finalRes);
  %abstractInhibitorBackward(s1, s2, finalRes) /\
      abstractInhibitorForward(s1, s2, finalRes);


array [1..44, 1..3] of Directions: inhibitorTable =
  [| No, No, No | No, Up, Do | No, Do, Up | No, NoN, No | No, UpN
      , Do | No, DoN, Up
   | Up, No, Up | Up, Up, Up | Up, Up, No | Up, Up, Do | Up, Do,
       Up | Up, NoN, Up | Up, UpN, Up | Up, UpN, No | Up, UpN, Do
       | Up, DoN, Up
   | Do, No, Do | Do, Up, Do | Do, Do, Up | Do, Do, No | Do, Do,
       Do | Do, NoN, Do | Do, UpN, Do | Do, DoN, Up | Do, DoN, No
       | Do, DoN, Do
   | NoN, No, NoN | NoN, Up, NoN | NoN, Do, NoN | NoN, NoN, NoN |
       NoN, UpN, NoN | NoN, DoN, NoN
   | UpN, No, UpN | UpN, Up, UpN | UpN, Do, UpN | UpN, NoN, UpN |
       UpN, UpN, UpN | UpN, DoN, UpN
   | DoN, No, DoN | DoN, Up, DoN | DoN, Do, DoN | DoN, NoN, DoN |
       DoN, UpN, DoN | DoN, DoN, DoN
   |];

% s1 (presumably a reaction flow) is inhibited by s2 (presumably
    a chemical species)
predicate abstractInhibitorTable(var Directions: s1, var
    Directions: s2, var Directions: finalRes) =
  let {
      array [1..3] of var int: dd;
      constraint dd[1] = s1;
      constraint dd[2] = s2;
      constraint dd[3] = finalRes;
  } in table(dd, inhibitorTable);
```

```
predicate abstractProduct(var int: s1, var int: s2, var int:
    finalRes) =
  abstractProductTable(s1, s2, finalRes);
  %abstractProductBackward(s1, s2, finalRes) /\
      abstractProductForward(s1, s2, finalRes);


array [1..40, 1..3] of Directions: productTable =
  [| No, No, No | No, Up, Up | No, Do, Do | No, NoN, NoN | No,
      UpN, UpN | No, DoN, DoN
   | Up, No, Up | Up, Up, Up | Up, Do, Up | Up, Do, No | Up, Do,
       Do | Up, NoN, NoN | Up, UpN, UpN | Up, DoN, DoN
   | Do, No, Do | Do, Up, Up | Do, Up, No | Do, Up, Do | Do, Do,
       Do | Do, NoN, NoN | Do, UpN, UpN | Do, DoN, DoN
   | NoN, No, NoN | NoN, Up, NoN | NoN, Do, NoN | NoN, NoN, NoN |
       NoN, UpN, NoN | NoN, DoN, NoN
   | UpN, No, UpN | UpN, Up, UpN | UpN, Do, UpN | UpN, NoN, NoN |
       UpN, UpN, UpN | UpN, DoN, NoN
   | DoN, No, DoN | DoN, Up, DoN | DoN, Do, DoN | DoN, NoN, NoN |
       DoN, UpN, NoN | DoN, DoN, DoN
   |];

predicate abstractProductTable(var Directions: s1, var Directions
    : s2, var Directions: finalRes) =
   let {
       array [1..3] of var int: dd;
       constraint dd[1] = s1;
       constraint dd[2] = s2;
       constraint dd[3] = finalRes;
   } in table(dd, productTable);


predicate abstractSum(var int: s1, var int: s2, var int: finalRes
    ) =
  abstractSumTable(s1, s2, finalRes);
  %abstractSumBackward(s1, s2, finalRes) /\ abstractSumForward(s1
      , s2, finalRes);
```

```
array [1..52, 1..3] of Directions: sumTable =
  [| No, No, No | No, Up, Up | No, Do, Do | No, NoN, No | No, UpN
      , Up | No, DoN, Do
   | Up, No, Up | Up, Up, Up | Up, Do, Up | Up, Do, No | Up, Do,
       Do | Up, NoN, Up | Up, UpN, Up | Up, DoN, Up | Up, DoN, No
       | Up, DoN, Do
   | Do, No, Do | Do, Up, Up | Do, Up, No | Do, Up, Do | Do, Do,
       Do | Do, NoN, Do | Do, UpN, Up | Do, UpN, No | Do, UpN, Do
       | Do, DoN, Do
   | NoN, No, No | NoN, Up, Up | NoN, Do, Do | NoN, NoN, NoN | NoN
       , UpN, UpN | NoN, DoN, DoN
   | UpN, No, Up | UpN, Up, Up | UpN, Do, Up | UpN, Do, No | UpN,
       Do, Do | UpN, NoN, UpN | UpN, UpN, UpN | UpN, DoN, Up | UpN
       , DoN, No | UpN, DoN, Do
   | DoN, No, Do | DoN, Up, Up | DoN, Up, No | DoN, Up, Do | DoN,
       Do, Do | DoN, NoN, DoN | DoN, UpN, Up | DoN, UpN, No | DoN,
        UpN, Do | DoN, DoN, DoN
   |];

predicate abstractSumTable(var Directions: s1, var Directions: s2
    , var Directions: finalRes) =
   let {
       array [1..3] of var int: dd;
       constraint dd[1] = s1;
       constraint dd[2] = s2;
       constraint dd[3] = finalRes;
   } in table(dd, sumTable);


predicate constant(var int: s) =
  s == No;


predicate monotonic(var int: s1, var int: s2) =
  s1 == s2;



%% global variables %%

  var Directions: x_CodY; constraint x_CodY in {Do, Up, No};
```

```
  var Directions: x_TnrA; constraint x_TnrA in {Do, Up, No};
  var Directions: o_1; constraint o_1 in {DoN, No, NoN};
  var Directions: o_14; constraint o_14 in {DoN, No, NoN};
  var Directions: o_3; constraint o_3 in {DoN, No, NoN};
  var Directions: o_7; constraint o_7 in {DoN, No, NoN};
  var Directions: o_8; constraint o_8 in {DoN, No, NoN};


constraint let {

  %% local variables %%
  var Directions: y_Leu; constraint y_Leu in {Do, Up, No};
  var Directions: z_BSCodY;
  var Directions: z_PIlv_Leu;
  var Directions: z_BSCcpA;
  var Directions: z_BSTnrA;
  var Directions: z_OpBkLBcd;
  var Directions: w_d4e6;
  var Directions: w_d1e43;
  var Directions: w_d1e57;
  var Directions: w_d1e59;
  var Directions: w_d1e64;
  var Directions: w_d4e19;
  var Directions: w_d1e97;
  var Directions: w_d4e33;
  var Directions: w_d1e115;
  var Directions: w_d4e51;
  var Directions: w_d1e205;
  var Directions: w_d1e217;
```

```
%% constants %%

var Directions: u_NoN; constraint u_NoN in {NoN};
var Directions: u_No; constraint u_No in {No};


%% steady state constraints %%
 constraint ( bool2int(o_1 in {DoN} ) + bool2int(w_d4e6 in {NoN,
      UpN} )) <= 1;
constraint abstractProduct(o_1,w_d4e6,z_BSCodY);
constraint abstractProduct(x_CodY,w_d1e43,w_d4e6);
constraint abstractInhibitor(u_No,o_14,w_d1e43);
constraint abstractSum(z_PIlv_Leu,y_Leu,w_d1e57);
constraint abstractProduct(w_d1e59,w_d1e64,w_d1e57);
constraint abstractInhibitor(u_No,w_d4e19,w_d1e59);
constraint abstractAccelerator(u_No,z_BSCcpA,w_d1e64);
var Directions: s_d1e92;
constraint abstractSum(z_BSCodY,y_Leu,s_d1e92);
constraint abstractSum(s_d1e92,z_BSTnrA,w_d4e19);
 constraint ( bool2int(o_3 in {DoN} ) + bool2int(w_d1e97 in {NoN
     , UpN} )) <= 1;
constraint abstractInhibitor(u_No,w_d4e33,w_d1e97);
constraint abstractProduct(o_3,w_d1e115,z_OpBkLBcd);
constraint abstractInhibitor(u_No,w_d4e33,w_d1e115);
constraint abstractSum(x_TnrA,x_CodY,w_d4e33);
 constraint ( bool2int(o_7 in {DoN} ) + bool2int(x_TnrA in {NoN,
      UpN} )) <= 1;
constraint abstractProduct(o_7,x_TnrA,z_BSTnrA);
 constraint ( bool2int(o_8 in {DoN} ) + bool2int(w_d4e51 in {NoN
     , UpN} )) <= 1;
constraint abstractProduct(o_8,w_d4e51,y_Leu);
constraint abstractProduct(z_PIlv_Leu,w_d1e205,w_d4e51);
constraint abstractInhibitor(u_No,z_OpBkLBcd,w_d1e205);
constraint abstractProduct(o_14,w_d1e217,z_BSCcpA);
constraint abstractInhibitor(u_No,z_BSTnrA,w_d1e217);
 constraint ( bool2int(o_14 in {DoN} ) + bool2int(u_No in {NoN,
     UpN} )) <= 1;
%% overproduce leucine %%
constraint y_Leu in {Up};
  %% no more than one knockout variable is $\DoN$ %%
 constraint ( bool2int(o_1 in {DoN} ) + bool2int(o_3 in {DoN} )
     + bool2int(o_14 in {DoN} ) + bool2int(o_7 in {DoN} ) +
     bool2int(o_8 in {DoN} )) <= 1;
%% no more than one reaction is removed: knockout variable is $
    \NoN$ %%
 constraint ( bool2int(o_1 in {NoN} ) + bool2int(o_3 in {NoN} )
```

```
      + bool2int(o_14 in {NoN} ) + bool2int(o_7 in {NoN} ) +
      bool2int(o_8 in {NoN} )) <= 1;

} in true;
solve satisfy;

output [
  "x_CodY: ", show(dirStrings[fix(x_CodY)]), "\n",
  "x_TnrA: ", show(dirStrings[fix(x_TnrA)]), "\n",
  "o_1: ", show(dirStrings[fix(o_1)]), "\n",
  "o_14: ", show(dirStrings[fix(o_14)]), "\n",
  "o_3: ", show(dirStrings[fix(o_3)]), "\n",
  "o_7: ", show(dirStrings[fix(o_7)]), "\n",
  "o_8: ", show(dirStrings[fix(o_8)]), "\n",
];
```