Computing Difference Abstractions of Linear Equation Systems

Emilie Allart, Joachim Niehren, Cristian Versari

▶ **To cite this version:**

Emilie Allart, Joachim Niehren, Cristian Versari. Computing Difference Abstractions of Linear Equation Systems. Theoretical Computer Science, 2021, ⟨10.1016/j.tcs.2021.06.030⟩. ⟨hal-03156136v4⟩

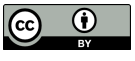
HAL Id: hal-03156136

https://hal.science/hal-03156136v4

Submitted on 13 May 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

Computing Difference Abstractions of Linear Equation Systems

Emilie Allart[a,c], Joachim Niehren[a,b], Cristian Versari[a,c]

[a] *BioComputing CRIStAL - Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL) - UMR 9189*
[b] *Linking Dynamic Data CRIStAL - UMR 9189, Inria Lille - Nord Europe*
[c] *Université de Lille*

Abstract

Abstract interpretation was proposed for predicting changes of reaction networks with partial kinetic information in systems biology. This requires to compute the set of difference abstractions of a system of linear equations under nonlinear constraints. We present the first practical algorithm that can compute the difference abstractions of linear equation systems exactly. We also present a new heuristics based on minimal support consequences for overapproximating the set of difference abstractions. Our algorithms rely on elementary modes, first-order definitions, and finite domain constraint programming. We implemented our algorithms and applied them to change prediction in systems biology. It turns out experimentally that the new heuristics is often exact in practice, while outperforming the exact algorithm.

The present document from 2025 revises a journal article published in Theoretical Computer Science [1] in 2021, which in turn extends on a paper published at the 17th International Conference on Computational Methods in Systems Biology (CMSB'2019) [2].

Keywords: abstract interpretation, constraint programming, linear programming, elementary modes, first-order definitions, gene knockout prediction, reaction networks, systems biology, synthetic biology, metabolic networks, boolean abstraction.

1. Introduction

Motivated by analysis questions for steady states [3, 4] of chemical reaction networks [5, 6, 7, 8] we study the problem of how to compute the set of difference abstractions of the solutions of a given linear equations system. The difference abstractions may be subject to constraints, which in the motivating application serve for expressing partial kinetic information on inhibitors and accelerators of the reactions of the network [9].

Problem. We consider systems of homogeneous linear equations with variables for positive real numbers in $\mathbb{R}_+$ including 0, possibly existentially quantified, such as for instance $E(X,Y)$ in (1):

$$\exists U.\exists V. \quad \begin{pmatrix} -1 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ U \\ V \end{pmatrix} = 0 \tag{1}$$

Given that the matrix is triangular and the variables U, V are existentially quantified, we can read off the set of solutions of $E(X,Y)$ easily. A variable assignment is a solution over $\mathbb{R}_+$ if and only if it maps X and Y to the same number:

$$sol^{\mathbb{R}_+}(E(X,Y)) = \{\alpha : \{X,Y\} \to \mathbb{R}_+ \mid \alpha(X) = \alpha(Y)\}$$

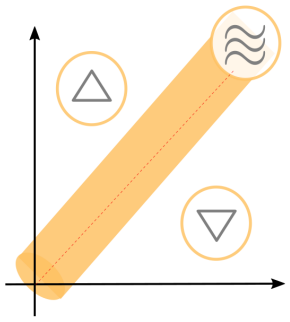


Figure 1: The difference abstraction to Δ_3.

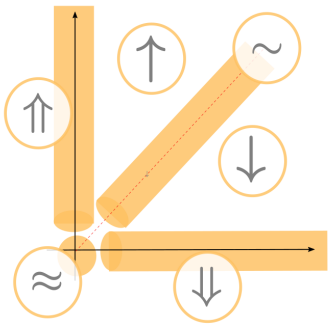


Figure 2: The difference abstraction to Δ_6.

As a logical formula with operators $+$ and $*$, the equation system $E(X, Y)$ can be rewritten as:

$$\exists U.\exists V. \quad (\ U = X + V \quad \wedge \quad U = Y + V \ )$$

This formula is equivalent to the linear equation $X = Y$ when interpreted over the real numbers, so that:

$$sol^{\mathbb{R}_+}(E(X, Y)) = sol^{\mathbb{R}_+}(X = Y)$$

An instance of the question studied in this article is, what happens to the value of Y if we increase the value of X, while jumping from one solution of $E(X, Y)$ to another? Clearly, the value of Y must be increased as well, since X and Y must have the same value in all solutions. More generally, we want to compute all possible relationships of any two solutions of $E(X, Y)$ over the positive reals.

For making this problem more precise, we consider a partition of the space $\mathbb{R}_+^2$ of concrete differences into the set of abstract differences $\Delta_3 = \{\triangle, \triangledown, \approxeq\}$, which is illustrated in Figure 1. The abstract difference $\triangle$ stands for an increase, $\approxeq$ for a no-change, and $\triangledown$ for a decrease. Given a finite set of variables V, we define the difference abstraction of two variable assignments to the positive reals $\alpha, \alpha' : V \to \mathbb{R}_+$ as the assignment $\beta : V \to \Delta_3$ such that for all $x \in V$:

$$\beta(x) = \begin{cases} \triangle & \text{if } \alpha(x) < \alpha'(x) \\ \approxeq & \text{if } \alpha(x) = \alpha'(x) \\ \triangledown & \text{if } \alpha(x) > \alpha'(x) \end{cases}$$

Our objective then is to compute for any given linear equation system the set of difference abstractions of any two positive real solutions. In our example system $E(X, Y)$, the expected answer is $\{\beta : \{X, Y\} \to \Delta_3 \mid \beta(X) = \beta(Y)\}$. In general, the computation can be done in exponential time by a generate and test algorithm based on Lemma 1 presented for the first time below, which reduces the test to solving a linear program. This algorithm, however, is too slow in practice, to be applicable to change prediction tasks in systems biology. This is why the previous approach by John et al. [9, 10, 11] computed an overapproximation only, by interpreting the equation system abstractly over the canonical relational structure with finite domain Δ_3 and signature $\Sigma = \Sigma_{pos\text{-}arith} = \{+, *, 0, 1\}$, and solving it by finite domain constraint programming [12, 13]. Each of the finitely many abstract solution obtained by abstract interpretation can then be tested based on Lemma 1, on whether it is the abstraction of some concrete solution. But still an exponential number of tests (each in polynomial time) may be needed in the worst case. The question we study in the present article is whether there exists a better algorithm useful for

change prediction in systems biology, that can compute the difference abstraction exactly while avoiding any overapproximation.

Furthermore, two variants of the above problem must be supported for practical application to systems biology. First, we need to be able to treat a refined difference abstraction with 6 values $\Delta_6 = \{\uparrow, \downarrow, \sim, \Uparrow, \Downarrow, \approx\}$ illustrated in Figure 2. Note that $\bigtriangleup = \uparrow \uplus \Uparrow$, $\bigtriangledown = \downarrow \uplus \Downarrow$ and $\approxeq = \sim \uplus \approx$, depending on whether the change started or ended with 0 or not. Second, for capturing partial kinetic information up to similarity, we must be able to impose additional constraints on the abstract solutions we are interested in. These will be given by some first-order formulas that are to be interpreted over the abstract domain.

Contributions. First, we generalize the difference abstractions $h_{\Delta_3} : \mathbb{R}_+^2 \to \Delta_3$ and $h_{\Delta_6} : \mathbb{R}_+^2 \to \Delta_6$ algebraically to homomorphisms between the Σ-structures. The set of concrete differences $\mathbb{R}_+^2$ becomes a Σ-algebra with signature $\Sigma = \Sigma_{pos\text{-}arith}$, where $+$ and $*$ are interpreted by pointwise addition and multiplication, the constant 0 as the pair $(0, 0)$, and the constant 1 as the pair $(1, 1)$. The sets of abstract differences Δ_3 and Δ_6 become $\Sigma_{pos\text{-}arith}$-structures that are naturally induced by the finite partitioning of $\mathbb{R}_+^2$ in Figures 1 and 2. The partionnings define homomorphism between $\Sigma_{pos\text{-}arith}$-structures $h_{\Delta_3} : \mathbb{R}_+^2 \to \Delta_3$ and $h_{\Delta_6} : \mathbb{R}_+^2 \to \Delta_6$.

Second, we can show that John's overapproximation theorem for difference abstractions [9, 11] can be generalized to any homomorphism $h : S \to \Delta$ between Σ-structures S and Δ. It states for any negation-free first-order Σ-formula ϕ, that the set of abstractions $h \circ \alpha$ of some S-solution α of ϕ can be approximated by the abstract interpretation of ϕ over Δ:

$$h \circ sol^S(\phi) \subseteq sol^\Delta(\phi)$$

This means that for any concrete solution α of ϕ that $h \circ \alpha$ is an abstract solution of ϕ. If furthermore Δ is finite, then we can compute $sol^\Delta(\phi)$ by finite domain constraint programming. This approach was shown to be applicable in the motivating application from systems biology [10].

The objective of this article, however, is to compute exactly the set of difference abstractions of the set of pairs of real solutions of a linear equation system for both cases $h_{\Delta_3} \circ sol^{\mathbb{R}_+^2}(\phi)$ and respectively $h_{\Delta_6} \circ sol^{\mathbb{R}_+^2}(\phi)$, where ϕ is a system of linear equations possibly with existential quantifiers (which clearly can be seen as Σ-formulas). The motivation is to overcome the main problematics of John's overapproximation, which can already be illustrated at the example of $E(X, Y)$. When interpreted over the Σ-structure Δ_3, the system $E(X, Y)$ admits the abstract solution $\beta = [X/\bigtriangleup, Y/\bigtriangledown]$, since $\bigtriangleup +^{\Delta_3} \bigtriangledown$ may be related to any value in Δ_3 nondeterministically. However, β is not the difference abstraction of any concrete $\mathbb{R}_+^2$-solution – i.e. any pair of $\mathbb{R}_+$-solutions –, given that $E(X, Y)$ implies $X = Y$. In other words, $X = Y$ is a logical consequence of $E(X, Y)$ over $\mathbb{R}_+$, but not over Δ_3, so precisely the information that we seek in our example is lost by abstract interpretation relying on John's overapproximation.

In order to avoid any overapproximation, one may want to enrich the linear equation system before solving it over the finite abstract structure, by adding all its logical consequences over $\mathbb{R}_+$, which correspond to all the linear combinations of the rows of the matrix (its row space). However this leads to an infinite number of consequences, so one would be forced to consider them modulo equivalence up to abstract interpretation. Since the abstract domains considered are finite, the number of equivalence classes would be finite too. However, finding a representative for each equivalence class corresponds to solving a linear programming problem, and the number of consequences to be added is exponential in the dimension of the matrix. Therefore, even if this approach leads to a finite representation of the set of logical consequences of the linear equation system, it is still unfeasible in practice for complexity reasons. A less inefficient approach but still infeasible in practice can be obtained by the *generate and test* algorithm metionned earlier.

The idea is to test the satisfiability of each abstract solution candidate individually by linear programming based on Lemma 1 below.

As a third contribution we propose a new heuristics based on minimal support consequences to improve John's overapproximation algorithm. Given a system of linear equations, the idea is to add all linear consequences that have a minimal number of variables and normalized coefficients, before computing the abstract solutions. We show how to compute this finite set of linear consequences based on elementary modes [14, 15] and orthogonal complements.

As a fourth and most important contribution, we present algorithms for computing the difference abstractions for linear equation system exactly. They can deal with Δ_3 and Δ_6 and with the addition of constraints on the abstract differences. The exact abstraction problems are reduced to finite domain constraint problems, that can be solved in practice with existing finite domain constraint solvers. The reductions are based on properties of first-order definitions that permit to reason with concrete differences in a first-order logic with pairs. Furthermore, we rely on a recent algorithm [16] for the exact rewriting of linear equation systems with respect to the boolean abstraction $h_{\mathbb{B}} : \mathbb{R}_+ \to \mathbb{B}$, which maps 0 to 0 and all other positive real numbers to 1. The linear equation system ϕ obtained by exact rewriting for the boolean abstraction satisfies $h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(\phi) = sol^{\mathbb{B}}(\phi)$ and is again based on the computation of elementary modes.

Fifth, we implemented the minimal support heuristics and our exact abstraction algorithm for Δ_6, and applied them to the prediction of leucine overproduction, a benchmark task for change prediction in systems biology [9, 10]. It turns out that the minimal support heuristics indeed computes the difference abstraction to Δ_6 exactly for this benchmark, while it does not do so in general. The main advantage of this heuristic is that it outperforms the exact algorithm dramatically in computation time: only 5 minutes are needed for the knockout prediction rather than 5 hours with the always exact algorithm.

The present article is an improved version the journal article in [1], which in turn extends on the conference paper at CMSB'2019 [2]. The conference paper was an extended abstract without any proofs, where most of the technical difficulties could not be exposed. In particular, we could not explain how to decompose the difference abstractions for Δ_3 and Δ_6 into the boolean abstraction based on functions defined in first-order logic. Also we could not describe in any sufficient detail how the addition of kinetic constraint can be dealt with. So the additional material mainly consists in detailed sections on how to compute difference abstractions to Δ_3 and Δ_6 exactly, also in the presence of kinetic constraints. The case of Δ_6 is considerably more difficult to be treated, since it requires a more advanced kind of first-order definitions. Furthermore, the presentation of the minimal support heuristics has been extended and equipped with a correctness argument.

The present article contains full proofs, of which some simpler inductions are delegated to the appendix.

Outline. In Sections 2 we discuss further related work and illustrate the application to change prediction in systems biology. In Section 3, we start with some general preliminaries before recalling the notion of homomorphism between Σ-structuresfrom universal algebra, that we illustrate by the boolean and the sign abstraction from the $\Sigma_{pos\text{-}arith}$-algebra $\mathbb{R}_+$ to the $\Sigma_{pos\text{-}arith}$-algebra $\mathbb{B}$. In Section 4, we recall the $\Sigma_{pos\text{-}arith}$-structures of abstract differences Δ_3 and Δ_6. The standard first-order logic for talking about Σ-structures by Σ-formulas is recalled in Section 5, concommittantly with the notion of difference abstractions of first-order formulas. The classes of arithmetic and polynomial equations studied later on are introducted in Section 6.

The first new results are presented in Section 7: there we generalize John's operapproximation theorem for difference abstractions of logical formulas to soundness theorem for the abstract interpretation of logical formulas with respect to homomorphic abstractions. We then recall a completeness result for the boolean abstraction from [16]. It shows that any linear equation

system can be rewritten equivalently into a $h_{\mathbb{B}}$-exact system. We introduce the *first-order tuple logic* in Section 8 and relate it to the standard first-order logic. The special case of the first-order pair logic serves in the proofs of our main result in Section 9. Based on the exact boolean rewriting and the first-order pair logic, we provide new characterizations of difference abstractions of linear equation systems, both for Δ_3 and Δ_6. These characterizations yield algorithms for computing difference abstractions of linear equation systems in both cases. In Section 10.1, we lift these algorithm to difference abstractions of linear equation systems with additional difference constraints. An overapproximation heuristics with minimal support is then presented in Section 11. In Section 12, an implementation and experimentation section compares these two approaches. In Section 13, finally, we prove a general result on first-order definitions in the first-order tuple logic, on which the characterizations of the difference abstractions rely. A conclusion follows. In Appendix, we add missing proofs on the basic properties of the first-order tuple logic.

Contents

1 Introduction **1**

2 Related Work **6**
2.1 Linear Programming . . . 6
2.2 Change Prediction in Systems Biology . . . 7
2.3 Flux Balance Analysis . . . 11

3 Preliminaries **11**
3.1 Sets and Tuples . . . 11
3.2 Sequences . . . 11
3.3 Functions and Relations . . . 12
3.4 Σ-Algebras . . . 12
3.5 Σ-Structures . . . 12
3.6 Homomorphisms . . . 13
3.7 Boolean Abstraction . . . 14
3.8 Sign Abstraction . . . 14

4 Abstracting Differences **14**
4.1 Tuple Algebras . . . 15
4.2 Homomorphic Abstractions . . . 15
4.3 The $\Sigma_{pos\text{-}arith}$-Structure Δ_3 . . . 16
4.4 The $\Sigma_{pos\text{-}arith}$-Structure Δ_6 . . . 16

5 First-Order Logic **17**
5.1 First-Order Expressions . . . 17
5.2 First-Order Formulas . . . 18
5.3 First-Order Definitions . . . 20
5.4 Difference Abstractions . . . 20

6 Equation Systems **21**
6.1 Arithmetic Equations . . . 21
6.2 Polynomial Equations . . . 21
6.3 Linear Equations . . . 22
6.4 Simple $h_{\mathbb{B}}$-Mixed Systems . . . 22

7 Abstract Interpretation **23**
7.1 John's Theorem . . . 23
7.2 Exact Formulas . . . 25
7.3 Exact Boolean Abstraction . . . 25

8 First-Order Tuple Logic **26**
8.1 Syntax and Semantics . . . 26
8.2 From Standard to Tuple Logic . . . 27
8.3 From Tuple to Standard Logic . . . 27
8.4 Commutation Property . . . 28
8.5 First-Order Definitions . . . 29
8.6 Defining Function Applications . . . 29

9 Characterizing Difference Abstractions **30**
9.1 Exact Difference Abstraction for Δ_3 . . . 30
9.2 Exact Difference Abstraction for Δ_6 . . . 34

10 Computing Difference Abstractions with Constraints **37**
10.1 Problem of Exact Abstraction with Constraints . . . 37
10.2 Mixed Structures: $\mathbb{B} \cup \Delta_3$ and $\mathbb{B} \cup \Delta_6$ . . . 38
10.3 Defining k-ary Function Applications in First-Order Logic . . . 39
10.4 Exact Difference Abstraction with Constraints for Δ_3 . . . 39
10.5 Exact Difference Abstraction with Constraints for Δ_6 . . . 42

11 Overapproximation Heuristics with Minimal Support Consequences **45**

12 Implementation and Experimentation **46**
12.1 Implementation . . . 46
12.2 Application to Change Prediction of Reaction Networks . . . 47
12.3 Counter Example for the Minimal Support Heuristics . . . 47

13 First-Order Definitions of Function Application **49**

14 Conclusion **52**

Appendix A Proofs for Section 8 (First-Order Tuple Logic) **55**

2. Related Work

We discuss how to test whether an abstraction solution is an difference abstraction of a concrete solution by using linear programming, applications to change prediction in systems biology, and the releationship to flux balance analysis

2.1. Linear Programming

We first discuss the generate and test algorithm for solving our problem based on existing results for linear equations systems and linear programming.

Given an abstract difference $d \in \Delta_3$ and two variables y, z we define a formula saying that the abstraction of the concrete difference denoted by (y, z) is equal to d:

$$abs_d(y,z) = \begin{cases} y < z & \text{if } d = \triangle \\ x \eqsim y & \text{if } d = \approx \\ z < y & \text{if } d = \triangledown \end{cases}$$

Let $\mathbf{x} = x_1 \dots x_m$, $\mathbf{y} = y_1 \dots y_m$ and $\mathbf{z} = z_1 \dots z_m$ be sequences of distinct variables. Given an assignment of the variable in $\mathbf{x}$ to abstract differences $\beta : \{\mathbf{x}\} \to \Delta_3$ we define a formula $abs_\beta(\mathbf{y}, \mathbf{z})$ as follows:

$$abs_\beta(\mathbf{y}, \mathbf{z}) = \bigwedge_{i=1}^{m} abs_{\beta(x_i)}(y_i, z_i)$$

The assignment β then is the difference abstraction of some pair of solutions in $sol^{\mathbb{R}_+}(E(\mathbf{x}))$ if and only if the following formula is satisfiable over $\mathbb{R}_+$.

$$E(\mathbf{y}) \wedge E(\mathbf{z}) \wedge abs_\beta(\mathbf{y}, \mathbf{z})$$

Lemma 1. *The satisfiability over $\mathbb{R}_+$ of systems of homogeneous linear equation and strict linear inequations without constants can be decided in polynomial time.*

Proof A strict linear inequation without constants $x < y$ is equivalent over $\mathbb{R}_+$ to $\exists z.\ x + z \eqsim y \wedge z \neq 0$. Therefore, it is sufficient to consider the satisfiability over $\mathbb{R}_+$ of systems of homogeneous linear equations $E(\mathbf{x})$ and nonzero equation $\mathbf{x}' \neq 0$. The solution set $sol^{\mathbb{R}_+}(E(\mathbf{x}))$ is a cone (while for more general linear programs it could be more general polytopes). The elements of a cone can be multiplied by positive reals without leaving the cone. Therefore, the nonzero equation $\mathbf{x}' \neq 0$ can be rewritten to $\mathbf{x}' \geq 1$ without affecting the satisfiability of the formula. This rewriting eliminates the strict inequations without constants at the cost of introducing nonstrict inequations with constants. The result is a linear program. Instead of interpretation over $\mathbb{R}_+$ we can add inequations $\mathbf{x} \geq 0 \wedge \mathbf{x}' \geq 0$ and change to an interpretation over $\mathbb{R}$. It well known that the satisfiability of linear programs over $\mathbb{R}$ can be tested in polynomial time [17]. □

So by a naive enumerate and test algorithm, we can compute in the case of Δ_3 the set of all difference abstractions for $sol^{\Delta_3}(E(\mathbf{x}))$ in time $O(3^m poly(|E(\mathbf{x})|))$. For Δ_6, the analogous argument yields $O(6^m poly(|E(\mathbf{x})|))$.

2.2. Change Prediction in Systems Biology

Reaction networks [5, 6, 7, 8] are widely used in systems biology to model the dynamics of biological systems, so that their behaviour can be simulated or analysed. We are interested in change predictions for reaction networks with partial kinetic information [9, 10, 11]. The steady state semantics of such networks yields a system of linear equations, and a set of nonlinear constraints about the differences abstraction in Δ_6 of solutions of the linear system.

A simple example of reaction network with partial formal kinetic information is given in Figure 3; the linear equation system and the nonlinear constraints on its difference abstraction are given in Figure 4. The networks has five species A, B, C, D, E and nine reactions 1, ..., 6, in_A, out_B, out_C. As with the graphical notation for Petri nets, the species are nodes drawn as circles and the reactions nodes drawn as boxes. The colors of the species indicate their biological role, but do not contribute to the semantics. Metabolites are drawn in yellow and enzymes in brown circles. Reaction in_A is an inflow of the metabolite A, while reactions out_B and out_C are outflows of the metabolites B and C respectively. The inflows are controlled externally, while the

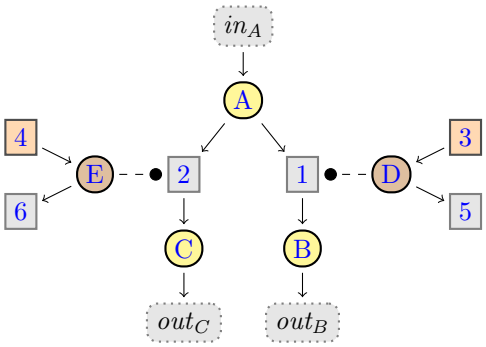


Figure 3: An example of reaction network with partial kinetic information.

Linear equation system over $\mathbb{R}_+$:	Nonlinear constraints over Δ_6:
$in_A = r_1 + r_2$	$r_1 = \mathrm{A} * \mathrm{D}$
$r_1 = out_B$	$r_2 = \mathrm{A} * \mathrm{E}$
$r_2 = out_C$	$r_3 \in \{\Downarrow, \sim\}$
$r_4 = r_6$	$r_4 \in \{\Downarrow, \sim\}$
$r_3 = r_5$	$r_5 = \mathrm{D}$
	$r_6 = \mathrm{E}$
	$in_A \in \{\uparrow, \downarrow, \sim\}$
	$out_B = \mathrm{B}$
	$out_C = \mathrm{C}$

Figure 4: Steady state semantics.

outflows are controlled internally in the system. Reactions 3 and 4 correspond respectively to the gene expression of the enzymes D and E. These reactions may be knocked out, modeling a gene knockout. The knockouts are changes that are controlled externally similarly to the change of inflow in_A. None of the other reactions can be changed externally. Reaction 6 degrades the enzyme E and reaction 5 degrades the enzyme D. Reaction 1 transforms its metabolic substrate A, into its metabolic product B, while being activated and accelerated by the enzyme D. Symmetrically, the reaction 2 transforms its metabolic substrate A, into its metabolic product C, while being activated and accelerated by the enzyme E.

In a steady state of the reaction network, the concentrations of all its species become stable. For each species we have a linear equation, that states that the rate of its production is equal to the rate of its consumption. For species A, for instance, this is the following equation over $\mathbb{R}_+$, since species A is produced by inflow of A at rate in_A, while it is consumed by reaction 1 at rate r_1 and reaction 2 at rate r_2:

$$in_A = r_1 + r_2$$

The network has partial kinetic information: we know the enzymes (and inhibitors) of the reactions, but not necessarily their precise kinetics. For instance, the precise kinetics of reaction 1 is unknown. But since D is an enzyme of reaction 1 it follows that the rate r_1 is zero if the concentration of species D is zero, and that r_1 increases if the concentration of species D increases. Furthermore, since A is a substrate of reaction 1 it follows that r_1 is zero if the concentration of species A is zero, and that r_1 increases if the concentration of species A increases. This means that the following nonlinear constraint holds after difference abstraction to Δ_6:

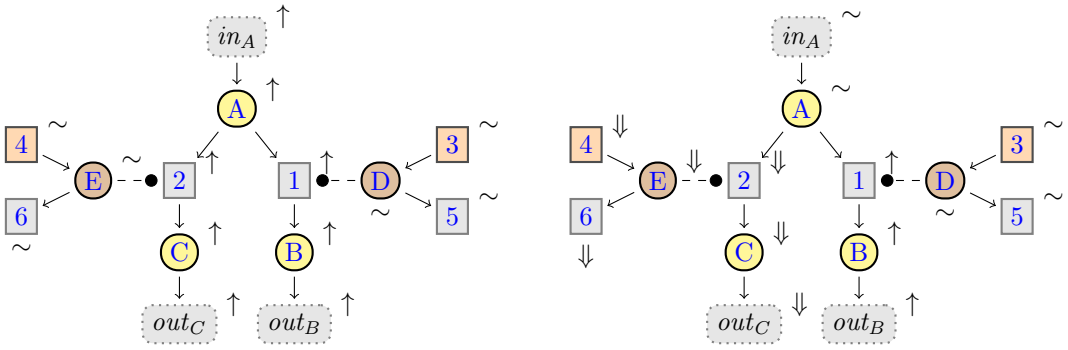
$$r_1 = \mathrm{A} * \mathrm{D}$$

For reactions 3 and 4 which may be knocked out, we have the following constraints about their difference abstraction to Δ_6:

$$r_3 \in \{\Downarrow, \sim\}$$
$$r_4 \in \{\Downarrow, \sim\}$$

So the rates of these reactions may either decrease to zero or remain unchanged but different from zero. The inflow of A may be increased, decreased, or keep unchanged while active:

$$in_A \in \{\uparrow, \downarrow, \sim\}$$



(a) Increase inflow of A.

(b) Knock down gene expression 4 of E.

Figure 5: Two changes leading to an increase of the outflow of B.

A typical question for change prediction is which changes can be applied to the example network in order to increase the outflow of B, that is $out_B = \uparrow$. The set of potential changes is to increase or decrease the inflow of A, i.e., $in_A \in \{\uparrow, \downarrow\}$, or to shut down reactions 3 or 4, that is $r_3 = \Downarrow$ or $r_4 = \Downarrow$. The two single-change predictions that answer this question are:

1. increase the inflow of A, that is $in_A = \uparrow$, or
2. knock down reaction 4, i.e., $r_4 = \Downarrow$ and thus of the gene expression of enzyme E.

These two predictions correspond to the two abstract solutions over Δ_6 in Figure 5. The first solution in Figure 5a motivates the prediction of an increase of in_A and the second solution in Figure 5b the prediction of a knock down of reaction 4, the gene expression producing enzyme E.

Both abstract solutions are difference abstractions over Δ_6 of real positive solutions of the linear equation system in Figure 4, so that these difference abstractions do also satisfy the nonlinear constraints over Δ_6 given there. We can find both predictions by applying John's overapproximation algorithm [9, 11], that is by computing the Δ_6 solutions of linear equation system and the nonlinear constraints. This can be done in practice by using finite domain constraint programming.

Qualitative reasoning can also be performed manually for this simple example. For increasing out_B be must increase the concentration of B and thus the rate of reaction 1. For this, we must either increase the concentration of enzyme D which is impossible by the available changes, or increase the concentration of A. The latter requires to either increase the inflow of A, leading to the first abstract solution in Figure 5a, or else decrease the rate of reaction 2. This is possible by decreasing the concentration of E by knocking out the reaction 4, the gene expression producing this enzyme. This yields the second abstract solution in Figure 5b.

John's algorithm does not lead to any overapproximation for this example. The main reason is that the graph of the reaction network in Figure 3 is acyclic, even in the absence of partial kinetic information. The situation changes for reaction networks with cycles. The simplest counterexample is the simple loop in Figure 6. The linear equation system of this network is exactly the system from the introduction, where John's algorithm predicts unjustified changes.

This network has two species A and B and four reactions: an inflow of A, an outflow of B, a reaction 1 transforming A to B and a inverse reaction 2. So each molecule A that flows

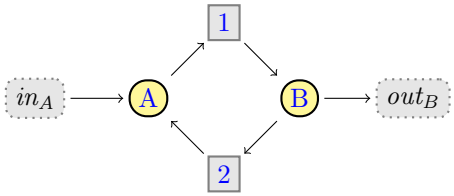


Figure 6: A reaction network with a simple loop.

$$r_1 = in_A + r_2$$
$$r_1 = out_B + r_2$$

Figure 7: Linear equation system of the reaction network in Figure 6 with the simple loop.

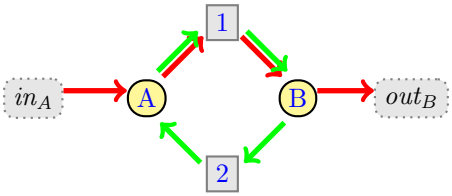


Figure 8: Elementary flux modes of the simple loop network.

into the system may loop for a while, changing to B and back, before eventually outflowing as B. In a steady state, the rate of the inflow of A is equal to the rate of the outflow of B. The argument can be understood more easily, when considering the elementary flux modes [15, 18] of this reaction network, which are shown graphically in Figure 8 in red and respectively in green. An elementary flux mode is a linear combination of reactions that can become steady. The simple loop network has two elementary flux modes, corresponding to the linear combination of reactions $1in_A + 1r_1 + 1out_B$ using red edges and the linear combination of reactions $1r_1 + 1r_2$ with green edges respectively.

While some molecules may loop with the green edges, transforming A's to B's and back, all inflowing A's must follow the red edges and thus eventually outflow as B. From an algebraic perspective, the elementary flux modes of a reaction network correspond exactly to the elementary modes of its stoichiometry matrix [14]. An elementary mode of a matrix A is a positive integer solution of $A\mathbf{x} = 0$ where $\mathbf{x}$ a sequence of fresh variables. Furthermore, we require that the solution has minimal support, meaning that a minimal subset of the variables is assigned to a nonzero value, and that the solution is normalized (so it cannot be devided by any natural number strictetly greater than 1). Note that the stoichiometry matrix of the simple loop network was given within the linear equation system (1) at the beginning of the introduction. Here, the vector of variables $\mathbf{x}$ corresponds to the order on reactions adopted by the stoichiometry matrix, so it can be chosen as (in_A, out_B, r_1, r_2). Its elementary modes are $(1, 1, 1, 0)$ and $(0, 0, 1, 1)$.

The difference abstraction of any two concrete solutions of the linear equations of the simple loop network must satisfy $in_A = out_B$. The corresponding abstract solution over Δ_6 satisfying $in_A = \uparrow$ and $out_B = \uparrow$ is illustrated in Figure 9. However, the alternative variable assignment to Δ_6 in Figure 10 is also a solution of all linear equations when interpreted over Δ_6, while not being justified by any pair of concrete solutions over $\mathbb{R}_+$.

The intuitive reason for this failure is that John's overapproximation algorithm performs only local reasoning, considering one linear equation at a time, i.e., one species of the reaction network. In this manner, it cannot see, that $X = Y$ is a logical consequence of the linear equation system of the network over $\mathbb{R}_+$, while it is not a consequence over Δ_6. So what we are searching is way

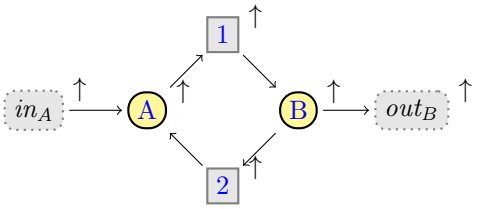


Figure 9: A justified abstract solution.

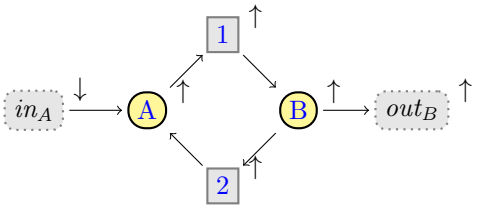


Figure 10: An unjustified abstract solution.

to reason globally with all species of a reaction network at a time. For this we have to take into account all linear combinations of the equations of the system.

2.3. Flux Balance Analysis

Flux balance analysis [3, 4] can be used to predict the effect of influx changes of metabolic networks at steady state. Such predictions can be based on reasoning with linear equation systems that describe the rates of the reactions in a steady state of the metabolic network, by using Gaussian elimination, elementary flux modes (EFMs) [19], or optimisation methods [20, 21]. Most importantly, precise quantitative kinetic information is not required in contrast to classical mathematical analysis methods for reaction networks. In fact, even when the kinetic functions associated to chemical reactions are known, the values of the rate constants are most often missing, since it is difficult to measure them experimentally in the precise state of the regulation of the metabolic network at the time point of interest.

3. Preliminaries

We present standard notion of sets, partial and total functions, relations. We then recall Σ-algebras, Σ-structures, and homomphisms that we illustrate at the boolean abstraction.

3.1. Sets and Tuples

We start with the usual notation for sets. Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}_+$ the set of positive real numbers, both including 0. For any set A and $n \in \mathbb{N}$, the set of n-tuples of elements in A is denoted by A^n. The i-th projection function on n-tuples of elements in A, where $1 \leq i \leq n$ is the function $\pi_i : A^n \to A$ such that $\pi_i(a_1, \ldots, a_n) = a_i$ for all $a_1, \ldots, a_n \in A$. If A is finite the number of elements of A is denoted by $|A|$.

3.2. Sequences

The set of all sequences $A^* = \cup_{n=0}^{\infty} A^n$ is the union of sets of tuples of size n where $n \geq 0$. We often write $a_1 \ldots a_n \in A^*$ instead of the tuple $(a_1, \ldots, a_n) \in A^n$. The concatenation of two sequences $\mathbf{a}, \mathbf{a}' \in A^*$ is denoted by the juxtaposition $\mathbf{a}\mathbf{a}' \in \mathcal{V}^*$. Given a function $\mu : A \to B$ and a sequence $\mathbf{a} = a_1 \ldots a_n \in A^*$ we can map μ to all elements of the sequence defining:

$$map_\mu(\mathbf{a}) = \mu(a_1) \ldots \mu(a_m)$$

For any sequence $\nu = (\nu_1, \ldots, \nu_k)$ of functions $\nu_i : A \to B$ where $1 \leq i \leq k$ and any sequence $\mathbf{a} \in A^*$ we define a sequence $map^k_\nu(\mathbf{a}) \in B^*$ by:

$$map^k_\nu(\mathbf{a}) = map_{\nu_1}(\mathbf{a}) \ldots map_{\nu_k}(\mathbf{a})$$

3.3. Functions and Relations

We continue with notations for total and partial functions. A partial function is a relation $f \subseteq A \times B$ such that for all $a \in A$ there exists at most one $b \in B$ such that $(a, b) \in f$. In this case, we write $f(a) = b$. The domain of the partial function is $dom(f) = \{a \mid \exists b \in B. f(a) = b\}$ and its range $ran(f) = \{b \mid \exists a \in A.\ f(a) = b\}$. A total function $f : A \to B$ is a partial function $f \subseteq A \times B$ such that $dom(f) = A$.

Given a total function $f : A \to B$ and a partial function $g \subseteq B \times C$ such that $ran(f) \subseteq dom(g)$ we define the function composition as the total function $g \circ f : A \to C$ such that $(g \circ f)(x) = g(f(x))$ for all $x \in A$.

Any $(n+1)$-ary relation $R \subseteq A_1 \times \ldots \times A_n \times B$ can be reinterpreted as an n-ary set valued function $R : (A_1 \times \ldots \times A_n) \to 2^B$ which satisfies for all $a_1 \in A_1$, ..., $a_n \in A_n$:

$$R(a_1, \ldots, a_n) = \{b \in B \mid (a_1, \ldots, a_n, b) \in R\}$$

Furthermore, if $F \subseteq \{f \mid f : A \to B\}$ is a subset of total functions and $g \subseteq B \times C$ a partial function, then we define the set value composition:

$$g \circ F = \{g \circ f : A \to C \mid f \in F,\ ran(f) \subseteq dom(g)\}$$

Note that $g \circ f$ is defined only if $ran(f) \subseteq dom(g)$, so functions $f \in F$ violating this condition will be ignored all over in the composition $g \circ F$. This is since we want all the functions in $g \circ F$ to be total even if g is partial.

3.4. Σ-Algebras

We next recall the notions of Σ-algebras, Σ-structures, and homomorphism between Σ-structures. These classical notions of universal algebra will be fundamental to our algebraic generalization of difference abstractions to the notion of Σ-abstractions in Definition 48.

Let $\Sigma = \cup_{n \geq 0} F^{(n)} \uplus C$ be a ranked signature. The elements of $f \in F^{(n)}$ are called the n-ary function symbols of Σ and the elements $c \in C$ its constants.

Definition 2. *A Σ-algebra $S = (dom(S), .^S)$ consists of a set $dom(S)$ and an interpretation $.^S$ such that $c^S \in dom(S)$ for all $c \in C$, and $f^S : dom(S)^n \to dom(S)$ for all $f \in F^{(n)}$ and $n \in \mathbb{N}$.*

Example 3. *The set of positive real numbers $\mathbb{R}_+$ can be turned into a $\Sigma_{pos\text{-}arith}$-algebra with domain $dom(\mathbb{R}_+) = \mathbb{R}_+$, by interpreting the operator $+$ as the addition operation of positive real numbers $+^{\mathbb{R}_+}$, the multiplication operator $*$ as the multiplication operation on positive real numbers $*^{\mathbb{R}_+}$. The constants are interpreted by themselves, i.e., $0^{\mathbb{R}_+} = 0$ and $1^{\mathbb{R}_+} = 1$. We will deliberatly confuse the set of postive real numbers $\mathbb{R}_+$ with the Σ-algebra $(dom(\mathbb{R}_+), .^{\mathbb{R}_+})$.*

Example 4. *The set of Booleans $\mathbb{B} = \{0, 1\} \subseteq \mathbb{R}_+$ can be turned into a $\Sigma_{pos\text{-}arith}$-algebra with domain $dom(\mathbb{B}) = \mathbb{B}$ by interpreting $+^{\mathbb{B}}$ as disjunction $\vee$, $*^{\mathbb{B}}$ as conjunction $\wedge$, and the constants by themselves $0^{\mathbb{B}}$ as 0 standing for* false*, and $1^{\mathbb{B}}$ as 1 standing for* true*. We will deliberatly confuse the set $\mathbb{B}$ with the Σ-algebra $(dom(\mathbb{B}), .^{\mathbb{B}})$. set of booleans.*

3.5. Σ-Structures

We can reuse the same signature Σ for defining relational Σ-structures. For this, we consider any symbol of $F^{(n)}$ as an $n+1$-ary relation symbol.

Definition 5. *A Σ-structure $\Delta = (dom(\Delta), .^\Delta)$ consists of a set $dom(\Delta)$ and an interpretation $.^\Delta$ such that $c^\Delta \in dom(\Delta)$ for all $c \in C$ and $f^\Delta \subseteq dom(\Delta)^{n+1}$ for all $f \in F^{(n)}$ and $n \in \mathbb{N}$.*

Any Σ-algebra is also a Σ-structure since any n-ary function is by definition an $n+1$-ary relation. Note also that symbols in $F^{(0)}$ are interpreted as monadic relations in Σ-structures, i.e., as subsets of the domain, in contrast to constants in C that are interpreted as elements of the domain.

Example 6. *The set of sigs $\mathbb{S} = \{-1, 0, 1\}$ can be turned into a $\Sigma_{pos\text{-}arith}$-structure $(dom(\mathbb{S}), .^{\mathbb{S}})$ with domain $dom(\mathbb{S}) = \mathbb{S}$. The interpretation of the operators as ternary relations, that we can consider as binary set valued functions, satisfies for all signs $s \in \mathbb{S}$:*

$$\begin{array}{ll} -1 +^{\mathbb{S}} 1 = 1 +^{\mathbb{S}} -1 = \{-1, 0, 1\} & -1 *^{\mathbb{S}} 1 = 1 *^{\mathbb{S}} -1 = \{-1\} \\ -1 +^{\mathbb{S}} -1 = \{-1\} & -1 *^{\mathbb{S}} -1 = \{1\} \\ 1 +^{\mathbb{S}} -1 = \{-1\} & 1 *^{\mathbb{S}} 1 = \{1\} \\ 0 +^{\mathbb{S}} s = s +^{\mathbb{S}} 0 = \{s\} & 0 *^{\mathbb{S}} s = s *^{\mathbb{S}} 0 = \{0\} \end{array}$$

Note that the operator $+^{\mathbb{S}}$ is a nonfunctional relation, since the sum of a negative with a positive number may have any sign. Therefore the relational structure of signs does not form an algebra. In contrast, the operation $*^{\mathbb{S}}$ is still a total function. So the restriction of the sign structure to multiplication is an algebra.

It is sometimes useful to add the elements of the domain of a Σ-structure A to the constants. Therefore, we define the extended signature:

$$\Sigma[dom(A)] = \Sigma \uplus dom(A)$$

The Σ-structure A can be lifted to a $\Sigma[dom(A)]$-structure by interpreting the new constants by themselves, i.e., $a^A = a$ for all $a \in dom(A)$, and all symbols in Σ as before.

3.6. Homomorphisms

We recall the notion of homomorphisms between Σ-structures from universal algebra, which generalizes on the notion of homomorphisms between Σ-algebras.

Definition 7. *A homomorphism between two Σ-structures S and Δ is a function $h : dom(S) \to dom(\Delta)$ such that for $c \in C$, $n \in \mathbb{N}$, $f \in F^{(n)}$, and $s_1, \ldots, s_{n+1} \in dom(S)$:*

1. *$h(c^S) = c^\Delta$, and*
2. *if $(s_1, \ldots, s_{n+1}) \in f^S$ then $(h(s_1), \ldots, h(s_{n+1})) \in f^\Delta$.*

If S and Δ are Σ-algebras, then the second condition for a homomorphism h from S to Δ is equivalent to:

$$h(f^S(s_1, \ldots, s_n)) = f^\Delta(h(s_1), \ldots, h(s_n))$$

When reinterpreting relations as set-valued functions, the second condition of homomophisms between general Σ-structures can be rewritten equivalently to:

$$h(f^S(s_1, \ldots, s_n)) \subseteq f^\Delta(h(s_1), \ldots, h(s_n))$$

3.7. Boolean Abstraction

The boolean abstraction of positive real numbers is our prime example for a homomorphism between Σ-algebras. It has the signature of positive arithmetic operators $\Sigma = \Sigma_{pos\text{-}arith}$ with two binary function symbols and two constants:

$$\Sigma_{pos\text{-}arith} = F^{(2)}_{pos\text{-}arith} \uplus C_{pos\text{-}arith} \qquad \text{where} \qquad \begin{cases} F^{(2)}_{pos\text{-}arith} = \{+, *\} \\ C_{pos\text{-}arith} = \{0, 1\} \end{cases}$$

For all Σ-algebras considered, the operators $+^S$ and $*^S$ are associative and commutative, with neutral element 0^S and 1^S respectively.

Definition 8. *We can abstract positive real numbers into booleans by defining a function $h_{\mathbb{B}} : \mathbb{R}_+ \to \mathbb{B}$ such that for all positive reals $r \in \mathbb{R}_+$:*

$$h_{\mathbb{B}}(r) = \begin{cases} 0 & \text{if } r = 0 \\ 1 & \text{if } r > 0 \end{cases}$$

Lemma 9. *The function $h_{\mathbb{B}} : \mathbb{R}_+ \to \mathbb{B}$ is a homomorphism between $\Sigma_{pos\text{-}arith}$-algebras.*

Proof For all $r, r' \in \mathbb{R}_+$ we have:

$$\begin{array}{rcccccl} h_{\mathbb{B}}(r +^{\mathbb{R}_+} r') = 1 & \Leftrightarrow & r +^{\mathbb{R}_+} r' \neq 0 & \Leftrightarrow & r \neq 0 \vee r' \neq 0 & \Leftrightarrow & h_{\mathbb{B}}(r) = 1 \vee h_{\mathbb{B}}(r') = 1 \\ h_{\mathbb{B}}(r *^{\mathbb{R}_+} r') = 1 & \Leftrightarrow & r *^{\mathbb{R}_+} r' \neq 0 & \Leftrightarrow & r \neq 0 \wedge r' \neq 0 & \Leftrightarrow & h_{\mathbb{B}}(r) = 1 \wedge h_{\mathbb{B}}(r') = 1 \end{array}$$

Hence $h_{\mathbb{B}}(r +^{\mathbb{R}_+} r') = h_{\mathbb{B}}(r) +^{\mathbb{B}} h_{\mathbb{B}}(r')$ and $h_{\mathbb{B}}(r *^{\mathbb{R}_+} r') = h_{\mathbb{B}}(r) *^{\mathbb{B}} h_{\mathbb{B}}(r')$. Finally, for both constants $c \in C$ we have that $h_{\mathbb{B}}(c^{\mathbb{R}_+}) = h_{\mathbb{B}}(c) = c = c^{\mathbb{B}}$. □

3.8. Sign Abstraction

As an example for a homomorphism between relational structures rather than only algebras, we consider the function $h_{\mathbb{S}} : \mathbb{R} \to \mathbb{S}$ that abstract any real number $r \in \mathbb{R}$ to its sign:

$$h_{\mathbb{S}}(r) = \begin{cases} -1 & \text{if } r < 0 \\ 0 & \text{if } r = 0 \\ 1 & \text{if } r > 0 \end{cases}$$

The set of all real numbers $\mathbb{R}$ can be turned into a $\Sigma_{pos\text{-}arith}$-algebra, in the same way then we did for $\mathbb{R}_+$. The sign abstraction $h_{\mathbb{S}}$ is a homomorphism of this algebra to the structure of structure of signs.

But in the present article, we will focus on the boolean abstraction for positive real numbers, while avoiding the more general sign abstraction, that also applies to negative numbers.

4. Abstracting Differences

Concrete differences are pairs of positive real numbers in $\mathbb{R}_+^2$. We show how to abstract concrete differences into abstract differences by using homomorphisms induced by partitions of $\mathbb{R}_+^2$ or more general tuple algebras. This permits us to define homomorphisms that abstract concrete difference in $\mathbb{R}_+^2$ to abstract differences in the finite $\Sigma_{pos\text{-}arith}$-structures Δ_3 and Δ_6 from [9] respectively.

4.1. Tuple Algebras

We start with a signature $\Sigma = F^{(2)} \cup C$ of constants and binary function symbols. For any Σ-algebra S and natural number $n \in \mathbb{N}$ we define the Σ-algebra of n-tuples $S^n = (dom(S)^n, .^{S^n})$ such that for all $s_1, \ldots, s_n, s'_1, \ldots, s'_n \in dom(S)$ and $\odot \in F^{(2)}$:

$$(s_1, \ldots, s_n) \odot^{S^n} (s'_1, \ldots, s'_n) = (s_1 \odot^S s'_1, \ldots, s_n \odot^S s'_n)$$

The constants $c \in C$ are interpreted as $c^{S^n} = (c^S, \ldots, c^S)$. Note that if 0^S is the neutral element of $+^S$, then 0^{S^n} is the also the neutral element of $+^{S^n}$. In analogy, if 1^S is the neutral element of $*^S$ then 1^{S^n} is also abthe neutral element of $*^{S^n}$. Furthermore, the associativity and commutativity of $+^{S^n}$ and $*^{S^n}$ inherit from $+^S$ and $*^S$ respectively.

Note that we deliberately confuse the set $\mathbb{R}^2_+$ with the Σ-algebra $(\mathbb{R}^2_+, .^{\mathbb{R}^2_+})$ with our notation. Given this, it follows from the above, that the algebra $\mathbb{R}^2_+$ has the neutral element $(0, 0)$ for $+^{\mathbb{R}^2_+}$ and the neutral element $(1, 1)$ for $*^{\mathbb{R}^2_+}$, and that these operations are associative and commutative.

For any function $h : A \to B$ and $n \in \mathbb{N}$ we define the function $h^n : A^n \to B^n$ such that $h^n(a_1, \ldots, a_n) = (h(a_1), \ldots, h(a_n))$ for all $a_1, \ldots, a_n \in A$.

Lemma 10. *Let S and Δ be Σ-structures. For any homomorphism h from S to Δ and $n \in \mathbb{N}$, the function h^n is a homomorphism between Σ-structures from S^n to Δ^n.*

Proof Let $\odot \in F^{(2)}$ and $t = (s_1, \ldots, s_n), t' = (s'_1, \ldots, s'_n) \in dom(S)^n$. Then we have:

$$\begin{array}{lll}
h^n(t \odot^{S^n} t') & = (h(s_1 \odot^S s'_1), \ldots, h(s_n \odot^S s'_n)) & \text{definitions of } h^n \text{ and } S^n \\
& = (h(s_1) \odot^\Delta h(s'_1), \ldots, h(s_n) \odot^\Delta h(s'_n)) & \text{since } h \text{ is a homomorphism} \\
& = (h(s_1), \ldots, h(s_n)) \odot^{\Delta^n} (h(s'_1), \ldots, h(s'_n)) & \text{definition of } \Delta^n \\
& = h^n(t) \odot^{\Delta^n} h^n(t') & \text{definition of } h^n
\end{array}$$

Finally, for any constant $c \in C$ we have:

$$\begin{array}{lcll}
h^n(c^{S^n}) & = & h^n(c^S, \ldots, c^S) & \text{definition } S^n \\
& = & (h(c^S), \ldots, h(c^S)) & \text{definition } h^n \\
& = & (c^\Delta, \ldots, c^\Delta) & \text{since } h \text{ is a homomorphism} \\
& = & c^{\Delta^n} & \text{definition of } \Delta^n
\end{array}$$

□

4.2. Homomorphic Abstractions

We now show how to abstract the n-tuples homomorphically and in a generic manner by partitionning the set all all n-tuples under consideration.

Let S be a Σ-structure, where $\Sigma = F^{(2)} \cup C$ continues to be a binary signature. The elements of S^n are called the *concrete differences*.

A generic manner to define a homomorphic abstraction on concrete difference in S^n is to start with some function $h : dom(S)^n \to \Delta$ to some arbitrary set Δ. The elements of Δ are then called *abstract differences*. The function h says how to map a concrete difference to some abstract differences. In other words, it partitions the set of concrete differences into equivalence classes $h^{-1}(\delta)$ where $\delta \in \Delta$.

For any partionning $h : dom(S)^n \to \Delta$, there is a unique manner to define an interpretation $.^\Delta$ such that $(dom(\Delta), .^\Delta)$ becomes $\Sigma_{pos\text{-}arith}$-structure with domain $dom(\Delta) = \Delta$ and h a homomorphism. For any constant $c \in C$ we have to define

$$c^\Delta = h(c^{S^n})$$

and for any binary symbol $\odot \in F^{(2)}$ we have to define a ternary relation $\odot^{\Delta}$, which seen as set-valued function $\odot^{\Delta} : \Delta \times \Delta \to 2^{\Delta}$ must satisfy for all abstract values $\delta_1, \delta_2 \in \Delta$:

$$\delta_1 \odot^{\Delta} \delta_2 = \{h(r_1^1 \odot^S r_2^1, \ldots, r_1^n \odot^S r_2^n) \mid h(r_1^1, \ldots, r_1^n) = \delta_1,\ h(r_2^1, \ldots, r_2^n) = \delta_2\}$$

Lemma 11. *Any partition $h : S^n \to \Delta$ is a homomorphism between $\Sigma_{pos\text{-}arith}$-structures with the interpretations $\odot^{\Delta}$ and c^{Δ} of elements of $\Sigma_{pos\text{-}arith}$ induced by h as defined above.*

Proof For any $p_1 = (r_1^1, \ldots, r_1^n)$, $p_2 = (r_2^1, \ldots, r_2^n) \in S^n$ the second condition for homomorphisms follows for all $\odot \in F^{(2)}_{pos\text{-}arith}$:

$$h(p_1 \odot^{S^n} p_2) \quad = h(r_1 \odot^{\mathbb{R}_+} r_2, r_1' \odot^{\mathbb{R}_+} r_2') \quad \in h(p_1) \odot^{\Delta} h(p_2)$$

Finally, for all constants $c \in C_{pos\text{-}arith}$ we have by definition that $h(c^{S^n}) = c^{\Delta}$. □

4.3. The $\Sigma_{pos\text{-}arith}$-Structure Δ_3

Our next objective is derive with the above construction the homormophic abstraction of concrete differences in the $\Sigma_{pos\text{-}arith}$-algebra $\mathbb{R}_+^2$ to the finite $\Sigma_{pos\text{-}arith}$-structure with domain $\Delta_3 = \{\bigtriangleup, \bigtriangledown, \approxeq\}$, where $\bigtriangleup$ stands for an increase, $\bigtriangledown$ for a decrease, and $\approxeq$ for a no-change. This abstract domain is well-known from qualitative reasoning (see e.g. [22]). In order to embed Δ_3 into $\mathbb{R}_+^2$, we define the symbols of Δ_3 as follows:

$$\bigtriangleup = (0,1), \qquad \bigtriangledown = (1,0), \qquad \approxeq = (0,0).$$

We then define a partition $h_{\Delta_3} : \mathbb{R}_+^2 \to \Delta_3$ such that for all $r, r' \in \mathbb{R}_+$:

$$h_{\Delta_3}(r, r') = \begin{cases} \bigtriangleup & \text{if } r < r' \\ \bigtriangledown & \text{if } r > r' \\ \approxeq & \text{if } r = r' \end{cases}$$

This difference abstraction is closely related to the sign abstraction via substraction. In order to see this, let be the finite function $h^{\mathbb{S}}_{\Delta_3} : \mathbb{S} \to \Delta_3$ mapping any sing to the corresponding abstract difference in Δ_3:

$$\begin{array}{lcl} h^{\mathbb{S}}_{\Delta_3}(-1) & = & \bigtriangledown \\ h^{\mathbb{S}}_{\Delta_3}(0) & = & \approxeq \\ h^{\mathbb{S}}_{\Delta_3}(1) & = & \bigtriangleup \end{array}$$

We then have for any concrete difference $(r, r') \in \mathbb{R}_+^2$ that:

$$h_{\Delta_3}(r, r') = h^{\mathbb{S}}_{\Delta_3}(h_{\mathbb{S}}(r - r'))$$

Given that h_{Δ_3} is a partition of $\mathbb{R}_+^2$, it turns the set Δ_3 into a $\Sigma_{pos\text{-}arith}$-structure $(dom(\Delta_3), .^{\Delta_3})$ with $dom(\Delta_3) = \Delta_3$, such that h_{Δ_3} becomes a homomorphism as shown by Lemma 11. The definition of the operators of Δ_3 was given before this Lemma and is summarized in the tables in Figure 11.

4.4. The $\Sigma_{pos\text{-}arith}$-Structure Δ_6

We next recall the abstraction of concrete differences to the finite Σ-structure with domain $\Delta_6 = \{\uparrow, \downarrow, \sim, \Uparrow, \Downarrow, \approx\}$ that was introduced for gene knockout prediction in [9]. In order to embed Δ_6 into $\mathbb{R}_+^2$, we define the symbols of Δ_5 as follows:

$$\begin{array}{lcllcllcl} \uparrow & = & (1,2), & \downarrow & = & (2,1), & \sim & = & (1,1), \\ \Uparrow & = & (0,2), & \Downarrow & = & (2,0), & \approx & = & (0,0). \end{array}$$

δ	δ'	$\delta +^{\Delta_3} \delta'$	$\delta *^{\Delta_3} \delta'$
$\triangle$	$\triangle$	$\{\triangle\}$	$\{\triangle\}$
$\triangle$	$\triangledown$	$\{\triangle, \approxeq, \triangledown\}$	$\{\triangle, \approxeq, \triangledown\}$
$\triangle$	$\approxeq$	$\{\triangle\}$	$\{\triangle, \approxeq\}$

δ	δ'	$\delta +^{\Delta_3} \delta'$	$\delta *^{\Delta_3} \delta'$
$\approxeq$	$\approxeq$	$\{\approxeq\}$	$\{\approxeq\}$
$\triangledown$	$\triangledown$	$\{\triangledown\}$	$\{\triangledown\}$
$\triangledown$	$\approxeq$	$\{\triangledown\}$	$\{\triangledown, \approxeq\}$

c	c^{Δ_3}
0	$\approxeq$
1	$\approxeq$

Figure 11: Interpretation of the $\Sigma_{pos\text{-}arith}$-structure Δ_3.

δ	δ'	$\delta +^{\Delta_6} \delta'$
$\uparrow$	$\uparrow$	$\{\uparrow\}$
$\uparrow$	$\downarrow$	$\{\uparrow, \sim, \downarrow\}$
$\uparrow$	$\sim$	$\{\uparrow\}$
$\uparrow$	$\Uparrow$	$\{\uparrow\}$
$\uparrow$	$\Downarrow$	$\{\uparrow, \downarrow, \sim\}$
$\uparrow$	$\approx$	$\{\uparrow\}$

δ	δ'	$\delta +^{\Delta_6} \delta'$
$\Uparrow$	$\downarrow$	$\{\uparrow, \sim, \downarrow\}$
$\Uparrow$	$\sim$	$\{\uparrow\}$
$\Uparrow$	$\Uparrow$	$\{\Uparrow\}$
$\Uparrow$	$\Downarrow$	$\{\uparrow, \sim, \downarrow\}$
$\Uparrow$	$\approx$	$\{\Uparrow\}$
$\Downarrow$	$\Downarrow$	$\{\Downarrow\}$

δ	δ'	$\delta +^{\Delta_6} \delta'$
$\sim$	$\sim$	$\{\sim\}$
$\sim$	$\approx$	$\{\sim\}$
$\sim$	$\downarrow$	$\{\downarrow\}$
$\sim$	$\Downarrow$	$\{\downarrow\}$
$\downarrow$	$\downarrow$	$\{\downarrow\}$
$\downarrow$	$\Downarrow$	$\{\downarrow\}$

δ	δ'	$\delta +^{\Delta_6} \delta'$
$\approx$	$\approx$	$\{\approx\}$
$\approx$	$\downarrow$	$\{\downarrow\}$
$\approx$	$\Downarrow$	$\{\Downarrow\}$

c	c^{Δ_6}
0	$\approx$
1	$\sim$

δ	δ'	$\delta *^{\Delta_6} \delta'$
$\uparrow$	$\uparrow$	$\{\uparrow\}$
$\uparrow$	$\downarrow$	$\{\uparrow, \sim, \downarrow\}$
$\uparrow$	$\sim$	$\{\uparrow\}$
$\uparrow$	$\Uparrow$	$\{\Uparrow\}$
$\uparrow$	$\Downarrow$	$\{\Downarrow\}$
$\uparrow$	$\approx$	$\{\approx\}$

δ	δ'	$\delta *^{\Delta_6} \delta'$
$\Uparrow$	$\uparrow$	$\{\Uparrow\}$
$\Uparrow$	$\sim$	$\{\Uparrow\}$
$\Uparrow$	$\Uparrow$	$\{\Uparrow\}$
$\Uparrow$	$\Downarrow$	$\{\approx\}$
$\Uparrow$	$\approx$	$\{\approx\}$
$\Downarrow$	$\Downarrow$	$\{\Downarrow\}$

δ	δ'	$\delta *^{\Delta_6} \delta'$
$\sim$	$\sim$	$\{\sim\}$
$\sim$	$\approx$	$\{\approx\}$
$\sim$	$\downarrow$	$\{\downarrow\}$
$\sim$	$\Downarrow$	$\{\Downarrow\}$
$\downarrow$	$\downarrow$	$\{\downarrow\}$
$\downarrow$	$\Downarrow$	$\{\Downarrow\}$

δ	δ'	$\delta *^{\Delta_6} \delta'$
$\approx$	$\approx$	$\{\approx\}$
$\approx$	$\downarrow$	$\{\Downarrow\}$
$\approx$	$\Downarrow$	$\{\Downarrow\}$

Figure 12: Interpretation of the $\Sigma_{pos\text{-}arith}$-structure Δ_6.

For defining this Σ-structure, we start with the function $h_{\Delta_6} : \mathbb{R}_+^2 \to \Delta_6$ such that for any two numbers $r, r' \in \mathbb{R}_+$:

$$h_{\Delta_6}(r, r') = \begin{cases} \uparrow & \text{if } 0 \neq r < r' \\ \downarrow & \text{if } r > r' \neq 0 \\ \sim & \text{if } r = r' \neq 0 \end{cases} \qquad h_{\Delta_6}(r, r') = \begin{cases} \Uparrow & \text{if } 0 = r < r' \\ \Downarrow & \text{if } r > r' = 0 \\ \approx & \text{if } r = r' = 0 \end{cases}$$

We define the ternary relation $+^{\Delta_6}$ as the relation that is symmetric in the first two arguments and satisfies the equations in Figure 12. The relation $*^{\Delta_6}$ is defined in the same style in Figure 12. The constants are interpreted as $0^{\Delta_6} = \approx$ and $1^{\Delta_6} = \sim$. By Lemma 11, $h_{\Delta_6} : \mathbb{R}_+^2 \to \Delta_6$ is a homomorphism.

5. First-Order Logic

We recall the standard first-order logic (FO) and show how to define difference abstractions of logical formulas.

5.1. First-Order Expressions

We fix a countably infinite set of variables $\mathcal{V}$. Typically, this set will be $\mathcal{V} = \mathbb{N}$.

Let $\Sigma = F^{(2)} \uplus C$ be a signature with binary operators only. This restriction is not essential but permits to ease the notation. The set of first-order expressions $\mathcal{E}_\Sigma$ is given by the abstract

For all $x \in V \subseteq \mathcal{V}$, $c \in C$, $\odot \in F^{(2)}$, and $e, e' \in \mathcal{E}_\Sigma$ such that $fv(e) \cup fv(e') \subseteq V$:

$$\begin{aligned}
[\![x]\!]^{\alpha,S} &= \{\alpha(x)\} \\
[\![c]\!]^{\alpha,S} &= \{c^S\} \\
[\![e \odot e']\!]^{\alpha,S} &= \cup\{s \odot^S s' \mid s \in [\![e]\!]^{\alpha,S}, s' \in [\![e']\!]^{\alpha,S}\}
\end{aligned}$$

Figure 13: Set-valued semantics $[\![e]\!]^{\alpha,S} \subseteq dom(S)$ of an expression $e \in \mathcal{E}_\Sigma$ of first-order logic, a Σ-structure S, and a variable assignment $\alpha : V \to dom(S)$.

syntax in Figure 13. Each expression $e \in \mathcal{E}_\Sigma$ is a term constructed from the constants in $c \in C$, the variables $x \in \mathcal{V}$, and the binary operators $\odot \in F^{(2)}$:

$$e, e' \in \mathcal{E}_\Sigma \quad ::= x \mid c \mid e \odot e' \quad \text{where } x \in \mathcal{V}, \odot \in F^{(2)}, \text{ and } c \in C$$

The set of free variables of an expression $fv(e)$ is the subset of variables that occurs in e. The semantics of expressions is set-valued, so any expression may denote a set of possible values. This is since the operations of relational structures may be relations, which are not always functions. For instance, consider the arithmetic signature $\Sigma[\Delta_3]$ extended with constants from Δ_3. We then have $\triangle +^{\Delta_3} \triangledown = \Delta_3$, so when interpreting the expression $\triangle + \triangledown$ in Δ_3, we may obtain any possible value of Δ_3 as a result.

More formally, for any expression $e \in \mathcal{E}_\Sigma$, the set-value semantics $[\![e]\!]^{\alpha,S} \subseteq dom(S)$ is defined in Figure 14, where S is a possibly relational Σ-structure and $\alpha : V \to dom(S)$ is a variable assignment with $fv(\phi) \subseteq V$.

Example 12. *In particular, for any variable assignment α, we have:*

$$[\![\triangle + \triangledown]\!]^{\alpha,\Delta_3} = \triangle +^{\Delta_3} \triangledown = \Delta_3$$

5.2. First-Order Formulas

Formulas $\phi \in \mathcal{F}_\Sigma$ of first-order logic are constructed according to the abstract syntax in Figure 14 from the expression $e \in \mathcal{E}_\Sigma$, the non-disjointness symbol $\eqsim$ that is also called the equality symbol, and the usual first-order connectives:

$$\phi \in \mathcal{F}_\Sigma \quad ::= e \eqsim e \mid \exists x.\phi \mid \phi \wedge \phi \mid \neg\phi \quad \text{where } x \in \mathcal{V}$$

We use few shortcuts for supporting richer formulas. We fix a variable $x_0 \in \mathcal{V}$ and define the formula $\mathit{true} =_{\text{def}} \exists x_0.\ x_0 \eqsim x_0$. Furthermore, for any sequence of formulas $\phi_1, \ldots, \phi_n$ we define the iterated conjunction:

$$\bigwedge_{i=1}^{n} \phi_i =_{\text{def}} \phi_1 \wedge \ldots \wedge \phi_n$$

such that $\bigwedge_{i=1}^{0} \phi_i = \mathit{true}$. We define negated equalities $e \neq e'$ by $\neg e \eqsim e'$, disjunctions $\phi \vee \phi'$ by $\neg(\neg\phi \wedge \neg\phi')$ and implications $\phi \to \phi'$ by $\neg\phi \vee \phi'$. The set of free variables $fv(\phi)$ can be lifted from expressions to formulas as usual:

$$\begin{aligned}
fv(\exists x.\phi) &= fv(\phi) \setminus \{x\} & fv(e \eqsim e') &= fv(e) \cup fv(e') \\
fv(\phi \wedge \phi') &= fv(\phi) \cup fv(\phi') & fv(\neg\phi) &= fv(\phi)
\end{aligned}$$

The semantics of expressions $[\![e]\!]^{S,\alpha} \subseteq dom(S)$ as defined in Figure 13 was set-valued, since the operators from the signature could be interpreted as relations in the structure S, ie., as set-valued

For all $e, e' \in \mathcal{E}_\Sigma$ and $\phi, \phi' \in \mathcal{F}_\Sigma$ with $fv(e) \cup fv(e') \cup fv(\phi) \cup fv(\phi') \subseteq V$, $x \in \mathcal{V}$:

$$[\![e \eqsim e']\!]^{\alpha,S} = \begin{cases} 1 & \text{if } [\![e']\!]^{\alpha,S} \cap [\![e'']\!]^{\alpha,S} \neq \emptyset \\ 0 & \text{else} \end{cases} \qquad [\![\phi \wedge \phi']\!]^{\alpha,S} = [\![\phi]\!]^{\alpha,S} \wedge^{\mathbb{B}} [\![\phi']\!]^{\alpha,S}$$

$$[\![\neg\phi]\!]^{\alpha,S} = \neg^{\mathbb{B}}([\![\phi]\!]^{\alpha,S}) \qquad [\![\exists x.\phi]\!]^{\alpha,S} = \begin{cases} 1 & \text{if exists } s \in dom(S). \\ & [\![\phi]\!]^{\alpha[x/s],S} = 1 \\ 0 & \text{else} \end{cases}$$

Figure 14: Semantics $[\![\phi]\!]^{\alpha,S} \in \mathbb{B}$ of formulas $\phi \in \mathcal{F}_\Sigma$ of first-order logic, where S is a Σ-structure and $\alpha : V \to dom(S)$.

functions. The nondisjointness or equality operator $\eqsim$ will be interpreted as set nondisjointness in general relational structures. Over algebras, however, this is equivalent to the interpretation by (set) equality. So when restricting the interpretation to algebras, it is fully justified to call the nondisjointness operator $\eqsim$ also equality symbol. In the general case, however, sets may be nondisjoint without being equal.

Example 13. *We want the following formula to become true in the relational structure Δ_3 over the arithmetic signature $\Sigma_{pos\text{-}arith}[\Delta_3]$ but extended with extra constants from Δ_3:*

$$\begin{array}{ll} & \triangle + \triangledown \eqsim \triangle \\ \wedge & \triangle + \triangledown \eqsim \triangledown \\ \wedge & \triangle + \triangledown \eqsim \approx \end{array}$$

The reason is that the operator $\eqsim$ stands for set nondisjointness, while $\triangle + \triangledown$ denotes the full set Δ_3, which is non-disjoint with all the singletons $\{\triangledown\}$, $\{\triangle\}$ and $\{\approx\}$.

Note that the difference between equality and nondisjointness doesn't matter for formulas like $x \eqsim y$, since variables are always interpreted as singletons. The difference may arise, however, for formulas like $x \eqsim y + z$, since the interpretation of the addition symbol $+$ in general structures may be a relation, i.e., it may be nondeterministic so that $y + z$ denotes a nonsingleton set. If this set contains the single possible value of x and some other value, then nondisjointness holds but not set equality.

Example 14. *The following formula is unsatisfiable over the structure Δ_3 with the extended signature $\Sigma_{pos\text{-}arith}[\Delta_3]$:*

$$\exists x.\ (x \eqsim \triangle \ \wedge \ x \eqsim \triangledown)$$

This is since for any variable assignment the expression x must be evaluated to a singleton, which cannot contain both $\triangle$ and $\triangledown$. Another way to see this is that $\exists x.\ (x \eqsim \triangle \wedge x \eqsim \triangledown)$ is equivalent to $\triangledown \eqsim \triangle$ which evaluates to false.

The formal definition of the semantics $[\![\phi]\!]^{\alpha,S} \in \mathbb{B}$ of first-order formulas $\phi \in \mathcal{F}_\Sigma$ is given in Figure 14. It is a truth value, that depends on a Σ-structures S and a variable assignment $\alpha : V \to dom(S)$ with $fv(\phi) \subseteq V$. As define earlier, any expressions $e \in \mathcal{E}_\Sigma$ with $fv(e) \subseteq V$ denotes a subset of $dom(S)$. It is a singleton if e is a variable or a constant, or if S is an algebra. The nondisjointness symbol in equations $e \eqsim e'$ is interpreted as set nondisjointness, i.e., $e \eqsim e'$ holds if e and e' are interpreted as nondisjoint subsets of $dom(S)$. The logical operators of conjunction $\wedge$ and negation $\neg$ are interpreted as the corresponding boolean operations $\wedge^{\mathbb{B}}$ and $\neg^{\mathbb{B}}$. Existentially quantified formulas $\exists x.\phi$ become true, if there exists some value s for x in the domain of S such that ϕ becomes true when mapping x to s.

The solution set of a formula $\phi \in \mathcal{F}_\Sigma$ over a Σ-structure S is the set of variable assignments from $fv(\phi)$ into the domain of S that makes the formula true:

$$sol^S(\phi) = \{\alpha : fv(\phi) \to dom(S) \mid [\![\phi]\!]^{\alpha,S} = 1\}$$

Given a set of variables $V \subseteq \mathcal{V}$ we can define set of solution sets with additional variables in V as follows:

$$sol^S_V(\phi) = sol^S(\phi \wedge \bigwedge_{x \in V} x = x)$$

5.3. First-Order Definitions

We define next what it means for a relation to be by some formula of first-order logic. Such relations permit us to become independent of the naming of the global variables of the formula.

Definition 15. *A* FO-definition *of arity m is a function $F : \mathcal{V}^m \to \mathcal{F}_\Sigma$ for which there exists a formula $\phi \in \mathcal{F}_\Sigma$ and a sequence of distinct variables $\boldsymbol{x} \in \mathcal{V}^m$ such that $fv(\phi) = \{\boldsymbol{x}\}$ and $F(\boldsymbol{y}) = \phi[\boldsymbol{x}/\boldsymbol{y}]$ for all $\boldsymbol{y} \in \mathcal{V}^m$. For any Σ-structure S and domain $D = dom(S)$, F defines the following relation $rel^S(F) \subseteq D^m$:*

$$rel^S(F) = \{(\alpha(\pi_1(\boldsymbol{x})), \ldots, \alpha(\pi_m(\boldsymbol{x}))) \in D^m \mid \alpha \in sol^S(\phi)\}$$

The formula $F(\mathbf{y})$ states that the values of $\mathbf{y}$ are in the relation defined by the fomula ϕ. Which precise sequence $\mathbf{y}$ of distinct variables is chosen, does not matter since $F(\mathbf{y}) = F(\mathbf{x})[\mathbf{x}/\mathbf{y}]$, since the solutions of $F(\mathbf{y})$ and $F(\mathbf{x})$ over structure S correspond to each other modulo renaming of variable

Lemma 16. *For any first-order definition $F : \mathcal{V}^m \to \mathcal{F}_\Sigma$ and sequence $\boldsymbol{y} = y_1 \ldots y_m$ of distinct variables:*

$$sol^S(F(\boldsymbol{y})) = \{[y_1/d_1, \ldots, y_m/d_m] \mid (d_1, \ldots, d_m) \in rel^S(F)\}.$$

Proof Straightforward.

Given some formula $\phi \in \mathcal{F}_\Sigma$ with $fv(\phi) = \{\mathbf{y}\}$ we often write $\phi(\mathbf{y})$ instead of ϕ. In other words, we identify the formula ϕ implicitely with the first-order definition that it implies with respect to the sequence $\mathbf{y}$. This first order definition satisfies $\phi(\mathbf{z}) = \phi[\mathbf{y}/\mathbf{z}]$ for all $\mathbf{z} \in \mathcal{V}^{|\mathbf{y}|}$.

5.4. Difference Abstractions

We next recast the notions of difference abstractions from [9, 11, 10] by applying homomorphisms to concrete difference in the $\Sigma_{pos\text{-}arith}$-algebra $\mathbb{R}^2_+$.

More generally, let S be a Σ-algebra and $V \subseteq \mathcal{V}$ a subset of variables. For any two variable assignments $\alpha, \alpha' : V \to dom(S)$, we define an assignment of variables to pairs of elements in the domain of the structure $pair(\alpha, \alpha') : V \to dom(S)^2$ that we call the differences of α and α', such that for all variables $x \in V$:

$$pair(\alpha, \alpha')(x) = (\alpha(x), \alpha'(x))$$

For any subset R of variable assignments of type $V \to dom(S)$ we define the *set of differences of assignments in R* by:

$$diff(R) = \{pair(\alpha, \alpha') \mid \alpha, \alpha' \in R\}$$

Definition 17. *Let S be a Σ-structure. For any homomorphism $h : S^2 \to \Delta$ and any formula $\phi \in \mathcal{F}_\Sigma$ we define the difference abstraction of the S-solution set of ϕ by:*

$$sol^S(\phi)^\Delta = h \circ diff(sol^S(\phi)))$$

The original definition of Δ-difference abstractions $sol(\phi)^\Delta$ in [9] was similar, but did not make the respective roles of *diff* and $h_\Delta : S^2 \to \Delta$ explicit. By having done so, we can now state that the difference abstraction of the S-solution sets of a formula is the S^2-solution set of the same formula.

Lemma 18. *For any formula $\phi \in \mathcal{F}_\Sigma$ and Σ-algebra S: $diff(sol^S(\phi)) = sol^{S^2}(\phi)$.*

Proof For all $\alpha : \mathcal{V} \to dom(S), \alpha' : \mathcal{V} \to dom(S), \alpha, \alpha' \in sol^S(\phi)$ we can construct the variable assignment $\alpha'' : \mathcal{V} \to dom(S)^2$ with $diff(\alpha, \alpha') = (\alpha(x), \alpha'(x)) = \alpha''(x)$. So we have $diff(sol^S(\phi)) \subseteq sol^{S^2}(\phi)$.

Conversely, for all variable assignment $\alpha'' : \mathcal{V} \to dom(S)^2$, $\alpha'' \in sol^{S^2}(\phi)$ we can generate two variable assignments $\alpha : \mathcal{V} \to dom(S), \alpha' : \mathcal{V} \to dom(S) \in sol^S(\phi)$ with $\forall x \in \mathcal{V}, \alpha(x) = \pi_1(\alpha''(x)) \wedge \alpha'(x) = \pi_2(\alpha''(x))$. So we have $sol^{S^2}(\phi) \subseteq diff(sol^S(\phi))$, and thus finally $diff(sol^S(\phi)) = sol^{S^2}(\phi)$. □

As an immediate consequence from Lemma 18, we have for any homomorphism $h : S^2 \to \Delta$ that the difference abstraction $sol(\phi)^\Delta = h \circ diff(sol^S(\phi))$ from Definition 48 satisfies:

$$sol(\phi)^\Delta = h \circ sol^{S^2}(\phi)$$

6. Equation Systems

We present the notions of equation systems that we will use in this article, including polynomial equations, positive linear equations, and $h_{\mathbb{B}}$-mixed equations.

6.1. Arithmetic Equations

We consider arithmetic equations that are first-order formula of the form $e = e'$ over the arithmetic signature, i.e., elements of $\mathcal{F}_{\Sigma_{pos\text{-}arith}}$. A system of arithmetic equations is a formula in $\mathcal{F}_{\Sigma_{pos\text{-}arith}}$. of the form:

$$\exists x_1 \ldots \exists x_m . (e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n)$$

Note that such the arithmetic equations do not support operators for substruction nor division, neither any other operators like for square roots, logarithm, etc.

6.2. Polynomial Equations

Systems of arithmetic equations in $\mathcal{F}_\Sigma$ can represent polynomial equations with natural coefficients, but not with integer coefficients. In order to see this, we introduce the following notation for writing polynomials with positive coefficients.

For any natural number $n \geq 0$ and any sequence of expressions $e_1, \ldots, e_n \in \mathcal{E}_{\Sigma_{pos\text{-}arith}}$, we define expressions for iterated sums and products:

$$\begin{array}{rclcrcl} \sum_{i=1}^n e_i & = & e_1 + \ldots + e_n & \text{where} & \sum_{i=1}^0 e_i & = & 0 \\ \prod_{i=1}^n e_i & = & e_1 * \ldots * e_n & \text{where} & \prod_{i=1}^0 e_i & = & 1 \end{array}$$

Furthermore, for all $e \in \mathcal{E}_{\Sigma_{pos\text{-}arith}}$ and natural numbers $n \geq 0$, let:

$$\begin{array}{rcl} ne & = & \sum_{i=1}^n e \\ e^n & = & \prod_{i=1}^n e \end{array}$$

A polynomial (with natural coefficients and no constant term) then is an expression in $\mathcal{F}_{\Sigma_{pos\text{-}arith}}$ that has the following form:

$$\sum_{i=1}^{n} n_i {x_i}^{m_i}$$

A polynomial equation (with natural coefficient and no constant term) is an equation $e = e'$ between two polynomials e and e'.

Example 19 (Solving equations in the pair algebra of concrete differences $\mathbb{R}_+^2$). *We consider the following polynomial equation $\phi(x,y) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ where x and y are distinct variables:*

$$3x + 4y^5 = 0$$

From the formal point of view, $\phi : \mathcal{V}^2 \to \mathcal{F}_\Sigma$ is a first-order definition, so that it can also be applied to other variables. In order to compute the solutions of $\phi(x,y)$ in the algebra of concrete differences $\mathbb{R}_+^2$, we consider the system of polynomial equations $\phi^2(x_1,y_1,x_2,y_2) = \phi(x_1,y_1) \wedge \phi(x_2,y_2)$ that contains two copies if $\phi(x,y)$ by with freshly renamed variables $x_1,x_2,y_1,y_2 \in \mathcal{V}$. The formula $\phi^2(x_1,y_1,x_2,y_2)$ is equal to:

$$3x_1 + 4y_1^5 = 0 \ \wedge \ 3x_2 + 4y_2^5 = 0$$

We then have:

$$sol^{\mathbb{R}_+^2}(\phi(x,y)) = \{[x/(\alpha(x_1),\alpha(x_2)), y/(\alpha(y_1),\alpha(y_2))] \mid \alpha \in sol^{\mathbb{R}_+}(\phi^2(x_1,y_1,x_2,y_2))\}$$

This example illustrate, that we can compute the solution set of any equation system in $\mathcal{F}_\Sigma$ over the pair algebra $\mathbb{R}_+^2$ by computing the solution set of some other equation system in $\mathcal{F}_\Sigma$ over $\mathbb{R}_+$.

6.3. Linear Equations

A *(homogeneous) linear equation (with natural coefficients and no constant term)* is a formula of $\mathcal{F}_{\Sigma_{pos\text{-}arith}}$ with the signature of positive arithmetics that has the form:

$$n_1x_1 + \ldots + n_mx_m = n_{m+1}x_{m+1} + \ldots + n_px_p$$

where $m,p,n_1,\ldots,n_p \in \mathbb{N}$ and $x_1,\ldots,x_p \in \mathcal{V}$.

Definition 20. *An equation $e = e'$ is* positive *if the right-hand side e' is equal to 0. It is called* quasi-positive *if it is positive or the right-hand side has the form nx for some natural n and some variable x.*

Example 21. *The equation system $x = 0 \wedge x = 1$ contains the positive linear equation $x = 0$ and the non-linear equation $x = 1$. The latter is non-linear, since having the constant term 1 on the right-hand side.*

6.4. Simple $h_{\mathbb{B}}$-Mixed Systems

In [16], systems of linear equations were generalized to *$h_{\mathbb{B}}$-mixed systems* by adding so called $h_{\mathbb{B}}$-exact and $h_{\mathbb{B}}$-invariant formulas. In the present article, we only need a simpler form of *$h_{\mathbb{B}}$-mixed systems* that we can define syntactically. Therefore we can delay the definition of $h_{\mathbb{B}}$-exact formulas to Definition 27 of Section 7, and avoid the introduction of $h_{\mathbb{B}}$-invariance all over.

Definition 22 (Simple $h_{\mathbb{B}}$-Mixed Systems). *A* simple $h_{\mathbb{B}}$-mixed system *is an existentially quantified conjunctive formula in $\mathcal{F}_{\Sigma_{pos\text{-}arith}}$ of the form $\exists \boldsymbol{z}.\ L \wedge P$ where L is a system of linear equations and P a system of positive polynomial equations (with positive coefficients and without out constant term).*

Note that simple $h_{\mathbb{B}}$-mixed systems may contain non-positive equations in the linear part L and non-linear equations in the positive part P.

Example 23. *The following system of polynomial equations is a simple $h_{\mathbb{B}}$-mixed system:*

$$\exists y_1.\exists y_3.\ y_1 + 4y_2 = 3y_3 \wedge y_1 * y_2 = 0$$

The linear part here consists of a single linear equation $y_1 + 4y_2 = 3y_3$, which is non-positive but quasi-positive. The positive part consists of the polynomial equation $y_1 * y_2 = 0$ which is non-linear.

Any simple $h_{\mathbb{B}}$-mixed system is indeed an $h_{\mathbb{B}}$-mixed system according to Definition 49 of [16], since any system of positive polynomial equations is $h_{\mathbb{B}}$-exact and $h_{\mathbb{B}}$-invariant by Theorem 4 of [16].

Example 24. *The equation $1 = 0$ is a $h_{\mathbb{B}}$-mixed system with the empty linear equation system true, since the equation is unsatisfiable over both $\mathbb{R}_+$ and $\mathbb{B}$. But the equation is not a simple $h_{\mathbb{B}}$-mixed system, since the polynomial on its left-hand side contains a constant term.*

7. Abstract Interpretation

Abstract interpretation can be used to approximate difference abstractions as stated by John's theorem. We start with a generalization of John's original theorem that applies to any kind of homomorphic abstraction. We then recall Theorem 39 of [16], showing that one can rewrite any linear system of equations, so that it becomes exact for the boolean abstraction. The same theorem also applies to $h_{\mathbb{B}}$-mixed systems. We note that the $h_{\mathbb{B}}$-exact rewriting relies on the generalization of John's theorem already, for which [16] refers to the present paper.

7.1. John's Theorem

We prove a generalization of John's Theorem [9, 11] stating the soundness of the abstract interpretation of first-order formulas with respect to some homomorphic abstraction. The same notion of soundness of abstract interpretation [23] was studied earlier for other kinds of abstractions.

Lemma 25. *Let h be a homomorphism between Σ-structures S and Δ, and $\alpha : V \to dom(S)$ a variable assignment. For any expression $e \in \mathcal{E}_\Sigma$ with $V(e) \subseteq V$:*

$$h([\![e]\!]^{S,\alpha}) \subseteq [\![e]\!]^{\Delta, h\circ\alpha}$$

Proof by induction on the structure of expressions $e \in \mathcal{E}_\Sigma$. Let α be a variable assignment into $dom(S)$. For any expressions $e = e_1 \odot e_2$ where $\odot \in F^{(2)}$ we have:

$$\begin{array}{lcll} h([\![e_1 \odot e_2]\!]^{S,\alpha}) & = & h([\![e_1]\!]^{S,\alpha} \odot^S [\![e_2]\!]^{S,\alpha}) & \\ & \subseteq & h([\![e_1]\!]^{S,\alpha}) \odot^\Delta h([\![e_2]\!]^{S,\alpha}) & \text{since } h \text{ is a homomorphism} \\ & \subseteq & [\![e_1]\!]^{\Delta,h\circ\alpha} \odot^\Delta [\![e_2]\!]^{\Delta,h\circ\alpha} & \text{induction hypothesis} \\ & = & [\![e_1 \odot e_2]\!]^{\Delta,h\circ\alpha} & \end{array}$$

For any expression $e = x \in V$ we have:

$$h([\![x]\!]^{S,\alpha}) = h(\{\alpha(x)\}) = [\![x]\!]^{\Delta,h\circ\alpha}$$

For constant expressions $e = c \in C$, the properties of h being a homomorphism yields:

$$h([\![c]\!]^{S,\alpha}) = h(c^S) = c^\Delta = [\![c]\!]^{\Delta,h\circ\alpha}$$

Proposition 26. *Let h be a homomorphism between Σ-structures S and Δ, and $\alpha : V \to dom(S)$ and a variable assignment. For any negation-free formula $\phi \in \mathcal{F}_\Sigma$ with $V(\phi) \subseteq V$:*

$$[\![\phi]\!]^{S,\alpha} \leq [\![\phi]\!]^{\Delta,h\circ\alpha}$$

Proof by induction on the structure of negation-free Σ-formulas $\phi \in \mathcal{F}_\Sigma$. If ϕ is some equation $e = e'$ then it holds by Lemma 25 that: $h([\![e]\!]^{S,\alpha}) \subseteq [\![e]\!]^{\Delta,h\circ\alpha}$ and $h([\![e']\!]^{S,\alpha}) \subseteq [\![e']\!]^{\Delta,h\circ\alpha}$. Hence:

$$\begin{array}{lcll} [\![e = e']\!]^{S,\alpha} = 1 & \Leftrightarrow & [\![e]\!]^{S,\alpha} \cap [\![e']\!]^{S,\alpha} \neq \emptyset & \\ & \Leftrightarrow & h([\![e]\!]^{S,\alpha}) \cap h([\![e']\!]^{S,\alpha}) \neq \emptyset & \\ & \Rightarrow & [\![e]\!]^{\Delta,h\circ\alpha} \cap [\![e']\!]^{\Delta,h\circ\alpha} \neq \emptyset & \text{by Lemma 25} \\ & \Leftrightarrow & [\![e = e']\!]^{\Delta,h\circ\alpha} = 1 & \end{array}$$

This shows that $[\![e = e']\!]^{S,\alpha} \leq [\![e = e']\!]^{\Delta,h\circ\alpha}$ as required. We next consider the case where ϕ is a conjunction of the form $\phi' \wedge \phi''$.

$$\begin{array}{lcll} [\![\phi' \wedge \phi'']\!]^{S,\alpha} & = & [\![\phi']\!]^{S,\alpha} \wedge^{\mathbb{B}} [\![\phi'']\!]^{S,\alpha} & \\ & \leq & [\![\phi']\!]^{\Delta,h\circ\alpha} \wedge^{\mathbb{B}} [\![\phi'']\!]^{\Delta,h\circ\alpha} & \text{induction hypothesis} \\ & = & [\![\phi' \wedge \phi'']\!]^{\Delta,h\circ\alpha} & \end{array}$$

The last case is where ϕ is an existentially quantified formula of the form $\exists x.\phi'$.

$$\begin{array}{lcll} [\![\exists x.\phi']\!]^{S,\alpha} = 1 & \Leftrightarrow & \exists s \in dom(S).\ [\![\phi']\!]^{\alpha[x/s],S} = 1 & \\ & \Rightarrow & \exists s \in dom(S).\ [\![\phi']\!]^{h\circ\alpha[x/s],\Delta} = 1 & \text{induction hypothesis} \\ & \Leftrightarrow & [\![\exists x.\phi']\!]^{\Delta,h\circ\alpha} = 1 & \end{array}$$

This shows that $[\![\exists x.\phi']\!]^{S,\alpha} \leq [\![\exists x.\phi']\!]^{\Delta,h\circ\alpha}$ as required. □

Theorem 1 (John's Theorem Generalized). *Let h be a homomorphism between Σ-structures S and Δ, and $\alpha : V \to dom(S)$ a variable assignment. For any negation-free formula $\phi \in \mathcal{F}_\Sigma$ with $fv(\phi) \subseteq V$:*

$$h \circ sol^S(\phi) \subseteq sol^\Delta(\phi)$$

Proof Let h be homomorphism from Σ-structure S to Σ-structure Δ and $\phi \in \mathcal{F}_\Sigma$ a negation-free formula. For any variable assignment α to $dom(S)$, we know that $[\![\phi]\!]^{S,\alpha} \leq [\![\phi]\!]^{\Delta,h\circ\alpha}$ by Proposition 26 since ϕ is negation-free. This is equivalent to $\{h \circ \alpha \mid \alpha \in sol^S(\phi)\} \subseteq sol^\Delta(\phi)$ and thus $h \circ sol^S(\phi) \subseteq sol^\Delta(\phi)$ as required. □

Corollary 2 (John's Original Theorem [9, 11]). *Let S and Δ be Σ-structures and h a homomorphism from S^2 to Δ. Any negation-free first-order formula $\phi \in \mathcal{F}_\Sigma$ then satisfies:*

$$sol^S(\phi)^\Delta \subseteq sol^\Delta(\phi)$$

Proof This follows from Lemma 18 and Theorem 1 applied to the homomorphism $h : S^2 \to \Delta$:

$$sol^S(\phi)^\Delta = h \circ diff(sol^{S^2}(\phi)) = h \circ sol^{S^2}(\phi) \subseteq sol^\Delta(\phi)$$

□

7.2. Exact Formulas

If $dom(\Delta)$ is finite then the set $sol^{\Delta}(\phi)$ is finite, while $sol^{\mathbb{R}_+}(\phi)$ is usually infinite. If furthermore ϕ is a conjunctive formula, we can compute the set $sol^{\Delta}(\phi)$ by a finite domain constraint solver from ϕ and the relations of Δ (such as e.g. Minizinc [13]). In contrast, it remains unclear how to compute the finite set $h \circ diff(sol^S(\phi))$ for infinite structures S. The problem is open, even if ϕ is a system of homogeneous linear equations and $S = \mathbb{R}_+$, so that the infinite set $sol^S(\phi)$ can be finitely represented by a triangular matrix.

This is the core of the problem that we will solve in the present paper. Our approach will be to rewrite formulas ϕ to $\mathbb{R}_+$-equivalent formulas that are h-exact in the following sense, which in terms of general abstract interpretation [23] means that abstract interpretation is complete for such formulas[24].

Definition 27. *Let h be a homomorphism between Σ-structures S and Δ. We call a Σ-formula ϕ h-*exact *if $h \circ sol^S(\phi) = sol^{\Delta}(\phi)$.*

For any h-exact formula ϕ, the homomorphic abstraction $h \circ sol^S(\phi)$ can be computed exactly by computing $sol^{\Delta}(\phi)$. This can be done by finite domain constraint programming if $dom(\Delta)$ is finite.

7.3. Exact Boolean Abstraction

We recall Theorem 5 of [16] that permits to rewrite any $h_{\mathbb{B}}$-mixed system to some some $h_{\mathbb{B}}$-exact formula that is $\mathbb{R}_+$-equivalent. Theorem 5 is based on the elementary modes rewriting of linear equation systems, as stated in Corollary 2 of [16], and the fact that this yields an $h_{\mathbb{B}}$-exact formula.

The development of this result was motivated by the needs of the present paper, but was presented independently for two reasons. First, these results require considerable effort with complementary techniques based on elementary modes, and second, they are of interest elsewhere, in particular for computing the sign abstraction as needed for the abstract interpretation of programming languages.

Proposition 28 (Elementary Modes Rewriting of Linear Equation Systems: Corollary 2 of [16]). *For any system of linear equations L of size n with m variables, we can compute in time $O(2^m poly(n))$ an $\mathbb{R}_+$-equivalent system of linear equations $\exists \boldsymbol{x}.\ L'$ of size $O(n + 2^m m)$ that is $h_{\mathbb{B}}$-exact.*

Proof sketch. An $\mathbb{R}_+$-EFM of L is a variable assignment in $\alpha \in sol^{\mathbb{R}_+}(L)$ with a minimal support, i.e. with a minimal number of variables x such that $\alpha(x) \neq 0$. It is also well-know that the set of all $\mathbb{R}_+$-EFMs of L can be computed in time $O(2^m poly(n))$ from L by using for example the reverse search method for the enumeration of polytope vertices and extreme rays [25]. Furthermore, any solution in $sol^{\mathbb{R}_+}(L)$ is equal to a positive linear combination of $\mathbb{R}_+$-EFMs of L. This can be expressed by an $\mathbb{R}_+$-equivalent system of linear equations $\exists \mathbf{x}.L'$ such that L' is a system of quasi-positive equations, in which all variables on some left-hand side belong to $\mathbf{x}$ and all variables on some right-hand side occur exactly once. Such linear equation systems are $h_{\mathbb{B}}$-exact by Theorem 3 of [16]. □

Theorem 3 ($h_{\mathbb{B}}$-Exact Rewriting of $h_{\mathbb{B}}$-Mixed Systems: Theorem 5 of [16]). *For any simple $h_{\mathbb{B}}$-mixed system ϕ of size n with m variables we can compute in time $O(2^m poly(n))$ a simple $h_{\mathbb{B}}$-mixed system $br(\phi)$ of size $O(n + 2^m m)$ that is $\mathbb{R}_+$-equivalent to ϕ and $h_{\mathbb{B}}$-exact.*

$$
\begin{array}{ll}
o, o' \in \mathcal{O}^n_\Sigma ::= \pi_i(x) \mid c \mid o \odot o' & \text{where } 1 \leq i \leq n,\ c \in C, \text{ and } \odot \in F^{(2)} \\
\psi, \psi' \in \mathcal{F}^{\mathrm{n}}_\Sigma ::= o = o' \mid \exists x.\psi \mid \psi \wedge \psi' \mid \neg\psi & \text{where } x \in \mathcal{V} \text{ g}
\end{array}
$$

Figure 15: Expressions and formulas of the first-order logic with n-tuples.

Proof sketch. Consider a simple $h_{\mathbb{B}}$-mixed system $\phi = \exists \mathbf{z}.(L \wedge P)$. We can then replace L its elemenary modes rewriting $\exists \mathbf{x}.L'$ from Proposition 28, which is $\mathbb{R}_+$-equivalent to L. Let $br(\phi)$ be the simple $h_{\mathbb{B}}$-mixed system $\exists \mathbf{z}.((\exists \mathbf{x}.L') \wedge P)$. Since P is a positive polynomial system, it is $h_{\mathbb{B}}$-exact and $h_{\mathbb{B}}$-invariant. Therefore, Proposition 41 of [16] implies that $br(\phi)$ is $h_{\mathbb{B}}$-exact and $\mathbb{R}_+$-equivalent to ϕ. □

In order to compute the boolean abstraction $h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(\phi)$ for a given simple $h_{\mathbb{B}}$-mixed system ϕ, we first compute its $h_{\mathbb{B}}$-exact rewriting $br(\phi)$ along the proof of Theorem 3, which is based on the elementary modes rewriting of the linear part given in Proposition 28. Second, we compute the solution set $sol^{\mathbb{B}}(br(\phi))$ by finite domain constraint programming. We can then use the $h_{\mathbb{B}}$-exactness of $br(\phi)$ to conclude:

$$sol^{\mathbb{B}}(br(\phi)) = h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(br(\phi)) = h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(\phi)$$

Therefore, $sol^{\mathbb{B}}(br(\phi))$ is the boolean abstraction of $sol^{\mathbb{R}_+}(\phi)$.

8. First-Order Tuple Logic

We next extend the first-order logic to n-tuples where the parameter n is fixed. The only case we need later on is the first-order pair logic, i.e., the first-order tuple logic with $n = 2$. Back and forth compilers from first-order logic with and without tuples will be allows us to reason about solution sets of equation systems over the pair algebra $\mathbb{R}^2_+$, i.e., about concrete differences.

8.1. Syntax and Semantics

The syntax of first-order logic with n-tuples is given in Figure 15. The expressions $o \in \mathcal{O}^n_\Sigma$ are like the expression $e \in \mathcal{E}_\Sigma$ except that variables x are now replaced by projection expressions $\pi_i(x)$ where $1 \leq i \leq n$. The reason is that any variable must denote an n-tuple of values, rather than a single value.

Given that the syntax is basically the same, the notions of polynomial equation, linear equations, and simple $h_{\mathbb{B}}$-mixed systems carry over literally from the standard first-order logic to the first-order tuple logic.

The only change in the semantics is when interpreting over an algebra S, then the variables are mapped to elements of the tuple algebra S^n. Hence we consider variables assignments β to n-tuples of $dom(S)^n$. The only change in hte definition of the interpretation needed is that:

$$[\![\pi_i(x)]\!]^{S,\beta} = \{\pi_i(\beta(x))\}$$

The set of n-ary solutions of a formula $\psi \in \mathcal{F}^{\mathrm{n}}_\Sigma$ over a Σ-structure S is defined as follows:

$$n\text{-}sol^S(\psi) = \{\beta : fv(\psi) \to dom(S)^n \mid [\![\psi]\!]^{\beta,S} = 1\}$$

8.2. From Standard to Tuple Logic

We next show how to express any first-order formulas in $\mathcal{F}_\Sigma$, interpreted over a tuple algebra S^n, by some formulas in $\mathcal{F}^n_\Sigma$, interpreted over S. In a first step, we convert expressions of the standard logic $e \in \mathcal{E}_\Sigma$ – that can be interpret over the tuple Σ-algebra S^n – to expressions of the tuple logic $\pi_i(e) \in \mathcal{O}^n_\Sigma$ for all $1 \leq i \leq n$. If $e = x$ for some variable $x \in \mathcal{V}$ then $\pi_i(x)$ is a term in $\mathcal{O}^n_\Sigma$ anyway. Otherwise, we define $\pi_i(e)$ such that for all operators $\odot \in F^{(2)}$ expressions $e', e'' \in \mathcal{E}_\Sigma$, and constants $c \in C$:

$$\begin{array}{rll} \pi_i(e' \odot e'') & =_{\text{def}} & \pi_i(e') \odot \pi_i(e'') \\ \pi_i(c) & =_{\text{def}} & c \end{array}$$

Lemma 29. *For any $e \in \mathcal{E}_\Sigma$, Σ-algebra S, $n \geq 1$, and $\beta : V \to dom(S)^n$ with $fv(\phi) \subseteq V \subseteq \mathcal{V}$:*

$$[\![e]\!]^{\beta,S^n} = [\![(\pi_1(e), \ldots, \pi_n(e))]\!]^{\beta,S}$$

Proof sketch. By induction on the structure of expressions in $\mathcal{E}_\Sigma$. □

We next lift the mapping of standart to to tuple expressions to formulas: we convert any standard logic formula $\phi \in \mathcal{F}_\Sigma$ to some formula $fot^n(\phi) \in \mathcal{F}^n_\Sigma$ while introducing tuple projection:

$$\begin{array}{ll} fot^n(e \doteq e') =_{\text{def}} \wedge_{i=1}^n \pi_i(e) \doteq \pi_i(e') & fot^n(\phi \wedge \phi') =_{\text{def}} fot^n(\phi) \wedge fot^n(\phi') \\ fot^n(\neg\phi) =_{\text{def}} \neg fot^n(\phi) & fot^n(\exists x.\phi) =_{\text{def}} \exists x. fot^n(\phi) \end{array}$$

The next proposition shows that for any standard logic formula ϕ, that its interpretation over the tuple algebra S^n has the same solutions, than interpreting the corresponding formula $fot^n(\phi)$ of the tuple logic over S.

Proposition 30. *For any $\phi \in \mathcal{F}_\Sigma$, Σ-algebra S, and $n \geq 1$: $sol^{S^n}(\phi) = n\text{-}sol^S(fot^n(\phi))$.*

Proof sketch. By induction on the structure of formulas in $\mathcal{F}_\Sigma$. We only show the base case of Σ-equations. So let ϕ is a Σ-equation of the form $e \doteq e'$ where $e, e' \in \mathcal{E}_\Sigma$ and β a variable assignment $\beta : V \to dom(S)^n$ such that $fv(\phi) \subseteq V \subseteq \mathcal{V}$. Then:

$$\begin{array}{ll} sol^{S^n}(e \doteq e') & = \{\beta \mid [\![e \doteq e']\!]^{\beta,S^n} = 1\} \\ \textit{Lemma 29} & = \{\beta \mid [\![\bigwedge_{i=1}^n \pi_i(e) \doteq \pi_i(e')]\!]^{\beta,S} = 1\} \\ & = \{\beta \mid [\![fot^n(e \doteq e')]\!]^{\beta,S} = 1\} \\ & = n\text{-}sol^S(fot^n(e \doteq e')) \end{array}$$

The inductive cases for the other formulas are straightforward. □

8.3. From Tuple to Standard Logic

We wish to decompose any FO-tuple formula $\psi \in \mathcal{F}^n_\Sigma$ into a standard FO formula $dec_\nu(\psi) \in \mathcal{F}_\Sigma$ for any given n-tuple of fresh variables generators $\nu = (\nu_1, \ldots, \nu_n)$ where $\nu_1, \ldots, \nu_n : \mathcal{V} \to \mathcal{V}$. For this we map any expression $o \in \mathcal{O}^n_\Sigma$ with projections to some expression $dec_\nu(o) \in \mathcal{E}_\Sigma$ with new variables instead of projections:

$$dec_\nu(\pi_i(x)) =_{\text{def}} \nu_i(x), \qquad dec_\nu(c) =_{\text{def}} c, \qquad dec_\nu(o \odot o') =_{\text{def}} dec_\nu(o) \odot dec_\nu(o').$$

We then map any formula $\psi \in \mathcal{F}^n_\Sigma$ with projections to some formula $dec_\nu(\psi) \in \mathcal{F}_\Sigma$ with fresh variables:

$$\begin{array}{ll} dec_\nu(o \doteq o') =_{\text{def}} dec_\nu(o) \doteq dec_\nu(o') & dec_\nu(\neg\psi) =_{\text{def}} \neg dec_\nu(\psi) \\ dec_\nu(\psi \wedge \psi') =_{\text{def}} dec_\nu(\psi) \wedge dec_\nu(\psi') & dec_\nu(\exists x.\psi) =_{\text{def}} \exists \nu_1(x) \ldots \exists \nu_n(x).\ dec_\nu(\psi) \end{array}$$

For any $V \subseteq \mathcal{V}$ and variable assignment $\alpha : \uplus_{i=1}^{n} \nu_i(V) \to dom(S)$ we define a composition assignment $cmp_\nu(\alpha) : V \to dom(S)^n$ such that for all $x \in V$:

$$cmp_\nu(\alpha)(x) = (\alpha(\nu_1(x)), \ldots, \alpha(\nu_n(x)))$$

The function cmp_ν is a bijection from assignments of type $\uplus_{i=1}^{n} \nu_i(V) \to dom(S)$ to assignments of type $V \to dom(S)^n$. Its inverse decomposition cmp_ν^{-1} satisfies for any variable assignment $\beta : V \to dom(S)^n$:

$$cmp_\nu^{-1}(\beta)(\nu_i(x)) = \pi_i(\beta(x))$$

Lemma 31. *For any expression $o \in \mathcal{O}_\Sigma^n$ and variable assignment $\beta : V \to dom(S)^n$ with $fv(o) \subseteq V \subseteq \mathcal{V}$ we have $[\![dec_\nu(o)]\!]^{S, cmp_\nu^{-1}(\beta)} = [\![o]\!]^{\beta, S}$.*

Proof sketch. By induction on the structure of expressions $o \in \mathcal{O}_\Sigma^n$. □

Proposition 32. *For any $\psi \in \mathcal{F}_\Sigma^n$, Σ-structure S, and $n \geq 1$:*

$$n\text{-}sol^S(\psi) = cmp_\nu(sol^S(dec_\nu(\psi)))$$

Proof We first prove the following claim is by induction on the structure of Σ-formulas in $\mathcal{F}_\Sigma^n$, where the base case follows from Lemma 31.

Claim 33. *For any variable assignment $\beta : V \to dom(S)^n$ with $V \subseteq \mathcal{V}$ and formula $\psi \in \mathcal{F}_\Sigma^n$ we have $[\![dec_\nu(\psi)]\!]^{S, cmp_\nu^{-1}(\beta)} = [\![\psi]\!]^{\beta, S}$.*

The proof of the claim is straightforward by induction on the structure of Σ-formulas in $\mathcal{F}_\Sigma^n$. Finally, the claim implies the proposition as follows:

$$\begin{array}{lll} \beta \in n\text{-}sol^S(\psi) & \Leftrightarrow & cmp_\nu^{-1}(\beta) \in n\text{-}sol^S(dec_\nu(\psi)) \qquad \text{previous claim} \\ & \Leftrightarrow & cmp_\nu(cmp_\nu^{-1}(\beta)) \in cmp_\nu(n\text{-}sol^S(dec_\nu(\psi))) \\ & \Leftrightarrow & \beta \in cmp_\nu(n\text{-}sol^S(dec_\nu(\psi))) \end{array}$$

□

8.4. Commutation Property

As above, we fix some n-tuple of fresh variable generators $\nu = (\nu_1, \ldots, \nu_n)$ and consider the operator cmp_ν that composes assignments for each component to an assignment to n-tuples. We next show a commutation property of the composition operator with homomorphisms.

Lemma 34. *For any homomorphism h between Σ-structures S and Δ and any subset $V \subseteq \mathcal{V}$, and variable assignment $\alpha : \uplus_{i=1}^{n} \nu_i(V) \to dom(S)$:*

$$cmp_\nu(h \circ \alpha) = h^n \circ cmp_\nu(\alpha)$$

Proof The terms on both side of the equations denote functions with domain V. Furthermore, for any variable $x \in V$ we have:

$$\begin{array}{lll} cmp_\nu(h \circ \alpha)(x) & = & (h(\alpha(\nu_1(x))), \ldots, h(\alpha(\nu_n(x)))) \\ & = & h^n((\alpha(\nu_1(x))), \ldots, \alpha(\nu_n(x))) \\ & = & h^n(cmp_\nu(\alpha)(x)) \\ & = & (h^n \circ cmp_\nu(\alpha))(x) \end{array}$$

□

Proposition 35. *For any subset $V \subseteq \mathcal{V}$, any subset R of variable assignments of type $\uplus_{i=1}^{n} \nu_i(V) \to dom(S)$, and any homomorphism h from Σ-structure S to Σ-structure Δ:*

$$cmp_\nu(h \circ R) = h^n \circ cmp_\nu(R)$$

Proof By Lemma 34: $cmp_\nu(h \circ R) = \{cmp_\nu(h(\alpha)) \mid \alpha \in R\} = \{h^n(cmp_\nu(\alpha)) \mid \alpha \in R\} = h^n \circ cmp_\nu(R)$. □

8.5. First-Order Definitions

Our strategy for characterizing difference abstractions will be to decompose them into boolean abstractions via first-order definable functions. Therefore, we define next what it means for function or relation to be defined by a formula of first-order tuple logic.

Definition 36. *A $\mathcal{F}^n_\Sigma$-definition of arity m is a function $F : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ for which there exists a formula $\psi \in \mathcal{F}^n_\Sigma$ and a sequence of distinct variables $\boldsymbol{x} \in \mathcal{V}^m$ such that $fv(\psi) = \{\boldsymbol{x}\}$ and $F(\boldsymbol{y}) = \psi[\boldsymbol{x}/\boldsymbol{y}]$ for all $\boldsymbol{y} \in \mathcal{V}^m$. Let S be a Σ-algebra, $n \in \mathbb{N}$, and $D = dom(S)^n$ the domain of the tuple algebra S^n. The first-order definition F then defines the following relation $n\text{-}rel^S(F) \subseteq D^m$:*

$$n\text{-}rel^S(F) = \{(\alpha(\pi_1(\boldsymbol{x})), \ldots, \alpha(\pi_m(\boldsymbol{x}))) \in D^m \mid \alpha \in n\text{-}sol^S(\psi)\}$$

The formula $F(\mathbf{y})$ states that the values of $\mathbf{y}$ are in the relation defined by the fomula ψ. Which precise sequence $\mathbf{y}$ of distinct variables is chosen, does not matter since $F(\mathbf{y}) = F(\mathbf{x})[\mathbf{x}/\mathbf{y}]$, since the solutions of $F(\mathbf{y})$ and $F(\mathbf{x})$ over structure S correspond to each other modulo renaming of variable

Lemma 37. *For any first-order definition $F : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ and sequence $\boldsymbol{y} = y_1 \ldots y_m$ of distinct variables:*

$$n\text{-}sol^S(F(\boldsymbol{y})) = \{[y_1/d_1, \ldots, y_m/d_m] \mid (d_1, \ldots, d_m) \in n\text{-}rel^S(F)\}.$$

Proof Straightforward.

8.6. Defining Function Applications

We will frequently have to apply functions defined in first-order tuple logic to relations defined in first-order tuple logic. We show next how to define in first-order logic the application of a function with one input and one output to all components of relation. Later on we will have to consider functions of higher arities too.

Let S be the algebra of interest and $\ell, m, n \geq 0$. Let $D = dom(S)^n$ be the domain of the tuple algebra S^n. Let $F : \mathcal{V}^{1+\ell} \to \mathcal{F}^n_\Sigma$ be a first-order definition that defines a total function $n\text{-}rel^S(F) : D \to D^\ell$. Let $G : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ be a first-order definition that defines a relation $n\text{-}rel^S(G) \subseteq D^m$. We can then define the m-fold application of $n\text{-}rel^S(F)$ to components of m-tuple $n\text{-}rel^S(G)$ by the first-order definition $apply^m_{1,\ell}(F, G)$ such that for all sequences of variables $\mathbf{y}^i = y^i_1 \ldots y^i_m \in \mathcal{V}^*$ where $1 \leq i \leq \ell$:

$$apply^m_{1,\ell}(F, G)(\mathbf{y}^1, \ldots, \mathbf{y}^\ell) \quad =_{\text{def}} \quad \exists \mathbf{z}.\ G(\mathbf{z}) \wedge \bigwedge_{i=1}^{m} F(z_i, y^1_i, \ldots, y^\ell_i)$$

where $\mathbf{z} = z_1 \ldots z_m \in \mathcal{V}^*$ is a sequence of fresh variables.

Proposition 38. *Let $\ell, m, n \in \mathbb{N}$ natural numbers, $G : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ and $F : \mathcal{V}^{1+\ell} \to \mathcal{F}^n_\Sigma$ first-order definitions, and $\boldsymbol{y} = y_1, \ldots y_m$, $\boldsymbol{y}^1 = y^1_1 \ldots y^1_m$, ..., $\boldsymbol{y}^\ell = y^\ell_1 \ldots y^\ell_m$, sequence of pairwise distinct variables. We assume a Σ-algebra S and consider the domain $D = dom(S)^n$ of the tuple algebra S^n. If the relation $n\text{-}rel^S(F) \subseteq D \times D^\ell$ is a partial function then:*

$$n\text{-}rel^S(F) \circ n\text{-}sol^S(G(\boldsymbol{y})) = \left\{ \begin{array}{l} [y_i/(\alpha(y^1_i), \ldots, \alpha(y^\ell_i)) \mid 1 \leq i \leq m] \mid \\ \qquad \alpha \in n\text{-}sol^S(apply^m_{1,\ell}(F, G)(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^\ell))\} \end{array} \right.$$

The proof is delayed to Section 13 where the same Proposition 52 is restated after the more general Lemma 60, which treats m-fold applications $apply^m_{k,\ell}(F, G)$ with $k, \ell \in \mathbb{N}$ for total functions defined in first-order tuple logic by a definition F of type $D^k \to D^\ell$ to all components of the relations of type $(D^k)^m$ defined by G yielding some relation of type $(D^\ell)^m$.

9. Characterizing Difference Abstractions

We show how to characterize difference abstractions with respect to Δ_3 and Δ_6 of solution sets of systems of linear equations. Our solution is based on the exact rewriting of linear and $h_{\mathbb{B}}$-mixed systems of equations for the boolean abstraction (Theorem 3). It permits us to compute difference abstractions of solution sets of linear equations systems in single exponential time. In contrast to the boolean abstraction, however, we cannot provide any exact rewriting of linear equation systems for difference abstractions, and do not know whether this may always be possible.

9.1. Exact Difference Abstraction for Δ_3

We start with the abstraction of concrete difference in $\mathbb{R}^2_+$ to abstract differences in Δ_3. We first decompose the difference abstraction h_{Δ_3} into the boolean abstraction $h_{\mathbb{B}}$ and the *minimal support projection* of concrete differences.

The minimal support projection of a pair of positive real numbers $p \in \mathbb{R}^2_+$ is the intersection point of the parallel of the diagonal through p with either the x-axis or else the y-axis, as illustrated geometrically in Figure 16. The minimal support projection is a total function of type $\mathbb{R}^2_+ \to \mathbb{R}^2_+$ and thus a binary relation included in $\mathbb{R}^2_+ \times \mathbb{R}^2_+$. We can define it in the first-order pair logic by using the first-order definition $msp : \mathcal{V}^2 \to \mathcal{F}^2_\Sigma$ that satisfies for all two variables $x, y \in \mathcal{V}$:

$$msp(x, y) \ =_{\text{def}} \ \pi_1(x) + \pi_2(y) = \pi_2(x) + \pi_1(y) \wedge \pi_1(y) * \pi_2(y) = 0$$

Note that $msp(x, y)$ is simple $h_{\mathbb{B}}$-mixed system in the first-order pair logic. It contains a non-positive linear equation and a positive polynomial equation that is non-linear. The minimal support projection is then the relation defined by msp over the pair algebra $\mathbb{R}^2_+$, ie., $2\text{-}rel^{\mathbb{R}_+}(msp)$.

For any solution $\alpha \in 2\text{-}sol^{\mathbb{R}_+}(msp(x, y))$, some component of $\alpha(y)$ must be equal to zero since $\pi_1(\alpha(y)) * \pi_2(\alpha(y)) = 0$. The other component must be equal to $|\pi_1(\alpha(x)) - \pi_2(\alpha(x))|$ since $\pi_1(\alpha(x)) - \pi_2(\alpha(x)) = \pi_1(\alpha(y)) - \pi_2(\alpha(y))$. Hence:

$$\begin{array}{rcl} 2\text{-}rel^{\mathbb{R}_+}(msp) & = & \{((r, r'), (0, r' - r)) \mid r \leq r'\} \\ & \cup & \{((r, r'), (r - r', 0)) \mid r \geq r'\} \end{array}$$

Lemma 39 Decomposition. $h_{\Delta_3} = h^2_{\mathbb{B}} \circ 2\text{-}rel^{\mathbb{R}_+}(msp)$.

Proof This follows immediately from the equation before the lemma. □

For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}^2_\Sigma$, we introduced in Section 8.6 the first-order application $apply^m_{1,1}(msp, G) : \mathcal{V}^m \to \mathcal{F}^2_\Sigma$ that defines in the first-order pair logic the application of function defined by msp to the m components of the relation defined by G.

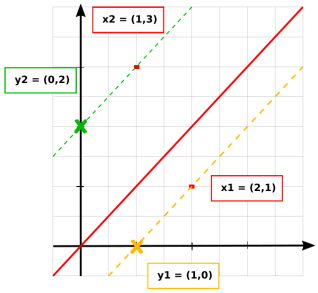


Figure 16: Two examples for the minimal support projection $f = 2\text{-}rel^{\mathbb{R}_+}(msp)$: $y_1 = f(x_1)$ and $y_2 = f(x_2)$.

Lemma 40 (First-order application of *msp*). *For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}^2_\Sigma$ and sequence $\boldsymbol{y} \in \mathcal{V}^m$:*

$$2\text{-}rel^{\mathbb{R}_+}(msp) \circ 2\text{-}sol^{\mathbb{R}_+}(G(\boldsymbol{y})) = 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,1}(msp, G)(\boldsymbol{y}))$$

Proof This lemma is a consequence of Proposition 38 on first-order definitions of function applications with $\ell = 1$, $n = 2$, $F = msp$, $S = \mathbb{R}_+$, and the fact that $2\text{-}rel^{\mathbb{R}_+}(msp)$ is a total function by Lemma 39. □

Theorem 4 (Characterization of difference abstraction to Δ_3). *Let ν be a pair of new variable generators. For any system of linear equations $L(\boldsymbol{y}) \in \mathcal{F}_\Sigma$ of size n with a sequence of m distinct free variables $\boldsymbol{y} \in \mathcal{V}^*$ we can compute in time $O(2^{2m}poly(n))$ a simple $h_{\mathbb{B}}$-mixed system $M(map^2_\nu(\boldsymbol{y})) \in \mathcal{F}_\Sigma$ with $2m$ free variables and size $O(n + 2^{2m}m)$ such that:*

$$h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L(\boldsymbol{y}))) = cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\boldsymbol{y}))))$$

Furthermore, if $L(\boldsymbol{y})$ is the application of some first-order definition $L : \mathcal{V}^m \to \mathcal{F}_\Sigma$ to the sequence $\boldsymbol{y}$ and $N : \mathcal{V}^{2m} \to \mathcal{F}_\Sigma$ is the first-order definition satisfying:

$$N(map^2_\nu(\boldsymbol{y})) = dec_\nu(apply^m_{1,1}(msp, fot^2 \circ L)(\boldsymbol{y}))$$

then $M : \mathcal{V}^{2m} \to \mathcal{F}_\Sigma$ can be choosen as the first-order definition such that $M(map^2_\nu(\boldsymbol{y}))$ is the $h_{\mathbb{B}}$-exact rewriting of the simple $h_{\mathbb{B}}$-mixed system $N(map^2_\nu(\boldsymbol{y}))$.

Proof Let $\mathbf{y} = y_1 \dots y_m$ and $L : \mathcal{V}^m \to \mathcal{F}_\Sigma$ be a first-order definition such that formula $L(\mathbf{y})$ is a linear system of equations.

	$h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y})))$
Proposition 18	$= h_{\Delta_3} \circ sol^{\mathbb{R}^2_+}(L(\mathbf{y}))$
Decomposition of h_{Δ_3} by Lemma 39	$= h^2_{\mathbb{B}} \circ 2\text{-}rel^{\mathbb{R}_+}(msp) \circ sol^{\mathbb{R}^2_+}(L(\mathbf{y}))$
First-order pair logic: Proposition 30	$= h^2_{\mathbb{B}} \circ 2\text{-}rel^{\mathbb{R}_+}(msp) \circ 2\text{-}sol^{\mathbb{R}_+}(L^2(\mathbf{y}))$
	with $L^2 = fot^2 \circ L$
First-order application of msp: Lemma 40	$= h^2_{\mathbb{B}} \circ 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,1}(msp, L^2)(\mathbf{y})))$

Not we would like to use $h_{\mathbb{B}}$-exact rewriting, but it applies only for equation systems of the standard first-order logic, and not for equation systems of the first-order pair logic such as $apply^m_{1,1}(msp, L^2)(\mathbf{y})$. This problem can be resolved by compiling formulas from the first-order

pair logic back to formulas of the standard first-order logic, while introducing fresh variables. For this we use the pair of fresh variable generators ν.

	$h^2_{\mathbb{B}} \circ 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,1}(msp, L^2)(\mathbf{y})))$
From pair to standard logic: Proposition 32	$= h^2_{\mathbb{B}} \circ cmp_\nu(sol^{\mathbb{R}_+}(dec_\nu(apply^m_{1,1}(msp, L^2)(\mathbf{y}))))$
Commutation property: Proposition 35	$= cmp_\nu(h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(dec_\nu(apply^m_{1,1}(msp, L^2)(\mathbf{y}))))$
$h_{\mathbb{B}}$-exact rewriting: Theorem 3	$= cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y})))$
	where $M(map^2_\nu(\mathbf{y}))$ *is the* $h_{\mathbb{B}}$*-exact rewriting of*
	the simple $h_{\mathbb{B}}$*-mixed system* $N(map^2_\nu(\boldsymbol{y}))$ *where:*
	$N(map^2_\nu(\mathbf{y})) = dec_\nu(apply^m_{1,1}(msp, L^2)(\mathbf{y}))$

By Theorem 3, the size of the simple $h_{\mathbb{B}}$-exact $h_{\mathbb{B}}$-mixed system $M(map^2_\nu(\mathbf{y}))$ is in $O(n + 2^{2m}m)$ and the time of its computation is in $O(2^{2m}poly(n))$. □

Lemma 41. *For any first-order definition $L : \mathcal{V}^m \to \mathcal{F}_\Sigma$, the simple $h_{\mathbb{B}}$-mixed system $N(map^2_\nu(\boldsymbol{y}))$ from Theorem 4 is equal to the following formula where μ_1 is some new variable generator:*

$$N(map^2_\nu(\boldsymbol{y})) = \exists map_{\mu_1}(\boldsymbol{y}).\ \begin{pmatrix} \bigwedge_{Y\in\{\boldsymbol{y}\}} dec_\nu(msp(\mu_1(Y), Y)) \wedge \\ \bigwedge_{i\in\{1,2\}} L(map_{\nu_i}(map_{\mu_1}(\boldsymbol{y}))) \end{pmatrix}$$

Proof The first-order definition $L^2 = fot^2 \circ L$ applied to the variable sequence $\mathbf{y}$ yields:

$$fot^2(L(\mathbf{y})) = \bigwedge_{i\in\{1,2\}} L(map_{\pi_i}(\mathbf{y}))$$

Let $\mu_1 : \mathcal{V} \to \mathcal{V}$ a fresh variable generator Applying the function $2\text{-}rel^{\mathbb{R}_+}(msp)$ in first-order pair logic to all components of tuples in the relation $n\text{-}rel^{\mathbb{R}_+}(fot^2 \circ L)$ while using μ_1 to create fresh existentially bound variables:

$$apply^m_{1,1}(msp, fot^2 \circ L)(\mathbf{y}) = \exists map_{\mu_1}(\mathbf{y}).\ \begin{pmatrix} \bigwedge_{Y\in\{\mathbf{y}\}} msp(\mu_1(Y), Y) \wedge \\ \bigwedge_{i\in\{1,2\}} L(map_{\pi_i}(\mathbf{y})) \end{pmatrix}$$

Applying the decomposition to with the pair of fresh variable generators $\nu = (\nu_1, \nu_2)$ yields that the formula $N(\mathbf{y}) = dec_\nu(apply^m_{1,1}(msp, fot^2 \circ L)(map^2_\nu(\mathbf{y})))$ is equal to the following formula as required by the lemma:

$$\exists map^2_\nu(map_{\mu_1}(\mathbf{y})).\ \begin{pmatrix} \bigwedge_{Y\in\{\mathbf{y}\}} dec_\nu(msp(\mu_1(Y), Y)) \wedge \\ \bigwedge_{i\in\{1,2\}} L(map_{\nu_i}(map_{\mu_1}(\mathbf{y})) \end{pmatrix} \quad \square$$

Example 42. *We reconsider the reaction network from biological example from Figure 4 with the sequence of variables there can be taken as the following sequence of reactions:*

$$\boldsymbol{y} = r_1 ... r_6\, in_A\, out_B\, out_C$$

The linear equation system that was derived form the reaction network there is:

$$L(\boldsymbol{y}) = \begin{pmatrix} in_A = r_1 + r_2 & \wedge \\ r_1 = out_B & \wedge \\ r_2 = out_C & \wedge \\ r_4 = r_6 & \wedge \\ r_3 = r_5 & \end{pmatrix}$$

$$M(map^2_\nu(\mathbf{y})) = \bigwedge_{Y\in\{\mathbf{y}\}} \nu_1(Y) * \nu_2(Y) = 0 \wedge M'(map_\nu(\mathbf{y}))$$

where

$$M'(map^2_\nu(\mathbf{y})) = \exists map^2_\nu(map_{\mu_1}(\mathbf{y}))\,.\; M''(map^2_\nu(map_{\mu_1}(\mathbf{y})), map^2_\nu(\mathbf{y}))$$

$$M''(map^2_\nu(map_{\mu_1}(\mathbf{y})), map^2_\nu(\mathbf{y})) = \exists x_0 \ldots \exists x_{22}.$$

$$\left(\begin{array}{llll}
\nu_1(\mu_1(\mathrm{r}_1)) = x_{16}+x_{17}+x_{18} & \wedge & \nu_1(\mathrm{r}_1) = x_{10}+x_{17}+x_{18} & \wedge\\
\nu_1(\mu_1(\mathrm{r}_2)) = x_{13}+x_{14}+x_{15} & \wedge & \nu_1(\mathrm{r}_2) = x_7+x_{14}+x_{15} & \wedge\\
\nu_1(\mu_1(r_3)) = x_{21}+x_{22} & \wedge & \nu_1(r_3) = x_8+x_{22} & \wedge\\
\nu_1(\mu_1(r_4)) = x_{19}+x_{20} & \wedge & \nu_1(r_4) = x_{11}+x_{20} & \wedge\\
\nu_1(\mu_1(\mathrm{r}_5)) = x_{21}+x_{22} & \wedge & \nu_1(\mathrm{r}_5) = x_{12}+x_{22} & \wedge\\
\nu_1(\mu_1(\mathrm{r}_6)) = x_{19}+x_{20} & \wedge & \nu_1(\mathrm{r}_6) = x_9+x_{20} & \wedge\\
\nu_1(\mu_1(in_A)) = x_{13}+x_{14}+x_{15}+x_{16}+x_{17}+x_{18} & \wedge & \nu_1(in_A) = x_6+x_{15}+x_{18} & \wedge\\
\nu_1(\mu_1(out_B)) = x_{16}+x_{17}+x_{18} & \wedge & \nu_1(out_B) = x_5+x_{17}+x_{18} & \wedge\\
\nu_1(\mu_1(out_C)) = x_{13}+x_{14}+x_{15} & \wedge & \nu_1(out_C) = x_4+x_{14}+x_{15} & \wedge\\
\nu_2(\mu_1(\mathrm{r}_1)) = x_1+x_{14}+x_{16} & \wedge & \nu_2(\mathrm{r}_1) = x_1+x_{10}+x_{14} & \wedge\\
\nu_2(\mu_1(\mathrm{r}_2)) = x_0+x_{13}+x_{17} & \wedge & \nu_2(\mathrm{r}_2) = x_0+x_7+x_{17} & \wedge\\
\nu_2(\mu_1(r_3) = x_3+x_{21} & \wedge & \nu_2(r_3) = x_3+x_8 & \wedge\\
\nu_2(\mu_1(r_4)) = x_2+x_{19} & \wedge & \nu_2(r_4) = x_2+x_{11} & \wedge\\
\nu_2(\mu_1(\mathrm{r}_5)) = x_3+x_{21} & \wedge & \nu_2(\mathrm{r}_5) = x_3+x_{12} & \wedge\\
\nu_2(\mu_1(r_6)) = x_2+x_{19} & \wedge & \nu_2(\mathrm{r}_6) = x_2+x_9 & \wedge\\
\nu_2(\mu_1(in_A)) = x_0+x_1+x_{13}+x_{14}+x_{16}+x_{17} & \wedge & \nu_2(in_A) = x_0+x_1+x_6 & \wedge\\
\nu_2(\mu_1(out_B)) = x_1+x_{14}+x_{16} & \wedge & \nu_2(out_B) = x_1+x_5+x_{14} & \wedge\\
\nu_2(\mu_1(out_C)) = x_0+x_{13}+x_{17} & \wedge & \nu_2(out_C) = x_0+x_4+x_{17} &
\end{array}\right)$$

Figure 17: Characterizing the difference abstraction for Δ_3 of the linear equation system in Figure 4 derived from the acyclic reaction network in Figure 3.

Lemma 41 shows for a pair of fresh variable generators $\nu = (\nu_1, \nu_2)$ and fresh variable generator μ_1 that:

$$\begin{array}{rcl}
N(map^2_\nu(\boldsymbol{y})) & = & dec_\nu(apply^m_{1,1}(msp, fot^2 \circ L)(map^2_\nu(\boldsymbol{y})))\\
& = & \exists map_{\mu_1}(\boldsymbol{y}).\exists map^2_\nu(map_{\mu_1}(\boldsymbol{y}))\,.\\
& & \bigwedge_{Y\in\{\boldsymbol{y}\}} \left(\begin{array}{ll} \nu_1(\mu_1(Y)) + \nu_2(Y) = \nu_2(\mu_1(Y)) + \nu_1(Y) & \wedge\\ \nu_1(Y) * \nu_2(Y) = 0 & \end{array}\right) \wedge\\
& & \bigwedge_{i\in\{1,2\}} \left(\begin{array}{l} \nu_i(\mu_1(in_A)) = \nu_i(\mu_1(r_1)) + \nu_i(\mu_1(r_2))\\ \nu_i(\mu_1(r_1)) = \nu_i(\mu_1(out_B))\\ \nu_i(\mu_1(r_2)) = \nu_i(\mu_1(out_C))\\ \nu_i(\mu_1(r_4)) = \nu_i(\mu_1(r_6))\\ \nu_i(\mu_1(r_3)) = \nu_i(\mu_1(r_5)) \end{array}\right)
\end{array}$$

The $h_{\mathbb{B}}$-exact rewriting of the simple $h_{\mathbb{B}}$-mixed system $N(map^2_\nu(\boldsymbol{y}))$ is as the simple $h_{\mathbb{B}}$-mixed system $M(map^2_\nu(\boldsymbol{y}))$ in Figure 17, that we computed with BioComputing's Reaction Network tool from the more than 22 elementary modes of the linear equation system. Note that the linear part of the equation system also contains the equations $\bigwedge_{Y\in\{\boldsymbol{y}\}} \nu_1(\mu_1(Y))+\nu_2(Y) = \nu_2(\mu_1(Y))+\nu_1(Y)$ originating from msp, and not only the two linear equation systems obtained by applying the first-order definition L, i.e., $L(map_{\nu_1}(map_{\mu_1}(\boldsymbol{y})))$ and $L(map_{\nu_2}(map_{\mu_1}(\boldsymbol{y})))$.

Theorem 4 induces a new algorithm for the exact computation of $h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y})))$. It inputs a system of linear equations $L(\mathbf{y}) \in \mathcal{F}_\Sigma$. By deviation over the first-order tuple logic,

it then computes the first-order logic formula $N(map^2_\nu(\mathbf{y})) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ where $N(map^2_\nu(\mathbf{y})) = dec_\nu \circ apply^m_{1,1}(msp, fot^2 \circ L)$. This formula is a simple $h_\mathbb{B}$-mixed system with $2m$ variables and size in $O(2n)$. One then computes the $h_\mathbb{B}$-exact rewriting $M(map^2_\nu(\mathbf{y}))$ of $N(map^2_\nu(\mathbf{y}))$. By Theorem 4 this can be done in time $O(2^{2m} poly(n))$ and yields an $h_\mathbb{B}$-exact simple $h_\mathbb{B}$-mixed system of size $O(n + 2^{2m}m)$ with $2m$ variables.

Therefore, the boolean solution set $sol^\mathbb{B}(M(\mathbf{y}))$ can then be computed from $M(\mathbf{y})$ by a naive generate and test algorithm in time $O(2^{2m}(n+2^{2m}m))$, which requires time in $O(2^{2m}n+2^{4m}m)$. In practice, we can improve this algorithm by computing the set $sol^\mathbb{B}(M(map^2_\nu(\mathbf{y})))$ from by finite domain constraint programming, rather than by a naive generate and test algorithm.

The set $diff(sol^\mathbb{B}(M(map^2_\nu(\mathbf{y}))))$ can be generated by enumerating all pairs of solutions in $sol^\mathbb{B}(M(map^2_\nu(\mathbf{y})))$. This can be done in time $O(2^{2m}m)$. The overall time is thus in $O(2^{2m} poly(n)+ 2^{2m}n + 2^{4m}m)$. Since $m \leq n$, the overall time for computing $h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y})))$ is in $O(2^{4m} poly(n))$. Note that this upper bound is simply exponential in the worst case, such as the alternative algorithm sketched previously in Section 2.

9.2. Exact Difference Abstraction for Δ_6

For the difference abstraction of concrete differences in $\mathbb{R}^2_+$ to Δ_6, we follow the same approach than for Δ_3, but need a considerably more evolved usage of first-order definitions.

We start with a decomposition of the difference abstraction h_{Δ_6} into the difference abstraction for h_{Δ_3} and the boolean abstraction $h_\mathbb{B}$. For this, we consider the algebra of total functions of type $\mathbb{R}^2_+ \to \mathbb{R}^2_+$ with addition. We define for any two functions $f, f' : \mathbb{R}^2_+ \to \mathbb{R}^2_+$ the function $f +^{\mathbb{R}^2_+ \to \mathbb{R}^2_+} f'$ such that for any pair $p \in \mathbb{R}^2_+$:

$$(f +^{\mathbb{R}^2_+ \to \mathbb{R}^2_+} f')(p) \quad = \quad f(p) +^{\mathbb{R}^2_+} f'(p)$$

The following lemma shows that h_{Δ_6} is the sum of h_{Δ_3} and $h^2_\mathbb{B}$ in this function algebra:

Lemma 43. *$h_{\Delta_6} = h^2_\mathbb{B} + h_{\Delta_3}$ where $+ = +^{\mathbb{R}^2_+ \to \mathbb{R}^2_+}$.*

Proof Let $p = (r, r') \in \mathbb{R}^2_+$. We distinguish the cases for all possible values for $h_{\Delta_6}(p) \in \Delta_6$.

Case $h_{\Delta_6}(p) = \uparrow$**.** Then $0 < r < r'$ so that $h^2_\mathbb{B}(p) = (1,1)$ and $h_{\Delta_3}(p) = \triangle = (0,1)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (1,2) = \uparrow = h_{\Delta_6}(p)$.

Case $h_{\Delta_6}(p) = \Uparrow$**.** Then $0 = r < r'$ so that $h^2_\mathbb{B}(p) = (0,1)$ and $h_{\Delta_3}(p) = \triangle = (0,1)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (0,2) = \Uparrow = h_{\Delta_6}(p)$.

Case $h_{\Delta_6}(p) = \downarrow$**.** Then $0 < r' < r$ so that $h^2_\mathbb{B}(p) = (1,1)$ and $h_{\Delta_3}(p) = \triangledown = (1,0)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (2,1) = \downarrow = h_{\Delta_6}(p)$.

Case $h_{\Delta_6}(p) = \Downarrow$**.** Then $0 = r' < r$ so that $h^2_\mathbb{B}(p) = (1,0)$ and $h_{\Delta_3}(p) = \triangledown = (1,0)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (2,0) = \Downarrow = h_{\Delta_6}(p)$.

Case $h_{\Delta_6}(p) = \sim$**.** Then $0 < r = r'$ so that $h^2_\mathbb{B}(p) = (1,1)$ and $h_{\Delta_3}(p) = \approxeq = (0,0)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (1,1) = \sim = h_{\Delta_6}(p)$.

Case $h_{\Delta_6}(p) = \approx$**.** Then $0 = r = r'$ so that $h^2_\mathbb{B}(p) = (0,0)$ and $h_{\Delta_3}(p) = \approxeq = (0,0)$. It follows that $(h^2_\mathbb{B} + h_{\Delta_3})(p) = (0,0) = \approx = h_{\Delta_6}(p)$.

□

We next introduce a total function of type $\mathbb{R}^2_+ \to (\mathbb{R}^2_+ \times \mathbb{R}^2_+)$ which maps each concrete difference in $\mathbb{R}^2_+$ to a pair of itself and its minimal support projection. We define this function

as a ternary relation of type $\mathbb{R}_+^2 \times (\mathbb{R}_+^2 \times \mathbb{R}_+^2)$ in the first-order pair logic. For this we use the first-order definition $\mathit{id\text{-}msp} : \mathcal{V} \times (\mathcal{V} \times \mathcal{V}) \to \mathcal{F}_\Sigma^2$ that satisfies for all $x, y_1, y_2 \in \mathcal{V}$:

$$\mathit{id\text{-}msp}(x, y_1, y_2) =_{\text{def}} \mathit{fot}^2(x = y_1) \wedge \mathit{msp}(x, y_2)$$

The relation defined by $\mathit{id\text{-}msp}$ is indeed the total function that maps any concrete difference $p \in \mathbb{R}_+^2$ to a pair of itself and its minimal support projection:

$$2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{id\text{-}msp})(p) = (p, 2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{msp})(p))$$

Furthermore, we define for any two functions $g : A \to B \times C$ and $f : B \times C \to D$ the pseudo composition $f \bullet g : A \to D$ such that for all $a \in A$:

$$(f \bullet g)(a) = f(\pi_1(g(a)), \pi_2(g(a)))$$

The homomorphism $h_{\mathbb{B}}^2 : \mathbb{R}_+^2 \to \mathbb{B}^2$ allows us to define another homomorphism $(h_{\mathbb{B}}^2)^2 : (\mathbb{R}_+^2)^2 \to (\mathbb{B}^2)^2$ by Lemma 10, that we can use for decomposing h_{Δ_6} as follows.

Lemma 44 Decomposition. $h_{\Delta_6} = +^{\mathbb{R}_+^2} \bullet (h_{\mathbb{B}}^2)^2 \circ 2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{id\text{-}msp})$.

Proof For any $p \in \mathbb{R}_+^2$, we have:

$$\begin{array}{rcl} h_{\Delta_6}(p) & = & h_{\mathbb{B}}^2(p) +^{\mathbb{R}_+^2} h_{\mathbb{B}}^2(2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{msp})(p)) \\ & = & +^{\mathbb{R}_+^2}(h_{\mathbb{B}}^2(p), h_{\mathbb{B}}^2(2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{msp})(p))) \\ & = & +^{\mathbb{R}_+^2}((h_{\mathbb{B}}^2)^2(p, 2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{msp})(p))) \\ & = & +^{\mathbb{R}_+^2}((h_{\mathbb{B}}^2)^2(\mathit{id}(p), 2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{msp})(p))) \\ & = & +^{\mathbb{R}_+^2} \bullet (h_{\mathbb{B}}^2)^2 \circ 2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{id\text{-}msp})(p) \end{array}$$

□

For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}_\Sigma^2$ we introduced in Section 8.6 the first-order application $\mathit{apply}_{1,2}^m(\mathit{id\text{-}msp}, G) : \mathcal{V}^m \to \mathcal{F}_\Sigma^2$ that defines in the first-order pair logic the application of the function defined by $\mathit{id\text{-}msp}$ to the m components of the relation defined by G.

Lemma 45 (First-order application of *id-msp*). *For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}_\Sigma^2$, sequence of pairwise distinct variables $\boldsymbol{y} = y_1 \dots y_m \in \mathcal{V}^*$, and pair of fresh variable generators $\mu = (\mu_1, \mu_2)$ where $\mu_1, \mu_2 : \mathcal{V} \to \mathcal{V}$:*

$$2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{id\text{-}msp}) \circ 2\text{-}\mathit{sol}^{\mathbb{R}_+}(G(\boldsymbol{y})) = \left\{ \begin{array}{c} [y_i/\alpha(\mu_1(y_i)), \alpha(\mu_2(y_i)) \mid 1 \le i \le m] \mid \\ \alpha \in 2\text{-}\mathit{sol}^{\mathbb{R}_+}(\mathit{apply}_{1,2}^m(\mathit{id\text{-}msp}, G)(\mathit{map}_\mu^2(\boldsymbol{y}))) \end{array} \right\}$$

Proof This lemma is a consequence of Proposition 38 on first-order definitions of function applications with $\ell = 2$, $n = 2$, $F = \mathit{id\text{-}msp}$, $S = \mathbb{R}_+$, and the fact that $2\text{-}\mathit{rel}^{\mathbb{R}_+}(\mathit{id\text{-}msp})$ is a total function by Lemma 44. □

Theorem 5 (Characterization of difference abstraction to Δ_6). *Let μ and ν be two pairs of fresh variable generators. For any system of linear equations $L(\boldsymbol{y})$ with a sequence of m distinct free variable $\boldsymbol{y} \in \mathcal{V}^*$ and size n we can compute in time $O(2^{4m}\mathit{poly}(n))$ an $h_{\mathbb{B}}$-exact $h_{\mathbb{B}}$-mixed system $\tilde{M}(\mathit{map}_\nu^2(\mathit{map}_\mu^2(\boldsymbol{y}))) \in \mathcal{F}_\Sigma$ with $4m$ free variables and of size $O(n + 2^{4m}m)$ such that:*

$$h_{\Delta_6} \circ \mathit{diff}(\mathit{sol}^{\mathbb{R}_+}(L(\boldsymbol{y}))) = \left\{ \begin{array}{c} [y/(\alpha^2(\mathit{map}_\nu^2(\mu_1(y))) +^{\mathbb{R}_+^2} \alpha^2(\mathit{map}_\nu^2(\mu_2(y))) \mid y \in \{\boldsymbol{y}\}] \mid \\ \alpha \in \mathit{sol}^{\mathbb{B}}(\tilde{M}(\mathit{map}_\nu^2(\mathit{map}_\mu^2(\boldsymbol{y})))) \end{array} \right\}$$

Furthermore, $\tilde{M}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ can be chosen as the $h_{\mathbb{B}}$-exact rewriting of the following simple $h_{\mathbb{B}}$-mixed system:

$$\tilde{N}(map^2_\nu(map^2_\mu(\boldsymbol{y}))) = dec_\nu(fot^2(L(\mu_1(\boldsymbol{y}))) \wedge \bigwedge_{i=1}^{m} msp(\mu_1(y_i), \mu_2(y_i))$$

Proof Let $L : \mathcal{V}^m \to \mathcal{F}_\Sigma$ be the first-order definition which when applied to $\mathbf{y}$ returns the linear formula $L(\mathbf{y})$.

$$\begin{array}{rl}
 & h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y}))) \\
\textit{Proposition 18} & = h_{\Delta_6} \circ sol^{\mathbb{R}^2_+}(L(\mathbf{y})) \\
\textit{Proposition 30} & = h_{\Delta_6} \circ 2\text{-}sol^{\mathbb{R}_+}(L^2(\mathbf{y})) \text{ with } L^2 = fot^2 \circ L \\
\textit{Decomp. Lemma 44} & = +^{\mathbb{R}^2_+} \bullet (h^2_{\mathbb{B}})^2 \circ 2\text{-}rel^{\mathbb{R}^2_+}(id\text{-}msp) \circ 2\text{-}sol^{\mathbb{R}_+}(L^2(\mathbf{y})) \\
\textit{First-order application of id-msp: Lemma 45} & = +^{\mathbb{R}^2_+} \bullet (h^2_{\mathbb{B}})^2 \circ \left\{ \begin{array}{l} [y_i/(\alpha(\mu_1(y_i)), \alpha(\mu_2(y_i))) \mid 1 \le i \le n] \mid \\ \quad \alpha \in 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,2}(id\text{-}msp, L^2)(map^2_\mu(\mathbf{y}))) \end{array} \right\} \\
 & = \left\{ \begin{array}{l} [y_i/\beta(\mu_1(y_i)) +^{\mathbb{R}^2_+} \beta(\mu_2(y_i)) \mid 1 \le i \le n] \mid \\ \quad \beta \in h^2_{\mathbb{B}} \circ 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,2}(id\text{-}msp, L^2)(map^2_\mu(\mathbf{y})))\} \end{array} \right\}
\end{array}$$

We can compute the $h^2_{\mathbb{B}}$ abstraction of the above solution set similarly to the case of Δ_3:

$$\begin{array}{rl}
 & h^2_{\mathbb{B}} \circ 2\text{-}sol^{\mathbb{R}_+}(apply^m_{1,2}(id\text{-}msp, L^2)(map^2_\mu(\mathbf{y}))) \\
\textit{Proposition 32} & = h^2_{\mathbb{B}} \circ cmp_\nu(sol^{\mathbb{R}_+}(dec_\nu(apply^m_{1,2}(id\text{-}msp, L^2)(map^2_\mu(\mathbf{y}))))) \\
\textit{Proposition 35} & = cmp_\nu(h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(dec_\nu(apply^m_{1,2}(id\text{-}msp, L^2)(map^2_\mu(\mathbf{y}))))) \\
\textit{Definition of } apply^m_{1,2} & = cmp_\nu(h_{\mathbb{B}} \circ sol^{\mathbb{R}_+}(dec_\nu(\exists \mathbf{y}.\ L^2(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^m id\text{-}msp(y_i, \mu_1(y_i), \mu_2(y_i))))) \\
\textit{Theorem 3 on } h_{\mathbb{B}}\textit{-exact rewriting} & = cmp_\nu(sol^{\mathbb{B}}(\tilde{M}(\mathbf{z}))) \\
 & \quad \textit{where } \mathbf{z} = map^2_\nu(map^2_\mu(\mathbf{y})) \text{ and } \tilde{M}(\mathbf{z}) \textit{ is an } h_{\mathbb{B}}\textit{-exact simple} \\
 & \quad h_{\mathbb{B}}\textit{-mixed system equivalent to the simple } h_{\mathbb{B}}\textit{-mixed system} \\
 & \qquad \tilde{N}'(\mathbf{z}) = dec_\nu(\exists \mathbf{y}.\ L^2(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^m id\text{-}msp(y_i, \mu_1(y_i), \mu_2(y_i))) \\
 & \quad \textit{and thus to the equivalent simple } h_{\mathbb{B}}\textit{-mixed system} \\
 & \qquad \tilde{N}(\mathbf{z}) = dec_\nu(L^2(\mu_1(\mathbf{y})) \wedge \textstyle\bigwedge_{i=1}^m msp(\mu_1(y_i), \mu_2(y_i)))
\end{array}$$

The combination of the above two calculations and the moving of $\beta = cmp_\nu(\alpha)$ to the left yields the equation stated in the theorem. By the $h_{\mathbb{B}}$-mixed systems Theorem 3, the size of $\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y})))$ is in $O(n + 2^{4m}m)$ and the time of its computation in $O(2^{4m}poly(n))$. □

Note that up to existential quantification, the first-order definitions N for Δ_3 and $\tilde{N}$ for Δ_6 are the same:

$$\exists map^2_\mu(map_{\mu_2}(\mathbf{y})).\ \tilde{N}(map^2_\nu(map^2_\mu(\mathbf{y}))) = N(map^2_\nu(map_{\mu_1}(\mathbf{y})))$$

Compared to the formula $N(map^2_\nu(\mathbf{y}))$ from Theorem 4, the variables in $map^2_\nu(\mathbf{y})$ are renamed to $map^2_\nu(map_{\mu_1}(\mathbf{y}))$ in Theorem 5, and the variables in $map^2_\nu(map_{\mu_2}(\mathbf{y}))$ are existentially quantified.

Lemma 46. *For any first-order definition $L : \mathcal{V}^m \to \mathcal{F}_\Sigma$, sequence of variables $\boldsymbol{y}$, and pair of new variables generators $\nu = (\nu_1, \nu_2)$ and $\mu = (\mu_1, \mu_2)$, the simple $h_{\mathbb{B}}$-mixed system $\tilde{N}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ from Theorem 5 has the following form:*

$$\tilde{N}(map^2_\nu(map^2_\mu(\boldsymbol{y}))) \quad = \quad \textstyle\bigwedge_{Y \in \{\boldsymbol{y}\}} dec_\nu(msp(\mu_1(Y), \mu_2(Y))) \wedge \bigwedge_{i \in \{1,2\}} L(map_{\nu_i}(map_{\mu_1}(\boldsymbol{y})))$$

Proof Analogous to the proof of Lemma 41 based on the above equality.

Example 47. *Reconsider the reaction network from the example in Figure 3 leading to the linear equation system $L(\boldsymbol{y})$ in Figure 4. According to Lemmas 46, we can choose:*

$$\tilde{M}(map^2_\nu(map^2_\mu(\boldsymbol{y}))) = \bigwedge_{Y\in\{\boldsymbol{y}\}} \nu_1(\mu_2(Y)) * \nu_2(\mu_2(Y)) = 0 \wedge M''(map^2_\nu(map^2_\mu(\boldsymbol{y})))$$

where M'' is the first-order definition given in Figure 17 but here applied to different variables.

Note that upper complexity bound of Theorem 5 is slightly different to that of Theorem 4, since we have to create $4m$ variables for Δ_6, in contrast to $2m$ variables for Δ_3. Theorem 5 induces a new algorithm for the exact computation of $h_{\Delta_6} \circ \mathit{diff}(sol^{\mathbb{R}_+}(L(\mathbf{y})))$. It requires time in $O(poly(n)2^{8m})$ where n is the size of $L(\mathbf{y})$. Hence it is simply exponential in the worst case, such as the existing algorithm sketched in Section 2. But now we can use finite domain constraint programming to avoid the naive generate and test approach. This will prove benefical in practice.

10. Computing Difference Abstractions with Constraints

We show how to compute difference abstraction exactly in the presence of constraints for both h_{Δ_3} and h_{Δ_6}.

10.1. Problem of Exact Abstraction with Constraints

We first generalize the difference abstractions h_{Δ_3} and h_{Δ_6} and the boolean abstraction $h_{\mathbb{B}}$ into a comon concept of a $\Sigma_{pos\text{-}arith}$-abstractions.

Definition 48. *A Σ-abstraction is a homomorphism between Σ-structures S and Δ such that $dom(\Delta) \subseteq dom(S)$.*

We assume that $dom(\Delta) \subseteq dom(S)$, since we want to talk at the same time about concrete values in $dom(S)$ and abstract values in $dom(\Delta)$ by first-order formulas interpreted over the Σ-structure S.

We can now define the algorithmic problem of *exact abstraction with constraints* for any Σ-abstraction. For this we recall that $\Sigma[dom(\Delta)]$ is the extension of signature Σ with additional constants from $dom(\Delta)$.

Definition 49. *Let $h : S \to \Delta$ be a Σ-abstraction into some finite Σ-structure Δ. The algorithmic problem of* exact abstraction with constraints with parameter h *has the following three inputs:*

System of linear equations $L \in \mathcal{F}_\Sigma$: *this system is to be interpreted over S.*

Constraint $C \in \mathcal{F}_{\Sigma[dom(\Delta)]}$: *a first-order formula which is to be interpreted over Δ.*

Set of observable variables $V \subseteq \mathcal{V}$: *a finite subset of variables that may be observed.*

The output is the set $abs^h_{V,C}(L)$ of abstractions of S-solutions of L by h, constrained to the Δ-solutions of C, and projected to the observable variables in V:

$$abs^h_{V,C}(L) = \{\beta_{|V} \mid \beta \in h \circ sol^S_{fv(C)}(L) \cap sol^\Delta_{fv(L)}(C)\}$$

In the example of the introductory reaction network in Figure 3, we consider the abstraction $h = h_{\Delta_6}$, and thus, $\Sigma = \Sigma_{pos\text{-}arith}$, $S = \mathbb{R}^2_+$, and $\Delta = \Delta_6$. The system of linear equations $L \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ is given in Figure 4. As non nonlinear constraint $C \in \mathcal{F}_{\Sigma_{pos\text{-}arith}[dom(\Delta_6)]}$ we can choose the kinetic constraints in Figure 4 in conjunction with the overproduction target $r_{out_B} = \uparrow$.

As set of observable variables, we may choose the variable for the speed of the inflow of A and the speed of the reactions subject to knockout 3 and 4.

$$V = \{in_A, r_3, r_4\}$$

In contrast to the system of linear equation system L, the constraint C may contain arbitrary arithmetic formulas including non-linear polynomial equations and universal quantifiers. This is needed to deal with nonlinear kinetic information. This does not make increase the difficulty of the problem to much, since the constraints are to be interpreted over the finite structure Δ, so that the universal quantifiers in C can be replaced by conjunctions, and the existential quantifieres by disjunctions.

The general problem could be simplified, if we could compute an h-exact formula M that is S-equivalent to L, since in this case:

$$abs^h_{V,C}(L) = sol^{\Delta}(\exists V.\ M \wedge C)$$

Furthermore, the solution set $sol^{\Delta}(\exists V.\ M \wedge C)$ can directely be computed by finite domain constraint programming. An exact rewriting exists for instance for the boolean abstraction as show in Theorem 3. So this simpler presentation of $abs^h_{V,C}(L)$ can be applied to solve the problem of exact boolean abstraction with constraints by finite domain constraint programming.

The characterizations of the difference abstractions h_{Δ_3} and h_{Δ_6} in Theorems 4, and 5, however, do not provide any such exact rewriting. So the question how to solve the problems of exact difference abstraction with constraints for h_{Δ_3} and h_{Δ_6} remain to be solved.

10.2. Mixed Structures: $\mathbb{B} \cup \Delta_3$ *and* $\mathbb{B} \cup \Delta_6$

For adding a treatment of kinetic constraints over Δ_n where $n \in \{3, 6\}$, we consider the union $\mathbb{B} \cup \Delta_n$ as a relational structure, unifying the functionalities of both structures $\mathbb{B}$ and Δ_n. For this, we defined the mixed signature by:

$$\Sigma^{mixed}_n = \{+^{\mathbb{B}}, *^{\mathbb{B}}, +^{\Delta_n}, *^{\Delta_n}\} \cup \mathbb{B} \cup \Delta_n$$

Here we reuse the binary functions of $\mathbb{B}$ and Δ_n as the binary function symbols of Σ^{mixed}_n and the values of $\mathbb{B} \cup \Delta_n$ as the constants of Σ^{mixed}_n. The interpretation of the structure Σ^{mixed}_n is such that each symbol is interpreted by itself. In particular, the constant 0 is *not* interpreted as 0^{Δ_n} but as:

$$0^{\mathbb{B}\cup\Delta_n} = 0 \in \mathbb{B}$$

Definition 50. *For any $n \in \{3, 6\}$, the mixed structure $\mathbb{B} \cup \Delta_n$ is the Σ^{mixed}_n-structure with the mixed domain $\mathbb{B} \cup \Delta_n$ in which all symbols of Σ^{mixed}_n are by themselves, but now with respect to the mixed domain.*

Since we will work on the mixed structure $\mathbb{B} \cup \Delta_n$, we need to state clearly whether we want to interpret symbols from $\Sigma_{pos\text{-}arith}$ over $\mathbb{B}$ or Δ_n. In order to do so, we introduce for any formula $\phi \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ and $\Sigma_{pos\text{-}arith}$-structure $S \in \{\mathbb{B}, \Delta_n\}$, a formula $in^S(\phi) \in \mathcal{F}_{\Sigma^{mixed}_n}$ by replacing the operators $+$ and $*$ from $\Sigma_{pos\text{-}arith}$ in expressions of ϕ by $*^S$ and $+^S$ and the constants $c \in \{0, 1\}$ by c^S. So we have for all operators $\odot F^{(2)}_{pos\text{-}arith}$, constant $c \in C_{pos\text{-}arith}$, expressions $e, e' \in \mathcal{E}_{\Sigma_{pos\text{-}arith}}$, and for all formulas $\phi, \phi' \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$:

$$\begin{array}{ll} in^S(e \odot e') = in^S(e) \odot^S in^S(e') & in^S(c) = c^S \\ in^S(\phi \wedge \phi') = in^S(\phi) \wedge in^S(\phi') & in^S(\neg\phi) = \neg in^S(\phi) \\ in^S(\exists x.\phi) = \exists x. \bigvee_{d \in dom(S)} x = d \wedge in^S(\phi) & \end{array}$$

Lemma 51. *For any $n \in \{3, 6\}$, formula $\phi \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$, and structure $S \in \{\mathbb{B}, \Delta_n\}$:*

$$sol^S(\phi) = sol^{\mathbb{B}\cup\Delta_n}(in^S(\phi))$$

Proof Straightforward by induction on the structure of formulas.

10.3. Defining k-ary Function Applications in First-Order Logic

We next want to define applications of k-ary function on the domains of mixed structures in standard first-order logic. Which structures are used does not matter for the theory, so we consider the case of general Σ-structures. We could also develop the theory for the first-order tuple logic, but do not do so for simplicity.

In order to deal with the k inputs, we consider a tuple of new variable generators $\nu = (\nu_1, \ldots, \nu_k)$ where $\nu_1, \ldots, \nu_k : \mathcal{V} \to \mathcal{V}$. For any $k, m \in \mathbb{N}$ and first-order definitions $F : \mathcal{V}^k \to \mathcal{V}$ and $G : \mathcal{V}^{km} \to \mathcal{F}_\Sigma$, we define a first-order definition $apply^m_{k,1}(F, G) : \mathcal{V}^m \to \mathcal{F}_\Sigma$ that describes an application of $rel^S(F)$ to all m k-tuples of any km-tuple in the relation $rel^S(G)$, such that it satisfies for all $\mathbf{y} = y_1, \ldots, y_m \in \mathcal{V}^*$:

$$apply^m_{k,1}(F, G)(\mathbf{y}) = \exists map^k_\nu(\mathbf{y}).\ G(map^k_\nu(\mathbf{y})) \wedge \bigwedge_{i=1}^{m} F(\nu_1(y_i), \ldots, \nu_k(y_i), y_i)$$

Proposition 52. *Let $G : \mathcal{V}^{mk} \to \mathcal{F}_\Sigma$ and $F : \mathcal{V}^k \times \mathcal{V} \to \mathcal{F}_\Sigma$ be first-order definitions, S a Σ-algebra, and $D = dom(S)^n$ the domain of S^n, such that the relation $n\text{-}rel^S(F)$ is a partial function of type $D^k \times D$. Then for any sequence of pairwise distinct variables $\boldsymbol{y} \in \mathcal{V}^m$ and any tuple of fresh variable generators $\nu = (\nu_1, \ldots, \nu_k)$ the sequence of variables $map^k_\nu(\boldsymbol{y})$ satisfies:*

$$rel^S(F) \circ cmp_\nu(sol^S(G(map^k_\nu(\boldsymbol{y})))) = sol^S(apply^m_{k,1}(F, G)(\boldsymbol{y}))$$

The proof will be given in Section 13. It will follow from the more general Proposition 62, which lifts the present Proposition 52 to the first-order tuple logic.

10.4. Exact Difference Abstraction with Constraints for Δ_3

We have $\Delta_3 = \mathbb{B}^2 \setminus (1, 1)$, so any pair of booleans but $(1, 1)$ is an element in Δ_3, and vice versa. For solving the general problem of exact abstraction with constraints for h_{Δ_3} we need to map pairs of Booleans to Δ_3 in the first-order logic of the mixed structure $\mathbb{B} \cup \Delta_3$. For this we consider the partial function $pair_{\Delta_3} \subseteq (\mathbb{B} \cup \Delta_3)^2 \times (\mathbb{B} \cup \Delta_3)$ such that for all $v_1, v_2 \in \mathbb{B} \cup \Delta_3$:

$$pair_{\Delta_3}(v_1, v_2) = \begin{cases} (v_1, v_2) & \text{if } v_1 *^{\mathbb{B}} v_2 = 0 \text{ so that } (v_1, v_2) \in \Delta_3 \\ \text{undefined} & \text{else} \end{cases}$$

The domain of this partial function is $dom(pair_{\Delta_3}) = \mathbb{B}^2 \setminus \{(1, 1)\} = \Delta_3$. Any pair from the domain is mapped to itself. We can define the ternary relation $pair_{\Delta_3}$ in the first-order logic of the mixed structure $B \cup \Delta_3$ by the function $Pair_{\Delta_3} : \mathcal{V}^2 \times \mathcal{V} \to \mathcal{F}_{\Sigma^{mixed}_3}$ such that for all variables $y_1, y_2, y \in \mathcal{V}$:

$$Pair_{\Delta_3}(y_1, y_2, y) \quad =_{\text{def}} \quad \begin{pmatrix} & (y_1 = 0 \wedge y_2 = 0 \wedge y = \approx) \\ \vee & (y_1 = 0 \wedge y_2 = 1 \wedge y = \triangle) \\ \vee & (y_1 = 1 \wedge y_2 = 0 \wedge y = \triangledown) \end{pmatrix}$$

The first-order definition $Pair_{\Delta_3}$ defines the relation $pair_{\Delta_3}$ in the mixed structure, that is $pair_{\Delta_3} = rel^{\mathbb{B}\cup\Delta_3}(Pair_{\Delta_3})$. We next consider definitions of applications of the binary function $pair_{\Delta_3}$ in the first-order logic of the mixed structure $\mathbb{B} \cup \Delta_3$.

Lemma 53. *Let $m \in \mathbb{N}$, $M : \mathcal{V}^{2m} \to \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ be a first-order definition, $\boldsymbol{y}$ be a sequence of distinct variables, and pair of fresh variable generators $\nu = (\nu_1, \nu_2)$:*

$$sol^{\mathbb{B}\cup\Delta_3}(apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\boldsymbol{y})) = cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\boldsymbol{y}))) \cap \{\alpha \mid \alpha : \{\boldsymbol{y}\} \to \Delta_3\}$$

Proof First note that solution sets over the booleans can also be computed in the mixed structure by using the conversion of formulas $in^{\mathbb{B}}$ as stated by Lemma 51:

$$sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))) = sol^{\mathbb{B}\cup\Delta_3}(in^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))))$$

The function $pair_{\Delta_3}$ maps all pairs of booleans in Δ_3 to themselves, while being undefined for all elements of $(\Delta_3 \cup \mathbb{B})^2 \setminus \Delta_3$. Hence:

$$cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))) \cap \{\alpha \mid \alpha : \{\mathbf{y}\} \to \Delta_3\} = pair_{\Delta_3} \circ cmp_\nu(sol^{\mathbb{B}\cup\Delta_3}(in^{\mathbb{B}}(M(map^2_\nu(\mathbf{y})))))$$

As stated in Section 3.3, the composition $pair_{\Delta_3} \circ \alpha$ is defined only for total functions α such that $ran(\alpha) \subseteq dom(pair_{\Delta_3}) = \Delta_3$. So a total function $pair_{\Delta_3} \circ \alpha$ may belong to the composition on the right only if α satisfies this condition.

Since $pair_{\Delta_3} = rel^{\mathbb{B}\cup\Delta_3}(Pair_{\Delta_3})$, Proposition 52 applied to the mixed structure $S = \mathbb{B} \cup \Delta_3$ and the first-order definition of the binary ($k = 2$) function $F = Pair_{\Delta_3} : \mathcal{V}^2 \times \mathcal{V} \to \mathcal{F}_{\Sigma_3^{mixed}}$ and the first-order definition $G = in^{\mathbb{B}} \circ M : \mathcal{V}^{2m} \times \mathcal{V} \to \mathcal{F}_{\Sigma_3^{mixed}}$ show that:

$$pair_{\Delta_3} \circ cmp_\nu(sol^{\mathbb{B}\cup\Delta_3}(in^{\mathbb{B}} \circ M(map^2_\nu(\mathbf{y})))) = sol^{\mathbb{B}\cup\Delta_3}(apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\mathbf{y}))$$

The combination of the two equations displayed above yields the lemma. □

The characterization of the difference abstraction for Δ_3 for linear equations systems from Theorem 4 can be specified in the first-order logic of the mixed structure $\mathbb{B} \cup \Delta_3$, leading to the following proposition.

Corollary 6 (of Theorem 4: Characterization of abstraction to Δ_3 in the mixed structure $B \cup \Delta_3$). *For any linear equation system $L(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ we can compute in at most exponential time a first-order formula in the mixed signature $G(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_3^{mixed}}$ such that:*

$$sol^{\mathbb{B}\cup\Delta_3}(G(\boldsymbol{y})) = h_{\Delta_3} \circ sol^{\mathbb{R}^2_+}(L(\boldsymbol{y}))$$

Furthermore, if $M(map^2_\nu(\boldsymbol{y}))$ is the $h_{\mathbb{B}}$-exact rewriting of the formula $N(map^2_\nu(\boldsymbol{y}))$ from Lemma 41 depending on $L(\boldsymbol{y})$, then we can choose:

$$G(\boldsymbol{y}) = apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\boldsymbol{y})$$

Proof Let ν be some pair a fresh variable generators. Let $M(map^2_\nu(\mathbf{y}))$ be the $h_{\mathbb{B}}$-exact rewriting of the formula $N(map^2_\nu(\mathbf{y}))$ from Lemma 41 depending on $L(\mathbf{y})$. By Theorem 4 we can compute the first-order definition $M : \mathcal{V}^{2m} \to \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ in at most exponential time, and it satisfies:

$$h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y}))) = cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y})))) \qquad (*)$$

Furthermore, Lemma 51 permits to reduce solution sets over the booleans to solutions sets in the mixed structure:

$$sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))) = sol^{\mathbb{B}\cup\Delta_3}(in^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))))$$

The formula $G(\mathbf{y})$ can then be computed in linear time from $M(map^2_\nu(\mathbf{y}))$. We need to show that it captures the difference abstraction by h_{Δ_3} of the solution set of $L(\mathbf{y})$. Lemma 53 on the first-order definition of $pair_{\Delta_3}$ and the equation $G(\mathbf{y}) = apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\mathbf{y})$ shows that:

$$sol^{\mathbb{B}\cup\Delta_3}(G(\mathbf{y})) = cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y})))) \cap \{\alpha \mid \alpha : \{\mathbf{y}\} \to \Delta_3\}$$

By equation $(*)$ it follows that:

$$cmp_\nu(sol^{\mathbb{B}\cup\Delta_3}(in^{\mathbb{B}}(M(map^2_\nu(\mathbf{y}))))) \subseteq \{\alpha \mid \alpha : \{\mathbf{y}\} \to \Delta_3\}$$

The corollary follows from the above equations, using $(*)$ in the last step:

$$sol^{\mathbb{B}\cup\Delta_3}(G(\mathbf{y})) = cmp_\nu(sol^{\mathbb{B}}(M(map^2_\nu(\mathbf{y})))) = h_{\Delta_3} \circ sol^{\mathbb{R}^2_+}(L(\mathbf{y}))$$

□

Example 54. *We continue the running example $L(\boldsymbol{y})$ where $\boldsymbol{y}$ is the sequence $r_1 ... r_6 in_A out_B out_C$ derived from the reaction network in Figure 4. in Example 42, we fixed a pair of fresh variable generators $\nu = (\nu_1, \nu_2)$ and computed the formula $M(map^2_\nu(\boldsymbol{y}))$ for characterizing the difference abstraction of $L(\boldsymbol{y})$ to Δ_3. We now want to characterize the difference abstraction of $L(\boldsymbol{y})$ to Δ_3 in the mixed structure $\mathbb{B} \cup \Delta_3$ via Corollary 6. For this the formula $M(\boldsymbol{y})$ is converted into the formula $G(\boldsymbol{y}) = apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\boldsymbol{y})$ over the mixed signature. This formula is equal to:*

$$G(\boldsymbol{y}) \quad = \quad \left(\begin{array}{l} \exists map^2_\nu(\boldsymbol{y}) \in \mathbb{B}. \\ \quad \bigwedge_{Y \in \{\boldsymbol{y}\}} Pair_{\Delta_3}(map^2_\nu(Y), Y) \wedge \nu_1(Y) *^B \nu_2(Y) = 0 \\ \quad \wedge\ in^{\mathbb{B}}(M'(map^2_\nu(\boldsymbol{y}))) \end{array} \right)$$

where the formula $M'(map^2_\nu(\boldsymbol{y})) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ is given in Figure 17.

Theorem 7 (Solving Exact Difference Abstraction with Constraints for h_{Δ_3}). *For any linear equation system $L(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$, constraint $C(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}[dom(\Delta_3)]}$, and finite subset of variables $V \subseteq \mathcal{V}$, we can compute in at most exponential time a formula over the mixed signature $H(\boldsymbol{y}) \in \mathcal{F}_{\Sigma^{mixed}_3}$ such that:*

$$sol^{\mathbb{B}\cup\Delta_3}(H(\boldsymbol{y})) = abs^{h_{\Delta_3}}_{V,C(\boldsymbol{y})}(L(\boldsymbol{y}))$$

Furthermore, if $M(map^2_\nu(\boldsymbol{y}))$ is the $h_{\mathbb{B}}$-exact rewriting of the formula $N(map^2_\nu(\boldsymbol{y}))$ from Lemma 41, then $H(\boldsymbol{y})$ can be defined as follows:

$$\begin{array}{rcl} G(\boldsymbol{y}) & = & apply^m_{2,1}(Pair_{\Delta_3}, in^{\mathbb{B}} \circ M)(\boldsymbol{y}) \\ H(\boldsymbol{y}) & = & \exists \{\boldsymbol{y}\} \setminus V.\ G(\boldsymbol{y}) \wedge in^{\Delta_3}(C(\boldsymbol{y})) \end{array}$$

Proof By Corollary 6 of Theorem 4 we can compute in at most exponential time the first-order definition $G : \mathcal{V}^{2m} \to \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ such that:

$$h_{\Delta_3} \circ diff(sol^{\mathbb{R}_+}(L)) = sol^{\mathbb{B}\cup\Delta_3}(G(\mathbf{y}))$$

Furthermore, Lemma 51 permits us to express any solution set over the structure Δ_3 by some solution set in the mixed structure $\mathbb{B} \cup \Delta_3$:

$$sol^{\mathbb{B}}(C(\mathbf{y})) = sol^{\mathbb{B}\cup\Delta_3}(in^{\Delta_3}(C(\mathbf{y})))$$

The formula $H(\mathbf{y}) \in \mathcal{F}_{\Sigma^{mixed}_3}$ then satisfies $sol^{\mathbb{B}\cup\Delta_3}(H(\mathbf{y})) = abs^{h_{\Delta_3}}_{V,C(\mathbf{y})}(L(\mathbf{y}))$ as required by the theorem. □

Example 55. *In the running example from Figure 4, the constraint imposed on the difference abstraction of $L(\boldsymbol{y})$ by h_{Δ_6} entails the following constraint $C(\boldsymbol{y})$ for Δ_3 that is to be imposed on the difference abstraction of $L(\boldsymbol{y})$ by h_{Δ_3}::*

$$C(\boldsymbol{y}) = \left(\begin{array}{llll} r_1 = A * D & \wedge & r_2 = A * E & \wedge \\ r_3 \in \{\bigtriangledown, \approx\} & \wedge & r_4 \in \{\bigtriangledown, \approx\} & \wedge \\ r_5 = D & \wedge & r_6 = E & \wedge \\ in_A \in \{\bigtriangledown, \approx, \bigtriangleup\} & \wedge & out_B = B & \wedge \\ out_C = C & & & \end{array} \right)$$

Theorem 7 shows that the abstraction $abs^{h_{\Delta_3}}_{V,C(\boldsymbol{y})}(L(\boldsymbol{y}))$ is the solution set $sol^{\mathbb{B}\cup\Delta_3}(H(\boldsymbol{y}))$ of the formula:

$$H(\boldsymbol{y}) = \exists\{\boldsymbol{y}\} \setminus V.\ (G(\boldsymbol{y}) \wedge in^{\Delta_3}(C(\boldsymbol{y}))$$

where $G(\boldsymbol{y})$ is given in Example 54.

The set $sol^{\mathbb{B}\cup\Delta_3}(H(\mathbf{y}))$ can be computed by a finite domain constraint programming from the formula $H(\mathbf{y})$, since the mixed structure $\mathbb{B} \cup \Delta_3$ is finite. Therefore, Theorem 7 yields an algorithm for solving the general problem of exact abstraction with constraints for the difference abstraction h_{Δ_3}.

10.5. Exact Difference Abstraction with Constraints for Δ_6

We next consider the partial function $\mathit{pair\text{-}sum}_{\Delta_6} \subseteq (\mathbb{B} \cup \Delta_6)^4 \times (\mathbb{B} \cup \Delta_6)$ that maps any two pairs of booleans to an abstract difference in Δ_6, so that for all $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{B}^2$:

$$\mathit{pair\text{-}sum}_{\Delta_6}(\mathbf{b}_1, \mathbf{b}_2) = \left\{ \begin{array}{ll} \mathbf{b}_1 +^{\mathbb{R}^2_+} \mathbf{b}_2 & \text{if } \mathbf{b}_1, \mathbf{b}_2 \in \mathbb{B}^2 \text{ and } \mathbf{b}_1 +^{\mathbb{R}^2_+} \mathbf{b}_2 \in \Delta_6 \\ \text{undefined} & \text{else} \end{array} \right.$$

The partial function $\mathit{pair\text{-}sum}_{\Delta_6}$ is used implicitely in Theorem 5. To see this, let μ, ν be two pairs of fresh variable generators. Theorem 5 states that any linear equation system $L(\mathbf{y})$ can be rewritten to some simple $h_{\mathbb{B}}$-mixed system $\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y})))$ such that:

$$h_{\Delta_6} \circ \mathit{diff}(sol^{\mathbb{R}_+}(L(\mathbf{y}))) = \left\{ \begin{array}{c} [y/\mathit{pair\text{-}sum}_{\Delta_6}(\beta^2(map^2_\nu(\mu_1(y))), \beta^2(map^2_\nu(\mu_2(y)))) \mid y \in \{\mathbf{y}\}] \mid \\ \beta \in sol^{\mathbb{B}}(\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y})))) \end{array} \right\}$$

We next define the relations on pairs $\mathit{pair\text{-}sum}_{\Delta_6}$ in the pair logic over the mixed structure $\mathbb{B} \cup \Delta_6$ by the function $\mathit{Pair\text{-}Sum}_{\Delta_6} : \mathcal{V}^4 \times \mathcal{V} \in \mathcal{F}_{\Sigma^{mixed}_6}$ such that for all variables $y_1, y_2, y_3, y_4, y \in \mathcal{V}$:

$$\begin{array}{rl} \mathit{Pair\text{-}Sum}_{\Delta_6}(y_1, y_2, y_3, y_4, y) =_{\text{def}} & (y_1 = 0 \wedge y_2 = 0 \wedge y_3 = 0 \wedge y_4 = 0 \wedge y = \approx) \\ & \vee (y_1 = 1 \wedge y_2 = 1 \wedge y_3 = 0 \wedge y_4 = 0 \wedge y = \sim) \\ & \vee (y_1 = 1 \wedge y_2 = 0 \wedge y_3 = 0 \wedge y_4 = 1 \wedge y = \Uparrow) \\ & \vee (y_1 = 1 \wedge y_2 = 1 \wedge y_3 = 0 \wedge y_4 = 1 \wedge y = \uparrow) \\ & \vee (y_1 = 0 \wedge y_2 = 1 \wedge y_3 = 1 \wedge y_4 = 0 \wedge y = \Downarrow) \\ & \vee (y_1 = 1 \wedge y_2 = 1 \wedge y_3 = 1 \wedge y_4 = 0 \wedge y = \downarrow) \end{array}$$

According to Definition 36, the first-order definition $\mathit{Pair\text{-}Sum}_{\Delta_6}$ indeed defines the partial function $\mathit{pair\text{-}sum}_{\Delta_6}$ in the mixed structure, that is: $\mathit{pair\text{-}sum}_{\Delta_6} = rel^{\mathbb{B}\cup\Delta_6}(\mathit{Pair\text{-}Sum}_{\Delta_6})$. The next lemma shows how to apply $\mathit{Pair\text{-}Sum}_{\Delta_6}$ to all 4 m-tuples of a tuple in some $4m$-ary relation.

Lemma 56. *For any first-order definition $O : \mathcal{V}^{4m} \to \mathcal{F}_{\Sigma_6^{mixed}}$, sequence of distinct variables $\boldsymbol{y} \in \mathcal{V}^m$, pairs of new variable generators $\mu = (\mu_1, \mu_2)$ and $\nu = (\nu_1, \nu_2)$, the solution set of the first-order application $apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, O)(\boldsymbol{y})$ over the mixed structure $\mathbb{B} \cup \Delta_6$ is equal to:*

$$sol^{\mathbb{B}\cup\Delta_6}(apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, O)(\boldsymbol{y})) = \left\{ \begin{array}{c} [y/pair\text{-}sum_{\Delta_6}(\beta^2(map^2_\nu(\mu_1(y))), \beta^2(map^2_\nu(\mu_2(y)))) \mid y \in \{\boldsymbol{y}\}] \\ \mid \beta \in sol^{\mathbb{B}\cup\Delta_6}(O(map^2_\nu(map^2_\mu(\boldsymbol{y})))) \end{array} \right\}$$

Proof We use the fact that $pair\text{-}sum_{\Delta_6} = rel^{\mathbb{B}\cup\Delta_6}(Pair\text{-}Sum_{\Delta_6})$ and the general property of first-order definitions from Proposition 52 with $F = Pair\text{-}Sum_{\Delta_6}$, $G = O$, $\ell = 4$, $k = 1$ and $n = 1$. As four new variable generators there we use $\{\nu_i \circ \mu_j \mid i, j \in \{1, 2\}\}$.

$$sol^{\mathbb{B}\cup\Delta_6}(apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, O)(\mathbf{y})) = pair\text{-}sum_{\Delta_6} \circ \left\{ \begin{array}{c} \alpha : \{\mathbf{y}\} \to \mathbb{B} \cup \Delta_6 \mid \exists \beta \in sol^{\mathbb{B}\cup\Delta_6}(O(map_\nu(map_\mu(\mathbf{y})))). \\ \forall y \in \{\mathbf{y}\}.\ \alpha(y) = \beta^2(map_\nu(\mu_1(y))), \beta^2(map_\nu(\mu_2(y))) \end{array} \right\}$$ □

Corollary 8 (of Theorem 5: Characterization of abstraction to Δ_6 in the mixed structure $\mathbb{B} \cup \Delta_6$). *For any linear equation system $L(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$, we can compute in at most exponential time a formula over the mixed signature $\tilde{G}(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_6^{mixed}}$ such that:*

$$sol^{\mathbb{B}\cup\Delta_6}(\tilde{G}(\boldsymbol{y})) = h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}(L(\boldsymbol{y})))$$

Furthermore, if $\tilde{M}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ is the $h_{\mathbb{B}}$-exact rewriting of the formula $\tilde{N}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ from Lemma 46, then $\tilde{G}(\boldsymbol{y})$ can be defined as follows:

$$\tilde{G}(\boldsymbol{y}) \quad = \quad apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, in^{\mathbb{B}} \circ \tilde{M})(\boldsymbol{y})$$

Proof Let $\nu = (\nu_1, \nu_2)$ and $\mu = (\mu_1, \mu_2)$ be two pairs of new variable generators, and $\tilde{M}(map_\nu(map_\mu(\mathbf{y}))) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ be the $h_{\mathbb{B}}$-exact rewriting of the formula $\tilde{N}(map^2_\nu(map^2_\mu(\mathbf{y})))$ for $L(\mathbf{y})$ from Lemma 46. By Theorem 5, the formula $\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y})))$ satisfies:

$$h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y}))) = \left\{ \begin{array}{c} [y/(\beta^2(map^2_\nu(\mu_1(y))) +^{\mathbb{R}^2_+} \beta^2(map^2_\nu(\mu_2(y)))) \mid y \in \{\mathbf{y}\}] \\ \mid \beta \in sol^{\mathbb{B}}(\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y})))) \end{array} \right\}$$

and can be computed in at most exponential time from $L(\mathbf{y})$. The definition of $pair\text{-}sum_{\Delta_6}$ yields:

$$\begin{array}{ll} & h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y}))) \\ = & \left\{ \begin{array}{c} [y/pair\text{-}sum_{\Delta_6}(\beta^2(\nu(\mu_1(y))), \beta^2(\nu(\mu_2(y)))) \mid y \in \{\mathbf{y}\}], \\ \mid \beta \in sol^{\mathbb{B}}(\tilde{M}(map^2_\nu(map^2_\mu(\mathbf{y}))))\} \end{array} \right\} \\ = & sol^{\mathbb{B}\cup\Delta_6}(apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, in^{\mathbb{B}} \circ \tilde{M})(\mathbf{y})) \quad \text{by Lemma 56 with } O = in^{\mathbb{B}} \circ \tilde{M} \\ = & sol^{\mathbb{B}\cup\Delta_6}(\tilde{G}(\mathbf{y})) \end{array}$$

□

Example 57. *We continue the running example with the linear equation system $L(\boldsymbol{y})$ in Figure 4 where $\boldsymbol{y} = r_1 ... r_6 in_A out_B out_C$. It was derived from the reaction network in Figure 3. In Example 47, we illustrated how to compute the formula $\tilde{M}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ for characterizing the difference abstraction to Δ_6 of $L(\boldsymbol{y})$, where $\nu = (\nu_1, \nu_2)$ and $\mu = (\mu_1, \mu_2)$ were pairs of fresh variable*

generators. This formula can be converted into the $\tilde{G}(\boldsymbol{y}) = apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, in^{\mathbb{B}} \circ \tilde{M})(\boldsymbol{y})$ *of the mixed signature* Σ^{mixed}_6 *by Corollary 8. This formula has the following form:*

$$\begin{array}{rcl} \tilde{G}(\boldsymbol{y}) & = & \exists map^2_\nu(map^2_\mu(\boldsymbol{y})) \in \mathbb{B}. \\ & & \bigwedge_{Y \in \{\boldsymbol{y}\}} Pair\text{-}Sum_{\Delta_6}(map^2_\nu(map^2_\mu(Y)), Y) \ \wedge \ \nu_1(\mu_2(Y)) *^B \nu_2(\mu_2(Y)) = 0 \\ & & \wedge \ in^{\mathbb{B}}(M''(map^2_\nu(map^2_\mu(\boldsymbol{y})))) \end{array}$$

where the formula $M''(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ *applies the first-order definition* M'' *from Figure 17 but to different variables than used there.*

Theorem 9 (Solving Exact Difference Abstraction with Constraints for h_{Δ_6}). *For any linear equation system* $L(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$*, constraint* $C(\boldsymbol{y}) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}[dom(\Delta_6)]}$*, and finite subet* $V \subseteq \mathcal{V}$ *we can compute in at most exponential time a formula* $\tilde{H}(\boldsymbol{y}) \in \mathcal{F}_{\Sigma^{mixed}_6}$ *that satisfies:*

$$sol^{\mathbb{B} \cup \Delta_6}(\tilde{H}(\boldsymbol{y})) = abs^{h_{\Delta_6}}_{V, C(\boldsymbol{y})}(L(\boldsymbol{y}))$$

Furthermore, if the formula $\tilde{M}(map_\nu(map_\mu(\boldsymbol{y}))) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ *is the* $h_{\mathbb{B}}$*-exact rewriting of the formula* $\tilde{N}(map^2_\nu(map^2_\mu(\boldsymbol{y})))$ *for* $L(\boldsymbol{y})$ *from Lemma 46, then* $\tilde{H}(\boldsymbol{y})$ *can be defined as follows:*

$$\begin{array}{rcl} \tilde{G}(\boldsymbol{y}) & = & apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, in^{\mathbb{B}} \circ \tilde{M})(\boldsymbol{y}) \\ \tilde{H}(\boldsymbol{y}) & = & \exists \{\boldsymbol{y}\} \setminus V.\ \tilde{G}(\boldsymbol{y}) \wedge in^{\Delta_6}(C(\boldsymbol{y})) \end{array}$$

Proof Let $\nu = (\nu_1, \nu_2)$ and $\mu = (\mu_1, \mu_2)$ be two pairs of new variable generators. Let the formula $\tilde{M}(map_\nu(map_\mu(\mathbf{y}))) \in \mathcal{F}_{\Sigma_{pos\text{-}arith}}$ be the $h_{\mathbb{B}}$-exact rewriting of the formula $\tilde{N}(map^2_\nu(map^2_\mu(\mathbf{y})))$ for $L(\mathbf{y})$ from Lemma 46 and $\tilde{G}(\mathbf{y}) = apply^m_{4,1}(Pair\text{-}sum_{\Delta_6}, in^{\mathbb{B}} \circ \tilde{M})(\mathbf{y})$. By Corollary 8, $\tilde{G}(\mathbf{y})$ can be computed in at most exponential time from $L(\mathbf{y})$ and satisfies:

$$h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}(L(\mathbf{y}))) = sol^{\mathbb{B} \cup \Delta_6}(\tilde{G}(\mathbf{y}))$$

The formula $\tilde{H}(\mathbf{y}) = \exists \{\mathbf{y}\} \setminus V.\ \tilde{G}(\mathbf{y}) \wedge in^{\Delta_6}(C(\mathbf{y}))$ then satisfies what is required by the theorem:

$$\begin{array}{rcl} sol^{\mathbb{B} \cup \Delta_6}(\tilde{H}(\mathbf{y})) & = & \{\beta_{|V} \mid \beta \in sol^{B \cup \Delta_6}_{\{\mathbf{y}\}}(\tilde{G}(\mathbf{y})) \cap sol^{\mathbb{B} \cup \Delta_6}_{\{\mathbf{y}\}}(C(\mathbf{y}))\} \\ & = & \{\beta_{|V} \mid \beta \in h_{\Delta_6} \circ diff(sol^{\mathbb{R}_+}_{\{\mathbf{y}\}}(L(\mathbf{y}))) \cap sol^{\mathbb{B} \cup \Delta_6}_{\{\mathbf{y}\}}(C(\mathbf{y}))\} \\ & = & abs^{h_{\Delta_6}}_{V, C(\mathbf{y})}(L(\mathbf{y})) \end{array}$$

□

Example 58. *In the running example from Figure 4 we imposed the following constraint* $\tilde{C}(\boldsymbol{y})$ *on the difference abstraction of* $L(\boldsymbol{y})$ *over* Δ_6*:*

$$\tilde{C}(\boldsymbol{y}) = \left(\begin{array}{llllll} r_1 = A * D & \wedge & r_2 = A * E & \wedge \\ r_3 \in \{\Downarrow, \sim\} & \wedge & r_4 \in \{\Downarrow, \sim\} & \wedge \\ r_5 = D & \wedge & r_6 = E & \wedge \\ in_A \in \{\downarrow, \uparrow, \sim\} & \wedge & out_B = B & \wedge \\ out_C = C \end{array} \right)$$

Theorem 7 shows that the abstraction $abs^{h_{\Delta_6}}_{V, C(\boldsymbol{y})}(L(\boldsymbol{y}))$ *is the solution set* $sol^{\mathbb{B} \cup \Delta_6}(\tilde{H}(\boldsymbol{y}))$ *where:*

$$\tilde{H}(\boldsymbol{y}) = \exists \{\boldsymbol{y}\} \setminus V.\ (\tilde{G}(\boldsymbol{y}) \wedge in^{\Delta_6}(\tilde{C}(\boldsymbol{y}))$$

and $\tilde{G}(\boldsymbol{y})$ *is given in Example 57.*

The set $sol^{\mathbb{B}\cup\Delta_6}(\tilde{H}(\mathbf{y}))$ can be computed by a finite domain constraint programming, since $\mathbb{B} \cup \Delta_6$ is a finite structure. Theorem 9 yields an algorithm for solving the exact abstraction problem with constraints (see Definition 49) for the $\Sigma_{pos\text{-}arith}$-abstraction h_{Δ_6}.

.

11. Overapproximation Heuristics with Minimal Support Consequences

We propose a new heuristics for approximating the problem of difference abstractions with constraints. Later on, we will see experimentally that this heuristics is close to exact in our main application while requiring considerably less computation time.

Let $h : \mathbb{R}_+^2 \to \Delta$ be some Σ-abstraction into a finite Σ-structure Δ. The general idea of the existing heuristics (see e.g. [9]) for approximating the problem of difference abstractions with constraints from Definition 49 is as follows. Given a linear equation system $L \in \mathcal{F}_\Sigma$, a constraint $C \in \mathcal{F}_{\Sigma[dom(\Delta)]}$ and subsets of variable $V \subseteq V' = fv(L) \cup fv(C)$, we first compute some linear equation system $L' \in \mathcal{F}_\Sigma$ that is a logical consequence of L over $\mathbb{R}_+$ with $fv(L') \subseteq V'$, and in a second step the set of abstract solutions:

$$sol^\Delta(\exists V.\ (L \wedge L' \wedge C))$$

by finite domain constraint programming. By John's theorem in Corollary 2, this set is an overapproximation of the target of the problem $\{\beta_{|V} \mid \beta \in sol^{\mathbb{R}_+}_{V'}(L)^\Delta \cap sol^\Delta_{V'}(C)\}$.

The choice of which $\mathbb{R}_+$-consequence L' of L to add to L is critical. Generally, L' is a finite conjunction of linear equations that are $\mathbb{R}_+$-consequences of L. These are all the linear combinations of equations in L. Unfortunately, there are infinitely many such linear combinations, of which L' has to choose some finite subset.

We call an equation $E \in \mathcal{F}_\Sigma$ *linear* if E has the form $n_1x_1 + \ldots n_kx_k = m_1y_1 + \ldots m_ly_l$ for some pairwise distinct variables x_i and y_j, natural numbers $k, l \geq 0$ and nonzero natural numbers $n_i, m_j > 0$. We call E *nontrivial* if not $k = 0$ and $l = 0$. The support of E is the set of its free variables $fv(E)$. We call E *normalized* if there not exists a natural number $p, n'_1, \ldots n'_l, m'_1, \ldots m'_n$ such that $n_i = p *^{\mathbb{N}} n'_i$ and $m_j = p *^{\mathbb{N}} m'_j$ for all i, j.

Definition 59 (Minimal support linear $\mathbb{R}_+$-consequences). *A linear equation $E \in \mathcal{F}_\Sigma$ is a minimal support linear $\mathbb{R}_+$-consequence of a system of linear equations $L \in \mathcal{F}_\Sigma$ if it satisfies the following three conditions:*

- *E is a nontrivial $\mathbb{R}_+$-consequence of L,*
- *not other nontrivial $\mathbb{R}_+$-consequence of L has a smaller support than E, and*
- *E is normalized.*

It is not difficult to see that no two different minimal support linear $\mathbb{R}_+$-consequences of L may have the same support. Therefore, the set of minimal support linear $\mathbb{R}_+$-consequences of L is finite and of cardinality at most $2^{|fv(E)|}$. Given a system of linear equations $L \in \mathcal{F}_\Sigma$ we denote the conjunction of all its minimal support linear $\mathbb{R}_+$-consequences by:

$$L_{msc} \in \mathcal{F}_\Sigma$$

We next show how to compute L_{msc} from L. First, we transform L into an integer matrix A in linear time, such L is equivalent to $A\mathbf{x} = 0$, where $fv(L) = \{\mathbf{x}\}$. The $\mathbb{R}_+$-consequences of L are thus the linear combination of the rows of A. Since we want to combine the rows of A and

not its columns, we consider the transposed matrix A^T. Given a sequence $\mathbf{z}$ of with as many fresh variables as A has rows, the linear combinations of the rows of A (the row space) can be identified with the following set of integer solutions:

$$sol^{\mathbb{Z}}(\exists \mathbf{y}.A^T\mathbf{y} = \mathbf{z})$$

Since the row space is the orthogonal complement of A's nullspace $sol^{\mathbb{Z}}(A\mathbf{x} = 0)$, we have that each vector corresponding to a $\mathbb{R}_+$-consequence of L must be orthogonal to every vector in the nullspace, and in particular to any of its bases. Let $A^\perp$ be some basis of the nullspace of A, which can be easily computed by using Gauß algorithm. Then the row space is given by:

$$sol^{\mathbb{Z}}(A^\perp\mathbf{z} = 0)$$

Since we are interested only in the subset of solutions of the above system which are nonzero, normalized and with a minimal support, the problem of finding them is simply a particular case of the computation of the elementary modes of the orthogonal complement of the nullspace of A, with basis $A^\perp$, but such that nonpositive solutions are considered too. The usual software packages for computing elementary modes can then be applied to $A^\perp$ to compute such "reversible" elementary modes [26], or equivalently the problem can be reduced to computing the extreme rays of a cone and solved with any library for the analysis of polytopes, such as [27, 25].

12. Implementation and Experimentation

We have implemented the algorithm solving the differences abstraction problem with constraints in the case of difference abstraction h_{Δ_6} and applied them to change prediction in systems biology

12.1. Implementation

First, we implemented the exact rewriting of linear equation systems for the boolean abstraction $h_{\mathbb{B}}$. We implemented the rewriting in Python while using the libcdd library for computing elementary modes [27, 28].

Second, we implemented a solver for first-order constraint in the mixed structure $\mathbb{B} \cup \Delta_6$ by finite domain constraint programming with the Minizinc tool [13].

Third, we implemented the exact computation of difference abstractions for h_{Δ_6}. This was done in Python based on implementation obtained in the first and second step.

Fourth, we use BioComputing's Reaction-Network tool to represent reaction networks with partial kinetic information in XML-format and to infer the linear equation systems and the nonlinear kinetic constraints. The tools does also support John's overapproximation algorithm. We then integrated our exact algorithm for computing difference abstraction with h_{Δ_6} into the tool, so that it can be applied the systems of linear equation and nonlinear constraints obtained from a reaction network.

Fifth, we implemented the minimal support heuristics again in Python. For this we had to compute "reversible" elementary modes. We again used the libccd library for this, by reducing the computation of reversible elementary modes to the computation of irreversible elementary modes. We used standard Python libraries to applying Gauß algorithm to compute the orthogonal matrix $A^\perp$.

Sixth, we integrated the minimal support heuristics into BioComputing's Reaction-Network tool [29]. We also added a support to compare the solutions sets obtained by John's overapproximation, the minimal support heuristics, and the exact algorithm.

Network	Count type	John's over-approx.	min. support consequences	exact
Simple loop (Figure 6)	abstract solutions	19	6	6
Leucine overproduction (Figure 22)	knockouts abstract solutions	16 292	14 228	14 228
Counter example (Figure 19a)	abstract solutions	≥ 10000	4454	4374

Figure 18: Predictions for the networks analysed in this paper, obtained respectively by pure abstract interpretation, the heuristics based on minimal support consequences and the exact algorithm.

The graphical output of reaction networks is done with BioCompting's Network-Graph tool. It also allows to annotate abstract solution or the elementary flux modes to the network graph, and is integrated into the Reaction-Network tool. BioComputing's Network-Graph tool is publicly available, while the other components of BioComputing Reaction-Network tool are not yet made publicly available.

12.2. Application to Change Prediction of Reaction Networks

The main application of the change prediction algorithm for reaction networks with partial kinetic information [9, 11] concerns overproduction of the branched chain amino acid Leucine by the reaction network in Figure 22. Leucine is a predecessor of of the surfactin, a nonribosomal peptide, that can be used as a surfactant and produced industrially by the bacteria B. Subtilis. Some of the change predictions obtained for this application where verified successfully in the bioreactor.

We compare the results of John's overapproximation, the minimal support heuristics and the exact algorithm in Figure 18. Beside of the leucine network we also consider the simple loop network and the counter example in Figure 19a.

For the simple loop network, the exact algorithm shows that there are 6 abstract solutions, one for each value of Δ_6. The minimal support heuristics finds the same 6 abstract solutions, while by John's overapproximation returns 19 abstract solutions, of which 13 are not justified.

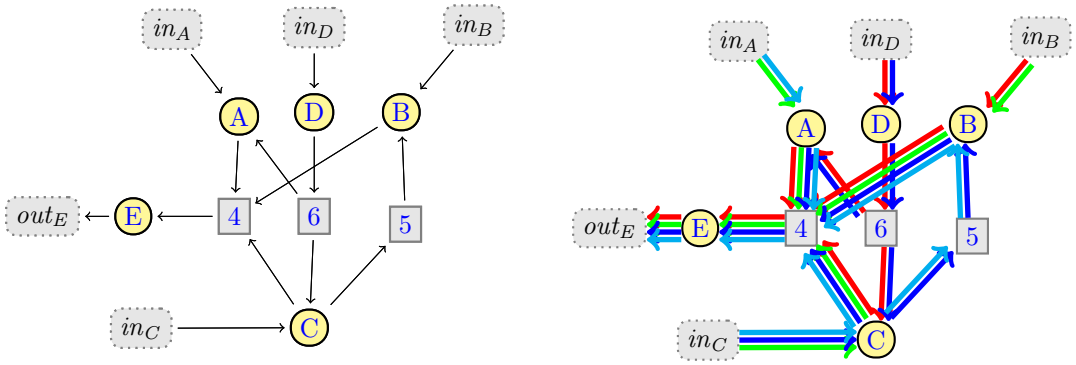
For the leucine network from Figure 22 the minimal support heuristics finds the same 228 solutions as the exact algorithm. John's overapproximation algorithm produces 292 solutions instead, including the 228 justified solutions.

On the other hand, the minimal support heuristics is remarkably faster than the exact algorithm – in the benchmark on leucine overproduction, we have 5 minutes versus 5 hours.

12.3. Counter Example for the Minimal Support Heuristics

We found and implemented the minimal support heuristics some years before finding the exact algorithm. At that time it was impossible to us to see whether the minimal support heuristics was exact or not. After having developed and implemented the exact algorithm, we could eventually evaluate this question. Our experiments showed that the heuristics is indeed exact for all applications to change prediction of reaction networks in systems biology that we tested. We then tried to prove in the case of Δ_3 that the minimal support heuristics was always exact, but failed to do so.

Next, we tried to find a counter example in the case of Δ_6. For this we developed a random generator of reaction networks and compared the minimal support heuristics with the exact algorithm that we implemented for Δ_6 only. This made us indeed find a counter example for Δ_6 that is given in Figure 19a.



(a) Graphical representation.

(b) Elementary flux modes.

Figure 19: Counter example for exactness of the minimal support heuristics for Δ_6.

$$
\begin{aligned}
2\ out_E &= in_B + in_C + r_6\\
2\ r_4 &= in_B + in_C + r_6\\
2\ out_E &= in_B + in_C + in_D\\
in_C + r_6 &= in_B + 2\ r_5\\
in_C + in_D &= in_B + 2\ r_5\\
2\ r_4 &= in_B + in_C + in_D\\
out_E + r_5 &= in_C + r_6\\
r_4 + r_5 &= in_C + r_6
\end{aligned}
\qquad
\begin{aligned}
out_E + r_5 &= in_C + in_D\\
r_4 + r_5 &= in_C + in_D\\
out_E &= in_A + in_D\\
r_4 &= in_A + in_D\\
out_E &= in_A + r_6\\
r_4 &= in_A + r_6\\
in_B + r_5 &= in_A + r_6\\
in_B + r_5 &= in_A + in_D\\
in_B + in_C &= 2\ in_A + r_6
\end{aligned}
\qquad
\begin{aligned}
in_B + in_C &= in_A + out_E\\
in_B + in_C &= 2\ in_A + in_D\\
in_B + in_C &= in_A + r_4\\
r_6 &= in_D\\
in_C &= in_A + r_5\\
r_4 &= out_E\\
r_4 &= in_B + r_5\\
out_E &= in_B + r_5
\end{aligned}
$$

Figure 20: Minimal support consequences

Why the minimal support heuristics admits abstract solutions that are not justified is not easy to understand.

We can see for instance that $r_4 = r_5 + in_B$ is a minimal support linear consequence, by looking at the elementary flux modes of the counter example network in Figure 19b. The list of all other minimal support linear consequences are given in Figure 20. We can also recognize, which of the abstract solutions are unjustified. An example is given in Figure 21. But we could not explain why this solution is unjustified, or find some nonminimal support consequence that it violates.

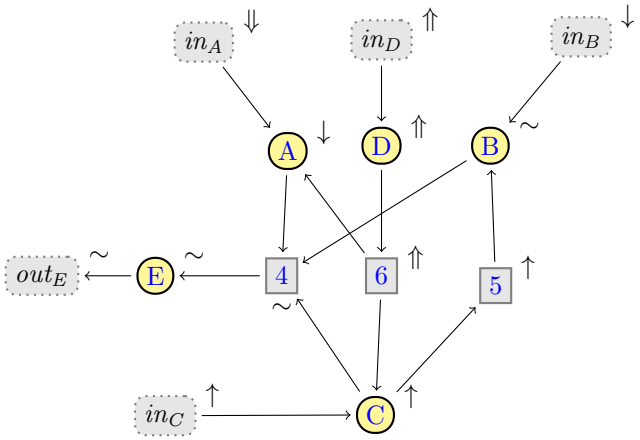


Figure 21: An unjustified solution found with the minimal support heuristics.

13. First-Order Definitions of Function Application

Our next objective is to generalize the definition of function applictions in first-order logic to functions with higher arities, as used already in special cases, and to prove the missing properties of such definitions stated earlier in Propositions 38 and 52.

We will use vector notation all over. We fix $\ell, k, n \in \mathbb{N}$ and consider first-order definitions F of type:

$$F : \mathcal{V}^k \times \mathcal{V}^\ell \to \mathcal{F}^n_\Sigma$$

For any Σ-algebra S and domain $D = dom(S)^n$ of the tuple algebra S^n, a partial function $n\text{-}rel^S(F) \subseteq D^k \times D^\ell$ is defined. For any m, we can lift the first-order definition F to a first-order definition $all^m_{k,\ell}(F) : \mathcal{V}^{mk} \times \mathcal{V}^{m\ell} \to \mathcal{F}^n_\Sigma$ where F is applied m-times, such that for all variables $x^1_1, \ldots, x^k_m, y^1_1, \ldots, y^\ell_m \in \mathcal{V}$, sequences $\mathbf{x}^j = x^j_1 \ldots x^j_m$ where $1 \leq j \leq k$, sequences $\mathbf{y}^j = y^j_1 \ldots y^\ell_m$ where $1 \leq j \leq \ell$, and sequences $\overline{\mathbf{x}} = \mathbf{x}^1 \ldots \mathbf{x}^k$ and $\overline{\mathbf{y}} = \mathbf{y}^1 \ldots \mathbf{y}^\ell$:

$$all^m_{k,\ell}(F)(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \;=_{\text{def}} \bigwedge_{i=1}^{m} F(x^1_i, \ldots, x^k_i, y^1_i, \ldots, y^\ell_i)$$

For any first-order definition $G : \mathcal{V}^{mk} \to \mathcal{F}^n_\Sigma$, we introduce a first-order definition $apply^m_{k,\ell}(G) : \mathcal{V}^{m\ell} \to \mathcal{F}^n_\Sigma$ such that for all $\overline{\mathbf{y}} \in \mathcal{V}^{m\ell}$:

$$apply^m_{k,\ell}(F, G)(\overline{\mathbf{y}}) =_{\text{def}} \exists \overline{\mathbf{x}}.\; G(\overline{\mathbf{x}}) \wedge all^m_{k,\ell}(F)(\overline{\mathbf{x}}, \overline{\mathbf{y}})$$

where $\overline{\mathbf{x}} \in \mathcal{V}^{mk}$ is some sequence of fresh variables. Note that $fv(all^m_{k,\ell}(F)(G)(\overline{\mathbf{y}})) = \{\overline{\mathbf{y}}\}$ so that the precise choice of $\overline{\mathbf{x}}$ is irrelevant. Note also for $k = 1$ that $apply^m_{k,\ell}(F, G)(\overline{\mathbf{y}})$ is equal to $apply^m_{1,\ell}(F, G)(\overline{\mathbf{y}})$ as defined earlier.

Lemma 60. *Let $F : \mathcal{V}^{k+\ell} \to \mathcal{F}^n_\Sigma$ and $G : \mathcal{V}^{mk} \to \mathcal{F}^n_\Sigma$ be first-order definitions and S a Σ-algebra, and $D = dom(S)^n$ the domain of S^n. If the relation $n\text{-}rel^S(F)$ is a partial function of type $D^k \times D^\ell$ then the relation $n\text{-}rel^S(all^m_{k,\ell}(F))$ is a partial function of type $D^{mk} \times D^{m\ell}$ such that:*

$$n\text{-}rel^S(all^m_{k,\ell}(F))(n\text{-}rel^S(G)) = n\text{-}rel^S(apply^m_{k,\ell}(F, G))$$

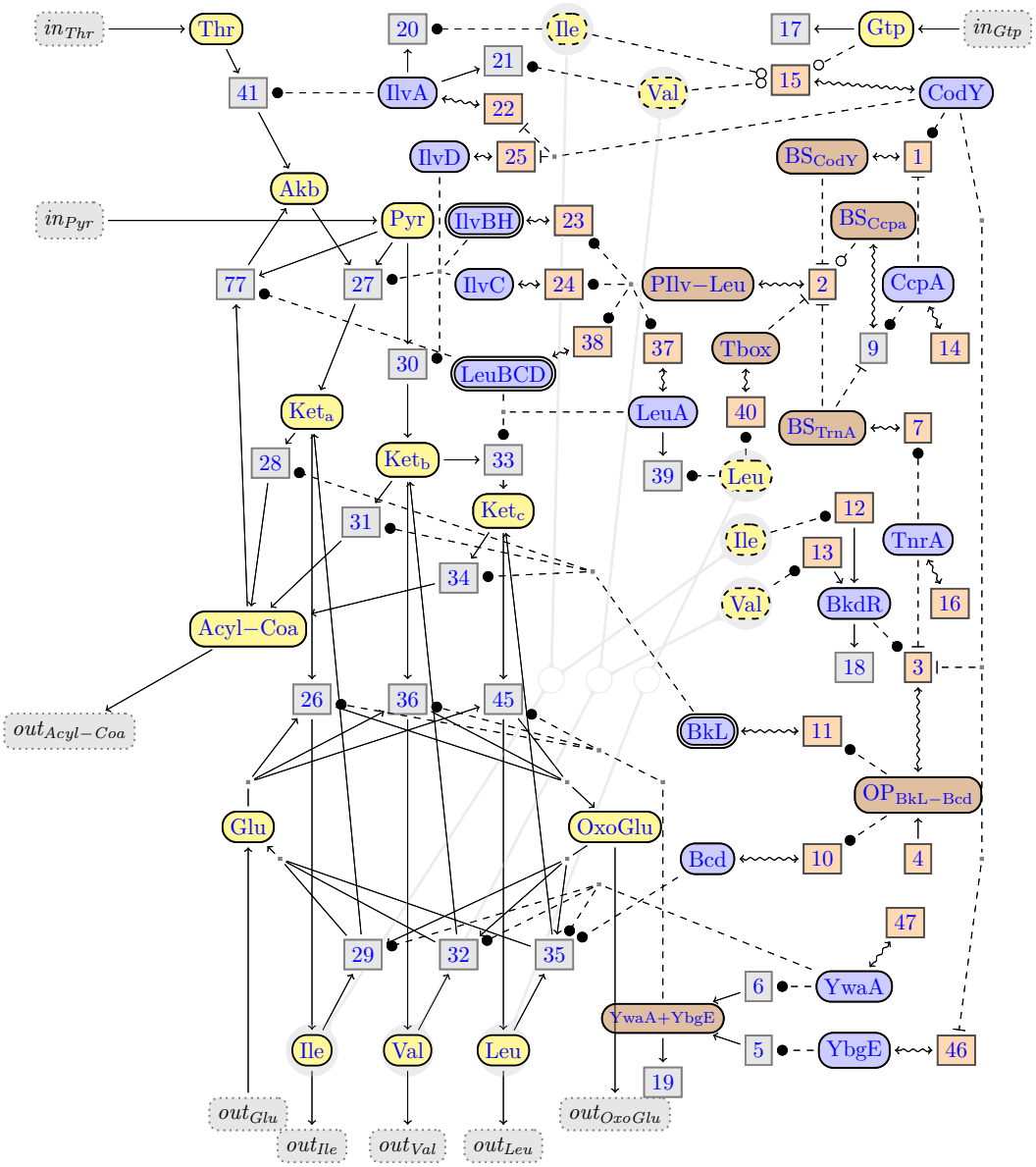


Figure 22: Graphical representation of the model that represents a part of the metabolism of the bacteria B. Subtilis, the branched chain amino-acid: Isoleucine, Leucine and Valine.

Proof Let $\mathbf{x}^1, \ldots, \mathbf{x}^k, \mathbf{y}^1, \ldots, \mathbf{y}^\ell \in \mathcal{V}^m$ be sequences of variables, so that no variable occurs twice in $\overline{\mathbf{x}}\overline{\mathbf{y}}$ where $\overline{\mathbf{x}} = \mathbf{x}^1 \ldots \mathbf{x}^k \in \mathcal{V}^{mkl}$ and $\overline{\mathbf{y}} = \mathbf{y}^1 \ldots \mathbf{y}^\ell \in \mathcal{V}^{m\ell}$. Then:

$$\begin{array}{rcl}
\textit{n-rel}^S(\textit{all}^m_{k,\ell}(F))(\textit{n-rel}^S(G)) & = & \{\textit{n-rel}^S(\textit{all}^m_{k,\ell}(F))(\alpha(\overline{\mathbf{x}})) \mid \alpha \in \textit{n-sol}^S(G(\overline{\mathbf{x}}))\} \\
& = & \{\alpha'(\overline{\mathbf{y}}) \mid \alpha' \in \textit{n-sol}^S(\textit{all}^m_{k,\ell}(F)(\overline{\mathbf{x}}, \overline{\mathbf{y}})),\ \alpha'_{|\{\overline{\mathbf{x}}\}} \in \textit{n-sol}^S(G(\overline{\mathbf{x}}))\} \\
& = & \{\alpha'(\overline{\mathbf{y}}) \mid \alpha' \in \textit{n-sol}^S(G(\overline{\mathbf{x}}) \wedge \textit{all}^m_{k,\ell}(F)(\overline{\mathbf{x}}, \overline{\mathbf{y}}))\} \\
& = & \{\alpha''(\overline{\mathbf{y}}) \mid \alpha'' \in \textit{n-sol}^S(\exists \overline{\mathbf{x}}.\ G(\overline{\mathbf{x}}) \wedge \textit{all}^m_{k,\ell}(F)(\overline{\mathbf{x}}, \overline{\mathbf{y}}))\} \\
& = & \textit{n-rel}^S(\textit{apply}^m_{k,\ell}(F, G))
\end{array}$$

For the previous last step note that for any $\alpha' \in n\text{-}sol^S(G(\overline{\mathbf{x}}) \wedge all^m_{k,\ell}(F)(\overline{\mathbf{x}},\overline{\mathbf{y}}))$ we can chose α'' as the restriction $\alpha'_{|\mathcal{V}\setminus\{\overline{\mathbf{x}}\}}$. Conversely, for any $\alpha'' \in n\text{-}sol^S(\exists\overline{\mathbf{x}}.\, G(\overline{\mathbf{x}}) \wedge all^m_{k,\ell}(F)(\overline{\mathbf{x}},\overline{\mathbf{y}}))$ there must exist a solution $\alpha' \in n\text{-}sol^S(G(\overline{\mathbf{x}}) \wedge all^m_{k,\ell}(F)(\overline{\mathbf{x}},\overline{\mathbf{y}}))$ such that α'' is the restriction $\alpha'_{|\mathcal{V}\setminus\{\overline{\mathbf{x}}\}}$. □

We first use Lemma 60 to prove a special case of Proposition 38 from Section 8.6 where $\ell = 1$.

Proposition 61 ($k = 1$ **and** $\ell = 1$)**.** *For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ and $F : \mathcal{V} \times \mathcal{V} \to \mathcal{F}^n_\Sigma$, sequence of variables $\boldsymbol{y} \in \mathcal{V}^m$, Σ-structure S and domain $D = dom(S)^n$, for which the relation $n\text{-}rel^S(F)$ is a partial function of type $D \times D$:*

$$n\text{-}rel^S(F) \circ n\text{-}sol^S(G(\boldsymbol{y})) = n\text{-}sol^S(apply^m_{1,1}(F,G)(\boldsymbol{y}))$$

Proof From Lemmata 37 and 60:

$$\begin{array}{ll} & n\text{-}rel^S(F) \circ n\text{-}sol^S(G(\mathbf{y})) \\ \text{Lemma 37} & = n\text{-}rel^S(F) \circ \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(G)\} \\ & = \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(all^m_{1,1}(F))(n\text{-}rel^S(G))\} \\ \text{Lemma 60} & = \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(apply^m_{1,1}(F,G))\} \\ \text{Lemma 37} & = n\text{-}sol^S(apply^m_{1,1}(F,G)(\mathbf{y})) \end{array}$$

□

We next use Lemma 60 to show the full Proposition 38 from Section 8.6 where $\ell \geq 1$.

Proposition 38 ($k = 1$ **and** $\ell \geq 1$)**.** *For any first-order definition $G : \mathcal{V}^m \to \mathcal{F}^n_\Sigma$ and $F : \mathcal{V} \times \mathcal{V}^\ell \to \mathcal{F}^n_\Sigma$, any Σ-algebra S, and domain $D = dom(S)^n$ of the typle algebra, such that the relation $n\text{-}rel^S(F)$ is a partial function of type $D \times D^\ell$, and any sequences of fresh variables $\boldsymbol{y}, \boldsymbol{y}^1, \ldots, \boldsymbol{y}^\ell \in \mathcal{V}^m$:*

$$n\text{-}rel^S(F) \circ n\text{-}sol^S(G(\boldsymbol{y})) = \{[\boldsymbol{y}/(\alpha(\boldsymbol{y}^1), \ldots, \alpha(\boldsymbol{y}^\ell)] \mid \alpha \in n\text{-}sol^S(apply^m_{1,\ell}(F,G)(\boldsymbol{y}^1, \ldots, \boldsymbol{y}^\ell))\}$$

Proof The proof generalizes on the proof of Proposition 61:

$$\begin{array}{ll} & n\text{-}rel^S(F) \circ n\text{-}sol^S(G(\mathbf{y})) \\ \textit{Lemma 37} & = n\text{-}rel^S(F) \circ \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(G)\} \\ & = \{[\mathbf{y}/(\mathbf{d}^1, \ldots, \mathbf{d}^\ell)] \mid (\mathbf{d}^1, \ldots, \mathbf{d}^\ell) \in (n\text{-}rel^S(all^m_{1,\ell}(F)))(n\text{-}rel^S(G))\} \\ \textit{Lemma 60} & = \{[\mathbf{y}/(\mathbf{d}^1, \ldots, \mathbf{d}^\ell)] \mid (\mathbf{d}^1, \ldots, \mathbf{d}^\ell) \in n\text{-}rel^S(apply^m_{1,\ell}(F,G))\} \\ \textit{Lemma 37} & = \{[\mathbf{y}/(\alpha(\mathbf{y}^1), \ldots, \alpha(\mathbf{y}^\ell)] \mid \alpha \in n\text{-}sol^S(apply^m_{1,\ell}(F,G)(\mathbf{y}^1, \ldots, \mathbf{y}^\ell))\} \end{array}$$

□

The next Propositions generalizes and thus proves Proposition 52.

Proposition 62 ($k \geq 1$ **and** $\ell = 1$)**.** *Let $G : \mathcal{V}^{mk} \to \mathcal{F}^n_\Sigma$ and $F : \mathcal{V}^k \times \mathcal{V} \to \mathcal{F}^n_\Sigma$ be a first-order definition, S a Σ-structure, and $D = dom(S)^n$ the domain of S^n, such that the relation $n\text{-}rel^S(F)$ is a partial function of type $D^k \times D$. Then for any $\boldsymbol{y} \in \mathcal{V}^m$ and a tuple of fresh variable generators $\nu = (\nu_1, \ldots, \nu_k)$ the sequence of variables $map^k_\nu(\boldsymbol{y})$ satisfies:*

$$n\text{-}rel^S(F) \circ cmp_\nu(n\text{-}sol^S(G(map^k_\nu(\boldsymbol{y})))) = n\text{-}sol^S(apply^m_{k,1}(F,G)(\boldsymbol{y}))$$

Proof Again from Lemmata 37 and 60, by generalizing the proof of Proposition 38:

$$\begin{array}{ll} & n\text{-}rel^S(F) \circ cmp_\nu(n\text{-}sol^S(G(map^k_\nu(\mathbf{y})))) \\ \text{Lemma 37} & = n\text{-}rel^S(F) \circ cmp_\nu(\{[map^k_\nu(\mathbf{y})/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(G)\}) \\ & = \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in (n\text{-}rel^S(all^m_{k,1}(F))(n\text{-}rel^S(G))\} \\ \text{Lemma 60} & = \{[\mathbf{y}/\mathbf{d}] \mid \mathbf{d} \in n\text{-}rel^S(apply^m_{k,1}(F,G))\} \\ \text{Lemma 37} & = n\text{-}sol^S(apply^m_{k,1}(F,G)(\mathbf{y})) \end{array}$$

□

14. Conclusion

We presented a new algorithm for computing the difference abstraction over Δ_3 and Δ_6 of the solution set of a system of linear equation systems with nonlinear constraints on the difference abstractions. The algorithm relies on an exact rewriting of linear equation systems with respect to the boolean abstraction, which can be based on elementary modes. Our reduction uses decompositions of the difference abstractions h_{Δ_3} and h_{Δ_6} into the boolean abstractions and functions on pair algebra $\mathbb{R}^2_+$ that can be defined in first-order logic with pairs. Eventually, we can compute the difference abstractions for systems of linear equation with constraints by finite domain constraint programming. We implemented our algorithm and applied it to change prediction of reaction networks with partial kinetic information in systems biology.

We also presented the minimal support heuristics, for approximating the difference abstraction over Δ_3 and Δ_6 of the solution set of a system of linear equation systems with nonlinear constraints. It turns out that the minimal support heuristics is exact for the prime application of change prediction while requiring much less computation time. It was difficult to find a counter-example shown that the minimal support heuristics is not always exact. We finally succeeded in for the case of Δ_6 by randomly generating and testing reaction networks. In the case of Δ_3 the question remains open though.

We believe that the presented algorithms are fundamental to develop better change prediction methods in the future. For this, it is important to not only deal with cycles in the metabolic parts of reaction networks but also deal with cycles through the regulatory part. An important challeange in practice is to provide multiple changes prediction. The current approaches, however, are not sufficiently precise to do so. This is due to the lack of kinetic information. Furthermore, the current approach can only abstract the differences of steady states, but not account for their relationship to initial states.

In the longer run, it would be of interest to obtain more quantitative predictions and not only qualitative predictions. But this would require more precise kinetic information in the reaction networks and to use more refined difference abstractions for the abstract interpretation.

Acknowledgements. We thank the anonymous reviewers of this journal version and also the reviewers of the CMSB'2019 version for their helpful and constructive feedback.

References

[1] Emilie Allart, Joachim Niehren, and Cristian Versari. Computing difference abstractions of linear equation systems. *Theor. Comput. Sci.*, 893:72–104, 2021. doi: 10.1016/J.TCS.2021.06.030. URL https://doi.org/10.1016/j.tcs.2021.06.030.

[2] Emilie Allart, Joachim Niehren, and Cristian Versari. Computing difference abstractions of metabolic networks under kinetic constraints. In Luca Bortolussi and Guido Sanguinetti, editors, *Computational Methods in Systems Biology - 17th International Conference, CMSB 2019, Trieste, Italy, September 18-20, 2019, Proceedings*, volume 11773 of *Lecture Notes in Computer Science*, pages 266–285. Springer, 2019. doi: 10.1007/978-3-030-31304-3_14. URL https://doi.org/10.1007/978-3-030-31304-3_14.

[3] Jeffrey D Orth, Ines Thiele, and Bernhard O Palsson. What is flux balance analysis? *Nature biotechnology*, 28(3):245–248, 2010.

[4] Jason A Papin, Joerg Stelling, Nathan D Price, Steffen Klamt, Stefan Schuster, and Bernhard O Palsson. Comparison of network-based pathway analysis methods. *Trends in biotechnology*, 22(8):400–405, 2004.

[5] Martin Feinberg. Chemical reaction network structure and the stability of complex isothermal reactors. *Chemical Engineering Science*, 42(10):2229 – 2268, 1987. ISSN 0009-2509. doi: http://dx.doi.org/10.1016/0009-2509(87)80099-4. URL http://www.sciencedirect.com/science/article/pii/0009250987800994.

[6] Laurence Calzone, Francois Fages, and Sylvain Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22 (14):1805–1807, July 2006. doi: 10.1093/bioinformatics/btl172. URL http://dx.doi.org/10.1093/bioinformatics/btl172.

[7] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasi—a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, 2006.

[8] François Fages, Steven Gay, and Sylvain Soliman. Inferring reaction systems from ordinary differential equations. *Theor. Comput. Sci.*, 599:64–78, 2015. doi: 10.1016/j.tcs.2014.07.032. URL http://dx.doi.org/10.1016/j.tcs.2014.07.032.

[9] Joachim Niehren, Cristian Versari, Mathias John, François Coutte, and Philippe Jacques. Predicting Changes of Reaction Networks with Partial Kinetic Information. *BioSystems*, 149:113–124, July 2016. URL https://hal.inria.fr/hal-01239198.

[10] François Coutte, Joachim Niehren, Debarun Dhali, Mathias John, Cristian Versari, and Philippe Jacques. Modeling Leucine's Metabolic Pathway and Knockout Prediction Improving the Production of Surfactin, a Biosurfactant from Bacillus Subtilis. *Biotechnology Journal*, 10(8):1216–34, August 2015. doi: 10.1002/biot.201400541. URL https://hal.inria.fr/hal-01153704.

[11] Mathias John, Mirabelle Nebut, and Joachim Niehren. Knockout Prediction for Reaction Networks with Partial Kinetic Information. In *14th International Conference on Verification, Model Checking, and Abstract Interpretation*, Rom, Italy, January 2013. URL http://hal.inria.fr/hal-00692499.

[12] Kim Marriott, Peter J. Stuckey, and Mark Wallace. Constraint logic programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 409–452. Elsevier, 2006. doi: 10.1016/S1574-6526(06)80016-7. URL https://doi.org/10.1016/S1574-6526(06)80016-7.

[13] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi: 10.1007/978-3-540-74970-7_38. URL https://doi.org/10.1007/978-3-540-74970-7_38.

[14] Julien Gagneur and Steffen Klamt. Computation of elementary modes: a unifying framework and the new binary approach. *BMC bioinformatics*, 5(1):1, 2004.

[15] Dr.Jürgen Zanghellini, David E. Ruckerbauer, Michael Hanscho, and Christian Jungreuthmayer. Elementary flux modes in a nutshell: Properties, calculation and applications. *Biotechnology Journal*, pages 1009–1016, 2013.

[16] Emilie Allart, Joachim Niehren, and Cristian Versari. Exact Boolean Abstraction of Linear Equation Systems. *Computation*, 9(11):32, October 2021. doi: 10.3390/COMPUTATION9110113. URL https://inria.hal.science/hal-03384058.

[17] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. doi: 10.1007/BF02579150. URL https://doi.org/10.1007/BF02579150.

[18] Stefan Schuster, Thomas Dandekar, and David A Fell. Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering. *Trends in biotechnology*, 17(2):53–60, 1999.

[19] K. Lotz, A. Hartmann, E. Grafahrend-Belau, and B.H. Schreiber, F.and Junker. Elementary flux modes, flux balance analysis, and their application to plant metabolism. *Plant Metabolism. Methods in Molecular Biology (Methods and Protocols)*, 2014.

[20] Costas D. Maranas and Ali R. Zomorrodi. *Flux Balance Analysis and LP Problems*, chapter 3, pages 53–80. Wiley-Blackwell, 2016. ISBN 9781119188902. doi: 10.1002/9781119188902.ch3. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119188902.ch3.

[21] Giuseppe Facchetti and Claudio Altafini. Partial inhibition and bilevel optimization in flux balance analysis. *BMC Bioinformatics*, pages 344–344, 2013.

[22] Kenneth D. Forbus. Qualitative reasoning. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 715–733. CRC Press, 1997. ISBN 0-8493-2909-4.

[23] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.

[24] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, March 2000. ISSN 0004-5411. doi: 10.1145/333979.333989. URL https://doi.org/10.1145/333979.333989.

[25] David Avis. A revised implementation of the reverse search vertex enumeration algorithm. In *Polytopes—combinatorics and computation*, pages 177–198. Springer, 2000.

[26] Marco Terzer and Jörg Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–2235, 2008.

[27] Komei Fukuda and Alain Prodon. Double description method revisited. In *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111. Springer, 1995.

[28] M Troffaes. pycddlib-a python wrapper for komei fukudals cddlib, 2018.

[29] BioComputing. Biocomputing's network-graph tool. http://researchers.lille.inria.fr/~niehren/BioComputing/Network-Graph/doc.html. 2018-07-10.

Appendix A. Proofs for Section 8 (First-Order Tuple Logic)

Lemma 29. *For any $e \in \mathcal{E}_\Sigma$, Σ-algebra S, $n \geq 1$, and $\beta : V \to dom(S)^n$ with $fv(\phi) \subseteq V \subseteq \mathcal{V}$:*

$$[\![e]\!]^{\beta,S^n} = [\![(\pi_1(e), \ldots, \pi_n(e))]\!]^{\beta,S}$$

Proof. By induction on the structure of expressions in $\mathcal{E}_\Sigma$.
Case of constants $c \in C$.

$$[\![c]\!]^{\beta,S^n} = c^{S^n} = (c^S, \ldots, c^S) = [\![(\pi_1(c), \ldots, \pi_n(c))]\!]^{\beta,S}$$

Case of variables $x \in \mathcal{V}$.

$$[\![x]\!]^{\beta,S^n} = \beta(x) = \beta((\pi_1(x), \ldots, \pi_n(x))) = [\![(\pi_1(x), \ldots, \pi_n(x))]\!]^{\beta,S}$$

Case of expressions $e_1 \odot e_2$ where $e_1, e_2 \in \mathcal{E}_\Sigma$ and $\odot \in F^{(2)}$.

$$\begin{array}{ll} [\![e_1 \odot e_2]\!]^{\beta,S^n} & = \cup\{(s_1 \odot^{S^n} s_2) \mid s_j \in [\![e_j]\!]^{\beta,S^n}\} \\ \textit{ind.hyp.} & = \cup\{(s_1 \odot^{S^n} s_2) \mid s_j \in [\![(\pi_1(e_j), \ldots, \pi_n(e_j))]\!]^{\beta,S}\} \\ & = [\![(\pi_1(e_1) \odot^S \pi_1(e_2), \ldots, \pi_n(e_1) \odot^S \pi_n(e_2)]\!]^{\beta,S} \end{array}$$

□

Proposition 30. *For any $\phi \in \mathcal{F}_\Sigma$, Σ-algebra S, and $n \geq 1$: $sol^{S^n}(\phi) = n\text{-}sol^S(fot^n(\phi))$.*

Proof. By induction on the structure of formulas in $\mathcal{F}_\Sigma$. The base case of Σ-equations follows essentially from Lemma 29. Let β be an assignment variables $\beta : V \to dom(S)^n$ with $V \subseteq \mathcal{V}$.
Case $e \doteq e'$ where $e, e' \in \mathcal{E}_\Sigma$.

$$\begin{array}{ll} sol^{S^n}(e \doteq e') & = \{\beta \mid [\![e \doteq e']\!]^{\beta,S^n} = 1\} \\ \textit{by prev. claim} & = \{\beta \mid [\![\bigwedge_{i=1}^n \pi_i(e) \doteq \pi_i(e')]\!]^{\beta,S} = 1\} \\ & = \{\beta \mid [\![fot^n(e \doteq e')]\!]^{\beta,S} = 1\} \\ & = n\text{-}sol^S(fot^n(e \doteq e')) \end{array}$$

Case $\phi \wedge \phi'$ where $\phi, \phi' \in \mathcal{F}_\Sigma$.

$$\begin{array}{ll} sol^{S^n}(\phi \wedge \phi') & = \{\beta \mid [\![\phi \wedge \phi']\!]^{\beta,S^n} = 1\} \\ & = \{\beta \mid [\![\phi]\!]^{\beta,S^n} \wedge [\![\phi']\!]^{\beta,S^n} = 1\} \\ \textit{ind.hyp.} & = \{\beta \mid [\![fot^n(\phi)]\!]^{\beta,S} \wedge [\![fot^n(\phi')]\!]^{\beta,S} = 1\} \\ & = \{\beta \mid [\![fot^n(\phi \wedge \phi')]\!]^{\beta,S} = 1\} \\ & = n\text{-}sol^S(fot^n(\phi \wedge \phi')) \end{array}$$

Case $\neg\phi$ were $\phi \in \mathcal{F}_\Sigma$.

$$\begin{array}{ll} sol^{S^n}(\neg\phi) & = \{\beta \mid [\![\neg\phi]\!]^{\beta,S^n} = 1\} \\ & = \{\beta \mid \neg[\![\phi]\!]^{\beta,S^n} = 1\} \\ \textit{ind.hyp.} & = \{\beta \mid \neg[\![fot^n(\phi)]\!]^{\beta,S} = 1\} \\ & = \{\beta \mid [\![\neg fot^n(\phi)]\!]^{\beta,S} = 1\} \\ & = \{\beta \mid [\![fot^n(\neg\phi)]\!]^{\beta,S} = 1\} \\ & = n\text{-}sol^S(fot^n(\neg\phi)) \end{array}$$

Case $\exists x.\phi$ were $\phi \in \mathcal{F}_\Sigma$.

$$\begin{aligned}
sol^{S^n}(\exists x.\ \phi) &= \{\beta \mid [\![\exists x.\ \phi]\!]^{\beta,S^n} = 1\} \\
&= \{\beta \mid \mathit{exists}\ s \in dom(S)^n.\ [\![\phi]\!]^{\beta[x/s],S^n} = 1\} \\
\textit{ind.hyp.} &= \{\beta \mid \mathit{exists}\ s \in dom(S)^n.\ [\![fot^n(\phi)]\!]^{\beta[x/s],S} = 1\} \\
&= \{\beta \mid [\![\exists x.\ fot^n(\phi)]\!]^{\beta,S} = 1\} \\
&= \{\beta \mid [\![fot^n(\exists x.\ \phi)]\!]^{\beta,S} = 1\} \\
&= n\text{-}sol^S(fot^n(\exists x.\ \phi))
\end{aligned}$$

□

Lemma 31. *For any expression $o \in \mathcal{O}^n_\Sigma$ and variable assignment $\beta : V \to dom(S)^n$ with $fv(o) \subseteq V \subseteq \mathcal{V}$ we have $[\![dec_\nu(o)]\!]^{S,cmp_\nu^{-1}(\beta)} = [\![o]\!]^{\beta,S}$.*

Proof. By induction on the structure of Σ-expressions $o \in \mathcal{O}^n_\Sigma$.
Case of constants $c \in C$.

$$[\![dec_\nu(c)]\!]^{S,cmp_\nu^{-1}(\beta)} = [\![c]\!]^{\beta,S}$$

Case $\dot{\pi}_i(x)$ where $x \in \mathcal{V}$ and $1 \le i \le n$.

$$\begin{aligned}
[\![dec_\nu(\dot{\pi}_i(x))]\!]^{S,cmp_\nu^{-1}(\beta)} &= \{cmp_\nu^{-1}(\beta)(cmp_{\nu_i}^{-1}(x))\} \\
&= \{\pi_i(\beta(x))\} \\
&= [\![o]\!]^{\beta,S}
\end{aligned}$$

Case $o_1 \odot o_2$ where $o_1, o_2 \in \mathcal{O}^n_\Sigma$ and $\odot \in F^{(2)}$.

$$\begin{aligned}
[\![dec_\nu(o_1 \odot o_2)]\!]^{S,cmp_\nu^{-1}(\beta)} &= [\![dec_\nu(o_1) \odot dec_\nu(o_2)]\!]^{S,cmp_\nu^{-1}(\beta)} \\
&= \cup\{(s_1 \odot^S s_2 \mid s_i \in [\![dec_\nu(o_i)]\!]^{S,cmp_\nu^{-1}(\beta)}\} \\
\textit{ind.hyp.} &= \cup\{(s_1 \odot^S s_2 \mid s_i \in [\![o_i]\!]^{\beta,S}\} \\
&= [\![o_1 \odot o_2]\!]^{\beta,S}
\end{aligned}$$

Case $\pi_i(o_1, \ldots, o_n)$ where $o_1, \ldots, o_n \in \mathcal{O}^n_\Sigma$.

$$\begin{aligned}
[\![dec_\nu(\pi_i(o_1, \ldots o_n))]\!]^{S,cmp_\nu^{-1}(\beta)} &= [\![dec_\nu(o_i)]\!]^{S,cmp_\nu^{-1}(\beta)} \\
\textit{ind.hyp.} &= [\![o_i]\!]^{\beta,S} \\
&= [\![\pi_i(o_1, \ldots o_n)]\!]^{\beta,S}
\end{aligned}$$

□

Claim 33. *For any variable assignment $\beta : V \to dom(S)^n$ with $V \subseteq \mathcal{V}$ and formula $\psi \in \mathcal{F}^n_\Sigma$ we have $[\![dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = [\![\psi]\!]^{\beta,S}$.*

Proof. The proof of the claim is by induction on the structure of Σ-formulas in $\mathcal{F}^n_\Sigma$.
Case $o = o'$ where $o, o' \in \mathcal{O}^n_\Sigma$.

$$\begin{aligned}
[\![o = o']\!]^{\beta,S} = 1 &\Leftrightarrow [\![o]\!]^{\beta,S} \cap [\![o]\!]^{\beta,S} \neq \emptyset \\
\text{Lemma 31} &\Leftrightarrow [\![dec_\nu(o)]\!]^{S,cmp_\nu^{-1}(\beta)} \cap [\![dec_\nu(o)]\!]^{S,cmp_\nu^{-1}(\beta)} \neq \emptyset \\
&\Leftrightarrow [\![dec_\nu(o) = dec_\nu(o')]\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
&\Leftrightarrow [\![dec_\nu(o = o')]\!]^{S,cmp_\nu^{-1}(\beta)} = 1
\end{aligned}$$

Case $\psi \wedge \psi'$ where $\psi, \psi' \in \mathcal{F}^n_\Sigma$.

$$
\begin{array}{ll}
[\![\psi \wedge \psi']\!]^{\beta,S} = 1 & \Leftrightarrow [\![\psi]\!]^{\beta,S} \wedge [\![\psi']\!]^{\beta,S} = 1 \\
\textit{ind.hyp} & \Leftrightarrow [\![dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} \wedge [\![dec_\nu(\psi)']\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
& \Leftrightarrow [\![dec_\nu(\psi) \wedge dec_\nu(\psi)']\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
& \Leftrightarrow [\![dec_\nu(\psi \wedge \psi')]\!]^{S,cmp_\nu^{-1}(\beta)} = 1
\end{array}
$$

Case $\neg\psi$ where $\psi \in \mathcal{F}^n_\Sigma$.

$$
\begin{array}{ll}
[\![\neg\psi]\!]^{\beta,S} = 1 & \Leftrightarrow \neg[\![\psi]\!]^{\beta,S} = 1 \\
\textit{ind.hyp.} & \Leftrightarrow \neg[\![dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
& \Leftrightarrow [\![\neg dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
& \Leftrightarrow [\![dec_\nu(\neg\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = 1
\end{array}
$$

Case $\exists x.\psi$ where $\psi \in \mathcal{F}^n_\Sigma$.

$$
\begin{array}{ll}
[\![\exists x.\psi]\!]^{\beta,S} = 1 & \Leftrightarrow \textit{exist } s \in dom(S)^n.[\![\psi]\!]^{S,\beta[x/s]} = 1 \\
\textit{ind.hyp.} & \Leftrightarrow \textit{exist } s \in dom(S)^n.[\![dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta[x/s])} = 1 \\
& \Leftrightarrow \textit{exist } s_1 \in dom(S) \ldots s_n \in dom(S).[\![\psi]\!]^{S,cmp_\nu^{-1}(\beta[cmp_{\nu_i}^{-1}(x)/s_i])} = 1 \\
& \Leftrightarrow [\![\exists cmp_{\nu_1}^{-1}(x) \ldots \exists cmp_{\nu_n}^{-1}(x).dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = 1 \\
& \Leftrightarrow [\![\exists x.dec_\nu(\psi)]\!]^{S,cmp_\nu^{-1}(\beta)} = 1
\end{array}
$$

□