Extracting reaction systems from function behavior

Genova, D.; Hoogeboom, H.J.; Prodanoff, Z.

Journal of Membrane Computing (2020) 2:194–206
https://doi.org/10.1007/s41965-020-00045-z

REGULAR PAPER

Extracting reaction systems from function behavior

Daniela Genova[1] · **Hendrik Jan Hoogeboom**[2] · **Zornitza Prodanoff**[3]

Received: 27 June 2020 / Accepted: 13 August 2020 / Published online: 1 September 2020

Abstract
Reaction systems, introduced by Ehrenfeucht and Rozenberg, are a theoretical model of computation based on the two main features of biochemical reactions: facilitation and inhibition, which are captured by the individual reactions of the system. All reactions, acting together, determine the global behavior or the result function, res, of the system. In this paper, we study decomposing of a given result function to find a functionally equivalent set of reactions. We propose several approaches, based on identifying reaction systems with Boolean functions, Boolean formulas, and logic circuits. We show how to minimize the number of reactions and their resources for each single output individually, as a group, and when only a subset of the states are considered. These approaches work both when the reactions of the given res function are known and not known. We characterize the minimal number of reactions through the minimal number of logical terms of the Boolean formula representation of the reaction system. Finally, we make applications recommendations for our findings.

Keywords Reaction systems · Boolean functions · Minimization · Boolean formulas · DNF · Prime implicants · Logic synthesis · Logic circuits

1 Introduction

Reaction systems were introduced by Ehrenfeucht and Rozenberg in [6], as a theoretical model of computation capturing the two main features of biochemical reactions: facilitation and inhibition. Each reaction system is defined over a finite set of background entities and contains reactions, that are triples of entities: reactants, inhibitors, and products. Reactions act together on each subset (state) of entities, such that, if all of the reactants and none of the inhibitors are present, the reaction is enabled in that state and its products are produced. The products of all reactions enabled in a state form the next state and, in that way, define the result function res. The aggregate behavior of the system, can then be represented by a one-out (exactly one edge goes out of each state (vertex)) graph, called the 0-context graph of the system, [9]. Two reaction systems that exhibit the same aggregate behavior (produce the same 0-context graph) are called functionally equivalent, since their result functions act the same way on every single state. Determining whether two reaction systems are functionally equivalent was proved in [6] to be Co-NP-complete. An extensive list of complexity results for reaction systems is obtained in [1]. Other equivalences of reaction system studied in the literature include enabling equivalence [7], process equivalence, enabling process equivalence, and more [12].

One of the new approaches to reaction systems we introduce here is that of *decomposing* the aggregate result function into separate reactions, as a kind of parallelization. This includes not only minimization, but also adds to it: starting with a *flat* state graph (which does not show which set of reactions led to it), we decompose it into more or less independent reactions. One of the novel approaches to minimization here is that we do not only minimize reaction systems that are known, but we also minimize reaction systems that are not known, i.e. the individual reactions defining the global result function are not known. To our knowledge, the topic of recovering of unknown reaction systems with known res function, is new.

✉ Daniela Genova
d.genova@unf.edu

Zornitza Prodanoff
zprodano@unf.edu

[1] Department of Mathematics and Statistics, University of North Florida, 32224 Jacksonville, FL, USA

[2] LIACS, Leiden University, Leiden, The Netherlands

[3] School of Computing, University of North Florida, Jacksonville, FL 32224, USA

Whether we start from a reaction system that is specified by its set of reactions or directly by a given result function, which may be given in the form of its 0-context graph, we seek to minimize the number of reactions adhering to certain properties. In the various minimization techniques that we propose, we always obtain a functionally equivalent reaction system to the (explicitly or implicitly) given one.

Minimization in terms of resources in reactions, more specifically minimal and almost minimal reactions, are discussed in [8, 17]. Minimizing reaction systems, in terms of obtaining functionally equivalent reaction systems with a minimal number of reactions, when the set of reactions are partitioned according to their product sets, were studied in [5].

We propose three types of minimizations that adhere to different types of requirements: (1) minimizing reactions (and their combined resources) for the purpose of producing a specific entity with minimum cost; (2) minimizing the number of reactions for the entire system when all outputs are considered; and (3) minimizing the number of reactions when only some states are considered. These can then be applied in various hybrid modes.

Two components of our approach are crucial for this paper: identifying reaction systems with Boolean functions and Boolean formulas and adapting minimization techniques for Boolean formulas to reaction systems. The first connection was made by Ehrenfeucht and Rozeneberg in [6] and later used in [5]. In [2], the logical formulas are used as formula based predictors for the occurrence of certain entities in a set number of steps. The second, spans well-known minimization tools from logic circuits minimization and logic synthesis: Karnaugh maps [11], the Quine-McCluskey algorithm [13] and Espresso algorithm [15, 16], which were also used for the minimization shown in [5].

This paper is organized as follows. Section 2 presents the basic notions of reaction systems. It explains how, given a set of reactions, one can generate its result function (conveniently represented by the 0-context graph) and, conversely, how given the result function, one can construct a set of maximally inhibited reactions that generates it. We show how every function on the power set of a finite set can be translated to a reaction system.

In Section 3, we discuss ways to identify reaction systems with Boolean functions and Boolean formulas. A crucial observation is made in Theorem 10, which basically states that common terms in DNF formulas correspond to reactions, i.e. to their resources, while the output is determined by the formulas involved. Thus, the minimization of DNF formulas can directly be applied to minimization of reaction systems.

Section 4 focuses on extracting a minimal set of reactions from a given result function. These methods can also be used to recover reactions from a reaction system that is not known, using only its result function. A popular example is used throughout to illustrate the different methods of minimization, e.g. Karnaugh maps and Espresso algorithm, as well as, the different types of minimization: as independent outputs, together as a group, and with *don't care* states.

Section 5 presents a pseudocode of the Espresso algorithm for the cases where a group minimization is the goal and also when do not care states may be present. It explains the main features of the algorithms and also points to relevant references where more about these algorithms may be found.

Finally, Section 6 discusses applications and future work.

2 Reaction systems and the result function

Reaction systems were introduced in [6] and we recall and follow the definitions from this reference. The power set of a set S is denoted by $\mathcal{P}(S)$.

Definition 1 A *reaction system* is a pair $\mathcal{A} = (S, A)$, where S is a finite nonempty set, the *background set* of $\mathcal{A}$, and $A \subseteq \mathcal{P}(S) \times \mathcal{P}(S) \times \mathcal{P}(S)$ is a set of *reactions* in S.

A *reaction* is a triple $a = (R_a, I_a, P_a)$ of finite subsets of S, called the *reactant* set of a, the *inhibitor* set of a, and the *product* set of a respectively.

We say that a is *enabled in* $X \subseteq S$ if and only if $R_a \subseteq X$ and $I_a \cap X = \emptyset$. Thus, a reaction a is enabled in a set X, if X separates R_a from I_a. If a is enabled in X, we define the *result* of a *on* X, as $\mathrm{res}_a(X) = P_a$. Otherwise, a is not enabled in X, and then we set $res_a(X) = \emptyset$. For a set of reactions A, define $res_A(X) = \bigcup_{a \in A} res_a(X)$ to be the result of the set of reactions A on X. Clearly, the result of a set A on a subset X of S is the union of the product sets of the reactions that are enabled in X. For a given reaction system $\mathcal{A} = (S, A)$, we set $res_{\mathcal{A}}(X) = res_A(X)$.

We recall a popular example, Example 7, from the overview paper *A Tour of Reaction Systems* [4].

Example 2 Let $\mathcal{A} = (S, A)$ be the reaction system with background set $S = \{1, 2, 3, 4\}$ and set of reactions A, consisting of the six reactions below.

$a_1 = (\{1\}, \{3\}, \{2\})$ $\quad a_2 = (\{2\}, \{1\}, \{1\})$
$a_3 = (\{2\}, \{3\}, \{3\})$ $\quad a_4 = (\{3\}, \{1, 2\}, \{1, 2, 4\})$
$a_5 = (\{4\}, \{3\}, \{1, 2\})$ $\quad a_6 = (\{1, 3\}, \{2, 4\}, \{2, 3\})$

Consider, for example, the following subsets of S, $X_1 = \{2, 3\}$, $X_2 = \{1, 3, 4\}$, and $X_3 = \{1, 2\}$. In subset $X_1 = \{2, 3\}$, only a_2 is enabled, so the result of $\mathcal{A}$ on X_1 equals P_{a_2}, i.e., $\mathrm{res}_{\mathcal{A}}(X_1) = \mathrm{res}_{\mathcal{A}}(\{2, 3\}) = \mathrm{res}_{a_2}(\{2, 3\}) = P_{a_2} = \{1\}$. In $X_2 = \{1, 3, 4\}$, no reactions are enabled, since X_2 contains an inhibitor from every reaction. Thus, $\mathrm{res}_{\mathcal{A}}(\{1, 3, 4\}) = \emptyset$.

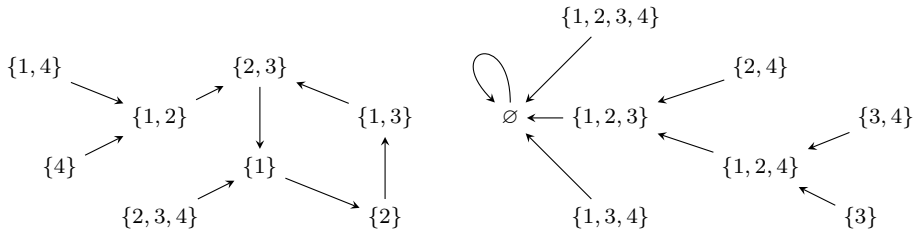


Fig. 1 The 0-context graph $G^0_{\mathcal{A}}$ from Example 4

In $X_3 = \{1,2\}$, reactions a_1 and a_3 are enabled, and so, $\text{res}_{\mathcal{A}}(X_3) = \text{res}_{\{a_1,a_3\}}(\{1,2\}) = \text{res}_{a_1}(\{1,2\}) \cup \text{res}_{a_3}(\{1,2\}) = P_{a_1} \cup P_{a_3} = \{2,3\}$. ⌟

Reaction systems are studied as computational devices. The state of the system is the set X of entities from S currently present. In a computational step of the system, the next state consists of all entities that are produced by the reactions of the system from the entities currently present, plus any entities that might arrive from independent parts elsewhere, called the *context*. Thus, a computational step of the system can be decomposed into two parts: collect the context and compute the new state: $X \mapsto X \cup C \mapsto \text{res}_{\mathcal{A}}(X \cup C)$, for some $X, C \in \mathcal{P}(S)$.

In this paper, we focus on the result function, $\text{res}_{\mathcal{A}}$, the second part of the computational step. It describes the *aggregate* change induced by *all* of the reactions in a state, as opposed to the *partial* contributions, res_a, made by the individual reactions a. Given a reaction system $\mathcal{A}$ over S, its result function $\text{res}_{\mathcal{A}}$ is a mapping from $\mathcal{P}(S)$ to $\mathcal{P}(S)$. It can be represented as a directed graph, the vertices of which are the states of the system. The target of an outgoing directed edge at each state, represents the result of the system at that state. Hence, this is a *one-out graph*, i.e. there is exactly one outgoing edge from each vertex. We assume that, at every state (vertex) no further context will be added. The following notion is from [9].

Definition 3 The 0-*context graph* of a reaction system $\mathcal{A}$ is the graph $G^0_{\mathcal{A}} = (\mathcal{P}(S), E)$ with edge set $E = \{ (X, \text{res}_{\mathcal{A}}(X)) \mid X \in \mathcal{P}(S) \}$.

Observe that, $(X, Y) \in E(G^0_{\mathcal{A}})$ if and only if $Y = res_{\mathcal{A}}(X)$.

Example 4 Consider the reaction system from Example 2. The 0-context graph for it, $G^0_{\mathcal{A}}$, is presented in Figure 1. The edges of the graph represent the result function $\text{res}_{\mathcal{A}}$. Thus, $(X_1, \text{res}_{\mathcal{A}}(X_1)) = (\{2,3\}, \{1\})$ is an edge on the graph from $\{2,3\}$ to $\{1\}$. Also, $(X_2, \text{res}_{\mathcal{A}}(X_2)) = (\{1,3,4\}, \varnothing)$ and $(X_3, \text{res}_{\mathcal{A}}(X_3)) = (\{1,2\}, \{2,3\})$ are edges in the graph. ⌟

Based on the result function $\text{res}_{\mathcal{A}}$, different types of equivalences of reaction systems were defined and studied, such as enabling equivalence, process equivalence, etc. [7, 12]. The first and most fundamental one is the functional equivalence defined in [6]. Two reactions a and b are *functionally equivalent*, if $\text{res}_a(X) = \text{res}_b(X)$, for every $X \subseteq S$. It was proved in [6], that this equivalence coincides with equality for individual reactions. When extended to sets, however, the notion of functional equivalence is not trivial, in general.

Definition 5 Let A, B be sets of reactions over background set S. Then A and B are *functionally equivalent*, denoted $A \sim B$, if $res_A(X) = res_B(X)$, for every $X \subseteq S$.

Extending this further to reaction systems, we say that two reaction systems $\mathcal{A} = (S, A)$ and $\mathcal{B} = (S, B)$ are functionally equivalent, denoted $\mathcal{A} \sim \mathcal{B}$, if their sets of reactions are functionally equivalent, i.e. if $A \sim B$. Whereas checking the equivalence of single reactions is a simple syntactic check, the equivalence of sets of reactions is computationally hard, see Corollary 12 below.

The following proposition states that *every* function on the subsets of a finite set, can be translated into a reaction system, thus, showing how powerful these systems are. The construction of a reaction system uses *maximally inhibited* reactions defined in [18], i.e., reactions where the reactant set and inhibitor set are complementary, a trick also used in Section 5 in [6].

Proposition 6 *Let S be a finite set. For every function $f : \mathcal{P}(S) \to \mathcal{P}(S)$, there exists a reaction system $\mathcal{A} = (S, A)$, such that $f = \text{res}_{\mathcal{A}}$.*

Proof Let S be a finite set and $f : \mathcal{P}(S) \to \mathcal{P}(S)$. Consider an ordered pair $(X, Y) \in f$, i.e., $X, Y \subseteq S$ and $Y = f(X)$. Construct the maximally inhibited reaction $a_X = (X, S \setminus X, Y)$, let $A = \{a_Z \mid Z \subseteq S\}$, and set $\mathcal{A} = (S, A)$. Since a_X is the only reaction enabled in X, $\text{res}_{\mathcal{A}}(X) = \text{res}_{a_X}(X) = Y$. Thus, every pair $(X, Y) \in f$ is represented by $\text{res}_{\mathcal{A}}(X) = Y$ and vice versa. □

From the above proposition, it becomes clear that we can identify any function on the power set of a given finite set with the result function of a reaction system. Equivalently, since such functions can also be identified with one-out graphs, every one-out graph with vertices that are the subsets

of a finite set can be identified with a reaction system whose 0-context graph equals the given graph.

In the next section, we discuss how these structures can be specified with Boolean functions.

3 Boolean functions and reaction systems

In this section, we take reaction systems to the digital domain, translating sets into bit-vectors. This has two motivations. First, as it was already illustrated in [6], bit-vectors (bit-strings) can be used to obtain complexity results for various computational or decision problems concerning reaction systems. Second, we will use bit-vectors to apply various tools to minimize the number of reactions in a reaction system, or construct reactions for a given res function behavior.

Boolean functions and logical formulas. In general, a Boolean function $H : \{0,1\}^n \to \{0,1\}$ can be specified with a Boolean formula. We will use n Boolean variables, $v_1, v_2, \ldots, v_n$, whose truth value assignments will be represented by the strings in $\{0,1\}^n$. (We do not distinguish between the boolean vector $(b_1, \ldots, b_n)$ and the n-bit string $b_1 \ldots b_n \in \{0,1\}^n$ and we use both terms interchangeably.) A *literal*, ℓ, is a variable v or its negation $\bar{v}$ (or v'). A *term* is a product, i.e., a conjunction ($\wedge$ or $\cdot$), of literals. A term is a *minterm*, if each of the variables is used exactly once. That is, every minterm is of the form $\ell_1 \wedge \ldots \wedge \ell_n$ (or simply $\ell_1 \ldots \ell_n$), where $\ell_i = v_i$ or $\ell_i = \bar{v}_i$, for $1 \le i \le n$. A Boolean formula φ is in *disjunctive normal form* (DNF), if it is a sum of products, i.e., a disjunction ($\vee$ or $+$) of terms.

Given the Boolean function $H : \{0,1\}^n \to \{0,1\}$, construct a Boolean formula in DNF φ_H (or simply φ, when H is understood) as follows. For each Boolean vector $x = b_1 \ldots b_n \in \{0,1\}^n$, such that $H(x) = 1$, construct the minterm $\gamma(x) = \ell_1 \ldots \ell_n$, such that for each $i, 1 \le i \le n$, the literal $\ell_i = v_i$ if and only if $b_i = 1$ and $\ell_i = \bar{v}_i$, otherwise. Then, take φ to be a disjunction of all these $\gamma(x)$. Then, $H(x) = 1$ if and only if $\varphi(x) = 1$. Note that if H has k input strings x that map into 1, then φ will have k minterms $\gamma(x)$, each representing one such string.

In the context of reaction systems, we need multi-valued Boolean functions $F : \{0,1\}^n \to \{0,1\}^n$ to represent functions $f : \mathcal{P}(S) \to \mathcal{P}(S)$. To handle a multi-valued function, one usually considers the partial functions $F_i : \{0,1\}^n \to \{0,1\}$, which are the projections onto the i^{th} coordinate of the output, defined as $F_i(b_1 \ldots b_n) = c_i$, if $F(b_1 \ldots b_n) = (c_1 \ldots c_n)$. Then, we represent each single-valued F_i by a Boolean formula φ_i, as described above, such that $F_i(x) = 1$ if and only if $\varphi_i(x) = 1$.

Given $F : \{0,1\}^n \to \{0,1\}^n$, define its *logical representation* $\boldsymbol{\Phi}_F$ (or simply $\boldsymbol{\Phi}$, if F is understood), to be the collection φ_i, for $1 \le i \le n$, where φ_i is in DNF, and represents the partial function F_i of F.

Sets as bit-vectors. Let $S = \{u_1, \ldots, u_n\}$ be an ordered set. The *characteristic function*, χ, maps every subset $X \subseteq S$ onto a Boolean vector x, i.e. $\chi(X) = x = (b_1, \ldots, b_n)$ such that $b_i = 1$ if and only if $u_i \in X$ and $b_i = 0$, otherwise. This establishes a bijection between $\mathcal{P}(S)$ and $\{0,1\}^n$ and, by extension, between functions f from $\mathcal{P}(S) \to \mathcal{P}(S)$ and functions F from $\{0,1\}^n \to \{0,1\}^n$, where $\bar{f}$ is the characteristic version of f, defined by $\bar{f} = \chi \circ f \circ \chi^{-1}$. In particular, $\overline{\text{res}}_{\mathcal{A}}$ is the characteristic version of the result function $\text{res}_{\mathcal{A}}$, with $\text{res}_{\mathcal{A}}(X) = Y$ if and only if $\overline{\text{res}}_{\mathcal{A}}(\chi(X)) = \chi(Y)$. Thus, $\overline{\text{res}}_{\mathcal{A}} : \{0,1\}^n \to \{0,1\}^n$ is defined by $\overline{\text{res}}_{\mathcal{A}} = \chi \circ \text{res}_{\mathcal{A}} \circ \chi^{-1}$.

Relating terms and reactions. Here, we study the relation between the size (number of terms) of the logical representation $\boldsymbol{\Phi}_F$ describing a Boolean function $F : \{0,1\}^n \to \{0,1\}^n$ and the number of reactions in the system $\mathcal{A}$ implementing $F = \overline{\text{res}}_{\mathcal{A}}$.

In Section 7 of [6], the authors show how Boolean formulas in DNF can be transformed into reaction systems, where the value of the formula is translated into generating a specific entity (or not).

This connection turns out to be crucial in our considerations. Apart from the output of the reaction, it shows that reactions behave like terms. Each reaction is enabled on the subset of the Boolean variables that are positive in the term and inhibited on the Boolean variables that are negated in the term.

The following lemma captures this correspondence.

Lemma 7 *For every logical term γ over n Boolean variables, there exists a reaction $a(\gamma)$ over a background set S of n entities, such that for every $X \subseteq S$, $a(\gamma)$ is enabled in X if and only if $\gamma(\chi(X)) = 1$.*

Proof Given a term γ over the Boolean variables $v_1, \ldots, v_n$, choose n entities and order them to form the background set $S = \{u_1, \ldots, u_n\}$ such that u_i corresponds to v_i, for $1 \le i \le n$. Designate a specific entity u_s as an output entity. We construct a reaction $a(\gamma)$, such that whenever this reaction is enabled, it will produce u_s.

Consider the literals in γ. Let P_γ be the set of indices of those literals that are variables v_i and N_γ be the set of indices of the literals that are negations of variables $\bar{v}_j$. Thus, $\gamma = \bigwedge_{i \in P_\gamma} v_i \cdot \bigwedge_{j \in N_\gamma} \bar{v}_j$. Then, $\gamma(b_1 \ldots b_n)$ is true if and only if each $b_i = 1, i \in P_\gamma$, while each $b_j = 0, j \in N_\gamma$.

For a set of indices $I \subseteq \{1, \ldots, n\}$, let $S_I = \{u_i \mid i \in I\}$ be the corresponding set of entities. Then, $a(\gamma) = (S_{P_\gamma}, S_{N_\gamma}, \{u_s\})$ satisfies the requirement, as it is enabled in a state X if and only if each of the u_i, for $i \in P_\gamma$ is present in X, while none of the u_j, for $j \in N_\gamma$ is present in X. That is, $a(\gamma)$ is enabled in X if and only if $\gamma(\chi(X)) = 1$. □

The construction used in the proof of the above lemma can be used to transform a logical representation Φ, i.e., a set of formulas $\varphi_1, \ldots, \varphi_n$ in DNF, into a reaction system, by converting the terms occurring in the formulas into reactions. To define a reaction system from Φ, we will make the convention that the entity u_i is produced when φ_i is true, for $i = 1, \ldots, n$. The following example illustrates this, for one output value.

Example 8 Suppose we are given the Boolean formula $\varphi_1 = \bar{v}_1 v_2 + \bar{v}_1 \bar{v}_2 v_3 + \bar{v}_3 v_4$ with variables $\{v_1, \ldots, v_4\}$ and terms γ_1, γ_2 and γ_3, respectively. Let $S = \{u_1, \ldots, u_4\}$ and we necessarily choose for output value $u_1 \in S$. Then, for γ_1, $P = \{2\}$ and $N = \{1\}$. Hence, $a(\gamma_1) = (\{u_2\}, \{u_1\}, \{u_1\})$. Similarly, for γ_2, $P = \{3\}$ and $N = \{1, 2\}$. Thus, $a(\gamma_2) = (\{u_3\}, \{u_1, u_2\}, \{u_1\})$. Finally, $a(\gamma_3) = (\{u_4\}, \{u_3\}, \{u_1\})$. ⌟

If a term γ occurs in several formulas of the logical representation Φ, then when γ is satisfied, according to the construction above, the output entities corresponding to each of these formulas must be generated by the reaction system implementing Φ. Rather than generating each entity with a separate reaction, these output entities can be combined in the product set of a single reaction. Hence, we have a direct correspondence between the number of distinct terms and the number of reactions.

Corollary 9 *Let $F : \{0,1\}^n \to \{0,1\}^n$ be a Boolean function, with a logical representation Φ in n variables, and with a total number of k distinct terms. Then, there exists a reaction system $\mathcal{A}$ over n background entities $S = \{u_1, \ldots, u_n\}$, with k reactions, such that $\overline{\text{res}}_{\mathcal{A}} = F$.*

Proof Let $\Phi = \varphi_1, \ldots, \varphi_n$. Consider a term γ that occurs in Φ. Define the index set $O_\gamma = \{\ i \mid \gamma \text{ occurs in } \varphi_i\ \}$ of γ occurrences in Φ.

As in the proof of Lemma 7, for a set of indices $I \subseteq \{1, \ldots, n\}$, let $S_I = \{u_i \mid i \in I\}$ be the corresponding set of entities. Now define $a(\gamma) = (S_{P_\gamma}, S_{N_\gamma}, S_{O_\gamma})$, where (again as in the proof of Lemma 7) the P_γ and N_γ are the indices of the variables occurring positively and negated in γ, respectively. As in the proof of Lemma 7, we argue that $a(\gamma)$ generates u_i in state X if and only if φ_i is true for $\chi(X)$. □

So, let F be a multi-valued Boolean function. If there exists a logical *representation* Φ of F with a total of k distinct terms, then there exists a reaction system *implementation* $\mathcal{A}$ with k reactions for F, such that $\overline{\text{res}}_{\mathcal{A}} = F$. It can easily be verified that the result also holds in reverse, by walking the construction backwards: the result function of a reaction system with k reactions can be logically represented using k distinct terms.

A reaction system $\mathcal{A} = (S, A)$ is called *minimal*, if for every reaction system $\mathcal{B} = (S, B)$, such that $\mathcal{A} \sim \mathcal{B}, |A| \leq |B|$. That is, a reaction system $\mathcal{A}$ is minimal if it has the smallest number of reactions of all reaction systems equivalent to it. A set $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ of logical formulas in DNF is *minimal* if its total number of distinct terms over all its formulas is minimal among all logically equivalent sets of formulas.

Theorem 10 *For every Boolean function $F : \{0,1\}^n \to \{0,1\}^n$ there exists a reaction system $\mathcal{A}$, such that, $\mathcal{A}$ is minimal if and only if the corresponding logical representation Φ of F is minimal.* □

Note that, the proof of Proposition 6 already indicates a way to translate Boolean functions to reaction systems, if as before, we equate sets with bit-vectors via their characteristic function. That construction yields maximally inhibited reactions, which correspond to minterms, rather than the more general terms, as all entities are present in the reactant or inhibitor sets. The size of the reaction system is 2^n, where n is the number of entities of the system. Its logical representation consists of almost the same number of minterms, except that we can omit states (minterms) that map into the empty set. The minterms corresponding to these reactions will not occur in any of the formulas, as the reactions do not contribute any entities.

Example 11 Consider the reaction system $\mathcal{A}$ from Example 2, as depicted by its 0-context graph in Figure 1. The edge $(\{2,4\}, \{1,2,3\})$ is translated into the maximally inhibited reaction $(\{2,4\}, \{1,3\}, \{1,2,3\})$. In the logical representation of $\overline{\text{res}}_{\mathcal{A}}$ this corresponds to the minterm $\bar{v}_1 v_2 \bar{v}_3 v_4$ (specifying when the reaction is enabled) that occurs in the formulas φ_1, φ_2, and φ_3 (specifying when the corresponding entity is produced).

The problem of deciding the equivalence of Boolean formulas has a known complexity, and by the strong correspondence between formulas and reaction systems, this can be translated to the complexity of the equivalence of reaction systems. It is well known that the problem of deciding satisfiability for formulas in conjunctive normal from is NP-complete. Dually, deciding whether a formula in DNF is satisfiable is co-NP-complete. As we can easily construct a reaction system representing a tautology (always produce an entity) one obtains the following result, stated in [6].

Corollary 12 *Functional equivalence of reaction systems is co-NP-complete.*

Minimization of reaction systems turns out to be a harder problem. The complexity of the related problem of minimization of Boolean formulas in DNF was obtained in [19].

Problem (MIN DNF) Given a DNF formula φ and an integer k, is there a DNF formula equivalent to φ with k or fewer occurrences of literals?

This is not exactly the problem we are looking for, counting literals instead of terms, but gives an indication of the complexity. As reported in [19, Theorem 4] problem MIN DNF is Σ_2^P-complete. (Compare: NP equals Σ_1^P, a lower level in the polynomial hierarchy [14].)

Corollary 13 *Minimization of reaction systems (counting the number of entities in reactants and inhibitors) is Σ_2^P-complete.*

4 Extracting reactions from the total result function

In Section 2, we demonstrated how given a reaction system $\mathcal{A}$ (specified by its set of reactions A), we can obtain its global behaviour, $\text{res}_{\mathcal{A}}$, represented by the 0-context graph $G^0_{\mathcal{A}}$ (see Examples 2 and 4). Also in that section, we described the reverse process, i.e., how given a function $f : \mathcal{P}(S) \to \mathcal{P}(S)$, which may be represented by a one-out graph G with vertices $\mathcal{P}(S)$, we can construct a reaction system $\mathcal{A}' = (S, A')$ using a set of maximally inhibited reactions A', such that $\text{res}_{\mathcal{A}'} = f$, or equivalently, such that $G^0_{\mathcal{A}'} = G$ (see Proposition 6).

In this section, we take the reverse process one step further: we minimize the set A' to obtain B, such that $\mathcal{B} = (S, B) \sim \mathcal{A}'$ and, consequently, $\text{res}_{\mathcal{B}} = \text{res}_{\mathcal{A}'}$. For that, we use the equivalence of reaction systems to Boolean formulas, as discussed in Section 3, and well-known methods for minimizing Boolean formulas like Karnaugh maps and the Espresso algorithm, see [11, 15, 16].

The Problem Assume the global $\text{res}_{\mathcal{A}}$ function of a reaction system $\mathcal{A} = (S, A)$ is given, presumably by a one-out graph $G^0_{\mathcal{A}}$ on vertices $\mathcal{P}(S)$, and assume further that the set A of individual reactions that produced the result function is not known. We want to *decompose* the result function $\text{res}_{\mathcal{A}}$ into a (preferably) minimal number of reactions $b_1, \ldots, b_k$, comprising a set of reactions B, which in turn, defines a reaction system $\mathcal{B}(S, B)$, such that, $\text{res}_{\mathcal{B}} = \text{res}_{\mathcal{A}}$, or equivalently, $G^0_{\mathcal{B}} = G^0_{\mathcal{A}}$.

Input A one-out graph $G^0_{\mathcal{A}}$ with $V(G) = \mathcal{P}(S)$ for some finite set $S = \{u_1, \ldots, u_n\}$, representing the result function $\text{res}_{\mathcal{A}}$ of a reaction system $\mathcal{A}$.

Output A minimal reaction system $\mathcal{B}$ such that $G^0_{\mathcal{B}} = G^0_{\mathcal{A}}$.

Step 1 For each (X, Y) edge, construct the (maximally inhibited) reaction $a_X = (X, S \setminus X, Y)$, as in Proposition 6, and consider the set $A' = \{a_X \mid X \subseteq S\}$ of the $2^{|S|}$ reactions describing $G^0_{\mathcal{A}}$ and the result function $\text{res}_{\mathcal{A}}$ that it represents. Thus, we obtain $\mathcal{A}' = (S, A')$ with $G^0_{\mathcal{A}'} = G^0_{\mathcal{A}}$ and $\text{res}_{\mathcal{A}'} = \text{res}_{\mathcal{A}}$, i.e. $\mathcal{A}' \sim \mathcal{A}$.

Step 2 For every reaction $a_X \in A'$ construct the corresponding minterm $\gamma(x) = \ell_1, \ldots, \ell_n$, using the bijection between terms and reactions as in Lemma 7, as follows. Let $x = \chi(X)$, and for $1 \le i \le n$, we choose the literal $\ell_i = v_i$ if and only if $u_i \in X$ and $\ell_i = \bar{v}_i$ otherwise. (At this point, we have 2^n minterms γ, one for each $X \subseteq S$.)

From our example, for $X = \{1, 2\}$, $x = 1100$ and we set $\gamma(x) = v_1 v_2 \bar{v}_3 \bar{v}_4$.

Step 3 We consider the elements of the domain. For each $u_i \in S$, construct a Boolean formula φ_i as a disjunction over all $\gamma(x)$ where $x = \chi(X)$ and (X, Y) is an edge such that $u_i \in Y$.

Note that, in this step, if a_X does not contain u_i in its third component Y, it will not be represented by the minterm $\gamma(x)$ in φ_i, where $x = \chi(X)$.

Example 14 The logical formulas φ_i for our example are as stated below. Consider for instance the first background entity $u_1 = 1$. In Figure 1, we see that there are nine edges leading into a set that contains 1, like $\{1, 4\} \to \{1, 2\}$ and $\{2\} \to \{1, 3\}$, which correspond to nine maximally inhibited reactions, like $(\{1, 4\}, \{2, 3\}, \{1, 2\})$ and $(\{2\}, \{1, 3, 4\}, \{1, 3\})$. and thus, to nine minterms in φ_1, like $v_1 \bar{v}_2 \bar{v}_3 v_4$ and $\bar{v}_1 v_2 \bar{v}_3 \bar{v}_4$.

In a similar way, we collect the edges where output value $u_i = i$ is generated, to obtain formula φ_i, for $i = 2, 3, 4$.

$$\begin{aligned}
\varphi_1 =& v_1 v_2 \bar{v}_3 v_4 + v_1 \bar{v}_2 \bar{v}_3 v_4 + \bar{v}_1 v_2 v_3 v_4 + \bar{v}_1 v_2 v_3 \bar{v}_4 + \bar{v}_1 v_2 \bar{v}_3 v_4 + \bar{v}_1 v_2 \bar{v}_3 \bar{v}_4 \\
&+ \bar{v}_1 \bar{v}_2 v_3 v_4 + \bar{v}_1 \bar{v}_2 v_3 \bar{v}_4 + \bar{v}_1 \bar{v}_2 \bar{v}_3 v_4 \\
\varphi_2 =& v_1 v_2 \bar{v}_3 v_4 + v_1 v_2 \bar{v}_3 \bar{v}_4 + v_1 \bar{v}_2 v_3 \bar{v}_4 + v_1 \bar{v}_2 \bar{v}_3 v_4 + v_1 \bar{v}_2 \bar{v}_3 \bar{v}_4 + \bar{v}_1 v_2 \bar{v}_3 v_4 \\
&+ \bar{v}_1 \bar{v}_2 v_3 v_4 + \bar{v}_1 \bar{v}_2 v_3 \bar{v}_4 + \bar{v}_1 \bar{v}_2 \bar{v}_3 v_4 \\
\varphi_3 =& v_1 v_2 \bar{v}_3 v_4 + v_1 v_2 \bar{v}_3 \bar{v}_4 + v_1 \bar{v}_2 v_3 \bar{v}_4 + \bar{v}_1 v_2 \bar{v}_3 v_4 + \bar{v}_1 v_2 \bar{v}_3 \bar{v}_4 \\
\varphi_4 =& \bar{v}_1 \bar{v}_2 v_3 v_4 + \bar{v}_1 \bar{v}_2 v_3 \bar{v}_4
\end{aligned}$$

⌟

Step 4 This is the minimization step, which can proceed in one of the following ways: *Step 4.1* Minimize the number of terms (and the total number of literals in them) for each individual φ_i separately, or *Step 4.2* Minimize the number of common terms for all φ_i in $\boldsymbol{\Phi}$ taken together, or *Step 4.3* Minimize the system when some states are disregarded (*don't care states*).

Each of these minimization problems is discussed in a separate subsection below.

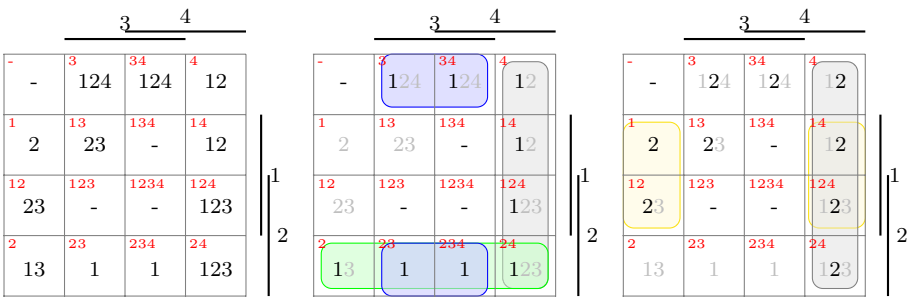


Fig. 2 (1) The 0-context graph, depicted in Karnaugh map style, see Example 2. (2) A minimal number of reactions generating entity 1. (3) Large clusters (most of the clusters) generating entity 2

4.1 Minimizing Reactions Producing a Single Entity

If we are considering a collection of biochemical reactions that produce a certain set of entities, we may be interested to control (or minimize), e.g. due to "cost", only the number of reactions that produce a specific entity, say u_j. Such minimization will also result in a minimal total number of literals, counting multiplicities, related to this output entity u_j, which means that we will use a minimal total number of reactants and inhibitors (resources) in the set of reactions producing that entity. When the cost considerations of using smallest number of reactants and inhibitors in the reactions producing a certain entity are more important than the minimal number of reactions producing all entities, then we use *Step 4.1*. Problems concerning reactions with minimal and almost minimal resources have been studied in [8, 17].

We illustrate our minimization approach using well-known methods for minimizing Boolean formulas, heuristic Karnaugh maps, and automatic minimization. Of course, the algorithmic approach is open for large scale models, but the Karnaugh approach will give us some additional insight.

Karnaugh maps. Karnaugh maps are a popular heuristic method to minimize single-valued Boolean functions [11]. We can represent the 0-context graph of a background set of limited size in Karnaugh map style, by identifying each edge (X, Y) with a single cell (square) in the Karnaugh map, see Figure 2 (1). The entities of each of the sixteen possible states $X \subseteq S$ are listed in the top left corner of each cell. The 16 states are organized in such a way that each fixed Boolean combination of entities forms a consecutive strip when the diagram is viewed as a torus. For example, if "1 and 2" are in X, the states (cells) are in the third row, while "not 1 and not 3" in X means the states are in the four corner cells. The result (output) Y of the system, for each of the sixteen states, is given inside the cells. Hence, each edge (X, Y) is represented by X in the top left corner and Y in the middle of the respective cell.

In this way, the diagram in Figure 2 (1) represents the graph $G^0_{\mathcal{A}}$ from Figure 1.

Let the designated output entity be $u_1 = 1$ and disregard other output entities in the cells for the moment, see Figure 2 (2). The nine cells that contain output 1 (i.e., u_1) are already represented by the nine minterms in the formula φ_1. We minimize that formula to a smaller number of terms, using "clustering" or glueing together neighboring cells, as many as possible, to form strips on the torus (when the Karnaugh map is folded and glued like a torus). This leads to the (blue, purple, and gray) regions described by the terms $\bar{v}_1 v_2$, $\bar{v}_3 v_4$, and $\bar{v}_1 v_3$, respectively. All together, these regions cover all 1's and hence describe φ_1 perfectly. That is, φ_1 reduces to $\varphi'_1 = \bar{v}_1 v_2 + \bar{v}_3 v_4 + \bar{v}_1 v_3$, equivalent to the following three reactions $(\{2\}, \{1\}, \{1\}), (\{4\}, \{3\}, \{1\})$, and $(\{3\}, \{1\}, \{1\})$.

For the output entity $u_2 = 2$, we immediately observe two large clusters corresponding to the terms $v_1 \bar{v}_3 + \bar{v}_3 v_4$, see Figure 2 (3). The remaining three cells can be recombined in several ways (partially over lapping with other 2 where possible), but we cannot do better than two 'domino's', for example $v_1 \bar{v}_2 \bar{v}_4 + \bar{v}_1 \bar{v}_2 v_3$ or $\bar{v}_2 v_3 \bar{v}_4 + \bar{v}_1 \bar{v}_2 v_4$. The last two entities are simple.

We summarize.

$$\varphi'_1 = \bar{v}_1 v_2 + \bar{v}_3 v_4 + \bar{v}_1 v_3$$
$$\varphi'_2 = v_1 \bar{v}_3 + \bar{v}_3 v_4 + v_1 \bar{v}_2 \bar{v}_4 + \bar{v}_1 \bar{v}_2 v_3$$
$$\varphi'_3 = v_2 \bar{v}_3 + v_1 \bar{v}_2 v_3 \bar{v}_4$$
$$\varphi'_4 = \bar{v}_1 \bar{v}_2 v_3$$

Note that we have 'accidentally' chosen the term $\bar{v}_1 \bar{v}_2 v_3$ in formula φ'_2 (we had several options) which also occurs in φ'_4. This means that we can join the two corresponding reactions $(\{3\}, \{1, 2\}, \{2\})$ and $(\{3\}, \{1, 2\}, \{4\})$ into $(\{3\}, \{1, 2\}, \{2, 4\})$. Note that choosing another option, might not give us this reduction. This is exactly one of the events that make minimizing multiple outputs hard.

Note that, the logical 'rule' behind finding larger Karnaugh blocks uses distributive law to combine two terms differing only in opposite literals, e.g., $\gamma \cdot v + \gamma \cdot \bar{v} = \gamma \cdot (v + \bar{v}) = \gamma$. This is similar to replacing the two reactions $(R \cup \{e\}, I, P)$ and $(R, I \cup \{e\}, P)$ by (R, I, P).

Table 1 Retrieving reactions from terms, Espresso minimization, separate outputs

term	used in	corresponding reaction
C′ D	FA, FB	$b_1 = (\{4\}, \{3\}, \{1,2\})$
A′ C	FA	$b_2 = (\{3\}, \{1\}, \{1\})$
A′ B	FA	$b_3 = (\{2\}, \{1\}, \{1\})$
A′ B′ C	FB, FD	$b_4 = (\{3\}, \{1,2\}, \{2,4\})$
B′ C D′	FB	$b_5 = (\{3\}, \{2,4\}, \{2\})$
A C′	FB	$b_6 = (\{1\}, \{3\}, \{2\})$
A B′ C D′	FC	$b_7 = (\{1,3\}, \{2,4\}, \{3\})$
B C′	FC	$b_8 = (\{2\}, \{3\}, \{3\})$

Espresso. As minimization is an important facet of digital design, several tools are available, from small scale to industrial strength. We have chosen the heuristic minimizer Espresso [15], together with Logic Friday[1], a free Windows program that provides a graphical interface to Espresso. Adhering to the syntax of Espresso, we use A to D to represent the variables v_1 to v_4 for the entities 1 to 4. The four logical formulas $\varphi_1, \ldots, \varphi_4$ from **Step 3** are entered as FA, FB, FC, and FD, listed below.

```
FA = A B' C' D + A' B' C' D + A' B C D' + A' B C D + A' B C' D'
   + A' B C' D + A B C' D + A' B' C D + A' B' C D' ;
FB = A B' C' D + A' B' C' D + A B C' D' + A B' C D' + A B' C' D'
   + A' B C' D + A B C' D + A' B' C D + A' B' C D'  ;
FC = A B C' D' + A B' C D' + A' B C' D' + A' B C' D + A B C' D ;
FD = A' B' C D + A' B' C D'  ;
```

We now use this tool with settings that "minimize each output separately" and we obtain the following equations:

```
FA = D C' + A' C + A' B ;
FB = A' B' C + D' B' C + A C' + D C';
FC = A D' B' C + B C';
FD = A' B' C;
```

One can see that Espresso returned mostly the same individually minimized formulas FA through FD, just as applying the Karnaugh heuristic did, where we obtained φ'_1, φ'_2, φ'_3 and φ'_4. The difference is that we made a different choice for φ'_2 vs. FB.

Translating the individually minimized formulas back into reactions is done in Table 1.

One feature of this minimization to notice is that for each individual i, φ'_i not only contains the smallest number of terms, but also, the smallest number of total literals used in these terms, which of course, is related to the fact that we aimed at maximal clusters or cells with output i in the Karnaugh map. This is due to maximizing the separate blocks of i to cover as many as possible, even if they overlap. That minimizes the variables used. Also, every term in such a formula φ'_i is essential, i.e. contains a cell that is not contained in any of the other clusters already used in the formula.

Hence, to minimize the number of reactions we will minimize the number of terms for the combined set of Boolean functions F_i.

4.2 Minimizing the number of reactions considering all output entities

In other cases, we will need to obtain the multiple outputs with a minimal total number of reactions. In other words, we start with the maximally inhibited set of reactions A' and find a set of reactions of minimal cardinality B, such that $\mathrm{res}_{A'} = \mathrm{res}_B$. Hence, to minimize the number of reactions, we will minimize the total number of distinct terms for the combined set of Boolean functions F_i.

Again, we input our original FA, FB, FC, and FD from the end of **Step 3** and our tool finds the following minimized set of equations. We have set the parameters of Espresso such that, rater than using a heuristic approach, the global minimum is obtained (at the cost of a longer computation, of course).

```
FA = A' B' C + C' D + A' B ;
FB = A' B' C + C' D + A B' C D' + A C';
FC = A B' C D' + B C';
FD = A' B' C;
```

[1] Logic Friday 1.1.4, (c) Steve Rickman.

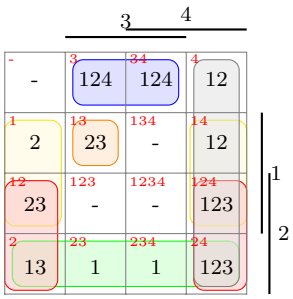


Fig. 3 Six reactions generating the original 0-context graph, see Example 2. Also terms as retrieved by Espresso minimization, see Section 4.2

Table 2 Retrieving reactions from terms, Espresso minimization, multiple outputs, Section 4.2

term	used in	corresponding reaction
A′ B′ C	FA, FB, FD	$a_4 = (\{3\},\{1,2\},\{1,2,4\})$
C′ D	FA, FB	$a_5 = (\{4\},\{3\},\{1,2\})$
A′ B	FA	$a_2 = (\{2\},\{1\},\{1\})$
A B′ C D′	FB, FC	$a_6 = (\{1,3\},\{2,4\},\{2,3\})$
A C′	FB	$a_1 = (\{1\},\{3\},\{2\})$
B C′	FC	$a_3 = (\{2\},\{3\},\{3\})$

The terms represent reactions, where terms common to one or more functions, can be combined in generating the corresponding entities. Translating this solution into reactions, we obtain our original reaction system $\mathcal{A}$ from Example 2 back, see Table 2. However, note that we did not start with the $\mathcal{A}$, because we assumed we do not know A. Rather, we started with its global result function $\mathrm{res}_{\mathcal{A}}$, built a maximally inhibited set of reactions A' that is functionally equivalent to it, and then, minimized A' to obtain a set B functionally equivalent to it. This process confirmed that the original set A was minimal.

It is important here to note that if, in general, we know the original set of reactions A, which is used to define $\mathrm{res}_{\mathcal{A}}$, we do not have to go through A' to find a minimal B (or verify that A is minimal). Espresso has an option to input the original reactions A (by coding them into logical formulas φ_i similar to the above) and obtain a functionally equivalent minimal set of reactions B (which may be the same as A). In this case, the initial φ_i will not necessarily consist of minterms, but of terms (possibly fewer than the maximum).

Comparison of the two models. We now compare the group minimization with the individual minimization. The individual minimization obtained in *Step 4.1* of our example produces the terms and the corresponding reactions presented in Table 1. We notice that the group minimization algorithm used in *Step 4.2* produces a smaller set of reactions, as seen in Table 2. However, the individual minimization algorithm from *Step 4.1* produces a smaller (or the same) total number of entities (counting multiplicities) necessary for producing each output entity.

For example, to produce the entity 1, the reactions b_j (namely b_1, b_2, and b_3) use six entities (counting multiplicities) in their combined reactant and inhibitor sets, that is, $\{4\},\{3\},\{3\},\{1\},\{2\},\{1\}$, while the reactions a_i (namely a_2, a_4, and a_5) use reactant and inhibitor sets $\{3\},\{1,2\},\{4\},\{3\},\{2\},\{1\}$, i.e. seven entities (counting multiplicities). This difference can be seen in the Karnaugh maps in Figure 2 (2) and Figure 3. In the individual case, represented in Figure 2 (2), the cluster consisting of the top middle and the bottom middle cells is bigger and, thus, defined by less literals, whereas the top middle cells only, in Figure 3, contribute more literals, as they form a smaller cluster. Such a minimization may be desirable in certain biological settings, when using less reactants and inhibitors is more cost effective, than having a smaller number of reactions.

4.3 Minimizing the number of reactions for a subset of the states

Often, there are situations where one wants to design a (biological) system where only a subset of all possible states are considered. The behavior of the system at the other states is irrelevant, because it is assumed that those states will not occur during the execution of the system.

Ignoring a part of the state space is very common in the design of digital systems. For example, if we want to design a system with ten states, we represent the state set with four bits. This will give sixteen available states, of which only ten will be used. When implementing the digital design by an electronic circuit, the output value for the superfluous states is specified as *don't care*. The output of these states can then be taken arbitrarily, and is chosen so that the implementation can be optimized.

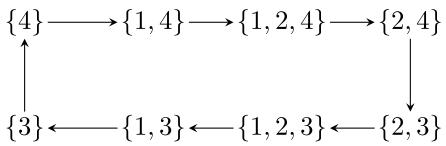
The Karnaugh map approach is particularly suited for situations like this. Cells that are marked as do not care *can* be

combined with surrounding cells, if that helps to obtain less, and thus larger, covering rectangles (clusters). Translated in the realm of reaction systems, this means that we will obtain less reactions with a smaller set of reactants or inhibitors.

We use an example from [12] to illustrate situations with do not care states. A *Gray code* is an ordering of the elements in $\{0,1\}^n$ such that the successive bitstrings differ by at most one bit. Example 2.3 from [12] implements a reaction system that mimics the forward and backward generation of a Gray code. Its six reactions are as follows.

$a = (\{4\}, \{2,3\}, \{1,4\})$ $b = (\{1,4\}, \{3\}, \{2,4\})$ $c = (\{2,3\}, \{4\}, \{1\})$
$d = (\{2\}, \{1\}, \{2,3\})$ $e = (\{1,3\}, \{4\}, \{3\})$ $f = (\{3\}, \{1,2\}, \{4\})$

As in our exposition above, the two bits are represented by the presence of two elements 1, 2 in the state space. Moreover, the Gray code is repeated and travelled twice, to obtain the state space below. The presence of elements 3 or 4 signals the forward or backward traversal of the Gray code.



Note that exactly one of the elements 3, 4 is present in each state of this design. Thus, half of the state space remains unspecified. Hence, using our Karnaugh map style presentation of the state space for the Gray code example, we obtain the diagram from Figure 4, where x denotes do not care states.[2]

After entering the required state transitions as a multi-valued function truth table to Espresso, minimization yields the following proposed implementation:

```
FA = B D' + B' D;
FB = A' B + A D;
FC = A D' + A' B;
FD = A' B' + A D;
```

Translating the 'gates' into reactions as before, we obtain six reactions, see Table 3.

Note that, if we look at the reactions, we see that the two do not care columns are just copied from their neighbors in the final solution. Also, observe that these reactions do not adhere to the 'modern' requirement that reactant and inhibitor sets must be nonempty. In the example, this is easily resolved. As entities 3 and 4 are complementary in the states for the Gray code, they can also be added in a complementary fashion to the reactant and inhibitor sets. This will change the global behavior of the system, but not in the eight state subspace we are interested in.

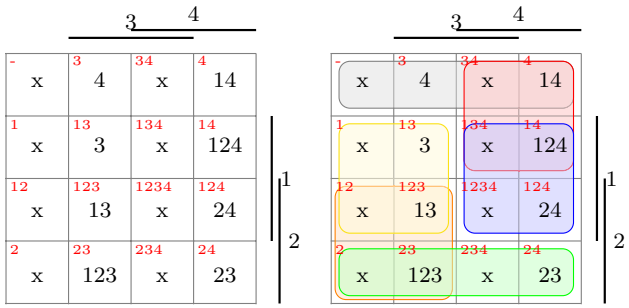


Fig. 4 The Gray code example with don't care states marked by 'x'

Table 3 Reactions obtained for the Gray code example from Section 4.3

product	used in	representative reaction
B D′	FA	$b_1 = (\{2\}, \{4\}, \{1\})$
B′ D	FA	$b_2 = (\{4\}, \{2\}, \{1\})$
A′ B	FB, FC	$b_3 = (\{2\}, \{1\}, \{1,3\})$
A D	FB, FD	$b_4 = (\{1,4\}, \emptyset, \{2,4\})$
A D′	FC	$b_5 = (\{1\}, \{4\}, \{3\})$
A′ B′	FD	$b_6 = (\emptyset, \{1,2\}, \{4\})$

[2] Although the cells of the Karnaugh map themselves are ordered according to a Gray code, it is still a surprise to see the code clearly spelled out in the diagram.

5 The Espresso Minimization Algorithm

Algorithm 1: ESPRESSO pseudocode for the minimization of the combined set F of functions F_i (adapted from [15])

input : 0-context graph $G^0_{\mathcal{A}}$ of a reaction system $\mathcal{A} = (S, A)$
output: A minimum number of reactions to obtain a functionally equivalent reaction system $\mathcal{B} = (S, B)$

```
1  begin
       // P is the set of 2^n possible truth value assignments for the set
          of n variables, v_1, v_2, ..., v_n
       // Str is an array of binary strings x_i ∈ {0,1}^n to hold the elements
          of P
       // F is the ON-set of the multiple-output function
       // D is the DC-set of the multiple-output function
       // R is the OFF-set of the multiple-output function
       // cost(F) first considers the number of cubes in F and then the
          number of literals to implement F
2      Strings ← GETSTRINGS(P);                          // Initialize the array
3      COMPUTECOVERS(Str);                  // Get F and D from G^0_A vertices
4      F_old ← F;                          // Save original cover for verification
5      R ← COMPLEMENT(F + D);                          // Compute the complement
6      F ← EXPAND(F, R);                                  // Initial expansion
7      F ← IRREDUNDANT(F, D);                           // Initial irredundant
8      E ← ESSENTIAL(F, D);                           // Detect essential primes
9      F ← F − E;                                    // Remove essentials from F
10     D ← D + E;                                        // Add essentials to D
11     while cost(F) < φ_2 do
12         φ_2 ← cost(F);
13         while |F| < φ_1 do
14             φ_1 ← |F|;
15             F ← REDUCE(F, D);
16             F ← EXPAND(F, R;
17             F ← IRREDUNDANT(F, D);
18         F_i ← LASTGASP(F, D, R);
19     F ← F + E ;                                    // Return essentials to F
20     D ← D − E;
21     if !VERIFY(F, D, F_old) then
22         exit(verify error);
23     return F;
24     return B;              // The set of reactions in B = (S, B);
       // This last step is the result of postprocessing of each term from
          the set of returned minimized functions F to record each reaction
          in B
25 
```

In this section, we briefly review the basic execution steps in *Espresso MV*, a multiple-input, multiple-output variant of the Espresso algorithm. The relevant pseudocode is shown in Algorithm 1. For further details regarding the description and operation of each execution step, the reader is referred to [15].

The set of binary strings $x \in \{0,1\}^n$ for which $\varphi(x) = 1$ is labeled as the ON-set in Espresso. Note that for a multiple-output function, the ON-set of the combined set of functions F_i is the combined set of the ON-sets for each function, where each string is labeled as x_i and the index i corresponds to the respective function index. Similar labels, OFF-set and DC-set, are used in an analogous way when the function values for a minterm are 0, and 0 or 1 respectively.

Pre-processing steps are performed first over the input, the 0-context graph $G^0_{\mathcal{A}}$ of a reaction system $\mathcal{A} = (S, A)$. As the total effect of the result function $\mathrm{res}_{\mathcal{A}}$ is represented by the edges in the 0-context graph $G^0_{\mathcal{A}}$, the total function $\mathrm{res}_a(X)$, for every $X \subseteq S$, is directly used to assign the 2^n states of the power set $\mathcal{P}(S)$ of the maximally inhibited reactions in $\mathcal{A}'$ to the set of binary strings in the three covers: the ON-set cover, the OFF-set cover, as well as the DC-set cover.

Note that to reduce the number of preprocessing operations, only two of the three covers could be produced in this way and COMPLEMENT can be used afterwards to compute the third set cover at the expense of adding processing complexity. Other heuristic algorithms have been proposed to mitigate that issue for functions with large complement size [10].

The Espresso algorithm uses a heuristic approach to reduce the original function cover with the help of a cost function that minimizes the number of cubes and the number of literals in each of the currently considered covers of the multiple output function that is viewed as the combined set of functions F_i. A cost function is used to minimize the number of cubes in the current cover and the total number of literals, where $|F|$ represents the number of 1s in the function cover.

The key processing steps include: EXPAND, IRREDUNDANT, ESSENTIAL, and REDUCE as follows. An *implicant* of the function is a term (a.k.a. *cube*), as shown in Figure 2, that does not contain any minterm representing a multiplication of the literals in string $x_i \in$ OFF-set of the function. A *prime implicant* of the function is an implicant that is not contained by any other implicant. Before processing begins, *essential prime implicants*, or implicants which contain some γ not contained in any other implicant, are identified using ESSENTIAL and are excluded. EXPAND is used to maximize the size of each implicant, so that other (smaller) implicants become covered and can be deleted. EXPAND is used to "expand" each cube (analogous to the color-coded clusters depicted in Figure 2) of the ON-set cover into a prime implicant.

In multiple-output function mode of operation, Espresso always 'expands' to multi-output prime implicants whenever applicable. The cubes are labeled with multiple indices accordingly: e.g. $cube_{ijp}$, to indicate that the prime implicant spans across the outputs of functions F_i, F_j, and F_p. EXPAND is followed by execution of IRREDUNDANT to select a minimal subset of this set of prime implicants that constitutes a function cover. IRREDUNDANT makes a cover *irredundant* by deleting a maximum number of *redundant* implicants. Note that an irredundant cover is minimal. This step can be implemented using a *containment test*, which in turn can be implemented using a *tautology test*.

As already noted in Section 2, the functional equivalence problem for reaction systems has been previously shown to be co-NP-complete [4, Section 7]. The proof presented in [4] reduces it to the tautology problem for Boolean formulas, which coincidentally comprises the core of the execution iterations in ESSENTIAL. The main idea is to iterate over the cover until no improvement in the cost function is seen and the solution is moved away from the local minimum without increasing the number of cubes in the cover. The latter goal is achieved by the processing inside the REDUCE module. The reader should note that an improvement in the cost function does not guarantee that a global minimum is reached. The optional execution of LASTGASP ensures that no implicant can replace any implicant in the cover to reduce the cardinality of the cover. When the outer loop terminates, an irredundant cover of prime implicants is found. At the end, *essential prime implicants* are returned to the cover before processing of the cover is completed.

Additional steps, not shown in Algorithm 1 can be added to find a better minimization solution at the expense of potentially adding excessive processing delays. The basic idea is to rely on generating the set of *all* prime implicants, then executing IRREDUNDANT over that set, and finally, solving the covering problem.

A variant of the Espresso MV algorithm, referred to as Espresso EXACT [15], presents an improvement over this strategy. Espresso EXACT is an approximate algorithm of polynomial complexity that can confirm the optimality of the solution in some of the input cases.

6 Concluding remarks

Reaction systems arose as a theoretical model of biochemical reactions. Due to various considerations for producing certain biomolecular products or biomolecules, we may be concerned with minimizing the number of reactions producing a single entity, or some of the entities, or all of the entities. In this paper, we explained how our approaches will vary regarding each of these different goals. In all cases, we converted the problem of minimizing the number of reactions to minimizing Boolean functions (single-valued or multi-valued) or, else, minimizing Boolean formulas.

By reducing this reaction systems problem to a Boolean function or a Boolean formula counterpart, we essentially reduce this problem to logic optimization for the purposes of logic synthesis in electronics, where many well-known approaches for such minimization exist.

Future directions of research include developing applications for the three types of minimization discussed here both in biochemical reactions and logic circuits, as well as, expanding the theoretical investigations of minimization of reaction systems.

Acknowledgements This research was initiated at and facilitated by the 2nd International Workshop in Reaction Systems and 1st School in Reaction Systems held on June 3, 2019, organized by Nicolaus Copernicus University, Toruń, Poland. DG and HJH acknowledge travel support from InterAPS (International Academic Partnerships in Sciences with Nicolaus Copernicus University) and from the University of North Florida, USA. The authors thank Matthew Thomas for useful comments on a previous version of this paper. The authors are also very grateful for the thoughtful suggestions from three anonymous referees, which have improved the presentation of this paper.

Compliance with ethical standards

Conflicts of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Azimi, S., Gratie, C., Ivanov, S., Manzoni, L., Petre, I., & Porreca, A. E. (2016). Complexity of model checking for reaction systems. *Theoretical Computer Science*, *623*, 103–113. https://doi.org/10.1016/j.tcs.2015.11.040.
2. Barbuti, R., Gori, R., Levi, F., & Milazzo, P. (2016). Investigating dynamic causalities in reaction systems. *Theoretical Computer Science*, *623*, 114–145. https://doi.org/10.1016/j.tcs.2015.11.041.
3. Brayton, R. K., Hachtel, G. D., McMullen, C. T., & Sangiovanni-Vincentelli, A. L. (1984). *Logic minimization algorithms for VLSI synthesis*. Amsterdam: Kluwer Academic Publishers.
4. Brijder, R., Ehrenfeucht, A., Main, M., & Rozenberg, G. (2011). A tour of reaction systems. *International Journal of Foundations of Computer Science*, *22*, 1499–1517. https://doi.org/10.1142/S0129054111008842.
5. Corolli, L., Maja, C., Marini, F., Besozzi, D., & Mauri, G. (2012). An excursion in reaction systems: From computer science to biology. *Theoretical Computer Science*, *454*, 95–108. https://doi.org/10.1016/j.tcs.2012.04.003.
6. Ehrenfeucht, A., & Rozenberg, G. (2007). Reaction systems. *Fundamenta Informaticae*, *75*, 263–280.
7. Ehrenfeucht, A., Kleijn, J., Koutny, M., & Rozenberg, G. (2017). Evolving reaction systems. *Theoretical Computer Science*, *682*, 79–99. https://doi.org/10.1016/j.tcs.2016.12.031.
8. Ehrenfeucht, A., Kleijn, J., Koutny, M., & Rozenberg, G. (2012). Minimal reaction systems. In: C. Priami, I. Petre, E. de Vink (eds) Transactions on Computational Systems Biology XIV. Lecture Notes in Computer Science, 7625, 102–122 https://doi.org/10.1007/978-3-642-35524-0_5
9. Genova, D., Hoogeboom, H. J., & Jonoska, N. (2017). A graph isomorphism condition and equivalence of reaction systems. *Theoretical Computer Science*, *701*, 109–119. https://doi.org/10.1016/j.tcs.2017.05.019.
10. Gurunath, B., & Biswas, N.N. (1989) An algorithm for multiple output minimization. IEEE Trans. on CAD of Integrated Circuits and Systems 8, 1007–1013. https://doi.org/10.1109/43.35553
11. Karnaugh, M. (1953). The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, *72*, 593–599. https://doi.org/10.1109/TCE.1953.6371932.
12. Kleijn, J., Koutny, M., & Mikulski, Ł. (2020). Reaction systems and enabling equivalence. *Fundamenta Informaticae*, *171*, 261–277. https://doi.org/10.3233/FI-2020-1882.
13. McCluskey, E. J, Jr. (1956). Minimization of Boolean functions. *Bell System Technical Journal*, *35*, 1417–1444. https://doi.org/10.1002/j.1538-7305.1956.tb03835.
14. Meyer, A.R., & Stockmeyer, L.J. (1972). The equivalence problem for regular expressions with squaring requires exponential space. Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, 125–129. https://doi.org/10.1109/SWAT.1972.29
15. Rudell, R.L., & Sangiovanni-Vincentelli, A.L. (1987). Multiple-valued minimization for PLA optimization. IEEE Trans. on CAD of Integrated Circuits and Systems 6, 727–750. https://doi.org/10.1109/TCAD.1987.1270318
16. Rudell, R., Sangiovanni-Vincentelli, A. (2003). Exact minimization of multiple-valued functions for PLA optimization. In: Kuehlmann A. (eds) The Best of ICCAD. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-0292-0_16
17. Salomaa, A. (2013). Minimal and almost minimal reaction systems. *Natural Computing*, *12*, 369–376. https://doi.org/10.1007/s11047-013-9372-y.
18. Salomaa, A. (2012). On state sequences defined by reaction systems. Kozen Festschrift, Lecture Notes in Computer Science 7230, 271–282. https://doi.org/10.1007/978-3-642-29485-3_17.
19. Umans, C. (2001). The minimum equivalent DNF problem and shortest implicants. *Journal of Computer and System Sciences*, *63*, 597–611. https://doi.org/10.1006/jcss.2001.1775.