Fundamenta Informaticae 75 (2007) 263–280
IOS Press

Reaction Systems

A. Ehrenfeucht
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309, USA

G. Rozenberg[*][†]
Leiden Institute of Advanced Computer Science
Leiden University, Niels Bohrweg 1
2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Abstract. Interactions between biochemical reactions lie at the heart of functioning of a living cell. In order to formalize these interactions we introduce reaction systems. We motivate them by explicitely stating a number of assumptions/axioms that (we believe) hold for a great number of biochemical reactions – we point out that these assumptions are very different from the ones underlying traditional models of computation. The paper provides the basic definitions, illustrates them by biology and computer science oriented examples, relates reaction systems to some traditional models of computation, and proves some basic properties of reaction systems.

Introduction

The functioning of a living cell consists of a huge number of individual reactions that interact with each other. These reactions are regulated, and the two main regulation mechanisms are facilitation/acceleration and inhibition/retardation. The interaction between individual biochemical reactions takes place through their influence on each other, and this influence happens through the mechanisms of facilitation and inhibition.

[*]Address for correspondence: Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
[†]Also works: and Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309, USA

The aim of this paper is to present a formal framework for the investigation of biochemical reactions. We introduce reaction systems whose goal is to formalize the *interactions between biochemical reactions*. To this goal a single reaction is defined in a natural way which includes the above mentioned facilitating and inhibiting components. Then the interaction between individual reactions does not have to be formalised – it is there automatically ("for free") through the definition of individual reactions.

More specifically a *reaction* is formalized as a triplet $a = (R_a, I_a, P_a)$, where R_a is the set of *reactants*, I_a is the set of *inhibitors*, and P_a is the set of *products*. The intuition behind this formal notion of a reaction corresponds in a straightforward way to the functioning of a biochemical reaction: reaction a converts/transforms the set of reactants R_a into the product set P_a providing that it is not inhibited by (one or more) inhibitors from I_a. Therefore, more formally, the result of applying reaction a to a set T, denoted by $res_a(T)$, is conditional: if T separates R_a from I_a (i.e., if R_a is included in T and I_a is disjoint with T), then a is enabled on (applicable to) T, otherwise a is not enabled on (not applicable to) T. If a is enabled on T, then a transforms the set of reactants into the product set, and so $res_a(T) = P_a$; otherwise $res_a(T)$ is the empty set.

Beginning with this basic notion of transformation by a reaction we extend it to sets of reactions, and then investigate the interactions between sets of reactions through the investigation of suitably defined interaction processes.

In developing the theory of reaction systems we adhere to a number of assumptions (axioms) that hold for a great number of biochemical reactions. First of all, in our approach *reactions are primary* while structures are secondary: reactions *create* states rather than *transform* states as is the case in traditional models in theoretical computer science. Another way to express this is to say that reactions create the environment rather than they work in an environment. We assume the "*threshold supply*" of elements (molecules): either an element is present, and then there is "enough" of it, or an element is not present. Thus we do not have counting here (we work with sets rather than multisets), and therefore we present here a qualitative rather than quantitative analysis of interactions between reactions. Another important assumption we make is that there is *no permanency of elements*: if "nothing" happens to an element (it is not a reactant for any active reaction) then it ceases to exist. The only way to keep an element present is to sustain it by a suitable reaction – sustaining an element requires an effort, it is not for free ("life/existence must be sustained!").

As we argue in the paper the axioms/assumptions underlying the functioning of biochemical reactions, and hence the underlying assumptions of our model, are very different (often ortogonal to) the underlying axioms of the vast majority models in theoretical computer science.

This paper introduces reaction systems to the audience of (theoretical) computer scientists. It provides the basic definitions, illustrates them by providing biology (genetic regulatory networks) as well as computer science (counters) oriented examples, relates this model to a couple of traditional models of computation (elementary net systems and boolean functions), and proves some basic properties of reaction systems. The paper is an extension of [4] into a full paper.

The paper is organised as follows. The basic notions of a reaction, a reaction system, and of an interactive process are introduced and discussed in Section 1. Then in Section 2 we consider a generic example of a genetic regulatory network and demonstrate how to model it by reaction systems. This example allows us to illustrate the *bottom-up modularity principle* for reaction systems: one combines reaction systems using the operation of set-theoretic union. Reversely, one can decompose reaction systems into “smaller” components through set-theoretic union. In Section 3 we illustrate the basic notions of reaction systems by showing how they can be used to construct counters. In Section 4 we discuss elementary net systems. Although we point out (already in Section 1) the very basic differences (underlying assumptions) between reaction systems and elementary net systems, we show here how to simulate elementary net systems by reaction systems. This gives us an opportunity to demonstrate a number of “programming tricks” for reaction systems. In Section 5 we describe a close relationship between reaction systems and boolean vector functions. Then in Section 6 we consider the functional equivalence of reactions – the most basic equivalence notion for single reactions. This notion is carried over to sets of reactions in Section 7. We show here the basic difference in considering single reactions versus considering sets of reactions. While deciding functional equivalence for single reactions is easy (trivial), deciding it for sets of reactions (and hence for reaction systems) is Co-NP-complete.

Preliminaries

Throughout the paper we use standard set-theoretical notation and terminology.
$\emptyset$ denotes the emtpy set. For sets A and B, we use $A \cup B$, $A \cap B$, and $A - B$ to denote their union, intersection and difference respectively. We write $A \subseteq B$ if A is included in B, and $A \nsubseteq B$ if A is not included in B. For a family $\mathcal{F} = \{A_i\}_{i \in I}$ we use $\bigcup_{i \in I} A_i$ to denote the union of all sets in $\mathcal{F}$. Also, $|A|$ denotes the cardinality of set A.
We use $\wedge$ to denote conjunction in boolean formulas.
No specific preliminary knowledge is required for reading this paper, except for Section 4 where some basic knowledge of elementary net systems is needed, and Section 5 and 7 where some basic knowledge of boolean functions is needed.

1. Reactions, Reaction Systems, and Interactive Processes

The basic mechanism of biochemical reactions is based on facilitating and inhibiting: a reaction, e.g., $x + y \rightarrow q$, can take place if all the reactants (x and y) are present and none of the inhibitors of the reaction is present. When a reaction takes place it produces the product (q in our example). Therefore a reaction transforms its set of reactants (here $\{x, y\}$) into its set of products (here $\{q\}$) providing that it is not inhibited. This leads to the following formalization of the notion of a reaction and the associated dynamics (transformation).

Definition 1. A *reaction* is a 3-tuple $a = (R, I, P)$ of finite sets. If S is a set such that $R, I, P \subseteq S$ then we say that a is a *reaction in* S.

The set R, also denoted by R_a, is the *reactant set* of a, the set I, also denoted by I_a, is the *inhibitor set* of a, and the set P, also denoted by P_a, is the *product set* of a. By *rac*(S) we denote the set of reactions in S. For a set A of reactions, $R_A = \bigcup_{a \in A} R_a$, $I_A = \bigcup_{a \in A} I_a$, and $P_A = \bigcup_{a \in A} P_a$.

The reaction $(\emptyset, \emptyset, \emptyset)$ is called the *empty reaction* and denoted by Φ.

Example 1. Let a be the reaction such that $R_a = \{c, x_1, x_2\}$, $I_a = \{y_1, y_2\}$, and $P_a = \{c, z\}$. A possible interpretation is that c is a catalyzer for this reaction, x_1, x_2 are compounds that enter into this reaction, y_1, y_2 are compounds inhibiting the functioning of c, and z is the compound that is the product of this reaction.

Definition 2. (1) For a reaction a and a finite set T, the *result of* a on T, denoted *res*$_a(T)$, is defined by: $res_a(T) = P_a$ if $R_a \subseteq T$ and $I_a \cap T = \emptyset$, and $res_a(T) = \emptyset$ otherwise.
(2) For a set of reactions A and a set T, the *result* of A on T, denoted $res_A(T)$, is defined by: $res_A(T) = \bigcup\{res_a(T) \mid a \in A\}$.

If $R_a \subseteq T$ and $I_a \cap T = \emptyset$, then we say that a is *enabled by* T; otherwise we say that a is *not enabled by* T. Then for a set of reactions A we say that A is *enabled by* T if each $a \in A$ is enabled by T.

We are ready now to define the main notion of this paper.

Definition 3. A *reaction system*, abbreviated rs, is an ordered pair $\mathcal{A} = (S, A)$ such that S is a finite set, and $A \subseteq rac(S)$.

The set S is called the *background (set) of* $\mathcal{A}$.

The main transformation defined by a reaction system is defined as follows.

Definition 4. For a reaction system $\mathcal{A} = (S, A)$ and a set $T \subseteq S$, the *result of* $\mathcal{A}$ on T, denoted $res_{\mathcal{A}}(T)$, is defined by $res_{\mathcal{A}}(T) = res_A(T)$. The set $\{a \in A \mid a$ is enabled by $T\}$ is called the T*-activity of* $\mathcal{A}$ (or *activity of* $\mathcal{A}$ *on* T), denoted by $en_{\mathcal{A}}(T)$.

We may drop the subscript $\mathcal{A}$ in our notations whenever $\mathcal{A}$ is understood from the context of considerations.

Thus a reaction a is enabled on a set T if T separates R_a from $I_a : R_a \subseteq T$ while $I_a \cap T = \emptyset$. Similarly a set of reactions A is enabled on T if T separates R_A from I_A. There are two important aspects of the definition of enabling of A on T and the definition of $res_A(T)$ that need to be pointed out here.
(1) If $x \in T$, A is enabled on T, and several reactions from A contain x in their reactant sets, then all of these reactions will “use” x to produce their product – there is no conflict of resources here! This reflects our *threshold assumption*: either a resource (here x) is present and then it is present in a “sufficient amount”, or it is not present. Hence there is no counting in reaction systems: one deals here with sets rather than multisets. Thus two reactions a and b with $R_a \cap R_b \neq \emptyset$ are not in conflict – they are in conflict if either $R_a \cap I_b \neq \emptyset$ or $R_b \cap I_a \neq \emptyset$, because then they can never be both present in a set of reactions that is enabled on some T.
This is a major difference with the existing models of concurrent systems, like, e.g., Petri nets (in this case elementary net systems is a good class of Petri nets to be compared with reactions systems, as they also have finite “background set” – the set of conditions).
(2) For A enabled on T, $res_A(T) = P_A$. This is independent of the set $T - R_A$; this set gets discarded, it vanishes (while R_A is transformed into P_A). This reflects our *assumption of no permanency*: an element of T vanishes unless it is produced/sustained by A (it belongs to P_A).

Also this is a major difference with existing models of concurrent systems. Thus, e.g., if in an elementary net system a global state M (marking) is transformed by a set E of events enabled on M, then elements of M that are not inputs to events in E just "carry through" (they are contained in the resulting marking). This permanency ("if not transformed then remains as it is") does not exist in reaction systems.

Definition 5. Let $\mathcal{A} = (S, A)$ be a rs. An *interactive process* π in $\mathcal{A}$ is a pair of finite sequences $\pi = (\gamma, \delta)$ such that, for some $n \geq 1$, $\gamma = C_0, C_1, ..., C_n$, $\delta = D_1, ..., D_n$ where $C_0, ..., C_n, D_1, ...D_n \subseteq S$, $D_1 = res_{\mathcal{A}}(C_0)$ and, for each $1 \leq i \leq n$, $D_i = res_{\mathcal{A}}(C_{i-1} \cup D_{i-1})$.

We will often use the "arrow notation" $\pi : C_0 \rightarrow (D_1, C_1) \rightarrow ... \rightarrow (D_n, C_n)$ to represent an interactive process π as above.

The sequence $C_0, ..., C_n$ is the *interaction sequence of* π, and the sequence $D_1, ..., D_{n+1}$ is the *result sequence of* π. For each $1 \leq i \leq n$, we define $W_i = C_i \cup D_i$, and $W_0 = C_0$. The sequence $W_0, W_1, W_2, ..., W_n$ is called the sequence of *states of* π; while W_0 is called the *initial state of* π. For each $0 \leq j \leq n$, the set C_j is called the *context of* W_j. Thus, for all $1 \leq i \leq n$, the i^{th} result of π (i.e., D_i) is obtained from the state $(i-1)$ of π by applying $res_{\mathcal{A}}$. The sequence $E_0, E_1, ..., E_{n-1}$ of subsets of A such that $E_i = en_{\mathcal{A}}(W_i)$, for all $1 \leq i \leq n-1$, is called the *activity sequence of* π.

Since there is no permanency in reaction systems, each element in $D_{i+1} = res_{\mathcal{A}}(W_i)$ is produced by E_i (and hence it is "new", i.e., it is not a "remnant" of W_i that was not a subject of the transformation by E_i). Also each element of the context C_{i+1} is new. In this sense states are *created* by (sets of) reactions rather than states are *transformed*/changed by (sets of) reactions. One can say that in reaction systems, reactions do not work in an environment but rather they create an environment. This is also a major difference with traditional models of computation.

Thus in this approach the notion of a reaction (process) is primary, while the notion of a structure is derived: reactions (processes) create structures (environment). This is well reflected in the definition of a reaction system which is essentially a set of reactions.

Although biochemical processes happen in space and time, we do not believe that there is one adequate interpretation to explain them. Therefore we do not propose one fixed interpretation of interactive processes and try to keep this formal framework independent of physical interpretations, and so amenable for various interpretations. An interesting interpretation is that the result sequence represents our knowledge about a modelled system while the state sequence represents the observations of this system. From a given observation W_i our model of the system $(\mathcal{A})$ yields D_{i+1} as the prediction of the "next" observation. However, the next observation is W_{i+1}, and if $W_{i+1} - D_{i+1} \neq \emptyset$ then we miss some knowledge about the system. Then C_{i+1} represents the missing knowledge and $W_{i+1} - D_{i+1}$ represents the "minimal" missing knowledge; here $C_{i+1} \cap D_{i+1}$ represents elements which may have different explanations (different sources of existence).
Another common interpretation (quite consistent with the above "knowledge" interpretation) is that the system we model is a part of a bigger system and so the interaction sequence $C_0, ..., C_n$ represents the interaction with the "rest" of the whole system.

The notion of an interactive process suggests a natural way to associate a transition system with a reaction system.

Definition 6. Let $\mathcal{A} = (S, A)$ be a rs. The *transition system of* $\mathcal{A}$, denoted by $tr(\mathcal{A})$, is the ordered pair (Q, τ) where the set of *states* $Q = 2^S$, and the *transition relation* $\tau \subseteq Q \times Q$ is defined by: for $X, Y \in Q$, $(X, Y) \in \tau$ if and only if $res_{\mathcal{A}}(X) \subseteq Y$.

Thus $tr(\mathcal{A})$ is a finite automaton (without the initial state) with the set of states equal to the set of all subsets of the background set S. An instructive intuition is to imagine two kinds of "component edges": a *result edge* is an ordered pair (X, Y) such that $res_{\mathcal{A}}(X) = Y$, and an *inclusion edge* is an ordered pair (X, Y) such that $X \subseteq Y$. Then a single transition in $tr(\mathcal{A})$ is a 2-tact step consisting of a result edge followed by an inclusion edge. If the result edge is (X, Y) and the inclusion edge is (Y, Z) then the resulting edge is labeled by $Z - Y$. In this way $tr(\mathcal{A})$ is a *subset transition system* where the set of states as well as the letters of the alphabet are subsets of one (background) set.
In the obvious way the interaction sequences in $\mathcal{A}$ are in one to one correspondence with the paths in $tr(\mathcal{A})$.

2. Genetic Regulatory Networks

In this section we consider an example which illustrates how to model a simple generic regulatory network regulating gene expression. This example will allow us to reflect on how to construct more complex reaction systems from component systems.

Example 2. Consider a simple generic example of a genetic regulatory system G from [7] given (as is mostly the case) in a graphical form depicted in Figure 1.

This network involves three genes a, b, c and four proteins A, B, C, D. Proteins B, C are repressor proteins for gene a – if any of them binds to a regulatory site of gene a (each of them binds to a different regulatory site) then RNA polymerase is prevented from transcribing gene a. Protein B is expressed by gene b, and protein C is expressed by gene c. Protein A which is expressed by gene a is a repressor protein for gene c. Moreover, protein A interacts with protein D to form a protein complex that is a repressor for gene b.
We will construct four reaction systems representing gene a, gene b, gene c, and repressor complex E (formed by proteins D and A) – the interpretation of these component systems is given after they are defined. Thus
(1) $\mathcal{A}_a = (S_a, K_a)$, where $K_a = \{g_a, p_a, t_a\}$ with
$g_a = (\{a\}, I_{g_a}, \{a\})$,
$p_a = (\{a\}, \{B, C\}, \{u\})$,
$t_a = (\{a, u\}, I_{t_a}, \{A\})$,
and $S_a = \{a, u, A, B, C\} \cup I_{g_a} \cup I_{t_a}$.
For reactions g_a and t_a we did not specify the inhibitors sets I_{g_a} and I_{t_a} as this is not relevant for our considerations here.
(2) $\mathcal{A}_b = (S_b, K_b)$, where $K_b = \{g_b, p_b, t_b\}$ with
$g_b = (\{b\}, I_{g_b}, \{b\})$,
$p_b = (\{b\}, E, \{u\})$,
$t_b = (\{b, u\}, I_{t_b}, \{B\})$,
and $S_b = \{b, u, E, B\} \cup I_{g_b} \cup I_{t_b}$.

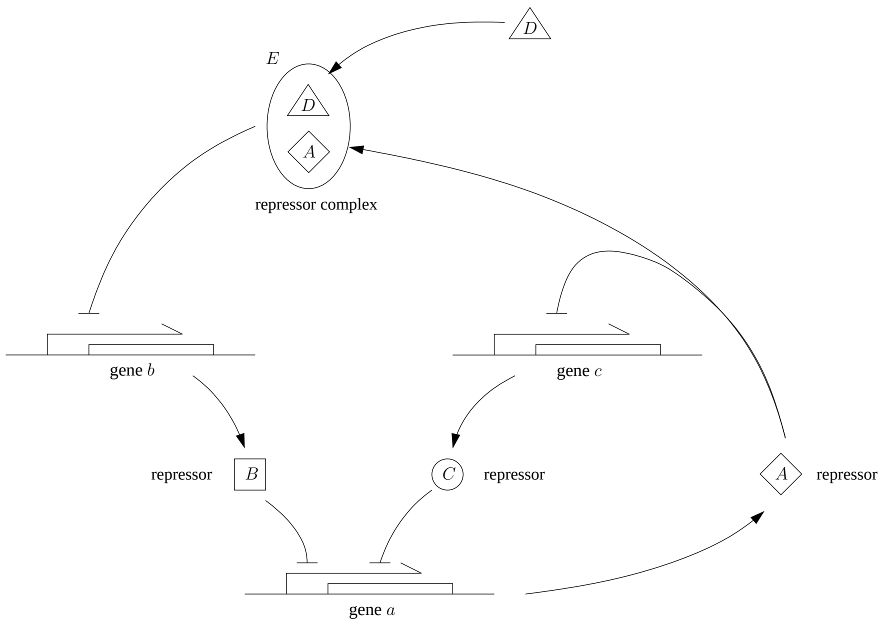

Figure 1. A simple generic regulatory network

Again we do not specify I_{g_b} and I_{t_b}.
(3) $\mathcal{A}_c = (S_c, K_c)$ where $K_c = \{g_c, p_c, c_c\}$ with
$g_c = (\{c\}, I_{g_c}, \{c\})$,
$p_c = (\{c\}, A, \{u\})$,
$t_c = (\{c, u\}, I_{t_c}, \{C\})$,
and $S_c = \{c, u, A, C\} \cup I_{g_c} \cup I_{t_c}$.
Again we do not specify I_{g_c} and I_{t_c}.
(4) $\mathcal{A}_E = (S_E, K_E)$, where $K_E = \{q_E\}$ with
$q_E = (\{D, A\}, I_{q_E}, \{E\})$.
Again we do not specify I_{q_E}.

Let's interpret $\mathcal{A}_a$.
Here reactions g_a, p_a and t_a represent the gene a itself, its promoter, and its coding region respectively. Reaction g_a says that unless damaged, e.g., by radiation (too high levels of radiation are easily specifiable by I_{g_a}), gene a is sustained, hence available. Reaction p_a says that if gene a is available, then, unless promoter region is blocked by either B or C (which bind to it), RNA polymerase u becomes available for gene a. Then reaction t_a says that when gene a is available and polymerase u is available, then (unless some adversary conditions specified by I_{t_a} hold) protein A is expressed.
Analogous interpretation holds for $\mathcal{A}_b$ and $\mathcal{A}_c$.
Then reaction q_E from K_E says that when proteins A and D are available they form protein complex E. Now we form the reaction system $\mathcal{A}_G$ modelling the regulatory system G by the union of component systems $\mathcal{A}_a, \mathcal{A}_b, \mathcal{A}_c$, and $\mathcal{A}_E$ as follows: $\mathcal{A}_G = (S_G, K_G)$, where $S_G = S_a \cup S_b \cup S_c \cup S_E$, and $K_G = K_a \cup K_b \cup K_c \cup K_E$.

Although this is not the subject of this paper, in actual modelling according to the above explanation, the modelling entity would be an ordered pair $(\mathcal{A}_G, \mathcal{F}_G)$, where $\mathcal{F}_G = \{\mathcal{A}_a, \mathcal{A}_b, \mathcal{A}_c, \mathcal{A}_E\}$. Thus $\mathcal{F}_G$ is a family of subsystems of $\mathcal{A}_G$ with each subsystem representing the "physical object of interest" in actual modelling. In this way $\mathcal{A}_G$ provides the description of G in the formalism of reaction systems, while $\mathcal{F}_G$ provides the description of its structure. The most important aspect, viz., the functionality (behaviour) of G is given by the formal model of reaction systems.

This example illustrates how to combine reaction systems to form a composed reaction system. We propose to do it in such a way that given component reaction systems one does not have to provide a description of an interface between these components in order to form a composed system. Rather, this interface/interaction is provided "automatically". We refer to this way of constructing composed systems from their components as *bottom-up modularity*, and our mathematical solution for it is the use of set-theoretic union for combining reaction systems. This can be done because reaction systems are build up on sets: a rs is an ordered pair of sets (and each single reaction is an ordered triplet of sets). This allows one to combine reaction systems (as well as single reactions) by using set-theoretic operations.

Definition 7. For reaction systems $\mathcal{A}_1 = (S_1, A_1), \mathcal{A}_2 = (S_2, A_2)$ their *union*, denoted by $\mathcal{A}_1 \cup \mathcal{A}_2$, is the rs $(S_1 \cup S_2, A_1 \cup A_2)$.

In the same way as we can compose reaction systems by union, we can decompose a reaction system $\mathcal{A} = (S, A)$ by decomposing its sets of reactions A into, not necessarily disjoint, subsets (which together by union yield A). Then by choosing for each subset A_i a suitable, not necessarily strict, subset S_i of S one obtains a decomposition through set-theoretic union of $\mathcal{A}$ into component reaction systems $\mathcal{A}_i = (S_i, A_i)$. Obviously, an atomic decomposition is a decomposition where each A_i is a singleton. To summarize: *the basic mechanism for composing and decomposing reaction systems is set-theoretic union.*

3. Counting

In this section we will illustrate the basic notions of reaction systems by showing how they can be used to construct counters.

Example 3. (counter *mod* 2^n)
Let $S_n = \{e_0, e_1, ..., e_n\}$.
We will consider numbers in binary notation of length n with e_1 standing for the first position (2^0) and e_n standing for the last position (2^{n-1}).
The binary number s of length n will be represented by a subset of S consisting of these e's which represent positions of s equal to 1. Thus, e.g., for $n = 5$, $s = 01001$ (hence 9 in decimal notation) is represented by the subset $\{e_1, e_4\}$ of S.
On the other hand, e_0 represents the action of adding 1 in binary (hence the successor function). Thus, if e_0 belongs to a subset of S then 1 will be added, while no addition will take place if e_0 does not belong to a subset. Hence, e.g., the subset $\{e_0, e_1, e_4\}$ of S represents the binary number 1001 and the instruction to perform a binary successor function on it.

For all i, j such that $0 \leq j < i \leq n$, the reaction a_{ij} is defined by $a_{ij} = (\{e_i\}, \{e_j\}, \{e_i\})$. Thus a_{ij} maintains the presence of e_i in a "current subset" of S (a current state) as long as e_j is not present. This reflects the fact that if e_0 is not present then the successor operation is not performed, while if e_0 is present (hence the successor operation is performed) then, if a binary number s has 1 on the ith position and 0 on the jth position, where $j < i$, then the resulting binary number $s + 1$ will still have 1 on the ith position.
For each $1 \leq i \leq n$, the reaction b_i is defined by $b_i = (\{e_0, ..., e_{i-1}\}, \{e_i\}, \{e_i\})$. Thus b_i produces e_i if e_i is not present while all of $e_1, ..., e_{i-1}$ are present and also e_0 (the successor instruction) is present. Hence b_i represents the carry over of 1 to the i'th position (in binary) when 1 is added to a number that has 0 on position i and 1 on each position smaller than i.

Let now $\mathcal{B}_n = (S_n, B_n)$ be the rs such that $B_n = \{a_{ij} \mid 0 \leq j < i \leq n\} \cup \{b_i \mid 1 \leq i \leq n\}$.
Let's set $n = 4$, and consider the interactive process
$\pi : C_0 \to (D_1, C_1) \to (D_2, C_2) \to (D_3, C_3) \to (D_4, C_4)$, where
$C_0 = \{e_1, e_4\}, C_1 = \emptyset, C_2 = \{e_0\}, C_3 = \{e_0\}, C_4 = \emptyset$, $D_1 = \{e_1, e_4\}, D_2 = \{e_1, e_4\}, D_3 = \{e_2, e_4\}$, and $D_4 = \{e_1, e_2, e_4\}$.
It is easily seen that indeed π is an interactive process of $\mathcal{B}_4$, because:
$res_{\mathcal{B}_4}(C_0) = \{e_1, e_4\} = D_1$ and $W_1 = D_1$,
$res_{\mathcal{B}_4}(W_1) = \{e_1, e_4\} = D_2$ and $W_2 = \{e_1, e_4, e_0\}$,
$res_{\mathcal{B}_4}(W_2) = \{e_2, e_4\} = D_3$ and $W_3 = \{e_2, e_4, e_0\}$,
$res_{\mathcal{B}_4}(W_3) = \{e_1, e_2, e_4\} = D_4$ and $W_4 = \{e_1, e_2, e_4\}$.
As a matter of fact,
$en_{\mathcal{B}_4}(C_0) = \{a_{43}, a_{42}, a_{40}, a_{10}\}$,
$en_{\mathcal{B}_4}(W_1) = \{a_{43}, a_{42}, a_{40}, a_{10}\}$,
$en_{\mathcal{B}_4}(W_2) = \{a_{43}, a_{42}, b_2\}$, and
$en_{\mathcal{B}_4}(W_3) = \{a_{43}, a_{41}, a_{21}, b_1\}$.

This interactive process can be interpreted as the following sequence of situations (states):
– the binary number 1001 is given and there is no demand for performing the successor operation (this is represented by C_0).
– the binary number has not changed (it is still 1001 represented by $D_1 = C_0$), and there is still no demand for performing the successor operation (this is represented by $C_1 = \emptyset$),
– the binary number has not changed (it is still 1001 represented by $D_2 = D_1$), but there is a demand for performing the successor operation (represented by $C_2 = \{e_0\}$),
– the binary number has changed to 1010 (represented by D_3), and there is a demand for performing the successor operation ($C_3 = \{e_0\}$),
– the binary number has changed to 1011 (represented by D_4).

Example 4. (Embedded counter *mod* 2^n)
We will modify now the counter from the previous example to obtain a counter embedded in a larger system which gets activated by receiving a trigger signal from the system to count until either again a trigger signal is received which switches off the counter or, by counting, $2^n - 1$ is reached.
To this aim we extend S_n to $S'_n = S_n \cup \{t\}$, where t represents the "trigger" signal for the counter to start counting up to $2^n - 1$.

Then we add to the rs $\mathcal{B}_n$ from the previous example the following three reactions: $q = (\{t\}, \{e_0\}, \{e_0\})$, $r = (\{e_0\}, \{t\}, \{e_0\})$, and $s = (\{e_0, e_1, ..., e_n\}, \{t\}, \{t\})$.
Reaction q (the *trigger*) switches on the counting by introducing e_0 when the trigger signal t is introduced (if it was not received already), then reaction r (the *continuator*) maintains the presence of e_0 as long as t is not received again, and reaction s (the *off switch*) stops the counting (by introducing t which eliminates e_0) when $2^n - 1$ is reached.
Let $\mathcal{B}'_n = (S'_n, B'_n)$, where $B'_n = B_n \cup \{q, r, s\}$ with B_n defined in the previous example.

Consider the interactive process π_1 in $\mathcal{B}'_4$, where
$\pi_1 : C_0 \rightarrow (D_1, C_1) \rightarrow (D_2, C_2) \rightarrow (D_3, C_3) \rightarrow (D_4, C_4) \rightarrow (D_5, C_5) \rightarrow (D_6, C_6)$, with
$C_0 = \{e_1, e_4\}, C_1 = \{t\}, C_2 = C_3 = \emptyset$, $C_4 = \{t\}, C_5 = C_6 = \emptyset$.
Then
$en(C_0) = \{a_{10}, a_{40}, a_{42}, a_{43}\}$, $D_1 = \{e_1, e_4\}$ and $W_1 = \{t, e_1, e_4\}$,
$en(W_1) = \{q, a_{10}, a_{40}, a_{42}, a_{43}\}$, $D_2 = \{e_1, e_4, e_0\}$ and $W_2 = D_2$,
$en(W_2) = \{r, a_{42}, a_{43}, b_2\}$, $D_3 = \{e_2, e_4, e_0\}$ and $W_3 = D_3$,
$en(W_3) = \{r, a_{21}, a_{41}, a_{43}, b_1\}$, $D_4 = \{e_1, e_2, e_4, e_0\}$ and $W_4 = \{e_1, e_2, e_4, e_0, t\}$,
$en(W_4) = \{a_{43}, b_3\}, D_5 = \{e_3, e_4\}$ and $W_5 = D_5$,
$en(W_5) = \{a_{30}, a_{31}, a_{32}, a_{40}, a_{41}, a_{42}\}, D_6 = \{e_3, e_4\}$ and $W_6 = D_6$.

Here the initial state represents the binary number 1001, and $t \in C_1$ represents the signal switching on the counter. The counter produces then consecutively representations of 1010, 1011 and then stops because $t \in C_4$.
The intuition behind this process is the following sequence of situations:
– the binary number 1001 is given and there is no demand (signal) from the "rest of the system" R to start counting (this is represented by C_0),
– the binary number has not changed (it is still 1001 represented by $D_1 = C_0$) but the trigger signal t sent by R arrived $(C_1 = \{t\})$,
– the binary number has not changed but the demand for performing successor function (e_0) is now present (this is represented by D_2),
– the successor function was implemented and so the current binary number is 1010, and the demand to continue counting (e_0) remains (this is represented by $D_3 = \{e_0, e_2, e_4\}$) because C_2 did not contain t,
– the successor function was implemented and so the current binary number is 1011 and the demand to continue counting (e_0) remains (this is represented by $D_4 = \{e_0, e_1, e_2, e_4\}$) because C_3 did not contain t; however the trigger signal t sent by R arrived $(C_4 = \{t\})$,
– the successor function was implemented and so the current binary number is 1100 and the demand to continue counting (e_0) disappeared (this is represented by $D_5 = \{e_4, e_3\}$) because C_4 contained t,
– the successor function was not applied and so the current binary number remains the same, viz. 1100, represented by $D_6 = D_5$.

For the contrast, consider now the interactive process π_2 in $\mathcal{B}'_n$ such that
$\pi_2 : C_0 \rightarrow (D_1, C_1) \rightarrow ... \rightarrow (D_9, C_9) \rightarrow (D_{10}, C_{10})$, where
$C_0 = \{e_1, e_4\}, C_1 = \{t\}, C_2 = C_3 = ... = C_{10} = \emptyset$, thus *none* of C_i for $i \geq 2$ contains t. Then $en(C_0), D_1, W_1, en(W_1), D_2, W_2, ..., en(W_3)$, and D_4 are the same as for π_1. Going further we get now:
$W_4 = D_4 = \{e_1, e_2, e_4, e_0\}$

$en(W_4) = \{r, a_{43}, b_3\}, D_5 = \{e_3, e_4, e_0\}$ and $W_5 = D_5$,
$en(W_5) = \{r, a_{31}, a_{32}, a_{41}, a_{42}, b_1\}$, $D_6 = \{e_1, e_3, e_4, e_0\}$ and $W_6 = D_6$,
$en(W_6) = \{r, a_{32}, a_{42}, b_2\}, D_7 = \{e_2, e_3, e_4, e_0\}$ and $W_7 = D_7$,
$en(W_7) = \{r, a_{21}, a_{31}, a_{41}, b_1\}$, $D_8 = \{e_1, e_2, e_3, e_4, e_0\}$ and $W_8 = D_7$,
$en(W_8) = \{r, s\}$, $D_9 = \{e_0, t\}$ and $W_9 = D_9$,
$en(W_9) = \emptyset$, $D_{10} = \emptyset$ and $W_{10} = D_{10}$.

Also here the initial state represents the binary number 1001, and $t \in C_1$ represents the signal switching on the counter. The counter produces then consecutively representations of 1010, 1011, 1100, ... up to 1111 (none of C_i for $i \geq 2$ contains t to stop the counting) followed by the production of $\{e_0, t\}$, and then it "stops" (no reaction is enabled anymore). One can consider $\{e_0, t\}$ as the representation of "stop" (which empties the counter: $D_{10} = \emptyset$).

4. Elementary Net Systems

In this section we will use reaction systems to simulate elementary net systems. This will allow us to demonstrate some useful "tricks" for programming reaction systems.
We assume the reader to be familiar with elementary net systems – the basic class of Petri nets (see, e.g., [5]). As usual we assume that each event has at least one input place (condition) and at least one output place (condition), and that there are no loops: no place can be an input and an output for the same event. For an event e we use $inp(e)$ to denote the set of its input places, $out(e)$ to denote the set of its output places, and $ngb(e)$ to denote its neighbourhood $(inp(e) \cup out(e))$.

Consider an event e with $inp(e) = \{p_1, ..., p_k\}$ and $out(e) = \{q_1, ..., q_l\}$.
A "naive" simulation of e by a reaction is to define reaction $sim_e^1 = (\{p_1, ..., p_k\}, \{q_1, ..., q_l\}, \{q_1, ..., q_l\})$. However there is a number of problems with this simulation.

First of all, in EN systems we have permanency, i.e., a place not involved in transformation will remain as it is (i.e., if a place p has a token and $p \notin ngb(e)$ that p will be present also in the resulting marking). To account for this we will add for each place p the place $\bar{p}$ and then the p-sustaining reaction $sus_p = (\{p\}, \{\bar{p}\}, \{p\})$. Hence, as long as $\bar{p}$ is not present in the marking to be transformed, p is sustained. Then we modify the e-simulating reaction to $sim_e^2 = (\{p_1, ..., p_k\}, \{q_1, ..., q_l\}, \{q_1, ..., q_l, \bar{p}_1, ..., \bar{p}_k\})$.

The second problem is that in EN systems for a firing event the removal of tokens from the input places and the introduction of tokens to the output places happens in one step. But sim_e^2 above will introduce $\bar{p}_1, ..., \bar{p}_k$ which only in the *next step* will inhibit the p_i-sustaining reactions sus_{p_i} for each $1 \leq i \leq k$. To resolve this issue we will have a 2-tact simulation where every other step/tact will actually simulate the EN system, and in-between steps will perform/accomplish some auxiliary simulation tasks. Thus we introduce a ("tic, toc") clock by the following two reactions (and new background elements t_1 and t_2): $tic = (\{t_1\}, \{t_2\}, \{t_2\})$ and $toc = (\{t_2\}, \{t_1\}, \{t_1\})$. Then we modify the e-simulating reaction in such a way that it operates only in "tic moments" (t_1). Thus $sim_e^3 = (\{p_1, ..., p_k, t_1\}, \{q_1, ..., q_l\}, (\{q_1, ..., q_k, \bar{p}_1, ..., \bar{p}_k\})$.

The final issue is conflicts in EN systems. Since each place may contain only one token, any two events e_1, e_2 such that $ngb(e_1) \cap ngb(e_2) \neq \emptyset$ are in conflict. To take care of conflicts we introduce names for each event: id_e will be the name of event e, all these names are new elements of the background set. Then we modify again the simulating reactions obtaining the final form of the reaction simulating event e: $sim_e = (\{p_1, ..., p_k, t_1, id_e\}, \{q_1, ..., q_l\} \cup C_e, \{q_1, ..., q_l, \bar{p}_1, ..., \bar{p}_k\})$ where $C_e = \{id_{e'} : e'$ is in conflict with $e\}$.

Thus the reaction sim_e simulating e is enabled in a set/state T if and only if T contains t_1 (hence this is the "tic" moment of simulation), T contains the name id_e of e and it does not contain the name of any event e' that is in conflict with e, T includes $inp(e)$, and T does not contain any element of $out(e)$. Hence, from the point of view of e the reaction system simulating M works as follows.
The simulating reaction sim_e is enabled at the tic moment (t_1) of simulation. This results in producing $q_1, ..., q_l$ and $\bar{p}_1, ..., \bar{p}_k$. Then in the toc moment (t_2) of simulation elements $q_1, ..., q_l$ will be propagating by their sustaining reactions, sus_{q_i} for $1 \leq i \leq l$ (unless these reactions will be inhibited by the results of other event-simulating reactions), elements $p_1, ..., p_k$ will disappear because their sustaining reactions are blocked by $\bar{p}_1, ..., \bar{p}_k$, and finally all elements $\bar{p}_1, ..., \bar{p}_k$ disappear because there are no sustaining reactions for them.
Each run ρ of M is simulated by the interactive process $\pi = (\gamma, \delta)$ where $\gamma = C_0, C_1, ..., C_n$ is such that:
– C_0 consists of t_1, the initial marking of M, and the names (identities id_e) of events from the initial set of events forming the first step of ρ,
– for each even $i, i = 2j$ for some j, C_i consists of the names of events from the j^{th} step of ρ, and
– for each odd i, $C_i = \emptyset$.

5. Boolean functions

In this section we demonstrate a close relationship between reaction systems and boolean (vector) functions – we assume the reader to be familiar with basic notions of boolean functions and boolean formulas, see, e.g., [2].

We need some additional terminology, notation and notational convention.
For sets X and Y such that $X \subseteq Y$ the characteristic function of X w.r.t. Y is the boolean function $ch_{X,Y} : Y \rightarrow \{0, 1\}$ such that, for each $y \in Y, ch_{X,Y}(y) = 1$ if and only if $y \in X$. If Y is an ordered set, $Y = (y_1, ..., y_n)$, then we will identify in the obvious way each vector of $\{0, 1\}$ of dimension n with a boolean function on Y and thus with a characteristic function $ch_{X,Y}$ for some $X \subseteq Y$.

First we translate reaction systems into boolean vector functions.
Let $\mathcal{A} = (S, A)$ be a rs.
Let for each $s \in S, u_s$ be a boolean variable, and let $V = \{u_s : s \in S\}$. We will consider S, and consequently V, to be ordered sets: $S = (s_1, ..., s_n)$, where $|S| = n$ for some $n \geq 0$, and $V = \{u_{s_1}, ..., u_{s_n}\}$. The case $n = 0$ is trivial; hence we can assume that $n \geq 1$.
Each boolean function $\varphi : S \rightarrow \{0, 1\}$ is translated into the boolean function $\varphi' : V \rightarrow \{0, 1\}$ as follows: for each $s \in S$, $\varphi(s) = 1$ if and only if $\varphi'(u_s) = 1$.
Let $W = P_A$, thus $W \subseteq S$. Also W is an ordered set, $W = (w_1, ..., w_m)$, where $|W| = m$ for some $1 \leq m \leq n$. For each $w \in W$, we set $rp(w) = \{a \in A : w \in P_a\}$ – thus $rp(w)$ are all reactions in A which produce w.
Now with each reaction $a \in A$ we associate a boolean formula β_a as follows: $\beta_a = u_{x_1} \wedge ... \wedge u_{x_k} \wedge \overline{u}_{y_1} \wedge ... \wedge \overline{u}_{y_\ell}$ where $R_a = \{x_1, ..., x_k\}$ and $I_a = \{y_1, ..., y_\ell\}$.

Finally we define the boolean vector function $F_{\mathcal{A}} : \{0,1\}^n \rightarrow \{0,1\}^m$ as follows (recall that, since S and W are ordered, we identify boolean functions on S and W with vectors over $\{0,1\}$). For each boolean function $\varphi : S \rightarrow \{0,1\}$, $F_{\mathcal{A}}(\varphi) = \psi$, where $\psi : W \rightarrow \{0,1\}$ is the boolean function defined by: for each $w \in W$, $\psi(w) = \bigvee_{a \in rp(w)} \varphi'(\beta_a)$.

It follows from the definition of $F_{\mathcal{A}}$ that for each $T \subseteq S$ and each $U \subseteq W$, $res_{\mathcal{A}}(T) = U$ if and only if $F_{\mathcal{A}}(ch_{T,S}) = ch_{U,W}$.

Now we translate boolean vector functions into reaction systems.
Let $F : \{0,1\}^n \rightarrow \{0,1\}^m$ be the boolean vector function, with the (ordered) set of input variables $V = (u_1, ..., u_n)$, and the ordered set of output variables $W = (w_1, ..., w_m)$; we assume that $W \subseteq V$.
For each $w \in W$, let $F_w : \{0,1\}^n \rightarrow \{0,1\}$ be the projection of F onto w, i.e., for each vector $\alpha \in \{0,1\}^n$, $F_w(\alpha)$ is the w-coordinate of $F_w(\alpha)$.
Let Γ_w be a boolean formula in disjunctive normal form representing F_w. Let $cl(F)$ be the set of all (conjunctive) clauses appearing in all boolean formulas Γ_w for all $w \in W$, and then for each $\delta \in cl(F)$ let $P_\delta = \{w \in W : \delta \text{ is a clause in } \Gamma_w\}$. Thus P_δ are all output variables whose value is influenced by (a valuation of) δ.
With each clause $\delta \in cl(F)$ we associate the reaction a_δ as follows. Let $\delta = u_{i_1} \wedge ... \wedge u_{i_k} \wedge \overline{u}_{j_1} \wedge ... \wedge \overline{u}_{j_\ell}$ for some $u_{i_1}, ..., u_{i_k}, u_{j_1}, ..., u_{j_\ell} \in V$. Let $S = \{s_u : u \in V\}$ and then let a_δ be a reaction over S such that $R_{a_\delta} = \{s_{u_{i_1}}, ..., s_{u_{i_k}}\}$, $I_{a_\delta} = \{s_{u_{j_1}}, ..., s_{u_{j_\ell}}\}$, and $P_{a_\delta} = \{s_w : w \in P_\delta\}$.
Finally, we define $\mathcal{A}_F$ to be the reaction system (S, A) where $A = \{a_\delta : \delta \in cl(F)\}$.

Each subset $T \subseteq V$ translates into a subset $T' \subseteq S$ in an obvious way: if $T = \{t_1, ..., t_q\}$, then $T' = \{s_{t_1}, ..., s_{t_q}\}$. Using this translation the relationship between F and $\mathcal{A}_F$ which follows from the construction of $\mathcal{A}_F$ can be expressed in the following way. For each $T \subseteq V$ and each $U \subseteq W$, $F(ch_{T,V}) = ch_{U,W}$ if and only if $res_{\mathcal{A}_F}(T') = U'$.

6. Functional equivalence of reactions

In this section we consider the most natural way to compare reactions, viz., through their functional equivalence.

Definition 8. For $a, b \in rac(S)$ we say that a, b are *functionally equivalent in* S, denoted by $a \sim_S b$, if for each $T \subseteq S, res_a(T) = res_b(T)$.

We can simply write $a \sim b$ if S is understood from the context of considerations.

Lemma 1. Let $a \in rac(S)$. Then $a \sim \Phi$ if and only if either $P_a = \emptyset$ or $R_a \cap I_a \neq \emptyset$.

Proof:
Let $a \in rac(S)$.
Clearly if either $P_a = \emptyset$ or $R_a \cap I_a \neq \emptyset$, then for each $T \subseteq S, res_a(T) = \emptyset$, and so $a \sim \Phi$.
Assume now that both $P_a \neq \emptyset$ and $R_a \cap I_a = \emptyset$. Then $res_a(R_a) = P_a \neq \emptyset$, while $res_\Phi(R_a) = \emptyset$; therefore $a \sim \Phi$ does not hold.
Thus the lemma holds. $\square$

A reaction a which is functionally equivalent to Φ is called *trivial*; otherwise a is called *nontrivial*. Based on the above lemma, unless clear otherwise, we will assume in the sequel that nonempty reactions are nontrivial, i.e., for each nonempty reaction a, we have $R_a \cap I_a = \emptyset$ and $P_a \neq \emptyset$.

We will prove now that the extensionality principle holds for nontrivial reactions: the transformation defined by a nontrivial reaction is unique for this reaction.

Theorem 1. Let $a, b \in rac(S)$ be nontrivial. Then $a \sim b$ if and only if $R_a = R_b$, $I_a = I_b$, and $P_a = P_b$.

Proof:
Let $a, b \in rac(S)$ be nontrivial.
Assume that $a \sim b$.

Since a is nontrivial, $res_a(R_a) = P_a \neq \emptyset$, and since $a \sim b$, $res_b(R_a) = res_a(R_a) \neq \emptyset$. Therefore $R_b \subseteq R_a$.
Similarly, $R_b \subseteq R_a$, and consequently $R_a = R_b$.

As already stated above, since $a \sim b$, $res_a(R_a) = res_b(R_a)$ and so $P_a = P_b$.

Assume now that $I_a \neq I_b$ – without loss of generality assume that $I_a - I_b \neq \emptyset$; let $c \in I_a - I_b$. Then $res_a(R_a \cup \{c\}) = \emptyset$ because $c \in I_a$.
Since $R_a = R_b$, $res_b(R_a \cup \{c\}) = res_b(R_b \cup \{c\})$. But $res_b(R_b \cup \{c\}) \neq \emptyset$, because $R_b \subseteq R_b \cup \{c\}$, $c \notin I_b$ and $P_b \neq \emptyset$ (because b is nonempty). This contradicts the assumption $a \sim b$. Consequently $I_a = I_b$.
Therefore $R_a = R_b, I_a = I_b$, and $P_a = P_b$. $\Box$

We will assume in the sequel that for each nonempty reaction a, $I_a \neq \emptyset$, hence for each nonempty reaction a there are circumstances under which the reaction is inhibited.
On the other hand we do allow $R_a = \emptyset$. A reaction $a = (\emptyset, I_a, P_a)$ describes a production of P_a in the absence of elements from I_a, however in this description we are not interested in *how* P_a is produced.

We end this section by discussing the straightforward way of determining the redundancy of a reaction when it is “combined” with another one. this is expressed through the cover relation.

Definition 9. For $a, b \in rac(S)$ we say that a *covers* b, denoted by $a \geq_S b$, if for each $T \subseteq S$, $res_{\{a,b\}}(T) = res_a(T)$.

We will write simply $a \geq b$ rather than $a \geq_S b$ whenever S is understood from the context of considerations.

It is easy to see that a covers b if and only if, for each $T \subseteq S, res_b(T) \subseteq res_a(T)$. This implies directly the following two properties:
(1) Let $a, b \in rac(S)$. Then $a \geq b$ and $b \geq a$ if and only if $a \sim b$.
(2) For each $a \in rac(S), a \geq \Phi$.

We get the following characterization of the cover relation.

Theorem 2. Let $a, b \in rac(S)$ be nontrivial. Then $a \geq b$ if and only if $R_a \subseteq R_b, I_a \subseteq I_b$ and $P_b \subseteq P_a$.

Proof:
Let a, b be nontrivial reactions over S.
(i) Assume that $R_a \subseteq R_b, I_a \subseteq I_b$, and $P_b \subseteq P_a$. Let $T \subseteq S$.
If b is enabled on T, then $R_b \subseteq T$ and $I_b \cap T = \emptyset$. Since $R_a \subseteq R_b$ and $I_a \subseteq I_b$, this implies that $R_a \subseteq T$ and $I_a \cap T = \emptyset$ – therefore also a is enabled on T. Thus $res_b(T) = P_b$ and $res_a(T) = P_a$, and since $P_b \subseteq P_a$, $res_b(T) \subseteq res_a(T)$. Thus $res_{\{a,b\}}(T) \subseteq res_a(T)$. Therefore $a \geq b$.
(ii) Assume now that it is not true that $R_a \subseteq R_b, I_a \subseteq I_b$ and $P_b \subseteq P_a$. Therefore one of the following three cases must hold: $R_a \nsubseteq R_b$, $I_a \nsubseteq I_b$, or $P_b \nsubseteq P_a$. We will consider each of these cases separately.
Case 1: $R_a \nsubseteq R_b$.
Since b is enabled on $R_b, res_b(R_b) = P_b$ and since b is nontrivial, $P_b \neq \emptyset$. Since $R_a \nsubseteq R_b$, a is not enabled on R_b, and therefore $res_a(R_b) = \emptyset$. Thus $res_{\{a,b\}}(R_b) = res_a(R_b) \cup res_b(R_b) = P_b \neq \emptyset$, while $res_a(R_b) = \emptyset$. Therefore it is not true that $a \geq b$.
Case 2: $I_a \nsubseteq I_b$.
Hence $I_a - I_b \neq \emptyset$. Let then $c \in I_a - I_b$, and let $T = R_b \cup \{c\}$.
Since $R_b \subseteq T$ and $I_b \cap T = \emptyset$, b is enabled on T and so $res_b(T) = P_b$. Thus, since b is nontrivial, $res_b(T) \neq \emptyset$.
But $c \in I_a$ and so a is not enabled on T; therefore $res_a(T) = \emptyset$. Consequently $res_{\{a,b\}}(T) = res_b(T) \neq res_a(T)$. Therefore it is not true that $a \geq b$.
Case 3: $P_b \nsubseteq P_a$.
Hence $P_b - P_a \neq \emptyset$. Let then $c \in P_b - P_a$.
Since b is enabled on R_b, $res_b(R_b) = P_b$. On the other hand (whether or not a is enabled on R_b), $c \notin res_a(R_b) \neq res_a(R_b)$. therefore it is not true that $a \geq b$.

Since in all three cases we concluded that it is not true that $a \geq b$, we conclude that, if it is not true that $R_a \subseteq R_b$, $I_a \subseteq I_b$, and $P_b \subseteq P_a$, then it is not true that $a \geq b$.
Now the lemma follows from (i) and (ii). □

7. Functional equivalence of sets of reactions

In this section we will extend the notion of functional equivalence to sets of reactions and hence to reaction systems.

Definition 10. (1) For $A, B \subseteq rac(S)$ we say that A, B are *functionally equivalent in* S, denoted by $A \sim_S B$, if for each $T \subseteq S$, $res_A(T) = res_B(T)$.
(2) For reaction systems $\mathcal{A} = (S, A), \mathcal{B} = (S, B)$ we say that $\mathcal{A}, \mathcal{B}$ are *functionally equivalent*, denoted by $\mathcal{A} \sim \mathcal{B}$, if $A \sim_S B$.

While for single reactions extensionality holds (Theorem 5.1) and so deciding the functional equivalence is easy, the situation changes quite dramatically when we move to consider sets of reactions. We will show that the functional equivalence problem for sets of reactions, and hence for reaction systems, is Co-NP-complete by reducing it to the tautology problem for boolean formulas.

For a set of boolean variables $V = \{v_1, ..., v_n\}$, and an assignment $f : V \rightarrow \{0, 1\}$, we set $V_f = \{v \in V \mid f(v) = 1\}$. Then we represent f by giving V_f.

Let β be a boolean formula over V in disjunctive normal form. Thus β is a disjunction of clauses where each clause is a conjunction of literals, i.e. variables or their negations.
Let $\gamma = v_{i_1} \wedge v_{i_2} \wedge ... \wedge v_{i_k} \wedge \bar{v}_{j_1} \wedge \bar{v}_{j_2} \wedge ... \wedge \bar{v}_{j_\ell}$ be a clause of β – hence $v_{i_1}, ..., v_{i_k}, v_{j_1}, ..., v_{j_\ell} \in V$.
We use the notations $var(\gamma) = \{v_{i_1}, ..., v_{i_k}\}$ and $nvar(\gamma) = \{v_{j_1}, ..., v_{j_\ell}\}$.
Let $S = V \cup \{x, y, t\}$, where $\{x, y, t\} \cap V = \emptyset$.
For each clause γ as above we define the reaction a_γ by $a_\gamma = (var(\gamma) \cup \{x\}, nvar(\gamma) \cup \{y\}, \{t\})$, and then we set $A = \{a_\gamma \mid \gamma \text{ is a clause of } \beta\}$. Let $\mathcal{A}_\beta = (S, A)$.
On the other hand we set $B = \{(\{x\}, \{y\}, \{t\})\}$ and then define the rs $\mathcal{B}$ by $\mathcal{B} = (S, B)$.

Lemma 2. β is a tautology if and only if $\mathcal{A}_\beta \sim \mathcal{B}$.

Proof:
(1) Assume that β is a tautology.
Let $T \subseteq S$.
We note first that if either $x \notin T$ or $y \in T$, then $res_{\mathcal{A}}(T) = \emptyset$ and $res_{\mathcal{B}}(T) = \emptyset$.
Assume then that $x \in T$ and $y \notin T$.
Let f be the assignment of β given by $V_f = T$.
Since β is a tautology, there exists a clause γ of β such that $var(\gamma) \subseteq T$ and $nvar(\gamma) \subseteq V - T$.
Therefore for the reaction a_γ we have, $R_{a_\gamma} \subseteq T$ and $I_{a_\gamma} \cap T = \emptyset$, and so $res_{a_\gamma}(T) = \{t\}$. Consequently $res_A(T) = \{t\}$, and so $res_{\mathcal{A}}(T) = \{t\}$.
On the other hand, since $x \in T$ and $y \notin T$, $res_{\mathcal{B}}(T) = \{t\}$. Thus we conclude that (if β is a tautology) for each $T \subseteq S$, $res_{\mathcal{A}}(T) = res_{\mathcal{B}}(T)$, and consequently $\mathcal{A}_\beta \sim \mathcal{B}$.
(2) Assume that β is not a tautology.
Let f be an assignment that falsifies β, i.e., the value of β under f equals 0.
Hence for each clause γ of β, either $var(\gamma) \cap (V - V_f) \neq \emptyset$ or $nvar(\gamma) \cap V_f \neq \emptyset$. Therefore if we choose $T = V_f \cup \{x\}$, then for each reaction $a \in A$, either $R_a \subseteq T$ does not hold, or $I_a \cap T \neq \emptyset$. This implies that $res_{\mathcal{A}}(T) = \emptyset$.
On the other hand, $x \in T$ and $y \notin T$, and so $res_{\mathcal{B}}(T) = \{t\}$.
Thus we conclude that (if β is not a tautology) then there exists a $T \subseteq S$ such that $res_{\mathcal{A}}(T) \neq res_{\beta}(T)$, and consequently it is not true that $\mathcal{A}_\beta \sim \mathcal{B}$.
From (1) and (2) it follows that $\mathcal{A}_\beta \sim \mathcal{B}$ if and only if β is a tautology. Thus the lemma holds. □

We get now the main result of this section.

Theorem 3. The problem of functional equivalence of sets of reactions is Co–NP–complete.

Proof:
Since determining whether a given boolean formula is a tautology is a Co–NP–complete problem (see, e.g., [6]), the theorem follows directly from Lemma 7.1 and its proof. □

8. Discussion

In this paper we have introduced reaction systems. In particular, we have motivated the way they (and their dynamics) is defined by stating explicitely assumptions that we believe hold for a great number of biochemical reactions. We have illustrated basic notions of reaction systems by both biology and computer science oriented examples, and we related reaction systems to some traditional models of computation. As for the development of the theory of reaction systems, we have presented only a handful of basic results.

But there are many more relevant properties of reaction systems. For example, some of them are related to the topic of simulation (and hence equivalence) between reaction systems based on simulation between interaction processes. The understanding of such simulations is needed, e.g., for establishing normal forms which are useful for the analysis of behaviour of reaction systems.

Although we do not have counting in reaction system, it may be needed for expressing some important parameters of biochemical reactions (such as, e.g., concentrations of reactants, products, ...). We suggest to introduce counting on top of reaction systems rather than as a "facility" included in the definition itself. How to count? By introducing multisets through suitably chosen equivalence relations, or in some other ways?

Another notion needed for developing the theory of dynamics of processes in reaction systems is the notion of time. How to introduce time (and clocks) into reaction systems is a challenging and interesting research topic.

One of the observed properties of biological systems is formation of low entropy structures. How and why such structures are formed in reaction systems is therefore a very relevant research topic. It leads to a formulation and investigation of the notion of (biochemical) modularity.

We are currently working on the above mentioned, and other topics, and hope to be able to report soon on the obtained results in the forthcoming papers.
We are not aware of any papers that approach a theory of biochemical reactions in the way we suggest in this paper. There are however very recent papers dealing with stochastic chemical reaction networks: [2] is based on stochastic Petri nets, and [1] is based on stochastic process algebra.

Acknowledgements

The authors are indebted to Marloes van der Nat for the expert typing of the manuscript, to D.M. Prescott for discussions on biological interpretation/relevance of reacton systems, and to R. Brijder and H.J. Hoogeboom for useful comments on the first version of the manuscript.

References

[1] L. Cardelli, From process to ODEs by chemistry, draft.

[2] P. Clote and E. Kranakis, Boolean functions and computation models, Springer, 2002.

[3] M. Cook, D. Soloveichik, E. Winfree and J. Bruck, Finite stochastic chemical reaction networks are Turing universal in probability, draft.

[4] A. Ehrenfeucht and G. Rozenberg, Basic notions of reaction systems, Lecture Notes in Computer Science, v. 3340, 27–29, Springer, 2004.

[5] J. Engelfriet and G. Rozenberg, Elementary net systems, Lecture Notes in Computer Science, v. 1491, 12–121, Springer, 1998.

[6] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP–Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[7] H. de Jong, Modeling and simulation of genetic regulatory systems: a literature review, Journal of Computational Biology, 9(1), 69–105, 2002.