On the Complexity of Concurrent Multiset Rewriting

Marin Bertier, Matthieu Perrin, Cédric Tedeschi

International Journal of Foundations of Computer Science

On the Complexity of Concurrent Multiset Rewriting

Marin Bertier
IRISA, INSA de Rennes, France

Matthieu Perrin
Université de Nantes, France

Cédric Tedeschi
IRISA, Université de Rennes, France

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

In this paper, we are interested in the runtime complexity of programs based on multiset rewriting. The motivation behind this work is the study of the complexity of chemistry-inspired programming models, which recently regained *momentum* due to their adequacy to the programming of large autonomous systems. In these models, data are most of the time left unstructured in a *container*, or more formally, a multiset. The program to be applied to this multiset is specified as a set of conditioned rules rewriting the multiset. At run time, these rewrite operations are applied concurrently, until no rule can be applied anymore (the set of elements they need cannot be found in the multiset anymore).

A limitation of these models stands in their complexity: each computation step may require a complexity in $O(n^k)$ where n denotes the number of elements in the multiset, and k is the size of the subset of elements needed to trigger a given rule. By analogy with chemistry, such elements can be called *reactants*.

In this paper, we explore the possibility of improving the complexity of searching reactants through a static analysis of the rules' condition. In particular, we give a characterisation of this complexity, by analogy to the subgraph isomorphism problem. Given a rule R, we define its *rank* $\mathrm{rk}(R)$ and its *calibre* $\mathcal{C}(R)$, allowing us to exhibit an algorithm with a complexity in $O(n^{\mathrm{rk}(R)+\mathcal{C}(R)})$ for searching reactants, while showing that $\mathrm{rk}(R)+\mathcal{C}(R) \leq k$ and that $\mathrm{rk}(R)+\mathcal{C}(R) < k$ most of the time.

Keywords: Complexity; multiset rewriting; concurrency; rule-based programming

1. Introduction

With the ever-growing complexity of computing environments, building *autonomic systems*, that can "*manage themselves in accordance with high-level guidance from humans*" [22], is an issue getting more and more attention. In these systems, humans should only be required to write a set of *high-level rules* defining the system's behaviour. Then the system should be able to run indefinitely, adhering to these

high-level, technical details-free rules, whatever the actual conditions on the underlying platform are.

The implementation of an autonomic system comes with several challenges that cannot be tackled at once. A prerequisite is to distinguish the development of the low-level machinery from the definition of adequate programming abstractions allowing this high-level human guidance. The quest for such high-level abstractions led recently to the re-emergence of rule-based programming as a promising paradigm, for instance making it in turn possible to specify distributed systems in a declarative manner [15]. Also, nature appears to be a great source of inspiration, biological and chemical system being analogies worth an exploration [1, 17, 25]. Note that the autonomic computing paradigm was initially proposed by analogy with the autonomic nervous system. The chemical analogy was at the origin of the so-called *chemical programming model*, a rule-based programming model enhanced with a chemically-inspired execution model, which exhibits adequate properties to develop autonomic systems [4, 12, 20].

Metaphorically speaking, a *chemical* program is envisioned as a *chemical solution* where molecules represent data and reactions are processes manipulating these data and producing new data (new molecules). More formally speaking, these artificial chemistries [12] rely on *concurrent multiset rewriting*. Let $CP = (\mathcal{T}, M, R)$ a chemical program, where $\mathcal{T}$ is the set of possible types of molecules, M represents an input multiset of molecules, each $m \in M$ having a type $m.T \in \mathcal{T}$ and R is the rule to be applied on M. R can be represented by:

$$\textbf{replace } x_1 :: T_1, \ldots, x_k :: T_k \textbf{ by } P(x_1, \ldots, x_k) \textbf{ if } C(x_1, \ldots, x_k) \tag{1}$$

This rule is composed of three parts : (1) a *pattern* multiset $x_1 :: T_1, \ldots, x_k :: T_k$ of molecules needed to apply the rule, where $T_i \in \mathcal{T}$ and x_i is the name of the variable, (2) a multiset of molecules $P(x_1, \ldots, x_k)$ produced by the rule and (3) the reaction's condition $C(x_1, \ldots, x_k)$, which is a formula of the propositional logic, in which literals are the application of a boolean function on the variables x_1 to x_k. The previous formalisation excludes program having multiple rules. We restrict the present study to chemical programs having only one rule. Extending this work to multiple-rules chemical programs does not present major difficulties. The model assumes that there is no restriction to parallelism. Whatever the number of rules of one program is, these rules are to be applied concurrently — and in no particular order — on the global multiset. The only theoretical limitation to this concurrency is the property of *atomic capture*, which ensures that a reactant can be used in at most one reaction. Once no reactions can be applied anymore, *i.e.*, when no subset of elements satisfying any of the reaction rules' condition can be found in the global multiset, the program is said to be *inert*. In this state, the solution is stable and contains the final result of the program. let us review a basic chemical program, for

the sake of illustration[a]:

$$\textbf{replace } x :: int, y :: int \textbf{ by } x \textbf{ if } x \geq y \textbf{ in } \langle 8, 5, 3, 9, 2, 2\rangle$$

The rule consumes any two integers x, y with $x \geq y$ and produces a new integer which takes the value of x. The input multiset is $\langle 8, 5, 3, 9, 2, 2\rangle$. Initially, several reactions are possible: the rule can be applied to any couple of integers satisfying the condition: 2 and 3, 2 and 5, 8 and 9, *etc.* One scenario among the set of possible scenarios is:

$$\langle 8, 5, 3, 9, 2, 2\rangle \rightarrow^* \langle 3, 5, 9\rangle \rightarrow^* \langle 9\rangle$$

where $\rightarrow^*$ models the application of the rule several times. Merely looking at these two execution steps is not enough to infer what pairs of numbers reacted together. Recall that the only needed constraint is the *atomic capture*, which states that one molecule can react at most once. A molecule is always consumed in a reaction. As a corollary, when x, y is replaced by x, the two x variables actually being two different molecules, the first one being consumed in the reaction, the second one being created in it. In other words, no molecule can *survive* a reaction.

While this model is envisioned as a promising way to specify autonomic systems, one of the main barrier towards its actual adoption is related to its execution complexity : each computation step (*i.e.*, the application of one rewrite rule) assumes that some *reactants* satisfying the rule's condition are found in the multiset. Let us assume the number of objects in the multiset is n, and the *arity* of the rule, (*i.e.*, the number of reactants needed for its application) is k. Then, in the worst case, an exhaustive exploration of all possible combinations of k molecules among n is needed, and the complexity involved is in $O(n^k)$ (assuming $n \gg k$), which is, when k increases, a problem. One question left largely open about the model is the possibility to improve the time of reactants search.

Contribution. In this paper, we explore the possibility of improving the complexity of searching reactants through a static analysis of the reaction condition. In particular, we give a characterisation of this complexity, by analogy to the subgraph isomorphism problem. Given a rule R, we define the *rank* $\mathrm{rk}(R)$ and the *calibre* $\mathcal{C}(R)$, allowing us to exhibit an algorithm with a complexity in $O(n^{\mathrm{rk}(R)+\mathcal{C}(R)})$ for searching reactants, while showing that $\mathrm{rk}(R)+\mathcal{C}(R) \leq k$, and that $\mathrm{rk}(R)+\mathcal{C}(R) < k$ most of the time.

Organisation of the paper. The article is organised as follows. Section 2 introduces the problem and devises a model for it. In Section 3, a characterisation of the complexity, by analogy with the subgraph isomorphism problem, is given. Then, we describe the PMJA (Purification of the Minimal Juncture Assignment) algorithm

[a]The following rule is written under the HOCL [3] formalism, a language following the chemical model.

putting this result into practice. We discuss its complexity. Section 4 presents some related works. Section 5 concludes and gives some hints for future works.

2. Model

The problem to be solved is the search for elements in a multiset satisfying a rule's condition. The algorithm to be designed takes two input parameters, namely, a chemical rule R, as described by Expression 1 in Section 1, and a multiset M composed of n molecules. It returns:

- a tuple $(m_1, \ldots, m_k)$ of molecules in M, where m_i is of type T_i for all i and $C(m_1, \ldots, m_k)$ is true, if such a tuple exists in M,
- $\perp$ otherwise.

2.1. *Modeling of the rule's condition*

Notice first that the case where the condition is absent is simple to solve, since it is only necessary to compare the number of available molecules for each type to the number of molecules required. In the existing approaches (reviewed in Section 4), the condition is viewed as a black box[b], which imposes to test all possible combinations of molecules. Nevertheless, some optimisation can be done at compile time by studying the reaction's condition.

As for any propositional formula, a reaction condition can be put in disjunctive normal form[c]:

$$C(x_1, \ldots, x_n) = \bigvee_{i=1}^{L} \bigwedge_{j=1}^{l_i} f_{ij}(X_{ij})$$

where, for all i and j, $X_{i,j}$ is a subset of variables of R. Since molecules $m_1, \ldots, m_k$ verify $C_1(x_1, \ldots, x_k) \vee C_2(x_1, \ldots, x_k)$ if and only if they verify either $C_1(x_1, \ldots, x_k)$ or $C_2(x_1, \ldots, x_k)$, the various terms of the disjunction can be searched separately. We can consequently replace R by L equivalent rules $\{R_i\}$ whom condition is one of the L terms of the disjunction, *i.e.,* a conjunction of boolean functions applied to a subset of variables, like the one shown in Equation 2:

$$R_i = \textbf{replace } x_1, \ldots, x_k \textbf{ by } P(x_1, \ldots,_k) \textbf{ if } f_1(X_1) \wedge \cdots \wedge f_l(X_l). \tag{2}$$

While the number L of such formulae/rules thus generated is potentially exponential over k, it only depends on the initial rule, and not on the size of the multiset. Moreover, as the inertia is detected if and only if no reactants can be found for every rule, the searching of molecules for different rules is independent, and each rule can be processed in parallel to the others.

[b]The computation time of this black box is supposed to be finite.
[c]Note that the type of a molecule can be seen as an individual condition on this molecule.

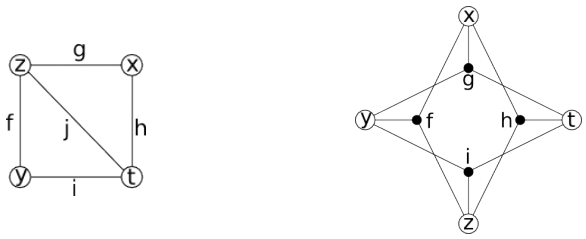

Fig. 1. The rule **replace** x, y, z, t **by** $P(x, y, z, t)$ **if** $f(z, y) \wedge g(x, z) \wedge h(x, t) \wedge i(y, t) \wedge j(z, t)$ has an arity 4 and a rank 2 and can be represented as a graph (left). The rule **replace** x, y, z, t **by** $P(x, y, z, t)$ **if** $f(x, y, z) \wedge g(x, y, t) \wedge h(x, z, t) \wedge i(y, z, t)$ has an arity 4 and a rank 3 and must be represented as a hyper-graph (right).

We now introduce some accessors to the elements of a rule R like the one given in Expression 2:

- $\mathrm{var}(R) = \{x_1, \ldots, x_k\}$ denotes the set of its variables. $|\mathrm{var}(R)|$ is called the *arity* of R.
- $\mathrm{pred}(R) = \bigcup_{i=1}^{l} \{(f_i, X_i)\}$ denotes the set of predicates to be tested on $\mathrm{var}(R)$. Each predicate $p = (f, X)$, associated with a literal in the condition, has a *function* $\mathrm{func}(p) = f$ and *arguments* $\arg(p) = X \subseteq \mathrm{var}(R)$.

Definition 1 (rank of a rule) *The* rank *of a rule R, denoted by* $\mathrm{rk}(R)$*, is the greatest arity of its predicates:* $\mathrm{rk}(R) = \max_{p \in \mathrm{pred}(R)} |\arg(p)|$.

A rule can be represented as a hyper-graph, in which the vertices are the variables and the hyper-edges are the predicates, as illustrated in Figure 1. Note that most of the problems encountered in the literature on artificial chemistries are solved by rules with a rank of 1, 2 or 3, as their predicates mostly involve comparisons of pairs of variables [12]. When the rank is 2, the representation is a simple graph (as the one on the left in Figure 1).

2.2. *Structuring of the multiset*

The previous section focused on defining the rules. We now devise a model for the multiset of molecules, and how to structure it according to the rule processed. The central definition in the following is the *axiom*. It can be seen as an instantiated predicate. In other words, it is a predicate for which an actual molecule has been found for each of its variables so as to make the predicate true. Note that, as reflected by the chosen term *axiom*, it can be seen as a minimal set of truth in regards to the rule's condition.

Definition 2 (axiom) *Let $p = (f, X)$ be a predicate. An* axiom *is a pair (p, m) where m is a function that associates a molecule to each variable of p, such that $f(m(x_1), \ldots, m(x_n))$ is true.*

We extend to axioms, the notations previously defined for predicates. Let an axiom $a = (p, m)$. Then, $\text{func}(a) = \text{func}(p)$ and $\arg(a) = \arg(p)$, $\text{pred}(a) = p$, $a[x] = m(x)$ for all $x \in \arg(p)$, $\text{mols}(a) = \{a[x] : x \in \arg(p)\}$. We finally extend these notations to sets of axioms: let A be a set of axioms. The set of molecules of A is:

$$\text{mols}(A) = \bigcup_{a \in A} \text{mols}(a)$$

Given a rule R and a set of molecules M, we can define the set of all the axioms that can be constructed.

Definition 3 (set of axioms induced) *For a solution M and a rule R, we define the* set of axioms induced *by R in M as the set of axioms composed of a predicate of R and of molecules of M:*

$$\mathcal{A}(R, M) = \{a : \text{pred}(a) \in \text{pred}(R) \wedge \forall x \in \arg(a), a[x] \in M\} \tag{3}$$

The goal is to associate each variable with a molecule such that this molecule is used in at least one axiom corresponding to each predicate of the rule. In this case, the variable is said *satisfied*.

Definition 4 (satisfaction of a variable) *Let $x \in \text{var}(R)$, A be a set of axioms and $m \in \text{mols}(A)$. The molecule m is said to* satisfy *x in A, denoted $m \models_A x$, if for every predicate of the rule pertaining x, we can find at least one axiom in A in which x is associated with m:*

$$\forall p \in \text{pred}(R), x \in \arg(p) \Rightarrow (\exists a \in A, \text{pred}(a) = p \wedge a[x] = m). \tag{4}$$

We are interested in finding sets of axioms leading to a possible reaction. A set of axioms specifies a possible reaction if there is a one-to-one relation between its variables and its molecules. Let us characterise sets of axioms so as to be able to define the subsets of axioms that can actually lead to a reaction.

- A set of axioms is *refined* if $\forall m \in \text{mols}(A), |\{x \in \text{var}(R) : m \models_A x\}| \geq 1$
- A set of axioms is *exclusive* if $\forall m \in \text{mols}(A), |\{x \in \text{var}(R) : m \models_A x\}| \leq 1$

Ensuring exclusivity can be done by adding inequality constraints in the rule so the same molecule cannot satisfy several variables[d]. From now on, we assume all sets of axioms are exclusive.

In a both refined and exclusive set of axioms, all molecules are assigned to one and only one variable. This does not mean that it specifies a possible reactions, as some variables may not be satisfied in it. Let us define sets of axioms that can lead to reactions:

- A set of axioms is *reactive* if $\forall x \in \text{var}(R), |\{m \in \text{mols}(A) : m \models_A x\}| \geq 1$

[d]Note that exclusivity is preserved by all the operations used in the following.

- A set of axioms is *purified* if $\forall x \in \text{var}(R), |\{m \in \text{mols}(A) : m \models_A x\}| \leq 1$

In other words, a *reactive* set of axioms contains at least one subset of axioms that specifies a reaction. Extracting one of them can be done by *purifying* it.

The algorithm presented later in this paper consists in taking the set of axioms induced and trying to assign molecules to variables so as to refine it, and then test the *reactivity* of such an *assignment*:

Definition 5 (assignment) *Let A be a set of axioms, $x_1, \ldots, x_p \in \text{var}(R)$ and $m_1, \ldots, m_p \in \text{mols}(A)$. An* assignment *of $m_1, \ldots, m_p$ to $x_1, \ldots, x_p$, denoted $A' = A[x_1 := m_1, \ldots, x_p := m_p]$, is the largest (in the sense of inclusion) refined subset of A verifying $\forall i \leq p, \forall m, m \models_{A'} x_i \Rightarrow m = m_i$.*

Algorithmically speaking, and as detailed in Section 3, given a set A of axioms induced, an assignment of A is obtained by removing all the molecules and axioms from A that cannot be in any refined subset of A, given the set of molecules chosen M_{chosen} for the subset $V_{assigned}$ of variables assigned. Firstly it means, given $V_{assigned}$, remove all the axioms corresponding to predicates containing variables in $V_{assigned}$ but built using molecules not in M_{chosen}. Secondly, it means remove all molecules consequently not used anymore, and the axioms in which they were, making in turn other molecules unused. This refinement is repeated until no more refinement is needed.

2.3. *NP-hardness and the subgraph isomorphism problem*

The reactants searching can be reduced to the subgraph isomorphism problem. In regard to the hyper-graph of the rules, a set of axioms can be modeled by a similar hyper-graph of molecules, where the vertices are the molecules contained in the set and each vertex corresponds to an axiom that links its arguments and is labelled by its predicate. Under this formalism, a purified reactive assignment is a sub-hyper-graph of the hyper-graph of molecules that is isomorphic to the hyper-graph of the rule, with respect to the labels of the edges.

The subgraph isomorphism problem is known to be NP-complete, as it contains the detection of a clique. This property gives clues on the intrinsic complexity of the reactants searching problem. Let $G = (V, E)$ be a graph. The chemical program with $M = V$ and $R =$ **replace** $x_1, \ldots, x_k$ **by** $P(x_1, \ldots, x_k)$ **if** $\bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (x_i, x_j) \in E$ can evolve if and only if G contains a clique. This shows that the reactants searching problem is NP-hard, depending on its arity k. It is actually NP-complete under the assumption that the evaluation of reaction conditions terminates (in a time necessarily independent of both n and k) as doing all tests between molecules non-deterministically can solve it in a polynomial time.

The rule used to show the NP-hardness of the reactants searching problem has a rank of 2, and its graph is a clique. In other words, it can be considered as a complicated rule since its reaction condition has as many literals as there are pairs of variables. We should therefore find a way to characterize the complexity of a rule.

This is what we do in the next section, which provides a study of the *calibre* of a rule, and an algorithm solving the reactants searching problem, based on it.

3. Calibre and the PMJA Algorithm

In this section, we present a more efficient algorithm for the reactants searching problem, based on a characterization of the complexity of a rule, using the notion of *calibre* of the rule. Then, we present the PMJA algorithm, which levers this characterisation to allow for a better complexity than the basic $O(n^k)$ case, most of the time. For brevity, we do not exhibit the complete proofs of properties and theorems in the following. Please refer to the research report [7] for the details.

3.1. *Calibre of a rule*

Determining the calibre of a rule relies on determining its minimal *juncture*:

Definition 6 (juncture of a rule) *Assuming the variables of a rule are completely sorted by an order $\lessdot$, we define the* juncture *of R for the order $\lessdot$, denoted $\mathcal{J}_{\lessdot}(R)$, as the set of variables which are not the smallest in several of their predicates:*

$$\mathcal{J}_{\lessdot}(R) = \{x \in \mathrm{var}(R) : |\{p \in \mathrm{pred}(R) : \exists y \lessdot x, \{x, y\} \subset \mathrm{arg}(p)\}| \geq 2\}\,. \quad (5)$$

Definition 7 (calibre of a rule) *The* calibre *of a rule is the size of its smallest juncture considering all possible orders:*

$$\mathcal{C}(R) = \min_{\lessdot} |\mathcal{J}_{\lessdot}(R)|. \quad (6)$$

A juncture $\mathcal{J}_{\lessdot}(R)$ such that $|\mathcal{J}_{\lessdot}(R)| = \mathcal{C}(R)$ is said to be minimal.

Let us illustrate the two previous definitions. As rules with a rank of 2 represent most of rules found in chemical programs in practice, we will discuss the calibre of some of these rules, by having a look at their corresponding graph:

- The calibre of a rule having a tree shape is 0. By definition, each node of a tree has a single parent except the root which is an orphan. Therefore, following the topological order, we find no node in the potential juncture.
- The calibre of a rule having a cycle shape is one. On one hand, regardless of the order chosen, the greatest element has necessarily two smaller neighbours: its predecessor and its successor in the cycle, making the calibre is at least equal to 1. On the other hand, for an order that follows the cycle, all the other elements have 0 or 1 parent, so the calibre is at most 1.
- As illustrated in Figure 2, other examples of graphs include the *bridge* and the *eight*, whose calibre is 1, as well as the *lattice*, whose calibre is 2.

Let us compare the calibre of a rule to its arity. As detailed later in Section 3.2, the complexity of our algorithm depends on $\mathcal{C}(R) + \mathrm{rk}(R)$. It is possible to group

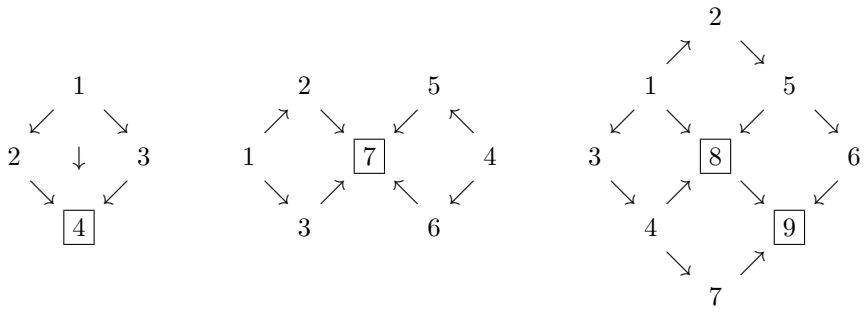

Fig. 2. The bridge, the height and the lattice with their minimal juncture.

predicates: it is equivalent to search for reactants x and y that verify both $f(x, y)$ and $g(x, y)$ or only $(f \wedge g)(x, y)$. This grouping may have an effect on both $\mathcal{C}(R)$ and $\mathrm{rk}(R)$. A direct remark is that, if we group all the predicates, we have $\mathrm{rk}(R) = |\mathrm{var}(R)|$ and $\mathcal{C}(R) = 0$, so it is always possible to find a grouping such that

$$\mathcal{C}(R) + \mathrm{rk}(R) \leq |\mathrm{var}(R)|. \tag{7}$$

The case of rules of rank 2 requires no particular grouping.

Theorem 8 (upper bound on the calibre of a rule with a rank of 2) *Let R be a rule of rank 2. The following inequality is verified, with equality if and only if the graph of R is a clique.*

$$\mathcal{C}(R) + 2 \leq |\mathrm{var}(R)|. \tag{8}$$

Proof. The inequality is due to the fact that regardless the order chosen, the two smallest variables have at most one smaller neighbor, so they cannot be part of any juncture. However, if the graph is a clique, all other variables have at least these two nodes as smaller neighbours, so they are in the juncture, whatever order is chosen.

Conversely, if the graph of R is not a clique, then there exists x and y that are not connected. Let R' be a rule with the same variables as R and a predicate connecting all pairs of variables, except (x, y). Firstly, let us remark that $\mathcal{C}(R) \leq \mathcal{C}(R')$. Secondly, let z be a variable different from x and y, and $\lessdot$ an order in which the three largest items are $z \lessdot x \lessdot y$ in that order. Then, $\mathcal{J}_{\lessdot}(R') = k - 3$, so $\mathcal{C}(R) \leq \mathcal{C}(R') \leq k - 3$. □

Theorem 9 (purification of the juncture's assignment) *Let R be a rule of arity k whose variables are ordered by $\lessdot$ with $\{x_1, \ldots, x_c\} = \mathcal{J}_{\lessdot}(R)$. Let A be a set of axioms. For all $m_1, \ldots, m_c \in \mathrm{mols}(A)$, $A_c = A[x_1 := m_1, \ldots, x_c := m_c]$ is reactive if and only if there are $m_{c+1}, \ldots, m_k \in \mathrm{mols}(A)$ such that $A_k = A[x_1 := m_1, \ldots, x_n := m_k]$ is reactive and purified.*

Proof. If A_k is reactive, given that $A_k \subset A_c$, we have:

$$\forall x \in \mathcal{J}_{\lessdot}(R), |\{m \in A_c : m \models x\}| \geq |\{m \in A_k : m \models x\}| \geq 1$$

so clearly A_c is reactive too.

Conversely, suppose A_c is reactive. The goal is to show that A_c can get purified by choosing one and only one molecule for every variable. This can be done by choosing one molecule for each variable, one by one, following the order $\lessdot$. When choosing a molecule for x_i, three cases can occur, depending on the number of predicates containing x_i and a smaller variable x_j:

(1) if x_i has no smaller neighbour, since A_c is reactive, there exists at least one molecule $m_i \models_{A_c} x_i$. We can choose any one of them.
(2) if there is one (and only one) such predicate p, since A_c is reactive, there is at least one axiom a such that $\text{pred}(a) = p$ in A_c. Then, one can choose $m_i = a[x_i]$.
(3) otherwise, $x_i \in \mathcal{J}_{\lessdot}(R)$, so x_i is already assigned in A_c. Since the assignment is reactive, there exists $m_i \in \text{mols}(A_c)$ that is suitable with all the already chosen variables. □

3.2. *The PMJA algorithm*

In this section, we present the PMJA (Purification of the Minimal Juncture Assignment) algorithm that solves the reactants searching problem. Algorithm 1 shows the global PMJA algorithm for a rule R, with an arity k and a juncture $\mathcal{J}_{\lessdot}(R) = \{x_1, \ldots, x_c\}$.

It takes as argument a set of axioms A of type **AxiomSet** organised like a graph when the rank is 2. A gives access to all the molecules used in these axioms, sorted by the variables they satisfy. Each molecule gives in turn access to a set of references to the axioms in which it is an argument, sorted by predicate. In terms of implementation, an AxiomSet could be implemented through a structure having: 1) a hash table of molecules, where a molecule is retrieved using the variable it satisfies as the key, 2) a hash table of the axioms these molecules satisfy retrieved using the predicate they implement as a key, 3) cross-references from molecules to axioms, andfrom variables to predicates.

According to this structuring, the physical size of the AxiomSet A for a rule R and a set of molecules M, is in $O(|\mathcal{A}(R, M)| + |\text{mols}(\mathcal{A}(R, M))|)$, and getting predicates and molecules from variables as well as getting axioms from molecules can be done in constant time, apart from cloning the structure itself which is necessarily linear in the size of A. By convention, the indices of an array $tab[]$ vary between 1 and tab.size.

The algorithm is based on Theorem 9, that suggests to test the reactivity of all the possible assignments of a juncture to detect inertia. Consequently, it is composed of a main loop, which is executed once for every tuple of molecules $(m_1, \ldots, m_c)$ that may be used to build an assignment of $\mathcal{J}_{\lessdot}(R)$. More precisely, as can be seen in Algorithm 1, the loop is composed of two main parts:

Algorithm 1: Reactants searching for R with $\text{var}(R) = \{x_1, \ldots, x_k\}$ and $\mathcal{J}_{\lessdot}(R) = \{x_1, \ldots, x_c\}$.

```
1 Molecule[] : findReactants(AxiomSet A) :
2     forall m_1 ⊨ x_1, ..., m_c ⊨ x_c do
          // m_i variables are global
3         AxiomSet A' ← A.clone();
4         buildAssignment(A');
5         if ¬(refineAndTestReactivity(A')) then
6             continue
7     return(purify(A'));
```

Algorithm 2: Assignment building given an AxiomSet.

```
1 AxiomSet : buildAssignment(AxiomSet A) :
      // A is passed by reference
2     for i ← 1 to c do
3         forall Molecule m ∈ A.molecules(i) s.t. m ≠ m_i do
4             m.removed ← true;
5             forall Axiom a ∈ m.axioms(*) do
6                 a.removed ← true;
```

(1) In the first part of the loop (Lines 3-5), the assignment $A' = A[x_1 \leftarrow m_1, \ldots, x_c \leftarrow m_c]$ is computed. If it is not reactive, it is dropped. The assignment is computed by a successive elimination of molecules, through two steps:

   (a) First, through the use of the *buildAssignment()* function, the assignment is built from the AxiomSet. As detailed in Algorithm 2, this function takes the cloned AxiomSet by reference, and removes all molecules that were not chosen from it. Subsequently, all axioms the removed molecule were in are removed on their turn.
   (b) Secondly, through the use of the *refineAndTestReactivity()* function detailed in Algorithm 3, the build assignment, still stored in the same AxiomSet variable is refined by removing all the molecules that cannot be in any refined subset of A'. It is done by repeatedly removing molecules that are not a member of any axiom left after removals made in the *buildAssignment()* function. When molecules are removed, all the axioms they took part in are also removed, making it potentially possible to remove other molecules, and so on, until either we cannot find any more non-used molecule or some variable

Algorithm 3: Refining and testing the reactivity of an AxiomSet.

```
1  Boolean : refineAndTestReactivity(AxiomSet A) :
       // A is passed by reference
2      Boolean reactive, changed;
3      repeat
4          changed ← false;
5          for i ← 1 to k do
6              reactive ← false;
7              forall Molecule m ∈ A.molecules(i) s.t. ¬m.removed do
8                  reactive ← true;
9                  forall Predicate p ∈ x_i.predicates() do
10                     Axiom at[ ] ← m.axioms(p);
11                     while m.first(p) ≤ at.size ∧ at[m.first(p)].removed do
12                         m.incrementFirst(p);
13                     if m.first(p) > at.size then // m.axioms(p) is empty
14                         changed ← true; m.removed ← true;
15                         forall Axiom a ∈ m.axioms(∗) do
16                             a.removed ← true;
17             if ¬reactive then break;
18     until ¬reactive ∨ ¬changed;
19     return(reactive)
```

cannot get satisfied anymore.[e] Remind that the axiomSet passed as a parameter to both *buildAssignment()* and *refineAndTestReactivity()* is passed by reference. In addition to refining the AxiomSet, the *refineAndTestReactivity()* function finally returns *true* if the refined AxiomSet is reactive, *false* otherwise.

(2) The second part in Algorithm 1 starts after the refinement in case the *refineAndTestReactivity()* function returned *true*, as if all molecules are still satisfied, according to Theorem 9, this assignment of the juncture can be purified so as to make a reaction. The *purify()* function achieves this purification and returns a set of molecules that can react. This set is finally returned by the global algorithm.

If no assignment of the juncture is reactive, the solution is inert and the algorithm returns $\perp$. From these observations, it can be inferred that the algorithm

[e]For each molecule m and predicate p for which m can be an argument, we keep an index on the first non-removed axiom corresponding for p and containing m, m.first(p) initialised to 1, in order to check efficiently if a molecule must be removed.

Algorithm 4: Purifying a reactive AxiomSet.

```
1  Molecule [] : purify(AxiomSet A) :
2      Molecule M[k];
3      for i ← 1 to k do //  x_σ(1) ≪ ··· ≪ x_σ(k)
4          switch {p ∈ pred(R) : ∃x_σ(j) ≪ x_σ(i), {x_σ(i), x_σ(j)} ⊂ arg(p)} do
5              case ∅ : // choose any
6                  forall Molecule m ∈ A.molecules(σ(i)) do
7                      if ¬m.removed then  M[i] ← m; break;
8                  break;
9              case {p} : //  x_σ(i) ≫ x_σ(j) ∈ arg(p)
10                 M[i] ← M[j].axioms(p)[M[j].first(p)].get(x_i); break;
11             otherwise //  x_σ(i) ∈ J_≪(R)
12                 M[i] ← m_i;
13     return M;
```

correctly returns a reactive purified assignment if there is one, and $\perp$ otherwise.

Proposition 10 (Time complexity) *In the worst case, the time complexity of Algorithm 1 is* $T(R, A) = \mathcal{O}\left(|\text{mols}(A)|^{\mathcal{C}(R)+\text{rk}(R)}\right)$.

Proof.

If the solution is inert, which is the worst case, there are $\prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}|$ executions of the main loop. We will show that the complexity of one iteration of the main loop is proportional to the size of the axiom set.

- In the *buildAssignment()* function, where the molecules that do not match the choice are removed, each axiom is considered at most once for each argument, leading to a complexity in $\text{rk}(R) \times |A|$[f].
- The complexity of the *refineAndTestReactivity()* function is defined by the complexity of its two inner loops. On Line 11 of Algorithm 3, the field $m.first(p)$ can only grow, so each axiom is checked only once for each molecule in its arguments. In the loop of Line 15, an axiom can only be removed once for each argument.
- The complexity of the *purify()* function is only $|\text{var}(R)| \ll |A|$.

The complexity of the main loop does not exceed $\text{rk}(R) \times |A|$ and is executed

[f]Here, $|A|$ is to be interpreted as the number of axioms, not the size of the AxiomSet structure.

$\prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}|$ times, so the complexity of the whole algorithm is

$$\mathcal{O}\left(\mathrm{rk}(R) \times |A| \times \prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}|\right).$$

We can simplify the writing. Since $|\{m \models x\}| \leq |\mathrm{mols}(A)|$, we have:

$$\prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}| \leq |\mathrm{mols}(A)|^{|\mathcal{J}_{\ll}(R)|}$$

and

$$|A| \leq \binom{|\mathrm{mols}(A)|}{\mathrm{rk}(R)} \leq \frac{|\mathrm{mols}(A)|^{\mathrm{rk}(R)}}{\mathrm{rk}(R)!}$$

Finally,

$$\mathrm{rk}(R) \times |A| \times \prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}| \leq \frac{1}{(\mathrm{rk}(R)-1)!} |\mathrm{mols}(A)|^{\mathcal{C}(R)+\mathrm{rk}(R)}$$

and the complexity of the algorithm is in $\mathcal{O}\left(|\mathrm{mols}(A)|^{\mathcal{C}(R)+\mathrm{rk}(R)}\right)$.

□

We have seen that it was possible to find an R such that $\mathcal{C}(R)+\mathrm{rk}(R) \leq |\mathrm{var}(R)|$. This establishes that the proposed algorithm has a complexity which is either similar or better than $\mathcal{O}(n^k)$. Note that in practice, we have actually $\mathcal{C}(R) + \mathrm{rk}(R) < |\mathrm{var}(R)|$ most of the time. Note also that this gain can be evaluated at compile time (it depends on the rank and the calibre of a rule, both obtained by static analysis of the rule, matter of Section 2).

4. Related Work

4.1. *Reactants searching problem*

To our knowledge, the problem of searching molecules to implement the chemical model as a computing paradigm, has only been addressed through exhaustive search, in particular in distributed settings, as studied in works like [14] in a shared memory architecture, or more recently, in the architectural framework in [21].

Artificial chemistries [12] have emerged primarily to provide computing models exhibiting interesting computational powers and artificial life models, for which the main objective is to show the match between the model and real settings.

The area of membrane computing [23], which devises a computing model based on the analogy of a set of possibly nested cells, is very similar to the chemical model when sub-solutions are allowed. The problems addressed in this area are closer to those of organic computing in general. Membrane computing consists mainly in devising models exhibiting interesting computational powers. As stated in [23], the problem of mapping such a model onto real computers is an open problem, would it be *in silico* or using real organic systems. Consequently, researches pursued in

this area did not actually tackle the problem. Nevertheless, interesting links between membrane systems and distributed computing are discussed in [9], specifically about how distributed algorithms can help implement P-Systems. Still, the problem of searching reactants is not addressed in [9], which focuses on classic issues in distributed systems such as mutual exclusion, and on how existing solutions to this problem can be adapted to the context of membranes. A similar problem, namely the allocation of multiple molecules in distributed settings, has been addressed in [6].

The present work is relevant for computing models based on multiset processing allowing to have complex conditions on rules. In literature, models with conditions, such as [2, 3, 10, 16] and models without conditions, such as [5, 23] have been proposed.

4.2. *Subgraph isomorphism problem*

The subgraph isomorphism problem has been well documented [11] since the first algorithm proposed by Ullman in 1976 [24], which was based on backtracking. In [19], a decision tree is built, allowing the search for isomorphic subgraphs in polynomial time, after a pre-treatment in exponential time. This work cannot be applied in our case, as the pre-treatment would have to be made on the graph of the molecules, which constantly changes depending on the reactions arising in the computation.

As shown in [18], it is possible to reformulate the subgraph isomorphism problem as a *Constraint Search Problem* (CSP). The distributed version of CSP, disCSP [8], is very close to the reactants searching problem. The disCSP problem is generally treated through an exhaustive exploration of the nodes of the network [13, 27], possibly with an optimisation using back-tracking [26].

5. Conclusion and Future Work

In this paper, we have shown that, through a static analysis of the reaction conditions, concurrent multiset rewriting using a rule of arity k can be solved in a time bounded by $n^{\mathrm{rk}(R)+\mathcal{C}(R)}$, with $\mathrm{rk}(R)+\mathcal{C}(R) \leq k$, and we exhibited the PMJA algorithm for this. For rules of rank 2, we were able to characterise the case of equality. An interesting point is that we are able to estimate the execution complexity at compile time, which makes it possible to provide the programmer with optimisation recommendations.

This work needs continuation at several levels. Firstly, there is a need to deal with compilation. So far we have only presented the effective search of molecules knowing the graph of the rule. We used arguments from logics to express that compilation was indeed possible. It would be interesting to find efficient algorithms, in particular to choose an optimal order of the variables in the rule. Choosing an order for which the juncture is minimal can be done once, at compile time, while providing guarantees on the complexity of the algorithm. However, the algorithm is flexible and works regardless of the juncture. The complexity could be further reduced by a finer choice of the order, with respect to the quantity of molecules for

each variable. (Among several minimal junctures, prefer those for which the number of actual molecules is minimised.)

Secondly, going further in the analysis of the reaction conditions could improve these results. For example, many literals are of the form $f(x) \circ g(y)$ where x and y are variables, f and g are functions to the same set E and $\circ$ is an order or equivalence relation on E. In this case, the search can be improved, for example through the sorting of the values of $f(m)$ and $g(n)$ for the suitable molecules m and n.

Finally, a planned work is the extension of this study to some practical aspects of the chemical programming model, including the possibility to have multiple rules in the solution, sub-solutions, and the higher order, that changes the behaviour of the program at run time.

References

[1] Ö. Babaoglu, G. Canright, A. Deutsch, G. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor and T. Urnes, Design patterns from biology for distributed computing, *TAAS* **1**(1) (2006) 26–66.

[2] J.-P. Banâtre, P. Fradet and Y. Radenac, Principles of Chemical Programming, *Electronic Notes in Theoretical Computer Science* **124**(1) (2005) 133 – 147, Proceedings of the 5th International Workshop on Rule-Based Programming (RULE 2004) Rule-Based Programming 2004.

[3] J.-P. Banâtre, P. Fradet and Y. Radenac, Generalised multisets for chemical programming, *Mathematical. Structures in Comp. Sci.* **16** (August 2006) 557–580.

[4] J.-P. Banâtre, Y. Radenac and P. Fradet, Chemical specification of autonomic systems, *ISCA 13th International Conference on Intelligent and Adaptive Systems and Software Engineering*, Nice, France (2004), pp. 72–79.

[5] G. Berry and G. Boudol, The chemical abstract machine, *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, *POPL '90*, (ACM, New York, NY, USA, 1990), pp. 81–94.

[6] M. Bertier, M. Obrovac and C. Tedeschi, Adaptive atomic capture of multiple molecules, *J. Parallel Distrib. Comput.* **73**(9) (2013) 1251–1266.

[7] M. Bertier, M. Perrin and C. Tedeschi, On the Complexity of Concurrent Multiset Rewriting, Research Report RR-8408 (December 2013).

[8] I. Brito, Synchronous, asynchronous and hybrid algorithms for discsp, *Principles and Practice of Constraint Programming (CP 2004)*, ed. M. Wallace, *Lecture Notes in Computer Science* **3258** (Springer Berlin Heidelberg, 2004), pp. 791–791.

[9] G. Ciobanu, R. Desai and A. Kumar, Membrane systems and distributed computing, *International Workshop on Membrane Computing 2002*, (2002), pp. 187–202.

[10] D. Cohen and J. M. Filho, Introducing a calculus for higher-order multiset programming, *Proceedings of the First International Conference on Coordination Languages and Models*, *COORDINATION '96*, (Springer-Verlag, London, UK, UK, 1996), pp. 124–141.

[11] D. Conte, P. Foggia, C. Sansone and M. Vento, Thirty years of graph matching, *International Journal of Pattern Recognition and Artificial Intelligence* **18**(3) (2004) 265–298.

[12] P. Dittrich, J. Ziegler and W. Banzhaf, Artificial chemistries – a Review, *Artificial Life* **7** (June 2001) 225–275.

[13] A. Doniec, S. Piechowiak and R. Mandiau, A discsp solving algorithm based on ses-

sions., *18th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, eds. I. Russell and Z. Markov (AAAI Press, 2005), pp. 666–670.

[14] K. Gladitz and H. Kuchen, Shared Memory Implementation of the Gamma-Operation, *J. Symb. Comput.* **21**(4) (1996) 577–591.

[15] S. Grumbach and F. Wang, Netlog, a rule-based language for distributed programming, *PADL*, eds. M. Carro and R. Peña *Lecture Notes in Computer Science* **5937**, (Springer, 2010), pp. 88–103.

[16] C. Hankin, D. L. Métayer and D. Sands, A Parallel Programming Style and Its Algebra of Programs, *Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, *PARLE '93*, (Springer-Verlag, London, UK, 1993), pp. 367–378.

[17] S. Hariri and M. Parashar, *Handbook of Bioinspired Algorithms and Applications* (CRC Press LLC, 2005), ch. The Foundations of Autonomic Computing.

[18] J. Larrosa and G. Valiente, Constraint satisfaction algorithms for graph pattern matching, *Mathematical. Structures in Comp. Sci.* **12** (August 2002) 403–422.

[19] B. T. Messmer and H. Bunke, Subgraph isomorphism in polynomial time, tech. rep., Institut fur Informatik und angewandte Mathematik, University of Bern, Switzerland (1995).

[20] A. Mostéfaoui, Towards a computing model for open distributed systems, *PaCT*, ed. V. E. Malyshkin *Lecture Notes in Computer Science* **4671**, (Springer, 2007), pp. 74–79.

[21] M. Obrovac and C. Tedeschi, Distributed chemical computing: A feasibility study, *International Journal of Unconventional Computing* **9**(3-4) (2013) 203–236.

[22] M. Parashar and S. Hariri, Autonomic Computing: An Overview, *International Workshop on Unconventional Programming Paradigms (UPP 2004)*, *LNCS* **3566**, (Springer, Le Mont Saint-Michel, France, 2005), pp. 257–269.

[23] G. Paun, G. Rozenberg and A. Salomaa, *The Oxford Handbook of Membrane Computing* (Oxford University Press, Inc., New York, NY, USA, 2010).

[24] J. R. Ullmann, An algorithm for subgraph isomorphism, *J. ACM* **23** (January 1976) 31–42.

[25] M. Viroli and F. Zambonelli, A biochemical approach to adaptive service ecosystems, *Inf. Sci.* **180**(10) (2010) 1876–1892.

[26] R. Zivan and A. Meisels, Parallel backtrack search on discsp, *Workshop on Distributed Constraint Reasoning (DCR-02)*, Bologna, Italy (2002).

[27] R. Zivan and A. Meisels, Message delay and discsp search algorithms, *Annals of Mathematics and Artificial Intelligence* **46** (April 2006) 415–439.