On the Complexity of Concurrent Multiset Rewriting

Marin Bertier, Matthieu Perrin, Cédric Tedeschi

On the Complexity of Concurrent Multiset Rewriting

Marin Bertier, Matthieu Perrin, Cédric Tedeschi

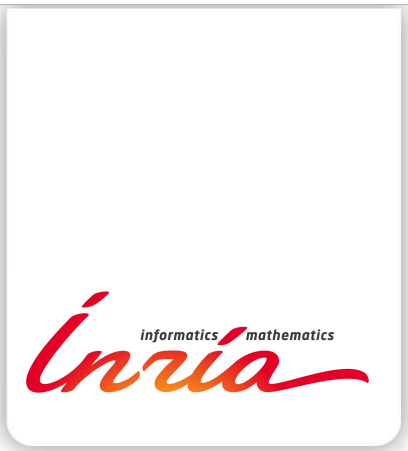
RESEARCH REPORT

N° 8408

December 2013

Project-Teams Myriads, ASAP

ISSN 0249-6399 ISRN INRIA/RR--8408--FR+ENG



On the Complexity of Concurrent Multiset Rewriting

Marin Bertier, Matthieu Perrin, Cédric Tedeschi

Project-Teams Myriads, ASAP

Research Report n° 8408 — December 2013 — 17 pages

Abstract:
While the complexity of the digital infrastructures surrounding us seems to increase continuously, autonomic computing appears as a promising paradigm to be explored. The quest for adequate high-level abstractions to program autonomic systems led to the reemergence of rule-based programming as a promising paradigm. Also, nature appears to be a great source of inspiration. 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, making it a promising candidate to develop autonomic systems. The metaphor envisions a computation as a set of concurrent reactions between molecules of data arising non-deterministically, until no more reactions can take place, in which case, the solution contains the final outcome of the computation.
More formally, such models strongly rely on concurrent multiset rewriting: the data are a multiset of molecules, and reactions are the application of a set of conditioned rewrite rules. At run time, these rewritings are applied concurrently, until no rule can be applied anymore (the elements they need do not exist anymore in the multiset). One of the main barriers towards the actual adoption of such models come from their complexity at run time: each computation step may require a complexity in $O(n^k)$ where n denotes the number of elements in the multiset, and k the size of the subset of elements needed to trigger one rule.
In this paper, we explore the possibility of improving the complexity of searching elements (or *reactants*) through a static analysis of the reaction condition. In particular, through the modeling of the problem with graphs, we give a characterisation of this complexity, based on the results of the well-known subgraph isomorphism problem. This allows us then to give an algorithm for searching reactants, that can be easily parallelised, and whose complexity is lower than the basic case, most of the time.

Key-words: Chemical Programming Model, Multiset Rewriting, Complexity

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Sur la complexité de la réécriture concurrente de multi-ensembles

Résumé :

La quête de modèles de programmation capable de programmer des systèmes autonomes a conduit à la réemergence de la programmation à base de règles comme un paradigme prometteur. Le modèle de programmation chimique combine ce paradigme avec un modèle d'exécution d'inspiration chimique, dans lequel les règles sont appliquées de façon non-déterministe, jusqu'à l'*inertie*, c'est-à-dire un état où la condition d'exécution d'aucune règle de *réaction* n'est satisfaite.

De façon plus formelle, ce modèle s'appuie foirtement sur la réécriture concurrente de multi-ensemble. Les données sont représentés par les *molécules* de ce multi-ensemble, et les règles sont des règles de réécriture de ce multi-ensemble. Le principal problème de ce modèle est sa complexité à l'exécution. Chaque pas de calcul peut avoir une complexité au pire cas en $O(n^k)$ où n est le nombre d'éléments du multi-ensemble et k la taille du sous-ensemble d'éléments nécessaire à l'application d'une règle.

Dans ce rapport, nous explorons la possibilité d'améliorer la complexité de la recherche de *réactifs* à travers une analyse statique des conditions de réactions. A travers la modélisation du problème par des graphes, une caractérisation de cette complexité est donnée par réduction au problème de sous-graphe isomorphe, ce qui permet de donner ensuite un algorithme de recherche des réactifs facilement parallélisable, et dont la complexité est meilleur qu'une recherche exhaustive, la plupart du temps.

Mots-clés : Modèle de programmation chimique, réécriture de multi-ensemble, complexité

Contents

1 Introduction

The amount of technologies surrounding us seems to grow continuously. The underlying infrastructure supporting such a computing power (*e.g.*, the Internet) is highly large, dynamic and heterogeneous. When coming to build applications on top of such platforms, its characteristics make the practical leveraging of such a computing power, a quite difficult problem.

One answer comes from *autonomic computing*, which consists in building systems that can "*manage themselves in accordance with high-level guidance from humans*" [15]. Put differently, humans should ideally only be required to write a set of *high-level rules* defining the behaviour of the system. Then the system should be able to run indefinitely, adhering to these high-level, technical details-free rules, whatever the conditions on the underlying platform are.

The implementation of an autonomic system offers several challenges that can not 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 adequate high-level abstractions led to the reemergence of rule-based programming as a promising paradigm, for instance to specify distributed systems in a declarative manner [8]. Also, nature appears to be a great source of inspiration. It was early suggested that biological and chemical system were analogies worth to be explored [1, 9, 17]. 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 shows good properties to develop autonomic systems [2, 13].

Metaphorically speaking, a *chemical* program is envisioned as a *chemical solution* where molecules of data float and react according to some *reaction rules* specifying the program, to produce new data (products of reactions). Formally speaking, these artificial chemistries [5] strongly rely on *concurrent multiset rewriting* : the solution is a multiset whose elements are the molecules, and reactions are conditioned rewrite rules to be applied on it. At run time, rules are applied concurrently on the multiset. 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 multiset, the program is said to be *inert*. In this state, the solution is stable and contains the final result of the program.

While such a model is today regarded 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* satisfaying the rule's condition has been located in the multiset. Let us assume the number of objects in the multiset is denoted by n, and that the arity of the rule (*i.e.*, the number of reactants needed to be found for its application, is denoted by 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 $O(n^k)$, which can become very problematic, especially with rules with a high arity.

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, through the modeling of the problem with graphs, we give a characterisation of this complexity, based on the subgraph isomorphism problem. This allows us then to give an algorithm for searching reactants, whose complexity is lower than the basic case of $O(n^k)$ in the sequential case, most of the time. Also, we show that this algorithm can be easily parallelised.

Organisation of the paper. The article is organised as follows. Section 2 introduces the problem and gives its modeling. Also, its NP-completeness is recalled. In Section 3, the characterisation of the complexity, by analogy with the subgraph isomorphism problem, is given. Then, the M2JA algorithm putting this result into practice is described. Its complexity, both in sequential and parallel settings is discussed. Section 5 draws some conclusion and gives some hints for future works.

2 Preliminaries

The problem to be solved is the search for elements in a multiset, that match a rule's condition. More specifically, the algorithm to be designed takes two input parameters, namely,

1. a chemical rule $R = \textbf{replace } x_1 :: T_1, \ldots, x_k :: T_k \textbf{ by } P(x_1, \ldots, x_k) \textbf{ if } C(x_1, \ldots, x_k)$;
2. a multiset M composed of n molecules

and returns:

- $\perp$ if $M \cup \langle R \rangle$ is inert, *i.e.* if no subset of molecules of M can satisfy R
- a tuple $(m_1, \ldots, m_k)$ of molecules in M, where $\forall i, m_i$ is of type T_i and $C(m_1, \ldots, m_k)$ is true, otherwise.

2.1 Study of the rules

If rules were not conditioned, the problem would be simple, since in this case, we would only need to compare the number of available molecules for each type to the number of molecules required. Thus, the remainder of the article will mostly focus on the study of the reaction condition.

This reaction condition is a formula of the propositional logic, whose literals are the application of a boolean function to the variables x_1 to x_k. The definition of these literals is left very open, as some implementations of the language allow Java expressions, which makes the reactants searching problem undecidable in theory. As evaluation of Java code is far from the subject of this paper, we assume the evaluation of reaction conditions finishes.

As any propositional formula, a reaction condition can be put in disjunctive normal form. Note that the type of a molecule can be seen as a condition on this molecule. Our formula is then of the following form, where the $X_{i,j}$ are subsets of variables:

$$R = \left(\textbf{replace } x_1, \ldots, x_k \textbf{ by } P(x_1, \ldots, x_k) \textbf{ if } \bigvee_{i=1}^{L} \bigwedge_{j=1}^{l_i} f_{i,j}(X_{i,j}) \right), \forall i, j, X_{i,j} \subset \{x_1, \ldots, x_k\}.$$

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 $C_2(x_1, \ldots, x_k)$, so the various terms of the disjunction can be searched separately. The rule R can be applied if and only if one of the rules of the following set can be applied:

$$\bigcup_{i=1}^{L} \left\{ \textbf{replace } x_1, \ldots, x_k \textbf{ by } P(x_1, \ldots, x_k) \textbf{ if } \bigwedge_{j=1}^{l_i} f_{i,j}(X_{i,j}) \right\}$$

Thus, from now on, we will restrict our analysis to the rules of the following form:

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

For such a rule, let us define $\mathcal{V}(R) = \{x_1, \ldots, x_k\}$ the set of its *variables* and $\mathcal{P}(R) = \bigcup_{i=1}^{l} \{(f_i, X_i)\}$ the set of its *predicates*, each predicate being associated to a litteral in the reaction condition. For a predicate $p = (f, X)$, we also define $\mathcal{F}(p) = f$ and $\arg(p) = X$.

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 all of the rules, the search process for different rules is independent, they can be fully parallelised.

In the following definition, we make possible to group predicates. We can then assume, modulo equivalence, that the arguments of the predicates of a rule are pairwise disjoint.

Definition 1 (equivalence of rules) *Let us define $\equiv$ as the smallest equivalence relation on the rules such that, for any rule R_1 and R_2:*

$$\left.\begin{array}{rcl} \mathcal{V}(R_1) & = & \mathcal{V}(R_2) \\ \wedge\ \mathcal{P}(R_1) & = & P \cup \{(f_1, E_1), (f_2, E_2)\} \\ \wedge\ \mathcal{P}(R_2) & = & P \cup \{(f_1 \wedge f_2, E_1 \cup E_2)\} \end{array}\right\} \Longrightarrow R_1 \equiv R_2.$$

Definition 2 (rank of a rule) *We define the* rank *of a rule R as the greatest arity of its predicates:*

$$\mathrm{rk}(R) = \max_{p \in \mathcal{P}(R)} |\arg(p)|. \tag{1}$$

A rule with a rank of 2 can be represented as a graph, in which the vertices are the variables and the edges are the predicates. Note that most of the problems encountered in literature, can be solved by rules with a rank of 1 or 2. For rules with a higher rank, some predicates connect more than two variables, making the rule impossible to be pictured with such a graph. Figure 1 illustrates this with two examples, of the following rules of arity 4 and of respective rank 2 and 3. While the second one can still be represented graphically, it cannot be expressed as a graph.

$$\textbf{replace } x, y, z, t \textbf{ by } P(x, y, z, t) \textbf{ if } f(z, y) \wedge g(x, z) \wedge h(x, t) \wedge i(y, t) \wedge j(z, t) \tag{2}$$

$$\textbf{replace } x, y, z, t \textbf{ by } P(x, y, z, t) \textbf{ if } f(x, y, z) \wedge g(x, y, t) \wedge h(x, z, t) \wedge i(y, z, t) \tag{3}$$

Figure 1: Graphical representations of the rules (2), of arity 4 and rank 2, and (3), of arity 4 and rank 3.

2.2 Structuration of the multiset

The previous section was focused on defining the *world of rules*. We now devise the characterisation of the *world of molecules*, and the relationships between both worlds. The central definition in the following is the *axiom*. It represents a predicate whose each variable has been associated to an actual molecule.

Definition 3 (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 $a = (p, m)$ be axiom. $\mathcal{F}(a) = \mathcal{F}(p)$ and $\arg(a) = \arg(p)$. Also, we note $\mathcal{P}(a) = p$, $a[x] = m(x)$ for all $x \in \arg(p)$ and $\mathcal{M}(a) = \{a[x] : x \in \arg(p)\}$. We extend this notations to set of axioms: let A be a set of axiom. $\mathcal{M}(A) = \bigcup_{a \in A} \mathcal{M}(a)$ is the set of the molecules contained in A.

Given a rule R and a set of molecules M, we can define the set of axioms that can be constructed. As one can expect, only some of these axioms are of interest for our problem. In the following, we give a characterisation of a set of axioms in regard to the rule it is based on.

Definition 4 (set of induced axioms) *For a solution M and a rule R, we define the* set of induced axioms *by R in M as the set of the axioms composed of a predicate of R and of molecules of M:*

$$\mathcal{A}(R, M) = \{a : \mathcal{P}(a) \in \mathcal{P}(R) \wedge \forall x \in \arg(a), a[x] \in M\} \tag{4}$$

Definition 5 (satisfaction of a variable) *Let $x \in \mathcal{V}(R)$, A be a set of axioms and $m \in \mathcal{M}(A)$. The molecule m is said to* satisfy *x in A, denoted $m \models_A x$, if for each predicate of the rule pertaining x, we can find at least one axiom in A in which x is associated to m:*

$$\forall p \in \mathcal{P}(R), x \in \arg(p) \Rightarrow (\exists a \in A, \mathcal{P}(a) = p \wedge a[x] = m). \tag{5}$$

Informally speaking, a molecule m does not satisfy a variable x if there is no subsets of molecules containing m in the solution, that verify a predicate in the rule that contains x.

Definition 6 (refined and exclusive set of axioms) *A set of axioms A is said to be* refined *if each of its molecule satisfies at least one variable,* i.e. *if condition 6 is true. A is said to be* exclusive *if each of its molecule satisfies at most one variable,* i.e. *if condition 7 is true.*

$$\forall m \in \mathcal{M}(A), |\{x \in \mathcal{V}(R) : m \models_A x\}| \geq 1 \tag{6}$$

$$\forall m \in \mathcal{M}(A), |\{x \in \mathcal{V}(R) : m \models_A x\}| \leq 1 \tag{7}$$

More importantly, we define the *validity* and *maximality* of a set of axioms:

Definition 7 (maximal and valid set of axioms) *A set of axioms A is said to be* valid *for the rule R if at least one molecule satisfies each of its variables,* i.e. *if condition 8 is true. A is said to be* maximal *if at most one molecule satisfies each of its variables,* i.e. *if condition 9 is true.*

$$\forall x \in \mathcal{V}(R), |\{m \in \mathcal{M}(A) : m \models_A x\}| \geq 1 \tag{8}$$

$$\forall x \in \mathcal{V}(R), |\{m \in \mathcal{M}(A) : m \models_A x\}| \leq 1 \tag{9}$$

A possible reaction for a rule R and a multiset M is a one-to-one relation between the variables of R and the molecules of M, such that each molecule satisfies the variable it is associated to. From the above definitions, such a relation corresponds to a subset of the set of axioms induced by R in M is all refined, exclusive, maximal and valid at the same time. This notion of affectation is devised in the following. Note that the exclusivity is preserved by all the transformations we do in the following. Thus, we assume that the set of induced axioms is already exclusive. Note that if it is not, it is possible to reduce to this case by copying the molecules for each of their variable and adding inequality constraints to the rule.

Definition 8 (affectation) *Let A be a set of axioms, $x_1, \ldots, x_p \in \mathcal{V}(R)$ and $m_1, \ldots, m_p \in \mathcal{M}(A)$. An* affectation *of $m_1, \ldots, m_p$ to $x_1, \ldots, x_p$, denoted $A[x_1 := m_1, \ldots, x_p := m_p]$, is the union of all the refined subsets of the set $\{a \in A : \forall i, x_i \in \arg(a) \Rightarrow a[x_i] = m_i\}$.*

Proposition 1 (characterisation of the affectation) *$A' = A[x_1 := m_1, \ldots, x_c := m_c]$ is the biggest refined subset of A, in the sense of inclusion, verifying $\forall i \leq c, \forall m, m \models_{A'} x_i \Rightarrow m = m_i$.*

Proof. Let $B = \{a \in A : \forall i, x_i \in \arg(a) \Rightarrow a[x_i] = m_i\}$ be the set introduced to define the affectation (Definition 8), and $A' = A[x_1 := m_1, \ldots, x_c := m_c]$. Three points must be demonstrated to obtain the property:

1. *The property that $\forall i, \forall m, m \models_{A'} x_i \Rightarrow m = m_i$.* This is true since, as $A' \subset B$, the molecules in A' are also in B, so $m \models_{A'} x_i \Rightarrow m \models_B x_i \Rightarrow m = m_i$.

2. *A' is refined.* Let A_1 and A_2 be two refined subsets of B. Let $m \in \mathcal{M}(A_1 \cup A_2) = \mathcal{M}(A_1) \cup \mathcal{M}(A_2)$. If $m \in \mathcal{M}(A_1)$, there is a variable x such that $m \models_{A_1} x$, so $m \models_{A_1 \cup A_2} x$, and symmetrically for $m \in \mathcal{M}(A_2)$. Therefore $A_1 \cup A_2$ is refined. As the number of subsets of B is finite, A' is refined.

3. *The fact that it is the biggest such set.* Let A'' be a refined subset of A verifying $\forall i, m \models x_i \Rightarrow m = m_i$. Then A'' is a refined subset of B. By definition of the affectation, $A'' \subset A'$.

□

In regard to the graph of the rule, the axiom set induced by a rule with a rank of 2 can be modeled by a similar graph of molecules. In this new kind of graphs, the vertices are the molecules of the solution, and an edge, labeled with p, links two molecules m_1 and m_2 if there is an axiom a in the axiom set such that $\mathcal{P}(a) = p$ and $\mathcal{M}(a) = \{m_1, m_2\}$. Under this formalism, a valid maximal affectation is a subgraph of the graph of molecules that is isomorphic to the graph of the rule, with respect to the labels of the edges.

This 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, as we detail now.

2.3 NP-hardness of the reactants searching problem

Under the assumption of the termination of the evaluation of reaction conditions — in a time then necessarily independent of both n and k — the reactants searching problem is in class NP, as doing all tests between molecules non-deterministically can solve it in a polynomial time. We will therefore study the NP-hardness of the problem, which, to our knowledge, has never been formally established before.

Proposition 2 (NP-hardness) *The reactants searching problem is NP-hard, depending on its arity.*

Proof.

Suppose we know an algorithm that solves the reactants searching problem in a time polynomial both in k, the arity of the rule and n, the number of molecules in the solution.

Let $G = (V, E)$ be a graph. We want to know if G contains a clique of k nodes. Consider the instance of our problem with $R = \Big(\textbf{replace } x_1, \ldots, x_k \textbf{ by } P(x_1, \ldots, x_k) \textbf{ if } \bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} (x_i, x_j) \in E\Big)$ and $M = V$.

Under our hypothesis, this chemical program finds a clique in G in polynomial time, which reduces our problem to that of finding a clique in a graph, establishing the property. □

The rule used to show Property 2 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 characterise the complexity of a rule. This is what we do in the next section, which defines the *stature* of a rule.

3 Efficient Reactants Searching

In this section, we first precisely characterise the complexity of the reactants searching problem for a rule, given its *stature*, which is defined first. Then, we present the M2JA algorithm, which leverages this characterisation to allow a better complexity than the basic $O(n^k)$ case, most of the time. Finally, we show that the M2JA algorithm can be parallelised easily.

3.1 Stature of a rule

Definition 9 (juncture of a rule) *Considering that the variables of a rule are completely sorted by an order $\preccurlyeq$, we define $\mathcal{J}_{\preccurlyeq}(R)$, the* juncture *of R for the order $\preccurlyeq$, as the set of variables for which several predicates contain other variables smaller than them:*

$$\mathcal{J}_{\preccurlyeq}(R) = \{x \in \mathcal{V}(R) : |\{p \in \mathcal{P}(R) : \exists y \prec x, \{x, y\} \subset \arg(p)\}| \geq 2\}\,. \tag{10}$$

Definition 10 (stature of a rule) *The* stature *of a rule is the size of its smallest juncture among all possible orders.*

$$\mathcal{S}(R) = \min_{\preccurlyeq} |\mathcal{J}_{\preccurlyeq}(R)|. \tag{11}$$

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

Let us illustrate the two previous definitions. As rules with a rank of 2 represents a very important part of chemical programs in practice, we will discuss the stature of some rules having this rank.

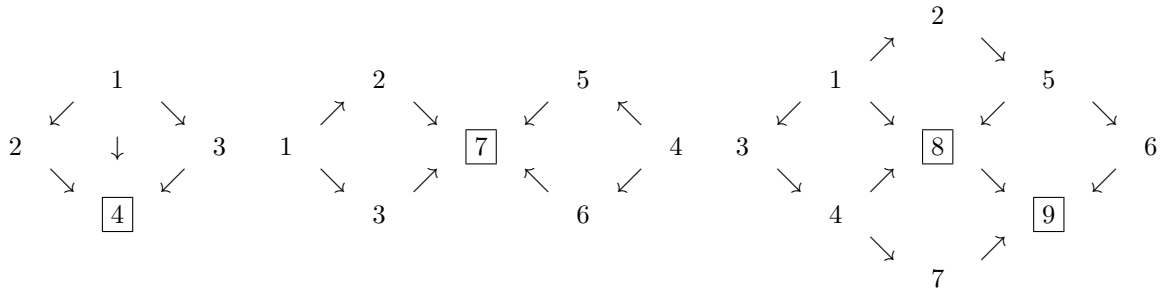


Figure 2: The bridge, the height and the lattice with their minimal juncture.

- The stature of a tree 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 stature of a cycle 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 stature 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 stature is at most one.
- Other examples of graphs (see Figure 2) include the *bridge* and the *eight*, with a stature of 1, as well as the *lattice*, which has a stature of 2.

Let us now compare the stature of a rule to its arity. Again, the case of the rules of rank 2 is interesting to start with.

Proposition 3 (growth of $\mathcal{S}$) *For all rules R_1 and R_2, if $\mathcal{V}(R_1) \subset \mathcal{V}(R_2)$ and $\mathcal{P}(R_1) \subset \mathcal{P}(R_2)$, then $\mathcal{S}(R_1) \leq \mathcal{S}(R_2)$.*

Proof.

The principle of the proof is to start by removing the elements of $\mathcal{P}(R_2) \setminus \mathcal{P}(R_1)$ one by one and then those of $\mathcal{V}(R_2) \setminus \mathcal{V}(R_1)$ to get from R_1 to R_2. We show that each step decreases the stature.

- We first show by induction the next property ($P(n)$) for all $n \geq 0$: every pair of rules (R_1, R_2) satisfying $\mathcal{V}(R_1) = \mathcal{V}(R_2)$, $\mathcal{P}(R_1) \subset \mathcal{P}(R_2)$ and $|\mathcal{P}(R_2)| - |\mathcal{P}(R_1)| = n$ verify $\mathcal{S}(R_1') \leq \mathcal{S}(R_2')$.
  - For $n = 0$, $R_1 = R_2$ so $\mathcal{S}(R_1) = \mathcal{S}(R_2)$.
  - Assume now that $P(n)$ is true for a $n \geq 0$.
    Let R_1 and R_2 such that $|\mathcal{P}(R_2)| - |\mathcal{P}(R_1)| = n+1$. Let $p \in \mathcal{P}(R_2) \setminus \mathcal{P}(R_1)$. Consider the rule R such that $\mathcal{V}(R) = \mathcal{V}(R_1) = \mathcal{V}(R_2)$ and $\mathcal{P}(R) = \mathcal{P}(R_1) \cup \{p\}$. We have $|\mathcal{P}(R_2)| - |\mathcal{P}(R)| = n$ then by induction, $\mathcal{S}(R) \leq \mathcal{S}(R_2)$.
    Let $\lessdot$ be an order on $\mathcal{V}(R)$ such that $\mathcal{J}_{\lessdot}(R)$ is minimal. For all variable x,

    $$\{p \in \mathcal{P}(R_1) : \exists y \lessdot x, \{x, y\} \subset \arg(p)\} \subset \{p \in \mathcal{P}(R) : \exists y \lessdot x, \{x, y\} \subset \arg(p)\}$$

    so $\mathcal{J}_{\lessdot}(R_1) \subset \mathcal{J}_{\lessdot}(R)$, and $\mathcal{S}(R_1) \leq |\mathcal{J}_{\lessdot}(R_1)| \leq |\mathcal{J}_{\lessdot}(R)| = \mathcal{S}(R) \leq \mathcal{S}(R_2)$, which shows $P(n+1)$.

By induction, for all rules R_1, R_2 with the same variables and $\mathcal{P}(R_1) \subset \mathcal{P}(R_2)$, $\mathcal{S}(R_1) \leq \mathcal{S}(R_2)$.

- Let us now remove variables. Let R_1 and R_2 such that $\mathcal{P}(R_1) = \mathcal{P}(R_2)$ and $\mathcal{V}(R_1) \subset \mathcal{V}(R_2)$. Regardless of the order $\lessdot$, for any $x \in \mathcal{V}(R_1)$, $(x \in \mathcal{J}_{\lessdot}(R_1)) \Rightarrow (x \in \mathcal{J}_{\lessdot}(R_2))$, so $\mathcal{J}_{\lessdot}(R_1) \leq \mathcal{J}_{\lessdot}(R_2)$, so $\mathcal{S}(R_1) \leq \mathcal{S}(R_2)$.
- Let us finally conclude with any rules R_1 and R_2 such that $\mathcal{V}(R_1) \subset \mathcal{V}(R_2)$, $\mathcal{P}(R_1) \subset \mathcal{P}(R_2)$. Let R be the rule with the variables $\mathcal{V}(R_2)$ and the predicates $\mathcal{P}(R_1)$. We have: $\mathcal{S}(R_1) \leq \mathcal{S}(R) \leq \mathcal{S}(R_2)$.

□

Theorem 1 (upper bound on the stature of a rule with a rank of 2) *Let R be a rule of rank 2. Then the following inequality is verified, with equality if and only if the graph of R is a clique.*

$$\mathcal{S}(R) + \mathrm{rk}(R) \leq |\mathcal{V}(R)| \tag{12}$$

Proof. Let R be a rule whose rank is 2, and $\lessdot$ an order on $\mathcal{V}(R)$. The two smallest variables x and y have, each, at most one smaller neighbour. Therefore, they are not part of $\mathcal{J}_{\lessdot}(R)$, so $\mathcal{S}(R) \leq |\mathcal{V}(R)| - 2 = |\mathcal{V}(R)| - \mathrm{rk}(R)$.

If the graph of R is a clique, any variable z different from x and y, has at least x and y as smaller neighbours, so $z \in \mathcal{J}_{\lessdot}(R)$, then $|\mathcal{J}_{\lessdot}(R)| = |\mathcal{V}(R)| - \mathrm{rk}(R)$, so $\mathcal{S}(R) = |\mathcal{V}(R)| - \mathrm{rk}(R)$.

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). By Proposition 3, $\mathcal{S}(R) \leq \mathcal{S}(R')$. 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 $\mathcal{S}(R) \leq \mathcal{S}(R') \leq k - 3$. □

Let us remark that the previous property is false for a rule with a rank greater than 2, and that even the property that $\mathcal{S}(R) \leq |\mathcal{V}(R)| - 2$ is false. This is because two variables can be connected by multiple predicates. This is the case for the rule (3) whose graph is illustrated on Figure 1. Note that this rule with 4 arguments has a stature equal to 3, which contradicts Theorem 1. This shows that in the case of higher rank rules, it is necessary to relax this theorem. This can be done using the notion of equivalence of rules presented before.

Theorem 2 (upper bound on the stature for a rule with a rank greater than 2) *For any rule R, we can find $R' \equiv R$ such that $\mathcal{S}(R') + \mathrm{rk}(R') \leq |\mathcal{V}(R')|$.*

Proof. Let R be a rule and R' be the rule with the same variables as R but only one predicate $(C, \{x_1, \cdots, x_k\})$, where C is the reaction condition of the rule. As only one predicate is present, the stature is 0. We have $\mathrm{rk}(R') \leq |\mathcal{V}(R')|$, so $\mathcal{S}(R') + \mathrm{rk}(R') \leq |\mathcal{V}(R')|$.

Remark 1 *The rule R' defined in the proof is usually not the one that minimises $\mathcal{S}(R') + \mathrm{rk}(R')$. It is simply given as an illustration.*

□

Theorem 3 (maximisability of the juncture's affectation) *Let R be a rule of arity k whose variables are ordered by $\lessdot$ with $\{x_1, \ldots, x_c\} = \mathcal{J}_{\lessdot}(R)$, and A be a set of axioms.*

For all $m_1, \ldots, m_c \in \mathcal{M}(A)$, $A_c = A[x_1 := m_1, \ldots, x_c := m_c]$ is valid if and only if there are $m_{c+1}, \ldots, m_k \in \mathcal{M}$ such that $A_k = A[x_1 := m_1, \ldots, x_n := m_k]$ is maximal and valid.

Proof. Suppose there are $m_{c+1}, \ldots, m_k$ such that A_k is valid, then

$$\begin{array}{rcl} A_k = A[x_1 := m_1, \ldots, x_k := m_k] & = & A[x_1 := m_1, \ldots, x_c := m_c][x_{c+1} := m_{c+1}, \ldots, x_k := m_k] \\ & \subset & A[x_1 := m_1, \ldots, x_c := m_c] = A_c. \end{array}$$

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

Conversely, suppose A_c is valid. We choose one molecule per variable, following the order $\lessdot$. When choosing a molecule for x_i, three cases can occur, depending on $in(x_i)$, the number of predicates that contain other variables smaller than x_i, which is defined as:

$$in(x_i) = |\{p \in \mathcal{P}(R) : \exists x_j \lessdot x_i, \{x_i, x_j\} \subset \arg(p)\}|$$

1. $in(x_i) = 0$: If x_i has no smaller neighbour, there is a molecule $m_i \models_{A_c} x_i$, as A_c is valid.
2. $in(x_i) = 1$.: Let p be this single predicate. There is an axiom a such that $\mathcal{P}(a) = p$ in A_c because A_c is valid. We choose $m_i = a[x_i]$.
3. $in(x_i) \geq 2$: In this case, $x_i \in \mathcal{J}_{\lessdot}(R)$ (see equation 10), so x_i is already affected in A_c. As this assignment is valid, $m_i \in \mathcal{M}(A_c)$ and is suitable with all the already chosen variables.

A_k made of these choices is maximal because a choice is made for each variable, and valid as we showed the existence in all three cases. □

3.2 The M2JA algorithm

In this section, we present the algorithm M2JA (Maximisation of the Minimal Juncture Affectation) that solves the reactants searching problem. Algorithm 1 shows the algorithm M2JA for a rule R, with an arity of 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 in rank 2. A gives access to all the molecules sorted by variables. Each molecule gives access to a set of references to the axioms in which it is an argument, sorted by predicate. According to this representation, the methods called on A in Algorithm 1 respond in constant time, apart from clone() whose complexity is linear in $|A|$. By convention, the indices of an array $tab[]$ vary between 1 and tab.size.

The algorithm is based on Theorem 3, that suggests to test the validity of all the affectations of a juncture to detect inertia. It is composed of a main loop (starting on line 2), executed for every tuple of molecules $(m_1, \ldots, m_c)$ that defines the affectation of $\mathcal{J}_{\lessdot}(R)$. The loop is composed of two parts:

1. In lines 3-26, the affectation $A' = A[x_1 \leftarrow m_1, \ldots, x_c \leftarrow m_c]$ is computed. If it is not valid, the computation is dropped. The affectation is computed by a successive elimination of molecules. In the first loop, the molecules that were not chosen are removed. In the second loop, the set is refined by removing all the molecules that cannot be in any refined subset of A'. For each molecules m and predicate p for which m can be an argument, we keep an index on the first unremoved axiom corresponding for p and containing m, m.first(p) initialised to 1, in order to check efficiently if a molecule must be removed. (See line 18.)
2. According to Theorem 3, a valid affectation of the jointure can be maximised into a valid maximal affectation. Lines 27-38 follow exactly the three cases of the proof of this theorem.

If no affectation of the juncture is valid, the solution is inert and the algorithm returns $\perp$ on line 39.

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

```
1  Molecule[] : findReactants(AxiomSet A) :
       /* Chose values for J_<(R) = {x_1, ..., x_c}                                  */
2      forall m_1 |= x_1, ..., m_c |= x_c do
           /* Compute A' = A[j_1 <- m_1, ..., j_c <- m_c]                            */
3          AxiomSet A' <- A.clone();
           /* A' <- {a in A : forall i <= c, x_i in arg(a) => a[x_i] = m_i}          */
4          for i <- 1 to c do
5              forall Molecule m in A'.molecules(i) s.t. m != m_i do
6                  m.removed <- true;
7                  forall Axiom a in m.axioms(*) do
8                      a.removed <- true;
           /* Refine A' :  compute a biggest fix point                               */
9          Boolean valid, changed;
10         repeat
11             changed <- false;
12             for i <- 1 to k do
13                 valid <- false;
14                 forall Molecule m in A'.molecules(i) s.t. !m.removed do
15                     valid <- true;
16                     forall Predicate p in x_i.predicates() do
17                         Axiom at[ ] <- m.axioms(p);
18                         while m.first(p) <= at.size & at[m.first(p)].removed do
                           m.incrementFirst(p);
19                         if m.first(p) > at.size then // m.axioms(p) is empty
20                             changed <- true;
21                             m.removed <- true;
22                             forall Axiom a in m.axioms(*) do
23                                 a.removed <- true;
24                 if !valid then break;
25         until !valid | !changed;
26         if !valid then continue ;// A' is not valid:  try next affectation
           (line 2)
           /* A' is valid:  there is a maximal valid affectation A'' subset A'.     */
           /* From this point, we know there is a reaction, let us find it          */
27         Molecule A''[k];
28         for i <- 1 to k do //  x_sigma(1) <| ... <| x_sigma(k)
29             switch {p in P(R) : exists x_sigma(j) <| x_sigma(i), {x_sigma(i), x_sigma(j)} subset arg(p)} do // Only
               depends on R
30                 case emptyset : // choose any
31                     forall Molecule m in A'.molecules(sigma(i)) do
32                         if !m.removed then  A''[i] <- m; break;
33                     break;
34                 case {p} : //  x_sigma(i) >> x_sigma(j) in arg(p)
35                     A''[i] <- A''[j].axioms(p)[A''[j].first(p)].get(x_i); break;
36                 otherwise //  x_sigma(i) in J_<(R)
37                     A''[i] <- m_i;
38         return A'';
39     return bottom;// No affectation is valid:  there is no solution
```

Proposition 4 (Computation of the affectation) *In Algorithm 1 after line 26, A' is valid and its molecules that are not marked as* removed *are $\mathcal{M}(A[x_1 := m_1, \ldots, x_c := m_c])$.*

Proof.
We will first use Proposition 1 to show that A' has the same molecules as the affectation. We need to show the three following points.

1. *The property $\forall i, \forall m, m \models_{A'} x_i \Rightarrow m = m_i$.* We never put the *remove* field of a molecule to false between lines 3 and 26. So if a molecule $m \neq m_i$ satisfies x_i in A' line 26, it also does line 9. But that is not possible because it would have been removed on line 6.

2. *A' is refined.* Line 26, *valid* is true, so *changed* is false, so the condition of line 19 was always false in the last iteration, which means exactly that A' is refined.

3. *The fact that it is the biggest such set.* Let B be a refined subset of A such that $\forall i, \forall m, m \models_{A'} x_i \Rightarrow m = m_i$. Suppose there is a molecule in $\mathcal{M}(B) \setminus \mathcal{M}(A')$. Let m be the first molecule from B removed from A' in Algorithm 1. It was after line 9 because B verifies the first condition, so it was removed on line 21. But m is at least in the same axioms there as in B, as it is the first molecule of $\mathcal{M}(B) \setminus \mathcal{M}(A')$ removed. Therefore, m can not be in B as B is refined. This is a contradiction, so $\mathcal{M}(B) \subset \mathcal{M}(A')$.

We will now show that A' is valid. If note, at least one variable would be associated to zero molecule in A'. A' is not modified during the last loop, so the same property is true at the beginning of the last loop. Consequently, the variable *valid* is set to $false$ on line 13, and line 26 can not be reached.

□

Proposition 5 (Correctness) *Algorithm 1 returns a valid maximal affectation if there is one and $\perp$ otherwise.*

Proof. If the algorithm does not return $\perp$, it ends on line 38. In this case, on line 26, $A' = A[x_1 := m_1, \ldots, x_c := m_c]$ is valid by Proposition 4, then by Theorem 3, there is a valid maximal affectation, which is the one returned by the algorithm, the three cases in the switch corresponding to the three cases in the proof of the theorem.

Conversely, if the algorithm returns $\perp$, let us show by contradiction that there is no maximum valid affectation. Suppose that there is a valid affectation $A[x_1 := m_1, \ldots, x_k := m_k]$. The affectation $A[x_1 := m_1, \ldots, x_c := m_c]$ was tested by the algorithm, but no value has been returned, since line 39 was reached. We deduce that $A[x_1 := m_1, \ldots, x_c := m_c]$ was invalid, which is contradictory since $A[x_1 := m_1, \ldots, x_k := m_k]$ is valid. □

Proposition 6 (Complexity) *We give two estimations of $C(R, A)$, the complexity in the worst case of Algorithm 1, one more precise and one simpler:*

$$C(R, A) = \mathcal{O}\left(\mathrm{rk}(R) \times |A| \times \prod_{x \in \mathcal{J}_{\ll}(R)} |\{m \models x\}|\right) \tag{13}$$

$$C(R, A) = \mathcal{O}\left(|\mathcal{M}(A)|^{\mathcal{S}(R)+\mathrm{rk}(R)}\right) \tag{14}$$

Proof. If the solution is inert, which is the worst case, there are exactly $\prod_{x\in\mathcal{J}_{\preccurlyeq}(R)} |\{m \models x\}|$ 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 first part, where the molecules that do not fit with the choice are removed, each axiom is considered at most once by argument, leading to a complexity $\mathrm{rk}(R) \times |A|$.
- The complexity of the refinement part is defined by the complexity of its two inner loops. On line 18, the field $m.first(p)$ can only grow, so each axiom is checked only once by molecule in its arguments. On Line 22, an axiom can only be removed once by argument.
- The complexity of the maximisation is only $|\mathcal{V}(R)| \ll |A|$.

The complexity of the main loop does not exceed $\mathrm{rk}(R)\times|A|$ and is executed $\prod_{x\in\mathcal{J}_{\preccurlyeq}(R)} |\{m \models x\}|$ times, so the complexity of the whole algorithm is $\mathcal{O}\left(\mathrm{rk}(R) \times |A| \times \prod_{x\in\mathcal{J}_{\preccurlyeq}(R)} |\{m \models x\}|\right)$. We can simplify the writing:

- $|\{m \models x\}| \leq |\mathcal{M}(A)|$, so $\prod_{x\in\mathcal{J}_{\preccurlyeq}(R)} |\{m \models x\}| \leq |\mathcal{M}(A)|^{|\mathcal{J}_{\preccurlyeq}(R)|}$,
- $|A| \leq \begin{pmatrix} |\mathcal{M}(A)| \\ \mathrm{rk}(R) \end{pmatrix} \leq \frac{|\mathcal{M}(A)|^{\mathrm{rk}(R)}}{\mathrm{rk}(R)!}$.

Finally $\mathrm{rk}(R) \times |A| \times \prod_{x\in\mathcal{J}_{\preccurlyeq}(R)} |\{m \models x\}| \leq \frac{1}{(\mathrm{rk}(R)-1)!}|\mathcal{M}(A)|^{\mathcal{S}(R)+\mathrm{rk}(R)}$, and the complexity of the algorithm can be expressed as $\mathcal{O}\left(|\mathcal{M}(A)|^{\mathcal{S}(R)+\mathrm{rk}(R)}\right)$. □

If the order $\preccurlyeq$ defines a minimal juncture, and if we choose R so that Theorem 2 is verified, the worst-case complexity of the algorithm is $\mathcal{O}\left(n^{\mathcal{S}(R)+\mathrm{rk}(R)}\right)$ for n molecules in the multiset. By Theorem 2, $\mathcal{S}(R) + \mathrm{rk}(R) \leq |\mathcal{V}(R)|$. This establishes that the algorithm proposed has a complexity which is either similar or better than $\mathcal{O}(n^k)$. Note also that this gain can be evaluated at compile time (it depends on the rank and the stature of a rule, both obtained by static analysis of the rule, matter of Section 2).

3.3 Parallelisation

Algorithm 1 is composed of a main loop over all the possible affectations of a juncture, in which the affectation is computed and, if it is valid, reactants are exhibited.

The iterations of the main loop are totally independent, *i.e.* the value of one affectation does not influence the value of another one. Hence, they can be done in parallel, which leads to the following corollary about the time complexity of a parallel execution of the algorithm:

Corollary 1 *The reactants searching problem for a rule R and n molecules can be solved by less than $n^{\mathcal{S}(R)}$ tasks in $\mathcal{O}\left(n^{\mathrm{rk}(R)}\right)$ operations each.*

Let us discuss the implementation of a such a parallelisation. Going further, let us notice that wo sub-problems must be addressed in order to make a fully decentralised chemical machine:

Axioms induction: the construction of the set of induced axioms is easy in the case of a sequential algorithm, as it is just necessary to make all the necessary tests between molecules, according to the predicates of the rule. This problem becomes more difficult in a distributed system. On the one hand, it is required to make each test at least once, to make sure that no reaction is forgotten. On the other hand, making a single test more than once might have a high impact on the results, as the time needed to evaluate a predicate only depends on the chemical program itself and is unpredictable. Thus, a good system would evaluate each predicate exactly once.

Affectations parallelisation: in practice, the set of axioms might not be directly copied on the correct number of nodes, each of them knowing of which affectation it has to take care. The set of axioms might even not be storable on a single machine. It is necessary to distribute the choice of the affectations. For the same reasons as above, each affectation must be treated at least once, and should be treated at most once.

4 Related work

We have seen that, for a rule of rank two, the reagents searching problem could be reduced to the subgraph isomorphism problem. This problem has been well documented [4] since the first algorithm proposed by Ullman in 1976 [16], which was based on backtracking. In [12], 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 be made on the graph of the molecules, which constantly change depending of the reactions arising in the computation.

As shown in [11], it is possible to reformulate the subgraph isomorphism problem as a *Constraint Search Problem* (CSP). The distributed version of CSP, disCSP [10], is very close to the reactants searching problem. The disCSP problem is generally treated by an exhaustive exploration of the nodes of the network [6, 19], possibly with an optimisation thanks to backtracking [18].

Naturally, the search for molecules has only been treated through exhaustive search, in particular in distributed settings, as studied in works like [7] in a shared memory architecture, or more recently, in the architectural framework in [14]. The problem of allocating molecules in concurrent settings have been addressed in [3].

5 Conclusion

Concurrent multiset rewriting requires the search for elements satisfying a reaction condition. Finding such *reactants* efficiently is a difficult algorithmic problem, so much that previous studies assumed that it was impossible to solve it in less than $\mathcal{O}(n^k)$ operations, for a solution with n molecules and a rule having k arguments.

In this paper, we showed that, through a static analysis of the reaction conditions, this problem can be solved in a time bounded by $n^{\mathrm{rk}(R)+\mathcal{S}(R)}$, with $\mathrm{rk}(R) + \mathcal{S}(R) \leq k$. For rules whose rank is 2, we were able to characterise the case of equality. We then showed an algorithm for this, and that it is well suited for a decentralised chemical machine, as it can be parallelised.

Apart from the already exposed problems inherent to the decentralisation, many issues remain to be solved in order to have a complete decentralised chemical machine.

The first issue concerns compilation. So far we have only presented the effective search of molecules knowing the graph of a rule. We used logics arguments to express that compilation was indeed possible. It would be interesting to find efficient algorithms, for example, to choose an optimal order of the variables in the rule. Choosing an order for which juncture is minimal allows a choice at compile time, while providing guarantees on the complexity of the algorithm. But the algorithm is more flexible and works regardless of the juncture. The complexity can be further reduced by a finer choice of the order, with respect to the quantity of molecules for each variable.

An other issue is the integration of other aspects of HOCL, including higher order, that changes the behaviour of the program over time, sub-solutions, and of course the ability to have

multiple rules in the same solution. In this last aspect, it is possible to parallelise the search upon the rules.

Finally, let us mention that we could go further in the analysis of the reaction conditions. 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 of set E and $\circ$ is an order or equivalence relation on E. In this case, the search can be improved, for example through a sorting on the values of $f(m)$ and $g(n)$ for the suitable molecules m and n.

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):26–66, 2006.

[2] J.-P. Banâtre, Y. Radenac, and P. Fradet. Chemical specification of autonomic systems. In *IASSE*, pages 72–79. ISCA, 2004.

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

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

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

[6] A. Doniec, S. Piechowiak, and R. Mandiau. A discsp solving algorithm based on sessions. In I. Russell and Z. Markov, editors, *FLAIRS Conference*, pages 666–670. AAAI Press, 2005.

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

[8] S. Grumbach and F. Wang. Netlog, a rule-based language for distributed programming. In M. Carro and R. Peña, editors, *PADL*, volume 5937 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2010.

[9] S. Hariri and M. Parashar. *Handbookf of Bioinspired Algorithms and Applications*, chapter The Foundations of Autonomic Computing. CRC Press LLC, 2005.

[10] B. Ismel and M. Pedro. Synchronous, asynchronous and hybrid algorithms for DisCSP. In M. Wallace, editor, *Principles and Practice of Constraint Programming*. Springer, 2004.

[11] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.*, 12(4):403–422, August 2002.

[12] B. T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical report, Institut fur Informatik und angewandte Mathematik, University of Bern, Neubruckstr. 10, Bern, Switzerland, 1995.

[13] A. Mostéfaoui. Towards a computing model for open distributed systems. In V. E. Malyshkin, editor, *PaCT*, volume 4671 of *Lecture Notes in Computer Science*, pages 74–79. Springer, 2007.

[14] M. Obrovac and C. Tedeschi. On the Feasibility of a Distributed Runtime for the Chemical Programming Model. In *14th International Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2012), in conjunction with IPDPS 2012*, Shanghai, China, May, 21 2012. IEEE. To appear.

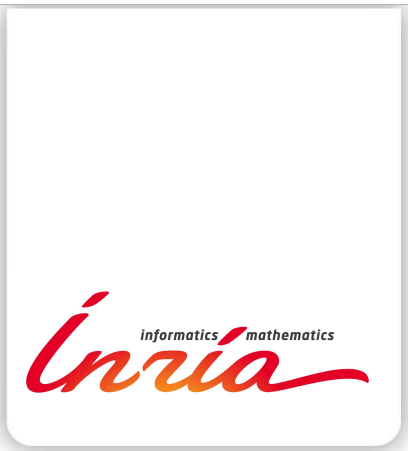
[15] M. Parashar and S. Hariri. Autonomic Computing: An Overview. In *International Workshop on Unconventional Programming Paradigms (UPP 2004)*, volume 3566 of *LNCS*, pages 257–269, Le Mont Saint-Michel, France, 2005. Springer.

[16] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976.

[17] M. Viroli and F. Zambonelli. A Biochemical Approach to Adaptive Service Ecosystems. *Information Sciences*, 2009.

[18] R. Zivan and A. Meisels. Parallel backtrack search on discsp, 2002.

[19] R. Zivan and A. Meisels. Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence*, 46(4):415–439, April 2006.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399