Generalized sorting problems: a symbolic view on reversible algorithms

Ali Akhavi

Generalized sorting problems: a symbolic view on reversible algorithms

Ali Akhavi

GREYC - Université de Caen
Département d'informatique - Campus II - Bd Marchal Juin
F-14032 Caen Cedex - FRANCE
ali.akhavi@unicaen.fr

Abstract

The paper presents a new framework for canonically associating an algorithm with a set of rewrite systems. We identify an algorithm with the set of all its execution traces. The rewrite systems are defined on the sequences of elementary transforms of the algorithm and are such that their normal forms are exactly the execution traces of the algorithm.
The considered algorithms here are those solving a class of problems we define and call *the generalized sorting problems*. It consists in finding a representative for generating sequences of a presented algebra. Under some reasonable hypotheses, the execution traces are a spanning diagram (colored, directed graph) on the Cayley diagram of the automorphism group of the presented algebra and thus also normal terms of rewrite systems.
Then we show that any *reversible*, always halting algorithm is solving a generalized sorting problem. By a reversible algorithm we mean essentially an algorithm that does not loose information during the executions.
Our approach provides a view of all algorithms that solve a given problem with a set of fixed elementary transforms and brings better understanding of the problem: For instance, when any input of the problem has a unique output and the automorphism group of the presented algebra is finite, the diameter of its Cayley graph is a lower bound for the complexity of all the algorithms solving the problem by using the set of fixed elementary transforms.

Keywords: ACM classification: G.4.1. Mathematical software - Algorithm design and analysis I.6.5. Simulation and modeling - Model development - Modeling methodologies.
Algorithm versus rewrite systems, modeling, symbolic and algebraic algorithms, specification of problems, abstract state machines.

1. Introduction

Let us consider the following model for a deterministic sequential algorithm: any algorithm A is given with a set of input data X, a set of output data $O \subset X$, a set of elementary transforms $F \subset X^X$, and a strategy function $\mathcal{S} : F^* \times X \to F$ as described in the following. At each step, the algorithm changes a data $x \in X$ to $f(x) \in X$ for some elementary transform in F. This continues until the modified data x satisfies some output condition: $x \in O \subset X$. An *elementary transform* is

a mapping from X to X, that can be performed by the algorithm in one step[1].

We assume that there is a deterministic strategy $\mathcal{S}$ to determine at each step the elementary transform f to apply to the data. This determination may a priori depend on the history of the execution. If $(f_1, f_2, \ldots, f_{i-1})$ are the elementary transforms used by the algorithm before arriving to the data x, the strategy $\mathcal{S}$ decides[2] the new elementary transform f_i to apply to x:

$$f_i := \mathcal{S}(x, (f_j)_{1 \leq j \leq i-1}).$$

Algorithm $A(X, O, F, \mathcal{S})$:
Input: $x \in X$.
Output: $y \in O$.
Initialization: $\mathbf{i := 1}$
While $x \notin O$ **do**
$f_i := \mathcal{S}(x, (f_j)_{1 \leq j \leq i-1})$.
$x := f_i(x)$
$i := i + 1$
return x

In this paper we are concerned by the case when the algorithm halts, when running on any input. Let call such an algorithm always halting. Considering F as an alphabet, the set F^* of finite words on F is then a monoid containing all finite executions of the algorithm and eventually words corresponding to no execution. Now fix a sequence of transforms $(f_1, f_2, \ldots, f_k) \in F^*$. We address the following question for the always halting algorithms:
Considering F as an alphabet, the set F^* of finite words on F is then a monoid containing all finite executions of the algorithm and eventually words corresponding to no execution. Now fix a sequence of transforms $(f_1, f_2, \ldots, f_k) \in F^*$. We address the following question for the always halting algorithms:

> *(Q) Is the sequence* $(f_1, f_2, \ldots, f_k) \in F^*$ *a possible execution for the algorithm?*
> More precisely, is there $(x, y) \in X \times O$ such that the algorithm A outputs y when running on the input x and following the exact sequence of transforms $(f_1, f_2, \ldots, f_k)$?

Main results:
a) The first aim of our paper is modeling algorithms. First, we introduce a class of problems called *the generalized sorting problems* (Section 2) (Algorithms that are solving a problem of this class happen to be well adapted for the question (Q)). Informally, a generalized sorting problem consists in finding a representative for the different encodings (of a same kind) of a fixed structure. Here a structure is a universal algebra and an encoding of a fixed kind is a generating sequence associated with a fixed *presentation* of the algebra. When running on an arbitrary generating sequence of an algebra (associated with a fixed presentation), an algorithm of our model outputs a particular generating sequence associated with the same presentation, after a finite number of

[1] An elementary transform may be decomposable in terms of other elementary transforms.

[2] Conventionally if the sequence of elementary transforms $(f_1, f_2, \ldots, f_{i-1})$ can never happen during some execution before dealing with the data x as an intermediate data, then the strategy $\mathcal{S}$ considers x as an input, i.e. it does not care about the sequence (the answer of $\mathcal{S}$ is the same than the case when $i = 1$).

steps. During an execution, any intermediate data, which is also a possible input, is always an encoding (of a fixed kind) of the same algebra.
Second we precise how elementary transforms operate on datas. The basic operations available to build an elementary transform are exclusively the fundamental operations of an algebra. An elementary transform is given with a sequence of elements of a *term algebra*. In the very particular case when the algebra is a set (the simplest possible algebra with no fundamental operation), a generating sequence is an enumeration of the elements of the set. Any elementary transform is just a permutation on the members of a generating sequence. Then the (generalized) sorting problem is to find a representative among the generating sequences of a set, for example one whose elements are sorted in an increasing order.
We propose a series of hypotheses that precise the definition of a suitable algorithm for solving a generalized sorting problem. In particular in this paper we deal with algorithms that halt, when running on any input. Notice that since any elementary transform moves an encoding of an algebra to another encoding (of the same kind) of the same algebra, any elementary transform is a permutation on the set of inputs (Fact 4).

b) We associate an algorithm with a set of rewrite systems and characterize (Theorem 1) the set of all the executions of the algorithm with the set of normal words of the associated rewrite systems[3]. The rewrite systems are defined on the sequences of elementary transforms of the algorithm. Under some reasonable hypotheses, the execution traces are a spanning diagram (colored, directed graph) on the Cayley diagram of the automorphism group of the algebra associated with the inputs and thus also normal terms of rewrite systems. If such a sequence is a normal form of one of these rewrite systems then the answer to the question (Q) is *yes*. Otherwise the answer is *no*. We propose a sufficient condition for the number of rewrite systems associated with an algorithm to be finite (Fact 17).

c) We use the seminal work of Gurevich modeling sequential algorithms. We introduce a special class of the so–called Gurevich's Abstract State Machines (ASM) that we call *reversible, always halting ASM*. We show (Theorem 2) that any algorithm that is step–for–step simulated by a reversible, always halting ASM, is in fact resolving a generalized sorting problem.

d) Our approach provides a view of all the reversible, always halting algorithms sharing a set of fixed elementary transforms and brings some highlights about the problem they solve: For instance, when the desired output associated (with any input) is unique and the automorphism group generated by the elementary transform is finite, then the radius of its Cayley graph is a lower bound for the complexity of all the algorithms solving the problem by using the set of fixed elementary transforms. (Corollary 1)

e) Sorting algorithms and Gaussian algorithm (lattice reduction algorithm in two dimensions) as well as Euclidean algorithm are given in the last section of the paper as examples to illustrate our approach.

Context, state of art:
Association of an algorithm with a rewrite system. There are already direct well-known approaches to associate a Turing machine (and so a precised version of an algorithm) with a rewrite system (6). The association is then used to show the indecidability of the problem of termination of some particular rewrite system. Here we propose a new different association. Informally, in the

[3]Considering a rewrite system as an algorithm (once an order is fixed on the set of rewriting rules and also on the place where one rule is first applied in a word), in this paper, an algorithm is associated with another algorithm.

classical approach rewriting is done in the world of datas, while in our framework, rewriting is done in the world of transformations on datas and our aim is to propose an alternative model for a class of algorithms. This paper is a beginning work: here, we are concerned by the number of rewrite systems associated with a fixed algorithm but the rewrite systems are not given explicitly. (The precise rewrite systems associated with some sorting algorithm and with the Gaussian algorithm are available in (1).)

The question (Q)**.** First notice that the question (Q) arises naturally and have been answered in a different context: rather than considering the executions of an algorithm if one considers orbits of a symbolic dynamical system(10), then there are established relationships between the properties of the set (language) of factors of the orbits and the properties of the dynamical system.
Second observe that answering the question (Q) in a general context of an arbitrary algorithm is difficult and the general problem even formulated more precisely seems to be undecidable.
In this paper we decide to answer the question (Q) for the reversible, always halting algorithms.

Relevance of our model of an algorithm.
To evaluate our model let us refer to the seminal work of Gurevich (8) modeling sequential algorithms. Gurevich defines a sequential algorithm to be an object that satisfies three natural postulates: the so-called Sequential Time, Abstract State and Bounded Exploration postulates.
The sequenciality of an algorithm is already expressed in our scheme since the data is modified step by step, according to a discrete time.
Since an algorithm is written down with a fixed finite text, but is applied automatically to many different (and possibly infinitely many) inputs, it is quite natural to assume that the datas manipulated by the algorithm do accept the same kind of elementary manipulations at different steps during different executions. In other words, with respect to the algorithm, the datas at each step are identically structured. Gurevich's Abstract State postulate requires this for an algorithm. This is expressed by our Hypothesis 1: We deal with the algorithms that find a representative for the generating sequences of a fixed presented algebra. So Hypothesis 1 together with the above scheme require datas at each step to be a generating sequence of a fixed presented algebra. In other words, states of an algorithm are the algebras generated by the datas at each step and this algebra which is also the set of accessible elements remains unchanged during any run.
In our framework, Gurevich's Bounded Exploration postulate is expressed with Hypothesis 4 (which states the bounded changeness of the data at each step) and Hypothesis 6 (which states the bounded exploreness of the stratetgy at each step)[4].

Outline of our method. We begin by specifying formally (Section 2) a generalized sorting problem (Definition 5). Many classical definitions about universal algebras are recalled, but are adapted to our needs. Then Section 3 describes some useful objects and some important hypotheses about the algorithm. In our framework elementary transforms are permutations on the set of inputs (or equivalently the set of intermediate datas or the set of *states*(8))) of an algorithm. We define the execution graph and the effective execution graph of the algorithm. The former is the graph of all possible trajectories of elementary transforms among the inputs (which are also the intermediate datas), while the latter is the graph of the effective trajectories taken by the algorithm, i.e. the execution traces. The vertices of the execution graph are inputs and there is an edge from x_1 to x_2 colored by an elementary transform f iff $f(x_1) = x_2$. The vertices of the effective execution graph are the inputs as well, but there is edge from x_1 to x_2 colored by an elementary

[4]The theorem associating an algorithm with a set of rewrite systems (Theorem 1) is expressed without requiring an algorithm to satisfy bounded exploreness, but requiring it to be bounded-change.

transform f iff x_1 is moved to x_2 by the elementary transformation f, in some run of the algorithm. The effective execution graph is a spanning graph of the execution graph. We identify vertices of any connected component of the execution graph with generating sequences of a fixed presented algebra. In this paper, we work with algorithms that are acting on fixed sized datas. In this context, we require all the connected components of the execution graph to be isomorphic and we identify any such connected component with a Cayley graph of the automorphism group of a fixed algebra (Section 4, Lemma 1). So the automorphisms group is identified with a factor of the monoid of finite words on the alphabet of elementary transforms.
Section 5 defines a *strategy-preserving isomorphism* and assemble different connected components up to strategy-preserving isomorphisms. We define the notion of *class of inputs* as equivalence class of connected components up to strategy-preserving isomorphisms. We show that there is a rewrite system $\mathcal{R}$ over the automorphism group of the presented algebra, satisfying the following property: the executions of the algorithm running on all inputs inside one class (vertices of connected components inside a class) are exactly normal forms of $\mathcal{R}$. Informally speaking, the behavior of the algorithm on a class of inputs is governed by one single mechanism.
Then using Gurevich's ASM framework, we consider in Section 6 an algorithm as an object that is step by step simulated by a so-called Abstract State Machine (ASM). We define reversible ASM's to be those that do not loose information during an execution. We show that any reversible, always halting algorithm is solving a generalized sorting problem.
Several examples are provided in the last section.

2. A specification of the problem

In this section, we recall basic and classical notions in universal algebras and define the notion of presented algebra. The advertised reader should skip the first subsection. Then we define the problem of finding a representative for generating sequences of a presented algebra. For a detailed introduction to universal algebras see for example (4), (2).

2.1. Preliminaries, universal algebras

A *language (or signature or type or vocabulary) of algebras* is a finite collection of function symbols, each of a fixed arity.
An *algebra* $\mathbf{A}$ of type Ψ is a nonempty set A (base set or universe of $\mathbf{A}$) together with interpretations of the function names in Ψ over A. The elements of the algebra are elements of the universe. A j-ary function name is interpreted as a function from A^j to A , which is a j-ary operation on A and f's are called the fundamental operations of $\mathbf{A}$.
Let $\mathbf{A}$ and $\mathbf{B}$ be two algebras of the same type. Then a function $\alpha : A \to B$ is called a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ if for every j-ary function symbol $f \in \Psi$, for $a_1, \ldots, a_j \in A$, we have $\alpha(f(a_1, \ldots, a_j)) = f(\alpha(a_1), \ldots, \alpha(a_j))$. If α is one-to-one and onto, then α is called an *isomorphism* from $\mathbf{A}$ to $\mathbf{B}$. Moreover if $\mathbf{A} = \mathbf{B}$ then an isomorphism is also called an *automorphism* on $\mathbf{A}$.
For a sequence $X = (x_i)_{i \in I}$ of elements of an algebra $\mathbf{A}$, we will use the classical notion of *algebra generated by* X as the intersection of all subalgebras of $\mathbf{A}$ containing all the elements of the sequence X. If the intersection is the whole universe A, then X is a *generating sequence* for A.
Given a nonempty set A and a positive integer j, a j-ary relation R on A is a subset of A^j. Equivalently a j-ary relation $\mathcal{R}$ is identified with its indicator function from A^j to $\{\texttt{true}, \texttt{flase}\}$. For a binary relation θ we also denote $a_1 \, \theta \, a_2$ for $\theta(a_1, a_2)$. A binary relation on A is an equivalence

relation if it is reflexive, symmetric and transitive. When m relations $R_1, \ldots, R_m$ are defined on the the universe of an algebra $\mathbf{A} = (A, \Psi)$, and we want to stress the existence of these relations, then we denote the algebra $\mathbf{A} = (A, \Psi, R_1, \ldots, R_m)$. In this case we call the *extended vocabulary* of the algebra the union of the set of function symbols Ψ and the set of relation symbols of the algebra[5].
Let $\mathbf{A}$ be an algebra of type Ψ and let θ be an equivalence relation defined on A. Then θ is a congruence on $\mathbf{A}$ if for each n-ary function symbol $f \in \Psi$ and elements $a_i, b_i \in A$, if $a_i\ \theta\ b_i$ holds for $1 \leq i \leq n$ then $f(a_1, \ldots a_n)\ \theta\ f(b_1, \ldots, b_n)$ holds. Any congruence θ introduces an algebraic structure on the set of equivalence classes A/θ, which is inherited from the algebra $\mathbf{A}$.
Let $S \subset A^2$. Define R as the smallest subset of A^2 containing S and such that $(i)(s, s) \in R, \forall s \in R$, $(ii)(s, t) \in R \Leftrightarrow (t, s) \in R$, $(iii)(s, t) \in R$ and $(t, v) \in R \Rightarrow (s, v) \in R$, and (iv) for all n-ary operation $f \in \Psi$, if $(s_i, t_i) \in R$ for $1 \leq i \leq n$, then $(f(s_1, \ldots s_n), f(t_1, \ldots, t_n)) \in R$. The relation θ defined on $\mathbf{A}$ by $(s\ \theta\ t \Leftrightarrow (s, t) \in R)$ is clearly a congruence on $\mathbf{A}$ called the *congruence generated by* S.
Let X be a set of objects called variables. *Terms* of type Ψ are defined classically by induction: A variable or a nullary function name is a term. If f is a function name of positive arity j and $t_1, \ldots, t_j$ are terms, then "the string" $f(t_1, \ldots, t_j)$ is a term. A ground term is a term that contains no variable. Such a term exists if there is at least one nullary function symbol in the vocabulary, which is always the case in this paper. The value $val(t, \mathbf{A})$ of a ground term t of type Ψ in an algebra $\mathbf{A}$ of type Ψ is given inductively as usual (A nullary function is identified with its value). If $val(t, \mathbf{A}) = val(t', \mathbf{A})$, we may say that $t = t'$ in $\mathbf{A}$.
The *term algebra* of type Ψ over X, written $\mathbf{T}(X)$, has as universe the set $T(X)$, and the fundamental operations satisfy

$$val\Big(\ f(t_1, \ldots, t_j)\ ,\ \mathbf{T}(X)\ \Big) = f(t_1, \ldots, t_j) \text{ for } f \in \Psi \text{ of arity } j \text{ and } t_i \in T(X), 1 \leq i \leq j.$$

If an algebra has an extended vocabulary, then the outermost symbol name of a term may be a relation symbol. Such a term is then called a *relational term* and its value is either true or false. Depending on the context, we may write both $val(R(a_1, \ldots, a_j), \mathbf{A}) = \texttt{true}$ (or $R(a_1, \ldots, a_j)$ holds in $\mathbf{A}$) for $(a_1, \ldots, a_j) \in R$; similarly $val(R(a_1, \ldots, a_j), \mathbf{A}) = \texttt{false}$ (or simply $(R(a_1, \ldots, a_j))$ does not hlod in $\mathbf{A}$) for $(a_1, \ldots, a_j) \notin R$.
Given a term $p \in T(X)$ of type Ψ where $X = \{x_1, \ldots, x_n\}$, and a algebra $\mathbf{A}$ of type Ψ, p defines (analogously to polynomials) an application from $\mathbf{A}^n$ to $\mathbf{A}$. The image of $(a_1, \ldots, a_n)$ is simply denoted $val\Big(\ p(a_1, \ldots, a_n), \mathbf{A}\Big)$, and is computed by strait forward induction.
An identity of type Ψ over $X = \{x_1, \ldots, x_n\}$ is an expression of the form $p \approx q$ where $p, q \in T(X)$. An algebra $\mathbf{A}$ of type Ψ satisfies an identity, if for every choice of $(a_1, \ldots, a_n) \in \mathbf{A}^n$, we have $p(a_1, \ldots, a_n) = q(a_1, \ldots, a_n)$. A class $\mathcal{K}$ of algebras satisfies a set of identities Σ, if each member of $\mathcal{K}$ satisfies each identity of Σ.
Let Σ be a set of identities of type Ψ and $M(\Sigma)$ be the class of all algebras of type Ψ satisfying Σ. A class $\mathcal{K}$ of algebras of type Ψ is a *variety* iff there is a set of identities Σ such that $\mathcal{K} = M(\Sigma)$. In this case the variety $\mathcal{K}$ is *axiomatized* by Σ.
Let us cite two very usual examples. The variety of groups is the class of all algebras with the vocabulary $\{.\ , ^{-1}, 1\}$ where . is a binary operation, $^{-1}$ is a unary operation and 1 is a nullary operation and satisfying the following identities: $(i)\quad x.(y.z) \approx (x.y).z$, $(ii)\quad x.1 \approx 1.x \approx x$ and

[5] In the mathematical logic context, a vocabulary contains function symbols as well as relation symbols. Then the analog of an algebra is a structure. However we find it more convenient to use the vocabulary of the universal algebra rather than the one of the mathematical logic, since we will extensively manipulate isomorphic algebras, variety of algebras and the presentation of an algebra.

(*iii*) $x.x^{-1} \approx x^{-1}.x \approx 1$. The variety of Abelian groups is the class of algebras of the same type satisfying the previous identities, and also $x.y \approx y.x$.
Another example is the variety of sets, where both the vocabulary and the set of identities satisfied by the variety are empty.

2.2. A presented algebra

Let $\mathcal{V}$ be a variety of algebras of type Ψ containing a nontrivial algebra[6]. Given a set X of variables, the congruence $\theta_{\mathcal{V}}(X)$ on $\mathbf{T}(X)$ is defined by $\theta_{\mathcal{V}}(X) = \bigcap \phi$, where ϕ is any congruence on $\mathbf{T}(X)$ such that $\mathbf{T}(X)/\phi$ is isomorphic to an algebra in $\mathcal{V}$. The *$\mathcal{V}$-free algebra over X* is then defined[7] by $\mathbf{F}_{\mathcal{V}}(X) = \mathbf{T}(X)/\theta_{\mathcal{V}}(X)$.
The $\mathcal{V}$-free algebra over X is generated by X and has the universal mapping property (for the variety $\mathcal{V}$ of algebras of type Ψ over X): Let $\mathbf{A}$ be an algebra of $\mathcal{V}$. For every map $\alpha : X \to \mathbf{A}$, there is a homomorphism $\hat{\alpha} : \mathbf{F}_{\mathcal{V}}(X) \to \mathbf{A}$ which extends α. If $\mathcal{V}$ is the variety of all algebras of a given type Ψ, then $\mathbf{F}_{\mathcal{V}}(X)$ is identified with $\mathbf{T}(X)$.
Let $\mathbf{A}$ be an algebra of a variety $\mathcal{V}$ of type Ψ and X be a set. Let $(x_i)_{i\in X}$ be a sequence (or family) of elements of $\mathbf{A}$. Define the mapping $\xi : X \to \mathbf{A}$ by $\xi(i) = x_i$, for all $i \in X$. Let $\phi : \mathbf{F}_{\mathcal{V}}(X) \to \mathbf{A}$, be the unique homomorphism that extends ξ.

- The sequence $(x_i)_{i\in X}$ generates $\mathbf{A}$ iff ϕ is onto, which depends only on the type Ψ and not on the variety $\mathcal{V}$.

  The algebra $\mathbf{A}$ is called *finitely generated* iff there is a finite sequence that generates $\mathbf{A}$. In this paper we always work with finitely generated algebras.

- The sequence is *independant* iff ϕ is one to one.

- The sequence is called *a basis* iff ϕ is an isomorphism. If there exists a family $(x_i)_{i\in X}$ in $\mathbf{A}$, which is a basis, then $\mathbf{A}$ is a free algebra in the variety $\mathcal{V}$.

Definition 1. *Let $\mathcal{V}$ be a variety of algebras. A* presentation *(relatively to $\mathcal{V}$) is a pair (n, S) where n is a positive integer and S a binary relation on the $\mathcal{V}$-free algebra $\mathbf{F}_{\mathcal{V}}(X)$, over a fixed set*[8] *$X = \{x_1, \ldots, x_n\}$. In the sequel a presentation will be denoted by (X, S) and $n = |X|$ is called the* rank *of the presentation.*

Let $\mathbf{A}$ be an algebra of $\mathcal{V}$. The presentation (X, S) is a presentation of $\mathbf{A}$ (relatively to $\mathcal{V}$) *or $\mathbf{A}$ is* presented by *(X, S) or $\mathbf{A}$ is an (X, S)*-presented algebra of the variety *$\mathcal{V}$ iff $\mathbf{A}$ is isomorphic to*

$$\mathbf{F}_{\mathcal{V},S}(X) = \mathbf{F}_{\mathcal{V}}(X)/\theta_S, \quad \textit{where}$$

θ_S is the congruence generated by S in $\mathbf{F}_{\mathcal{V}}(X)$. Let $\mathbf{A}$ be an algebra of the variety. Equivalently, $\mathbf{A}$ is (X, S)-presented iff there is a mapping $\xi : X \to A$ such that $(\xi(i))_{i\in X}$ is a generating sequence for $\mathbf{A}$ and the unique homomorphism $\phi : \mathbf{F}_{\mathcal{V}}(X) \to \mathbf{A}$ extending ξ to $\mathbf{F}_{\mathcal{V}}(X)$ satifises for all $t_1, t_2 \in F_{\mathcal{V}}(X)$, $\phi(t_1) = \phi(t_2) \iff (t_1, t_2) \in \theta_S$. The generating sequence $(\xi(i))_{i\in X}$ is then called a

[6] If a family $\mathcal{K}$ contains only a trivial algebra $\mathbf{A}$ and $|X| \geq 2$, then two distinct members x, y of X cannot be separated by any homomorphism $\alpha : \mathbf{T}(X) \to \mathbf{A}$. So in the general case, the *$\mathcal{K}$-free algebra over X* is defined by $\mathbf{F}_{\mathcal{K}}(\hat{X}) = \mathbf{T}(X)/\theta_{\mathcal{K}}(X)$, where $\hat{X} = X/\theta_{\mathcal{K}}(X)$. In the sequel a family of algebras contain always a nontrivial algebra. So $\hat{X}$ is identified with X.

[7] $\mathbf{F}_{\mathcal{V}}(X)$ exists iff $X \neq \emptyset$ or Ψ contains nullary function symbols.

[8] Changing X by another set Y of the same cardinal n is just renaming the variables and does not change the presentation.

covering *of the algebra* **A** *with respect to the presentation* (X, S) *or an* (X, S)-covering *or simply a covering of* **A** *when there is no ambiguity.*
A presented algebra *of a variety is a pair composed by an algebra* **A** *of the variety and a fixed presentation* (X, S) *such that* (X, S) *is a presentation of* **A**. *The* rank *of the presented algebra is the cardinality of X, which is always a fixed integer in this paper. The presentation is then finite and the presented algebra is finitely presented.*

A very common example of presentation and presented algebra concerns the variety of groups. They have been extensively studied (see (11) for example). Notice that the same algebra with two different presentations corresponds to two distinct presented algebras. Consider in the variety of groups, the group Σ_3 of permutations of three objects. It is presented by $(\{x_1, x_2\}, S_1 = \{(x_1^3, 1), (x_2^2, 1), (xy, yx^2)\})$ but also by $(\{x_1, x_2\}, S_2 = \{(x_1^2, 1), (x_2^2, 1), ((xy)^3, 1)\})$. $(\Sigma_3, (X, S_1))$ and $(\Sigma_3, (X, S_2))$ are two distinct presented algebras of the variety of groups.
Notice also that there is not a trivial notion of rank for a general algebra, while we defined the rank of a (X, S)-presented algebra to be $n = |X|$.
Given a variety $\mathcal{V}$ and a presentation (X, S) relatively to $\mathcal{V}$, the trivial algebra may be the unique algebra of the variety with the presentation (X, S). On the other hand, the algebra presented by $(X, \emptyset)$ is isomorphic to $\mathbf{F}_{\mathcal{V}}(X)$, the $\mathcal{V}$-free algebra over X.
Two algebras of type Ψ of the same variety $\mathcal{V}$ and with the same presentation (X, S) are obviously isomorphic, since they are both isomorphic to $\mathbf{F}_{\mathcal{V},S}(X)$.

2.3. Automorphism group of the presented algebra

Here we precise the notion of *automorphism group of the presented algebra* which is central in our approach. Indeed we will identify elementary transforms of an algorithm with automorphisms of a presented algebra and require the elementary transforms to generate all this group.

Notation 1. *Let* $\mathcal{V}$ *be a variety of type* Ψ *and* (X, S) *a presentation. Let* $\mathcal{I}$ *be a presented algebra. The set of all* (X, S)*-coverings of* $\mathcal{I}$ *is denoted by* $\tilde{\mathcal{I}}$. *Any application* ϕ *between two presented algebras* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *that conserves* (X, S)*-coverings (any* (X, S)*-covering of* $\mathcal{I}_1$ *is associated with a* (X, S)*-covering of* $\mathcal{I}_2$*) is extended naturally to an application between* $\tilde{\mathcal{I}}_1$ *and* $\tilde{\mathcal{I}}_2$*, denoted by* $\tilde{\phi}$.

Fact 1. *Let* (X, S) *be a presentation relatively to a variety* $\mathcal{V}$ *of type* Ψ. *Let* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *be two algebras of* $\mathcal{V}$ *with presentation* (X, S). *An application* ϕ *between* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *is* an isomorphism *iff it is an homomorphism and the image of a* (X, S)*-covering* c_1 *of* $\mathcal{I}_1$ *is a* (X, S)*-covering* c_2 *of* $\mathcal{I}_2$. *Moreover* ϕ *is completely defined by a pair* (c_1, c_2) *where* c_1 *is a* (X, S)*-covering of* $\mathcal{I}_1$, c_2 *is a* (X, S)*-covering of* $\mathcal{I}_2$ *and* $c_2 = \tilde{\phi}(c_1)$. *The image of any element x of* $\mathcal{I}_1$ *is obtained by first expressing x according to* c_1 *and then replacing* c_1 *by* c_2 *in this expression.*

Proof. The previous characterization of an isomorphism between two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ defines non-ambiguously a mapping between $\mathcal{I}_1$ and $\mathcal{I}_2$: if two terms constructed with a covering of $\mathcal{I}_1$ are equal, then they are equal as well, when the covering of $\mathcal{I}_1$ is replaced by a covering of $\mathcal{I}_2$, since c_1 and c_2 are two coverings relatively to the same presentation (X, S).
A homomorphism ϕ transforming a covering c_1 of $\mathcal{I}_1$ into a covering c_2 of $\mathcal{I}_2$, i.e. a homomorphism ϕ given by the pair (c_1, c_2) is indeed a bijective application: the reciprocal isomorphism ϕ^{-1} is simply given by the pair (c_2, c_1).
Let ϕ be an isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$, c_1 a covering of $\mathcal{I}_1$ relatively to (X, S) and $c_2 = \tilde{\phi}(c_1)$. Clearly c_2 is a generating sequence for $\mathcal{I}_2$: Since ϕ is onto, any element x of $\mathcal{I}_2$ has a preimage in $\mathcal{I}_1$ that is expressed as $t(c_1)$, a term on the covering c_1. Since ϕ is a morphism

$x = \phi(t(c_1)) = t(\tilde{\phi}(c1)) = t(c_2)$. Now let $t(c_1)$ and $t'(c_1)$ be two terms on c_1. Since ϕ is a one to one morphism, $t(c_1) = t'(c_1)$ iff $t(c_2) = t'(c_2)$, which shows that c_2 is a covering relatively to the presentation (X, S). □

Definition 2. *The set of all automorphisms of a presented algebra with the composition of applications has a group structure. Since any algebra of type Ψ of a variety $\mathcal{V}$ and with the presentation (X, S) is isomorphic to $\mathbf{F}_{\mathcal{V},S}(X)$, the automorphism groups of algebras with the presentation (X, S) are all isomorphic to the automorphism group of $\mathbf{F}_{\mathcal{V},S}(X)$, called* the automorphism group of the presented algebra *and denoted* $\mathbf{Aut}(\mathbf{F}_{\mathcal{V},S}(X))$ *or* $\mathbf{Aut}(\Psi, \mathcal{V}, n, S)$ *where $n = |X|$. If one (X, S)-covering of a presented algebra is fixed then this group is naturally identified with the set of (X, S)-coverings of the presented algebra.*

Let $\mathcal{I}$ be an algebra of the variety $\mathcal{V}$ and let (X, S) be a presentation for $\mathcal{I}$. Another natural way to see the previous assertion is the following. $\mathbf{Aut}(\mathbf{F}_{\mathcal{V},S}(X))$ acts freely and transitively (i.e. regularly) on the set of coverings $\tilde{\mathcal{I}}$: Once one (X, S)-covering c^* of the presented algebra is fixed then any element $f \in \mathbf{Aut}(\mathbf{F}_{\mathcal{V},S}(X))$ is naturally identified with the automorphism $(c^*, \tilde{f}(c^*))$. So by the classical orbit stabilizer theorem, there is bijective mapping between $\mathbf{Aut}(\mathbf{F}_{\mathcal{V},S}(X))$ and $\tilde{\mathcal{I}}$, namely the one that associates f to $\tilde{f}(c^*)$.

Remark 1. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two algebras of a variety with the same presentation and ϕ an isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. Then ϕ defines a group isomorphism between the automorphism groups of $\mathcal{I}_1$ and $\mathcal{I}_2$: The application that associates to an automorphism f of $\mathcal{I}_1$ defined by two generating sequences (c_1, c_2) the automorphism, say $\phi(f)$, of $\mathcal{I}_2$ defined by the two generating sequences $(\tilde{\phi}(c_1), \tilde{\phi}(c_2))$ is clearly a group isomorphism. Moreover for all $(c_1, c_2) \in \tilde{\mathcal{I}}_1^2$, and for all ω automorphism of $\mathcal{I}_1$, $c_1 = \tilde{\omega}(c_2) \Longrightarrow \tilde{\phi}(c_1) = \phi(\tilde{\omega})(\tilde{\phi}(c_2))$.*
Indeed $c_1 = \tilde{\omega}(c_2)$ means that ω may be defined by the pair of generating sequences (c_1, c_2). So the remark is just a consequence of the definition of $\phi(f)$.
Observe that from any function f (or permutation) on $\mathcal{I}_1$, one defines analogously $\phi(f)$.

Remark 2. *Let (c, c') and (d, d') be two pairs of coverings of a presented algebra. They represent the same automorphism if and only if the terms expressing $d'_1, \ldots, d'_n$ according to $c'_1, \ldots, c'_n$ are equivalent to those representing $d_1, \ldots, d_n$ according to $c_1, \ldots, c_n$.*

There are commonly other (somehow more constructive) ways to represent an automorphism: An automorphism $((c_1, \ldots, c_n), (c'_1, \ldots, c'_n))$ may be defined by the initial covering $c_1, \ldots, c_n$ and the way $c'_1, \ldots, c'_n$ are expressed in terms of $c_1, \ldots, c_n$, i.e. each c'_i given by an element W_i of the term algebra $T(x)$, such that $c'_i = W_i(c_1, \ldots, c_n)$. For example, if the presented algebra is a set of cardinality n, presented with an enumeration of its elements, then $c'_i = c_{\sigma(i)}$, where σ is a permutation of $\{1, \ldots, n\}$. If there is just one associative fundamental operation in the presented algebra then $c'_1, \ldots, c'_n$ are words $W_1(c_1, \ldots, c_n), \ldots, W_n(c_1, \ldots, c_n)$. An automorphism of the presented algebra is then defined by an initial generating sequence together with the words $W_1, \ldots, W_n$. Alternatively rather than fixing the initial (preimage) covering[9] one may fix the final (image) covering $c'_1, \ldots, c'_n$. Then an automorphism f is given by $c'_1, \ldots, c'_n$ together with the words $W'_1, \ldots, W'_n$ such that $f((W'_i(c'_1, \ldots, c'_n))) = c'_i$. Notice once more that when a covering is fixed (no matter it is the preimage or the image) then any automorphism is given by a sequence of terms on this covering.

[9] One may also fixe an initial generating sequence c and a final one c' and give the words expressing images of the components of c in c'.

Fact 2. *Let $x = (x_1, \dots x_n)$ be a a fixed covering of an (X, S)-presented algebra A. Any automorphism f of the (X, S)-presented algebra is then uniquely labeled by a sequence of terms $(T_1, \dots, T_n)$ of $\mathbf{F}_{\mathcal{V},S}(X)$. The set of labels of all the automorphisms does not depend on the initially chosen (X, S)-covering x.*

Proof. The automorphism f is defined by $(x, (T_i(x)))$ and so labeled by $(T_1, \dots, T_n)$. Obviously two sequences of terms $T_1, \dots, T_n$ and $T'_1, \dots, T'_n$ satisfy $T_i = T'_i$ in $\mathbf{F}_{\mathcal{V},S}(X)$ for all $i \in \{1, \dots, n\}$, if and only if they label the same morphism. We will now prove that for any (X, S)-covering $c = (c_1, \dots, c_n)$, $(T_1(c), \dots, T_n(c))$ is an (X, S)-covering. (So fixing initially another (X, S)-covering c rather than x does not change the set of labels whereas it does change the label of a given automorphism). Let $z \in A$. Let y be the preimage of z by the automorphism defined by (x, c). y is expressed as a term on $(T_i(x))$. So z is a term on $(T_i(c))$, i.e. any element of A is expressed as a term on $(T_i(c))$. Moreover any equality relation in A is expressed as $\Omega((T_i(x))) = \Omega'((T_i(x)))$. Since c is also an (X, S)-covering, one has $\Omega((T_i(c))) = \Omega'((T_i(c)))$ as well. □

Remark 3. *Let $x = (x_1, \dots x_n)$ be a a fixed initial covering. If the automorphisms are labelled by elements of the term algebra $\mathbf{T}(X)$ rather than elements of $\mathbf{F}_{\mathcal{V},S}(X)$, then a given automorphism may have several different labels but of course a given n-tuple of elements of $\mathbf{T}(X)$ is the label of at most one automorphism.*

Fact 3 whose proof is straight forward exhibits an advantage of representing an automorphism as sequence of terms on a covering.

Fact 3. *Let $\omega = ((c_1^*, \dots, c_n^*)\ ,\ (c_1, \dots, c_n))$ and*

$$f = ((c_1^*, \dots, c_n^*)\ ,\ (W_1(c_1^*, \dots, c_n^*), \dots, W_n(c_1^*, \dots, c_n^*)))$$

be two automorphisms of a presented algebra. Then the automorphism $\omega \circ f$ is represented by $((c_1^*, \dots, c_n^*)\ ,\ (W_1(c_1, \dots, c_n)), \dots, W_n(c_1, \dots, c_n)))$.

Fact 4. *Let $\mathcal{I}$ be a finitely generated algebra of a variety $\mathcal{V}$ of type Ψ with presentation (X, S) where $n = |X|$. Let $\tilde{\mathcal{I}}$ be the set of coverings of $\mathcal{I}$ relatively to (X, S). Let $W_1, \dots, W_n$ be n elements of the term algebra $\mathbf{T}(X)$ and $(c_1^*, \dots, c_n^*) \in \tilde{\mathcal{I}}$ such that $(W_1(c_1^*, \dots, c_n^*), \dots, W_n(c_1^*, \dots, c_n^*)) \in \tilde{\mathcal{I}}$. The application*

$$f : \left\{ \begin{array}{rcl} \tilde{\mathcal{I}} & \to & \tilde{\mathcal{I}} \\ (x_1, \dots, x_n) & \mapsto & (W_1(x_1, \dots, x_n)), \dots, W_n(x_1, \dots, x_n)) \end{array} \right. \tag{1}$$

is a permutation on $\tilde{\mathcal{I}}$.

Proof. Assume there are three coverings $(a_1, \dots, a_n)$, $(b_1, \dots, b_n)$ and $(c_1, \dots, c_n)$ such that

$$f(a_1, \dots, a_n) = f(b_1, \dots, b_n) = (c_1, \dots, c_n).$$

So $W_i(a_1, \dots, a_n) = W_i(b_1, \dots, b_n)$, for all $i \in \{1, \dots n\}$. Since $(c_1, \dots, c_n)$ is a covering of $\mathcal{I}$ any element of the covering $(a_1, \dots, a_n)$ is expressed as terms on (c_i): for all $i \in \{1, \dots n\}$, $a_i := \Phi_i(c_1, \dots, c_n)$. So

$$a_i = \Phi_i(W_1(a_1, \dots, a_n), \dots, W_n(a_1, \dots, a_n)).$$

But f transforms a covering of a presented algebra into another covering of the presented algebra and any equality relation on a covering is also an equality relation on any other covering sequence. In particular,

$$b_i = \Phi_i(W_1(b_1, \dots, b_n), \dots, W_n(b_1, \dots, b_n)).$$

So $a_i = \Phi_i(c_1, \dots c_n)$ and $b_i = \Phi_i(c_1, \dots c_n)$, $\forall i \in \{1, \dots n\}$, which leads to $a_i = b_i$.
Moreover from the definition of the terms $(\Phi_1, \dots, \Phi_n)$ one deduces that $c_i := W_i(a_1, \dots a_n) = W_i(\Phi_1(c_1, \dots c_n), \dots, \Phi_n(c_1, \dots c_n))$. Once more notice than any equality relation satisfied by one covering of the instance is satisfied by all the coverings of the instance since we deal with presented algebras. So any covering $(x_1, \dots x_n)$ satisfies :

$$x_i = W_i(\Phi_1(x_1, \dots, x_n), \dots \Phi_n(x_1, \dots, x_n)), \quad \forall i \in \{1, \dots n\},$$

which means that $(\Phi_1(x_1, \dots, x_n), \dots, \Phi_n(x_1, \dots, x_n))$ is the preimage of the generating sequence $(x_1, \dots, x_n)$.

□

Remark 4. (*i*) *Let $(c_1^*, \dots, c_n^*)$ be a fixed (X, S)-covering in $\tilde{\mathcal{I}}$. Then any other (X, S)-covering $(x_1, \dots, x_n)$ is expressed as terms in $(c_1^*, \dots, c_n^*)$. When the algebra is a group and $(X, S) = (X, \emptyset)$, the transformation f defined in (1) is called* a Nielsen transformation[10] *in (11). There is a bijective association between Nielsen transformations and automorphisms of a free group, once a covering $(c_1^*, \dots, c_n^*)$ is fixed. So when the algebra is a group and $(X, S) = (X, \emptyset)$, the previous fact just claims that any Nielsen transformation defines a permutation on the set of generating sequences of a free group.*
(*ii*)*The restriction of f to any closed subset of $\tilde{\mathcal{I}}$ under f is obviously a permutation on this subset.*
(*iii*) *The previous fact is true even if if there are additional equalities in the presented algebra between terms that are not consequences of the identities of the variety nor consequences of equalities in S. Indeed, the last proof does not use these additional equalities.*

2.4. A domain, an instance, the problem

Let $\mathcal{V}$ be a variety of algebras of type Ψ, (X, S) be a fixed presentation relatively to the variety $\mathcal{V}$ of rank $n = |X|$. Let $\mathbf{D} = (D, \Psi)$ be a fixed algebra of the variety $\mathcal{V}$ that we call the *domain*. We call indifferently domain the algebra $\mathbf{D}$ and its universe D.
Finally let us assume that there are $m \geq 0$ additional relations $\mathcal{R}_1, \dots \mathcal{R}_m$ of finite arities defined on D. For example, $m = 1$, R_1 is a binary relation and the domain D is ordered or preordered. We recall that $\Psi \cup \{\mathcal{R}_1, \dots \mathcal{R}_m\}$ is then called the extended vocabulary of $\mathbf{D}$.

Definition 3. *A valid input (or an input) is an n-tuple $x = (x_1, \dots, x_n) \in D^n$, satisfying all equality relations in S. The set of all valid inputs is called the* input set.

As precised by Fact 2 when a (X, S)-covering $x = (x_1, \dots, x_n)$ of $\mathbf{F}_{\mathcal{V},S}(X)$ is fixed then any element of the automorphism group of $\mathbf{F}_{\mathcal{V},S}(X)$ is identified with a sequence of terms $(T_1, \dots, T_n)$ in $\mathbf{F}_{\mathcal{V},S}(X)$. Any such sequence of terms is not only associated with an automorphism but also with a transformation of n-tuples of elements of $\mathbf{F}_{\mathcal{V},S}(X)$.
For example, if the algebra is a group, and $|X| = n = 2$, and $S = \emptyset$, then $(x_1, x_2) \to (x_1 x_2, x_2)$ defines an automorphism of the free group on two generators. It also defines a transformation

[10] It is associated with a free substitution.

on pairs of words on (x_1, x_2): The pair $(W_1(x_1, x_2), W_2(x_1, x_2))$ is transformed into the pair $(W_1(x_1, x_2)W_2(x_1, x_2), W_2(x_1, x_2))$.
Hence the automorphism group $\mathbf{Aut}(\Psi, \mathcal{V}, n, S)$ of the (X, S)-presented algebra, acts naturally on the input set: Once an arbitrary initial covering in $\mathbf{F}_{\mathcal{V},S}(X)$ is fixed, any automorphism is identified with a sequence of terms which is identified with a transformation on the input set. We call this action *the natural action* on the input set.

Definition 4. *Orbits of the the natural action of* $\mathbf{Aut}(\Psi, \mathcal{V}, n, S)$ *on the input set are called* instances of the problem.

Fact 5. *Two elements inside one instance of the problem generate the same subalgebra of the domain* **D**.

Proof. As described by Fact 2, any automorphism ϕ of $\mathbf{F}_{\mathcal{V},S}(X)$ can be expressed by a sequence of terms $(T_1, \ldots, T_n)$ of $\mathbf{F}_{\mathcal{V},S}(X)$ that fixes how ϕ acts on a sequence $x = (x_1, \ldots, x_n)$ of n elements of the domain. (Fact 2 shows that if x generates an (X, S)-presented algebra, then any other element in the orbit of x is an (X, S)-covering of the same algebra similarly to x.) The image of x by ϕ is $y := (T_1(x), \ldots, T_n(x))$. So y generates obviously a subalgebra of the algebra generated by x. Now since ϕ is an automorphism of $\mathbf{F}_{\mathcal{V},S}(X)$, there are terms $(T'_1, \ldots, T'_n)$ of $\mathbf{F}_{\mathcal{V},S}(X)$ such that $x_1 := T'_1(y), \ldots, x_n := T'_n(y)$. Thus the algebra generated by x is a subalgebra of the one generated by y. □

Consider a valid input $x \in D^n$. Consider the subalgebra A generated by x. The equality between pairs of terms in A may be just consequences of the identities of the variety or the equalities in S. But it may also exist additional equalities between terms in A. In all cases, the input x is formally considered by the action as an (X, S)-covering by identifying x with X. Any element in A is accessed as an element of the term algebra $\mathbf{F}_{\mathcal{V},S}(X)$. Even if additional equalities are satisfied in A, they are not used by the action. For example if in a valid input $x = (x_1, \ldots, x_n)$ two elements x_1 and x_2 are the same element of D (without $x_1 = x_2$ being implied by S) then the action formally considers x_1 and x_2 as two a priori different elements of D.
So the natural action is a free action on the inputs, since an (X, S)-covering is a fixed point for a transformation associated with an automorphism of $\mathbf{F}_{\mathcal{V},S}(X)$, if and only if the automorphism is the identity. Equivalently the algebra generated by a valid input $x \in D^n$ is always isomorphic to a factor of an (X, S)-presented algebra. It is either an (X, S)-presented algebra, which is the generic case or a proper factor of an (X, S)-presented algebra, when it satisfies additional equalities that are not consequences of the identities of the variety nor the equalities in S. These are the degenerate cases.
In the sequel, even in the degenerate cases, the additional equalities that are not consequences of the identities of the variety nor the equalities in S, are never used by the action. So the algebra generated by a valid input $x \in D^n$ will always be formally indentified with $\mathbf{F}_{\mathcal{V},S}(X)$ and called *an instance of the presented algebra*. Depending on the context, the word "instance" will be either an instance of the problem or an instance of the presented algebra (i.e. an algebra that is identified with $\mathbf{F}_{\mathcal{V},S}(X)$). When an instance of the presented algebra is presented by (X, S) then the associated instance of the problem is the set of all its (X, S)-coverings. Usually an instance of the presented algebra is noted $\mathcal{I}$ and the associated instance of the problem is denoted $\tilde{\mathcal{I}}$.

Remark 5. *Whenever the input set contains an n-tuple x, it contains all the instance containing x. The input set is then called* complete.

Definition 5. *Let $\mathcal{V}$ be a variety of algebras of type Ψ, (X, S) a presentation relatively to $\mathcal{V}$ with $|X| = n$ and D a domain, i.e. (D, Ψ) is an algebra of the variety, where m finitary relations $\mathcal{R}_1, \ldots \mathcal{R}_m$ are defined. A* generalized sorting problem *consists in finding a representative for the generating sequences of a presented algebra as precised in the following:*
Input: *a sequence of n elements of D satisfying the equality relations of S.*
Output: *a sequence of n elements of D in the same orbit than the input under the action of* **Aut**$(\Psi, \mathcal{V}, n, S)$ *(i.e. in the same instance than the input) and satisfying an additional property that is expressed with the extended vocabulary $\Psi \cup \{\mathcal{R}_1, \ldots \mathcal{R}_m\}$, i.e. with the fundamental operations and relations (including the equality) existing on the domain D, and usual logical connectors.*

One important particular case is when $S = \emptyset$. In this case, the algebra presented by (X, S) is the free algebra of the variety. Any sequence of $|X| = n$ elements of the domain is then a valid input and the input set is D^n. An instance is an orbit of the action of the automorphism group of the free group of the variety on the input set. With an input sequence $x \in D^n$ on hand, our problem is to find one particular element of the instance containing x.
All examples considered in the paper are in this particular case.
As a first example, consider the variety of (multiplicative) groups, $n = 3$, and $S = \emptyset$. Let D be the set of unimodular 2×2 matrices. We may define a preorder on D. The input is a family of three 2×2 unimodular matrices and the output a family of three matrices generating the same group than the input and satisfying some additional property that is expressed with the preorder among matrices.
As a second example, consider the variety of additive abelian groups $n = 2$, and $S = \emptyset$, and $D = \mathbb{Z}^2$. Define the preorder on D that compares two vectors according to their Euclidean norm. The input is a family of two vectors of $\mathbb{Z}^2$. The output is a family of two vectors (a_1, a_2) generating the same additive abelian group and satisfying $a_1 \leq a_2$ and $a_2 \leq a_2 + a_1$ and $a_2 \leq a_2 + a_1$.
As a third example, consider the variety of sets and $S = \emptyset$. Choose $D = \mathbb{Q}$ with the usual order. An input is a list of n rationals and the output the same rationals sorted in increasing order. The last two examples are detailed in the appendix.
A generic case is when there is binary relation defined on the domain which is an order (or a preorder). Then the coverings are lexicographically ordered and the desired output may be the minimal one with respect to this lexicographic order.
Usually the natural length of an input (the space needed to store an input in a computer) is defined by the domain. The elementary manipulations on datas are the elementary operations on the domain. So the cost of elementary manipulations on datas depends on the domain as well. For example when sorting n items, the domain might be $\mathbb{R}$ or $\mathbb{N}$. The length of inputs and the cost of elementary manipulations will not be the same in these two different cases.

3. An algorithm

Now let A be a deterministic algorithm for a generalized sorting problem. The algorithm A is modeled as in the beginning of the introduction: Its input is chosen in $\mathcal{X}$ which is a set of coverings of presented algebras taken in a domain. The algorithm has a set $F \subset \mathcal{X}^{\mathcal{X}}$ of elementary transformations and a non ambiguous strategy $\mathcal{S}$ for determining the adequate elementary transformation f during any execution. This determination depends on the current data manipulated by the algorithm and eventually on the past history of the execution. For convenience we will always suppose that F contains the identity application id.

As said in the previous section, with the operations of the type Ψ, the input of the algorithm generates a subalgebra of the domain. During the execution, at each step, the strategy determines the elementary transform to apply to the data and the determined elementary transform is then applied to the data. We will require the algorithm to preserve the subalgebra of the domain generated by the input sequence. The elementary transforms are expressed with the operations of the type Ψ. So they can be applied indifferently to any instance of the problem.
Now let us define and briefly discuss some hypotheses for our main theorem.

Hypothesis 1. *The algorithm A is solving a generalized sorting problem (introduced by Definition 5).*

Hypothesis 2. *Running on any input, the algorithm A halts and outputs an element of $\mathcal{X}$ satisfying the required property.*

Definition 6. *Let A be an algorithm and $\mathcal{S}$ the strategy used for determining the adequate elementary transformation f ($x \leftarrow f(x)$ with $f \in F$) at any moment during any execution. If $\mathcal{S}$ does not depend on the time during the execution, i.e. determining the adequate elementary transformation to apply to x depends only on data (x) manipulated and not on the past history of the algorithm, then the strategy (and the algorithm) is a* memory-less*, i.e. iff*

$$\forall (w, w') \in F^2, \forall x \in \mathcal{X}, \mathcal{S}(x, w) = \mathcal{S}(x, w').$$

Remark 6. *Let us give two examples of strategies. Assume that $F = \{f_0, f_1, \ldots, f_{n-1}\}$ is finite and there is preorder on $\mathcal{X}$. For all $x \in \mathcal{X}$,let $F(x)$ be defined by $F(x) := \{g(x) : g \in F\}$. There is always a minimal element in $F(x)$ which may be not unique. Both the following strategies are greedy since the chosen transformation f is such that $f(x)$ is the minimal element of $F(x)$ (with respect to the preorder on $\mathcal{X}$). When $F(x)$ has a unique minimal element both strategies choose the same elementary transformation. Now assume that $F(x)$ has several minima.*
(1) The first strategy is memory-less. The chosen elementary transform is f_i, with i defined by: $i = \min\{0 \leq r \leq n-1 : f_r(x)$ is a minimum of $F(x)\}$. This is the strategy associated with the well known the insertion sort.
(2) The second strategy determines the elementary transform f by a computation that depends on the most recent elementary transform used. If the most recent elementary transform used is f_k, then the chosen elementary transform is $f_{(k+i) \bmod n}$ with i defined by: $i = \min\{0 \leq r \leq n : f_{(k+r) \bmod n}(x)$ is a minimum of $F(x)\}$. This is the strategy associated to the well known the bubble sort.

We have to distinguish the output of a generalized sorting problem (Definition 5) and the output of an algorithm solving the problem. Running on a fixed input, an algorithm (sequential and deterministic in this paper) outputs always the same output, even if the problem may accept several outputs for the fixed input. Let us precise the notion of output for an algorithm. Observe that if the algorithm is not memory-less then some data may be an output for the algorithm in some context and not an output in another context.

Definition 7. *A data x^* in the input set $\mathcal{X}$ is an* output of the algorithm *(or an output) if it is at least once returned by the algorithm, i.e. if there is $x \in \mathcal{X}$ such that the algorithm returns x^* when running on x.*

A fixed point of the algorithm is an input which is returned by the algorithm without any modification, i.e. $x \in \mathcal{X}$ such that $A(x) = x$. Obviously any fixed point of the algorithm is an output. Now consider an output. The elementary transform determined by the algorithm to deal with any output is the identity during at least one execution (during the last iteration). If the algorithm is memory-less then the action of the algorithm on the data does not depend on the history of the execution. So,

Fact 6. *If an algorithm is memory-less then the outputs are exactly those inputs that are fixed points under the action of the algorithm.*

The following directed and colored graph visualizes all the possible trajectories of elementary transforms among datas. Any execution of the algorithm is a path in this graph but of course there are paths that do not correspond to any execution.

Definition 8. *The following directed and colored graph $G = (V, E)$ is called the execution graph of the algorithm (respectively the execution graph of an instance $\mathcal{I}$).*
The set of vertices V is the set of all inputs (respectively the set of all inputs inside an instance $\mathcal{I}$). For any two vertices c_1 and c_2, $(c_1, c_2) \in E \Leftrightarrow$ there is an elementary transform $f \in F$ such that $c_2 = f(c_1)$. If (c_1, c_2) is an edge, it is always labelled or colored by the name of the elementary transform f it represents.

Remark 7. *Consider an edge of the execution graph. By definition it is labelled by an elementary transform that is used at least once during an execution. This makes it impossible to add artificial and useless elementary transforms to the set F of elementary transforms.*

Now let us recall the definition of a weakly connected (or connected) component of a directed graph.

Definition 9. *Let G be a directed multi-graph and H be a subgraph of G. Let G' and H' be obtained from G and H by preserving the same vertices and replacing the directed edges of G and H with undirected edges. H is a connected component of G if and only if H' is a connected component of G'.*

Hypothesis 3. *The execution graphs of instances are the connected components of the execution graph of the algorithm.*

Hypothesis 3 defines a partition of the input set $\mathcal{X}$ into sets that are stable under the action of the set of elementary transforms F and identifies any such set with an instance of the problem. It express the whole compatibility of the set of elementary transforms of the algorithm with the partitioning of the input set into the sets of coverings of instances of the presented structure.

Definition 10. *Let $A(\mathcal{X}, O, F, \mathcal{S})$ be an algorithm. The set of all directed and colored*[11] *paths corresponding to executions of the algorithm running on all inputs (respectively all inputs inside an instance) is called* the effective execution graph of the algorithm *(respectively* the effective execution graph of the instance*).*
Two algorithms are equivalent *if and only if there is an isomorphism (of directed, colored graphs) between their effective execution graphs.*

[11] The colors are naturally the names of the elementary transforms.

Fact 7. *Let $A(\mathcal{X}, O, F, \mathcal{S})$ be an algorithm. Suppose Hypothesis 2 and 3 are satisfied. Let $\mathcal{I}$ be an instance of size n. Let $\mathcal{G}_{\mathcal{I}}$ be the execution graph of the instance $\mathcal{I}$ and $\mathcal{T}_{\mathcal{I}}$ be the effective execution graph of the instance.*

(*i*) *$\mathcal{T}_{\mathcal{I}}$ is a spanning sub-diagram of $\mathcal{G}_{\mathcal{I}}$.*

(*ii*) *If the algorithm is memory-less then $\mathcal{T}_{\mathcal{I}}$ is a spanning directed and colored forest of $\mathcal{G}_{\mathcal{I}}$.*

(*iii*) *If the instance $\mathcal{I}$ has a unique output x, then $\mathcal{T}_{\mathcal{I}}$ is a connected spanning sub-diagram of $\mathcal{G}_{\mathcal{I}}$.*

(*iv*) *If the instance $\mathcal{I}$ has a unique output x and the algorithm is memory-less then $\mathcal{T}_{\mathcal{I}}$ is a spanning directed and colored tree of $\mathcal{G}_{\mathcal{I}}$.*

Proof. (*i*) is trivial since the algorithm is running on any input. So $\mathcal{T}_{\mathcal{I}}$ is containing all the vertices of $\mathcal{G}_{\mathcal{I}}$. (*ii*) Under Hypthesis 2, no directed path corresponding to one single execution contains a circuit. Moreover $\mathcal{T}_{\mathcal{I}}$ has no circuit, i.e. there is no circuit whose edges belong to different executions, otherwise running on a vertex of the circuit, since the algorithm is memory-less, it would loop and never end. More generally $\mathcal{T}_{\mathcal{I}}$ has even no cycles, otherwise there would be a vertex on the cycle with two different outgoing edges which is impossible since the algorithm is deterministic and memory-less: there is only one path followed by the algorithm from any vertex to the corresponding output. (*iii*) is trivial since then $\mathcal{T}_{\mathcal{I}}$ contains a path from any vertex to the vertex corresponding to the output. (*iv*) is just (*ii*) and (*iii*) together. □

Remark 8. *For the same reasons, the effective execution graph $\mathcal{T}$ of the algorithm is always a spanning sub-diagram of the execution graph $\mathcal{G}$. In the same vain under Hypothesis 2 if the algorithm is memory-less then $\mathcal{T}$ is a spanning directed and colourd forest of $\mathcal{G}$.*

Definition 11. *Assume that there is no circuit in the effective execution graph of the algorithm*[12] *(Definition 10). Consider the relation on $\mathcal{X}$ defined by "c is related to c′" if and only if there is a directed path in the effective execution graph from c to c′. The reflexive closure of this relation is an order relation on $\mathcal{X}$ called the algorithm order.*

Fact 8. *Any element of $\mathcal{X}$ that is minimal for the algorithm order is an output. Moreover if the algorithm is memory-less, then any element in $\mathcal{X}$ which is not an output has a unique successor and any output is a minimal element.*

Proof. A minimal element for the algorithm order is by definition an output. Now consider a data in $\mathcal{X}$. It may be an output during an execution and an intermediate non terminal data during another execution. Such an output is not minimal for the algorithm order as defined above, but this cannot happen if the algorithm is memory-less. Moreover since the elementary transformation applied to a data is then the same in all executions, any data has a unique successor that is the modified data under the action of the elementary transform. □

Since the algorithm has to work correctly with all (often infinitely many) instances taken in a domain, it is quite reasonable to require that the elementary transforms operate on any data (an

[12] Observe that if the algorithm is memory-less and no execution is containing a loop then the execution graph has no circuit. Otherwise the execution graph may contain circuits while no execution is containing a loop.

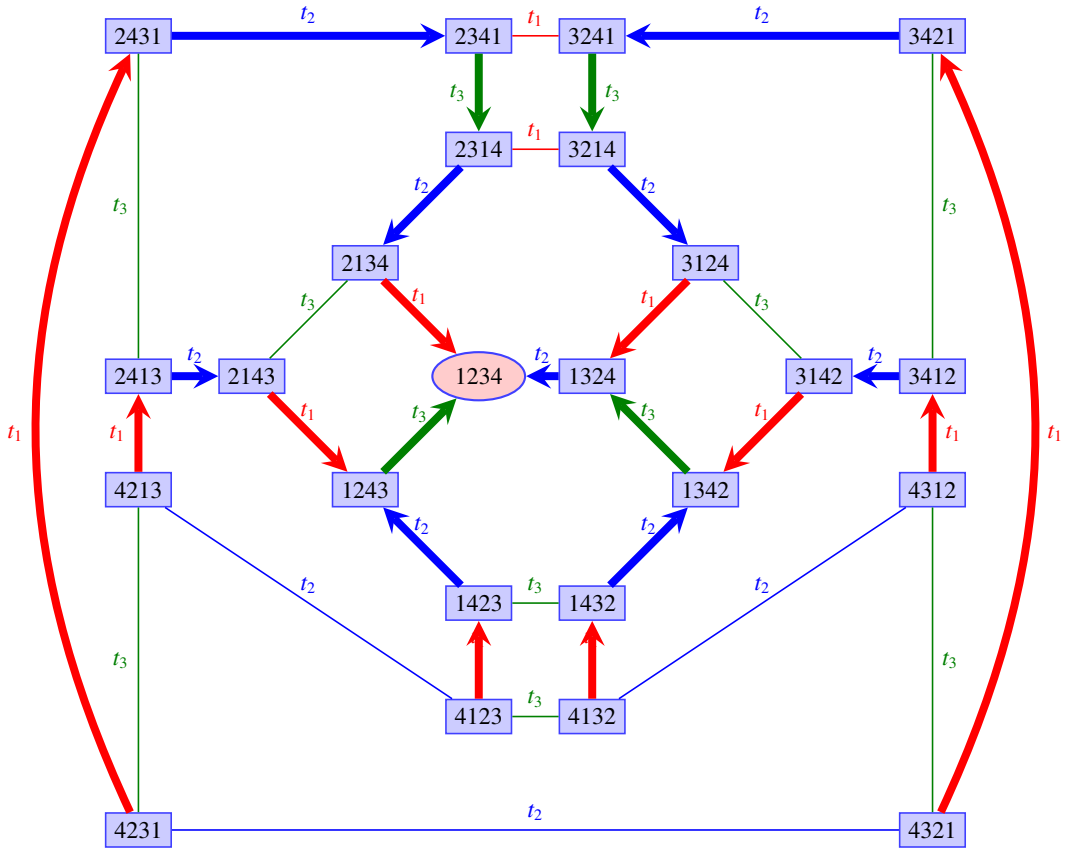

Figure 1: The insertion sort algorithm running on the instance $\mathcal{I} = \{1, 2, 3, 4\}$. There are three elementary transforms: for $1 \leq i \leq 3$, t_i is the swap of the i^{th} and $(i+1)^{th}$ elements of the covering. The agorithm is memory-less and there is a unique ouput. So the effective execution graph of the instance $\mathcal{I}$ is a spanning tree of the execution graph of $\mathcal{I}$.

n-tuple of the domain) independently of the instance, i.e. automatically: They act in the same way on different instances. It is also reasonable to assume that the elementary transforms on datas can be performed by using only the fundamental operations of the algebra. The next hypothesis proposes one way to achieve this.

Hypothesis 4. *Let Ψ be a vocabulary, $\mathcal{V}$ a variety of type Ψ and (X, S) a presentation relatively to $\mathcal{V}$. For all f in F, f is given by a sequence of $n = |X|$ terms $T_1, \dots T_n$ of type Ψ over X, i.e. a sequence of n elements of the term algebra $\mathbf{T}(X)$. If $c = (c_1, \dots, c_n)$ is an input of the elementary transform f, the output $f(c)$ is defined by*

$$f(c) = (T_1(c_1, \dots, c_n), \dots, T_n(c_1, \dots, c_n)) .$$

Remark 9. *(i) If the presented algebra has no fundamental operation (the variety of sets, i.e. when $\Psi = \emptyset$) then the only possible terms are the the variables $x_1, \dots, x_n$.*

(ii) In this paper we always work with presented algebras of a fixed rank. This remark is an exception: If an algorithm works automatically on presented algebras of arbitrary rank, then it is reasonable to require a compatibility condition with the rank. For example, the n terms corresponding to the n first (or last or consecutive) elements of the output covering of any elementary transform for presented algebras of rank $n + 1$, are the terms associated with an elementary transform $f' \in F$ that is used when working with presented algebras of rank n.

(iii) By Hypothesis 4 the algebra generated by the output sequence is a subalgebra of the algebra generated by the input sequence. Hypothesis 3 ensures that both input and output sequences are coverings of the same algebra and the set of automorphisms labelled by the elementary transforms (see Fact 2) generates the automorphism group of the algebra.

(iv) Hypotheses 3 and 4 could be replaced by one unique hypothesis asserting that any elementary transform is an n-tuple of terms that labels an automorphism of the presented algebra as detailed in Fact 2. Moreover the set of automorphisms so labelled generates the automorphism group of the presented algebra. If the presented algebra is the free group of rank n, then the elementary transforms are a set of Nielsen transformations generating the group of all Nielsen transformations. It can be for example, the set of elementary Nielsen transformations *(see (11)).*

Fact 9. *Let $\tilde{\mathcal{I}}$ be an instance. Let f be an elementary transform satisfying Hypothesis 4 and c^* be an element of $\tilde{\mathcal{I}}$. Under Hypothesis 3, $\hat{f} := (c^*, f(c^*))$ is an automorphism of $\mathcal{I}$, that is naturally asociated to the pair (c^*, f).*

Proof. Recall we work on presented algebras with fixed presentation (X, S). The sequence $f(c^*)$ generates a subalgebra of the algebra generated by c^*. Moreover it is in the same instance $\tilde{\mathcal{I}}$ since Hypothesis 3 is satisfied and an edge labelled by f makes c^* and $f(c^*)$ being in the same connected component of the execution graph. □

Fact 10. *Suppose that Hypotheses 1, 3 and 4 are satisfied. Let $\tilde{\mathcal{I}}$ be an instance. The restriction of any element of F to $\tilde{\mathcal{I}}$ is a permutation on this set.*

Proof. Under Hyptheses 3 and 4, f associates an element of $\tilde{\mathcal{I}}$ to another element of $\tilde{\mathcal{I}}$ and it is defined by the terms expressing the output covering in the input covering. So the fact praphrases Fact 4. □

Fact 11. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two instances and ϕ be any isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. Let $\tilde{\phi}$ be the extension of ϕ to the sets of coverings of $\mathcal{I}_1$ and $\mathcal{I}_2$.*

$$\forall \omega \in F^*,\ \forall c \in \tilde{\mathcal{I}},\quad \tilde{\phi}(\omega(c)) = \omega(\tilde{\phi}(c)).$$

Proof. We show the fact with $\omega = f \in F_{\mathcal{I}}$ (Then an obvious induction on the length of ω lead to the fact with $\omega \in F_{\mathcal{I}}^*$). The isomorphism ϕ is defined with the pair of coverings $(c, \tilde{\phi}(c))$. The elementary transform f moves the covering c to another covering c' by mean of terms $(W_1, \ldots, W_n)$. So $c' = (W_1(c), \ldots, W_n(c))$. By the definition of the isomorphism ϕ, $\phi(W_i(c)) = W_i(\tilde{\phi}(c))$, for all i and so $\tilde{\phi}(f(c)) = f((\tilde{\phi}(c))$. □

4. Identifying inputs, automorphisms and class of terms on elementary transforms

Lemma 1. *Under Hypothesis 1, 3 and 4, the execution graph of any instance (i.e. all connected components of the execution graph of the algorithm with vertices labelled by a list of n element of the algebra) is isomorphic to the Cayley diagram of the automorphism group of the presented algebra.*

Proof. Let us construct one Cayley diagram for the automorphism group of the presented algebra. Consider an instance $\mathcal{I}$ of the (X, S)-presented algebra. Hypothesis 1 together with Definition 8 assert that the vertices of the execution graph of an instance are labelled by all the (X, S)-coverings of the instance. But we have already noticed that when one arbitrary generating sequence c^* is fixed, then any generating sequence c is identified with an automorphism, say with the automorphism (c^*, c) and $\{(c^*, c) : c \text{ generating sequence of } \mathcal{I}\}$ is exactly the set of all automorphisms of the instance $\mathcal{I}$. Hence vertices of the execution graph of the instance are labelled by automorphisms of the instance and of course c^* is the only vertex identified with the trivial automporphism.
Once more from Definition 8, for any two vertices c_1 and c_2, (c_1, c_2) is a directed edge if and only if there is an elementary transform $f \in F$ such that $c_2 = f(c_1)$. Moreover by Hypothesis 4 the elementary transform f is given by terms $W_1, \ldots W_n$ that relate the components of the output covering to the components of the input covering. Now to any elementary transform f (that is in $\mathcal{X}^X$) we associate an automorphism $\hat{f}$ defined by $\hat{f} = (\,(c_1^*, \ldots, c_n^*)\,,\ (W_1(c^*), \ldots, W_n(c^*))\,)$. Then thanks to Fact 3, (c_1, c_2) is an edge if and only if $(c^*, c_1) \circ \hat{f}$ is represented by $(\,(c_1^*, \ldots, c_n^*)\,,\ (W_1(c_1), \ldots, W_n(c_n))\,)$. Hence following a directed edge labelled by f corresponds to the right multiplication in the automorphism group by $\hat{f}$.
Now by Fact 10, any elementary transform (restricted to an instance) is a permutation on the instance. So for any vertex of the execution graph of $\mathcal{I}$ and for any elementary transform $f \in F$, there is exactly one incoming and exactly one outgoing edges labelled by f.
Finally Hypothesis 3 states that the execution graph of any instance is connected. So any vertex is connected to the one labelled by c^* by means of at least one path labelled by $f_{i_1}^{\varepsilon_{i_1}} \ldots f_{i_r}^{\varepsilon_{i_r}}$ where ε_i is ± 1. So the vertex represents $\hat{f}_{i_1}^{\ \varepsilon_{i_1}} \circ \ldots \circ \hat{f}_{i_r}^{\ \varepsilon_{i_r}}$ and F is a set of generators for the automorphism group of $\mathcal{I}$. Moreover any other path from this vertex to c^* represents the same automorphism, i.e. is an equivalent member of the automorphism group. □

Remark 10. (*i*) *The previous lemma does not depend on the fixed generating sequence c^* used in Fact 9 to identify elementary transforms with automorphisms.*
(*ii*) *Lemma 1 is also a direct corollary of the well–known orbit-stabilizer theorem applied to the natural action of the automorphism group of the (X, S)–presented algebra on an orbit. As we already noticed this action is free, so it is regular on an orbit. Hence there is a natural bijection between the automorphism group of the presented algebra and any orbit. All the orbits are isomorphic G-sets (Any orbit is a principal homogeneous space or a torsor for the automorphism group of the presented algebra)*

Corollary 1. *Consider an algorithm that is solving a generalized problem. Assume that the automorphism group of the presented algebra is finite and for any input, the problem accepts e unique output . Let $F = (f_i)_{1\leq i\leq n}$ be a set of elementary transform such that Hypotheses 2, 3 and 4 are satisfied and let $\hat{F}$ be the set of automorphisms, obtained from F by fixing an arbitrary covering c^* as in Fact 9. The diameter of the Cayley graph of the automorphism group of the presented algebra, with $\hat{F}$ as a set of generators is a lower bound for the complexity of any algorithm that solves the problem, with the elementary transforms $f_1, \ldots, f_n$.*

Proof. An execution trace of the algorithm is a path on the finite Cayley graph of the automorphism group of the presented algebra and from any input there is a path to the output. Consider an instance. Any vertex of the Cayley graph is an input of the instance and there is a unique vertex which is also an output. The distance between a vertex (an input) and the output, i.e. the minimal number of edges separating this input from the output is also a lower bound for number of elementary transforms used by the algorithm when running on this input. The eccentricity of the output is (by definition) the maximum of the distances from an arbitrary vertex (an arbitrary input of the instance) to the output, i.e. a lower bound for the number of elementary transforms used by any algorithm with the elementary transforms $f_1, \ldots, f_n$, when running on any input of the instance. If the output is in the center of the graph, its eccentricity is minimal. The minimum which is then by definition the radius of the execution graph is a lower bound for the number of iterations used by any algorithm with the elementary transforms $f_1, \ldots, f_n$, when running on any input of the instance. Finally, since the execution graph of the instance is a Cayley graph, it is vertex transitive. So all the vertices have the same eccentricity. The minimal eccentricity of different vertices of the graph (the radius) is then also the maximal eccentricity (the diameter of the graph). □

Let us illustrate the previous result with an easy consequence: Let c be a generating sequence of a set of cardinal n. Consider the Cayley graph of the permutation group on n elements generated the transpositions that swap two consecutive elements. This graph is well known and usually called *Bubble sort graph* (see (9)): Its diameter is $n(n-1)/2$. From the previous corollary one deduces immediately,

Fact 12. *Consider a sorting algorithm: when running on all sequences of n elements without repetition taken in a linearly ordered set, it outputs the ordered sequence after a finite number of steps. Any such algorithm whose elementary transforms are the $n-1$ transpositions corresponding to swapping the i-th element and the $(i+1)$-th element, is requiring in the worst case, at least $n(n-1)/2$ steps.*

Different sorting algorithms, including bubble sort and insertion sort are satisfying the conditions of the previous fact.

5. Input classes, rewrite systems

Definition 12. *Two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of the presented algebra taken in a domain are called compatible with respect to an algorithm $A(\mathcal{X}, O, F, \mathcal{S})$, if and only if there is an isomorphism of the presented algebra $\Phi : \mathcal{I}_1 \to \mathcal{I}_2$, called* a strategy–preserving isomorphism*, such that the extension $\tilde{\Phi}$ to the sets $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ of coverings of $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfies the following property: the strategy $\mathcal{S}$ is invariant under the action of $\tilde{\Phi}$, i.e. for all x covering of $\mathcal{I}_1$ and for all sequence*

of elementary transforms $(f_j)_{1\leq j\leq i} \in (F_{\mathcal{I}})^i$*, corresponding to a possible history of the algorithm before dealing with the data*[13] *x,*

$$\mathcal{S}(\tilde{\Phi}(x), (f_j)_{1\leq j\leq i}) = \mathcal{S}(x, (f_j)_{1\leq j\leq i}). \tag{2}$$

The compatibility relation is clearly an equivalence relation on the set of instances of the problem. An equivalence class for the compatibility relation is called an input class*.*

Let us define a useful notation and state two facts.

Notation 2. *Consider an execution trace in m steps: Running on the input* c_0 *the algorithm is using successively the elementary transforms* $f_0, \ldots, f_{m-1}$ *and is obtaining successively the intermediate datas* $c_1, \ldots, c_{m-1}$ *and finally outputs* c_m*. This execution trace is denoted by*

$$c_0 \xrightarrow{f_0} c_1 \ldots c_{m-1} \xrightarrow{f_{m-1}} c_m. \tag{3}$$

Fact 13. *Let* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *be two compatible instances and* $\Phi : \mathcal{I}_1 \to \mathcal{I}_2$ *be an isomorphism such that the strategy* $\mathcal{S}$ *is invariant under the action of* $\tilde{\Phi}$*. Then there is a bijection between the set of execution traces of the instance* $\mathcal{I}_1$ *and the set of execution traces of the instance* $\mathcal{I}_2$*. More precisely the bijection* $\hat{\Phi}$ *extending* Φ *from* $\tilde{\mathcal{I}}_1 \times (F \times \tilde{\mathcal{I}}_1)^*$ *to* $\tilde{\mathcal{I}}_2 \times (F \times \tilde{\mathcal{I}}_2)^*$ *defined by:*

$$\hat{\Phi}(c_0, f_0, c_1, f_1, \ldots, c_{m-1}, f_{m-1}, c_m) := \left(\tilde{\Phi}(c_0), f_0, \tilde{\Phi}(c_1), f_1, \ldots \tilde{\Phi}(c_{m-1}), f_{m-1}, \tilde{\Phi}(c_m)\right) \tag{4}$$

is a bijection between the set of execution traces of the instance $\mathcal{I}_1$ *and the set of execution traces of the instance* $\mathcal{I}_2$*.*

Proof. By construction $\hat{\Phi}$ is a bijection from from $\tilde{\mathcal{I}}_1 \times (F \times \tilde{\mathcal{I}}_1)^*$ to $\tilde{\mathcal{I}}_2 \times (F \times \tilde{\mathcal{I}}_2)^*$. So to prove the fact, since $\mathcal{I}_1$ and $\mathcal{I}_2$ (respectively Φ and Φ^{-1}) play a symmetric role, we have just to show that whenener an element of $\tilde{\mathcal{I}}_1 \times (F \times \tilde{\mathcal{I}}_1)^*$ is an execution trace in $\mathcal{I}_1$, its image by $\hat{\Phi}$ is an execution trace in $\mathcal{I}_2$. Let m be a positive integer. Consider any execution trace in m steps,

$$c_0 \xrightarrow{f_0} c_1 \ldots c_{m-1} \xrightarrow{f_{m-1}} c_m$$

in $\mathcal{I}_1$. We will show by induction on the number k of steps ($1 \leq k \leq m$) that

$$\tilde{\Phi}(c_0) \xrightarrow{f_0} \tilde{\Phi}(c_1) \ldots \tilde{\Phi}(c_{k-1}) \xrightarrow{f_{k-1}} \tilde{\Phi}(c_k)$$

is a piece of an execution trace in $\mathcal{I}_2$. Running on the input $\tilde{\Phi}(c_0)$ since Φ is strategy–preserving, the elementary transform used by the algorithm is f_0 as well. Moreover by Fact 11, $\tilde{\Phi}(c_1) = \tilde{\Phi}(f_0(c_0)) = f_0(\tilde{\Phi}(c_0))$. Suppose that this is true after $k < m$ steps, i.e. when dealing with the input $\tilde{\Phi}(c_0)$, during k steps the algorithm has used the same elementary transforms than the ones used when dealing with the input c_0. So $\tilde{\Phi}(c_0) \xrightarrow{f_0} \tilde{\Phi}(c_1) \ldots \tilde{\Phi}(c_{k-1}) \xrightarrow{f_{k-1}} \tilde{\Phi}(c_k)$ is a beginning piece of an execution trace of the algorithm in the instance $\mathcal{I}_2$. Now consider $\tilde{\Phi}(c_k)$ as an intermediate data during the execution running on $\tilde{\Phi}(c_0)$. Once more, since Φ is strategy–preserving, there is in $\mathcal{I}_2$ an execution using f_k to deal with $\tilde{\Phi}(c_k)$.
Hence the execution trace $\tilde{\Phi}(c_0) \xrightarrow{f_0} \tilde{\Phi}(c_1) \ldots \tilde{\Phi}(c_{m-1}) \xrightarrow{f_{m-1}} \tilde{\Phi}(c_m)$ does exist in $\mathcal{I}_2$.

□

[13] i.e. there is $y \in \mathcal{I}$ such that running on y the algorithm A follows the sequence $(f_j)_{1\leq j\leq i}$ of elementary transforms and then continues with x as intermediate data.

Fact 14. *Assume that the algorithm is memory-less and Hyptheses 1 and 2 are satisfied. Then two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class if and only if there is an isomorphism Φ between $\mathcal{I}_1$ and $\mathcal{I}_2$ such that the extension $\tilde{\Phi}$ to the sets of coverings $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ is an increasing bijection for the algorithm order.*

Proof. The algorithm order is well defined provided that the execution graph has not circuit, which is clearly the case here: when an algorithm is memory-less a circuit in the execution graph would be exactly a non-terminating execution. First assume that $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class. Let Φ be any strategy–preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. By the previous fact, Φ can be extended as in relation (4) to a bijection $\hat{\Phi}$ between the execution traces of $\mathcal{I}_1$ and $\mathcal{I}_2$. Then for c, c' elements of $\tilde{\mathcal{I}}_1$, $c \leq c'$ iff $c \xrightarrow{f_1} c_1 \dots c_k \xrightarrow{f_k} c'$ is part of an execution in $\tilde{\mathcal{I}}_1$, iff $\tilde{\Phi}(c) \xrightarrow{f_1} \tilde{\Phi}(c_1) \dots \tilde{\Phi}(c_k) \xrightarrow{f_k} \tilde{\Phi}(c')$ is an an execution $\tilde{\mathcal{I}}_2$, that is $\tilde{\Phi}(c) \leq \tilde{\Phi}(c')$. Hence the extension of any strategy–preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$ to the sets of coverings is an increasing bijection for the algorithm order.
Now assume there is an isomorphism Φ between $\mathcal{I}_1$ and $\mathcal{I}_2$ such that the extension $\tilde{\Phi}$ to the sets of coverings $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ is an increasing bijection for the algorithm order. As Φ is an isomorphism, Fact 11 asserts that if $c \xrightarrow{f} c'$ is part of an execution in $\tilde{\mathcal{I}}_1$, then $\tilde{\Phi}(c) = f(\tilde{\Phi}(c'))$. Let us show that $\tilde{\Phi}(c) \xrightarrow{f} \tilde{\Phi}(c')$.
From $c \xrightarrow{f} c'$, one deduces that there is an execution in $\tilde{\mathcal{I}}_2$ where $\Phi(c)$ and $\Phi(c')$ are two intermediate datas and $\Phi(c')$ is obtained from $\Phi(c)$ within k steps:

$$\tilde{\Phi}(c) \xrightarrow{f_1} \tilde{\Phi}(c_1) \dots c_{k-1} \xrightarrow{f_k} \tilde{\Phi}(c').$$

Otherwise one would have $c \leq c'$, but without $\tilde{\Phi}(c) \leq \tilde{\Phi}(c')$. Now from $\tilde{\Phi}(c) \xrightarrow{f_0} \tilde{\Phi}(c_1)$ being part of an execution in $\tilde{\mathcal{I}}_2$ one deduces that there is an execution in $\tilde{\mathcal{I}}_1$ where $\Phi(c_1)$ and $\Phi(c)$ are two intermediate datas and c_1 is obtained from c within k' steps:

$$c \xrightarrow{f'_1} c'_1 \dots c'_{k'} \xrightarrow{f'_{k'}} c_1.$$

Otherwise one would have $\tilde{\Phi}(c) \leq \tilde{\Phi}(c_1)$, without $c \leq c_1$. But since the algorithm is deterministic and memory-less, $c \xrightarrow{f'_1} c'_1$ and $c \xrightarrow{f} c'$ being parts of executions in $\tilde{\mathcal{I}}_1$ leads to $c' = c'_1$. Then one has $c \leq c' \leq c_1$, but $\tilde{\Phi}(c) \leq \tilde{\Phi}(c_1) \leq \tilde{\Phi}(c')$, which contradicts the fact that $\tilde{\Phi}$ is increasing. So $k' = 1$ and $c_1 = c'$. Since there are no loop in any execution trace (otherwise the algorithm would not sop), $k = 1$. Finally since the action of the elementary transforms on the input set is free, one has also $f = f_1$. So for any element c of $\tilde{\mathcal{I}}_1$, when dealing with $\Phi(c)$ the strategy will apply the same elementary transform than when dealing with c. That is $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class. □

Next corollary exhibits some invariants for an input class, the number of acceptable outputs, and the way the outputs are related.

Corollary 2. *Let $A(X, O, F, \mathcal{S})$ be an algorithm and $\mathcal{I}_1$ and $\mathcal{I}_2$ be two compatible instances.*
(i) $\mathcal{I}_1$ and $\mathcal{I}_2$ have the same number k of acceptable outputs.
(ii) If $x_1, \dots, x_k$ and $x'_1, \dots, x'_k$ are the acceptable outputs of $\mathcal{I}_1$ and $\mathcal{I}_2$, then there are at most k strategy–preserving isomorphisms between $\mathcal{I}_1$ and $\mathcal{I}_2$ defined by pairs of coverings (x_1, x'_i) for $i \in \{1, \dots k\}$.

(iii)Assume that two outputs x_1 and x_2 in $\mathcal{I}_1$ are related with a sequence ω of elementary transformations: $x_2 = \omega(x_1)$ and let Φ be a strategy preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. Then $\tilde{\Phi}(x_1)$ and $\tilde{\Phi}(x_2)$ are two outputs related with the same same sequence of elementary transforms: $\tilde{\Phi}(x_2) = \omega(\tilde{\Phi}(x_1))$.

Proof. Let Φ be a strategy–preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$ (see Definition 12) and $\tilde{\Phi}$ be the bijection that extending $\tilde{\Phi}$ to the sets of coverings $\tilde{\mathcal{I}_1}$ and $\tilde{\mathcal{I}_2}$. From Fact 13, x is an output in $\mathcal{I}_1$ if and only if $\tilde{\Phi}(x)$ is an output in $\mathcal{I}_2$. This shows (*i*) and (*ii*). In particular, if $\mathcal{I}_1$ and $\mathcal{I}_2$ have unique outputs then there is a unique strategy–preserving isomorphism between them.
Now let $\Psi : \mathcal{I}_1 \to \mathcal{I}_2$ be any isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. Ψ defines naturally a bijection $\tilde{\Psi}$ between the sets of coverings of $\mathcal{I}_1$ and $\mathcal{I}_2$. Then Fact 11 states that for any sequence of elementary transforms w, $\tilde{\Psi}(w(c)) = w(\tilde{\Psi}(c))$. □

Now let us point out the theorem. It shows that in order to characterize executions of the algorithm, one has to study just the executions on one instance inside each input class. Then inside one such instance (representing one input class), one has *a priori* to characterize separately executions of the algorithm, for any set of inputs associated with an output (the sets of preimages of the outputs). In fact this depends on the number of strategy–preserving automorphisms inside an instance of the class: For an input class with k outputs, if there are k isomorphisms satisfying Definition 12, then all the outputs play the same role and one has to characterize the executions of the set of preimages of only one output inside the class[14].

Hypothesis 5. *In any input class with k outputs, i.e. where all instances have exactly k outputs, there are exactly k strategy–preserving isomorphisms between any pair of instances of the class.*

Corollary 3. *Let (x^*, y^*) be two outputs of an input class. Under the previous hypothesis, the isomorphism (x^*, y^*) is strategy-preserving.*

Proof. Let x^* and y^* be in two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of the input class. Let $(x_1^*, \ldots, x_k^*)$ be all the outputs of $\mathcal{I}_1$ and $(y_1^*, \ldots, y_k^*)$ all the outputs of $\mathcal{I}_2$. Let us fixe $r \in \{1, \ldots, k\}$. Clearly (and Fact 13 shows it) the image of an output by any strategy–preserving isomorphism Φ is an output. So, there is $j \in \{1, \ldots, k\}$ such that $\tilde{\Phi}(x_r^*) = y_j^*$. Since there is no other strategy-preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$ than those associating x_r^* to one of the y_i^*'s, the k strategy–preserving isomorphisms of Hypothesis 5 are exactly (x_r^*, y_i^*), $1 \leq i \leq k$. The argumentation holds for any $r \in \{1, \ldots, k\}$. So the k^2 isomorphisms (x_r^*, y_i^*), $1 \leq r, i \leq k$ are all strategy–preserving and are precisely k distinct isomorphisms.

□

Theorem 1. *Let $A(X, O, F, \mathcal{S})$ be an algorithm as specified precisely in the introduction: X is the set of inputs, $O \subset X$ is the set of outputs, $F \subset X^X$ is a set of elementary transforms and $\mathcal{S}$ is a non ambiguous strategy for determining the adequate elementary transform $f \in F$ at any moment during any execution. So running on an input in X, the algorithm outputs an element in O, after a sequence of elementary transforms. Assume that Hypothesis 1, 2, 3, 4 and 5 are satisfied. Then X is partitioned into input classes such that for any input class C there is a rewrite system over F^* such that executions of A running on any input inside C are exactly normal forms of the rewrite system.*

[14] If the number of isomorphisms satisfying Definition 12 is strictly less than k, then the outputs of the class are partitioned into $1 < k' < k$ sets and for each one, the set of preimages of one output has to be considered (k' sets of preimages to consider)

Moreover whenever an element of F^* is not a possible execution of the input class the associated normal form provides a new sequence of elementary transform close to the initial sequence (as a global transformation) that is a possible execution of the input class.

Proof. Consider an input class C. All instances inside C have the same size n and so an execution graph isomorphic to the Cayley diagram of the automorphism group of the inputs of size n. Thanks to the first assertion of Corollary 2, they have also all the same set of elementary transforms, say F_C.
Now consider an input class C. To find how the word $\omega \in F_C^*$ is rewritten, first choose any output, say x^*, of the class. Since ω is a permutation on the set of coverings of any instance of the class, there is a unique input x (a generating sequence in the same instance than x^*) such that $\omega(x) = x^*$. Running on x the algorithm outputs x'^* which is an (eventually other) output of the same instance, after a finite sequence of elementary transforms $\omega' \in F_C^*$. Let the word ω' be the result of rewriting ω. If $\omega' = \omega$ then ω is a normal form of this rewrite system and this happens if and only if ω corresponds to an execution trace. Moreover this does not depend on the output inside the input class: Let y^* be another output of the class. By Corollary 3 of Hypothesis 5, the isomorphism $\Phi = (x^*, y^*)$ is strategy–preserving (in other words the outputs of the class play all a symetric rôle). Then by Fact 11, $y^* = \tilde{\Phi}(x^*) = \tilde{\Phi}(\omega(x)) = \omega(\tilde{\Phi}(x))$. Finally Fact 13 asserts that running on $\tilde{\Phi}(x)$ the algorithm outputs $\tilde{\Phi}(x'^*)$ after the previously considered sequence of elementary transforms $\omega' \in F_C^*$. So the word ω' does not depend on the instance used during the rewriting method.

□

Remark 11. *Usually a rewrite system consists in some local rewriting rules. They are applied as soon as possible. Then one asks two typical questions about the rewrite system: termination and confluence. Here any term is rewritten* globally *with the algorithm. Since we require the considered algorithm to always terminate, our rewrite system is always terminating. There is a unique rewrite algorithm. Alternatively one can consider that we have one rewrite rule per term. Now let us focus the question of the confluence, i.e. whether the order of applying different rules matters. First observe that any term here has a unique normal form. This is a direct consequence of normal forms being exactly execution traces of the algorithm inside an input class and the fact that our algorithms are always deterministic.*

Fact 15. *Suppose we are rewriting inside an input class with one unique output. Let $w = w_1 w_2$ be a word in F^*. Let w', w'_1, w'_2 be the results of rewriting w, w_1, w_2. Let w'' be the result of rewriting $w'_1 w'_2$. Then $w' = w''$.*

Proof. Let $\mathcal{I}$ be an instance in an input class with one unique output. Let x^* be the output of $\mathcal{I}$. Consider a word u and call u' the result of rewriting u. We adopt the same notation as in Fact 9: to the pair (x^*, u) we associate the automorphism $\hat{u} := (x^*, u(x^*))$. By construction u and u' are the words labelling two directed paths in the execution graph of the instance, from the same starting vertex to the same ending vertex (the unique output). So the automorphisms $\hat{u} = (x^*, u(x^*))$ and $\hat{u'} = (x^*, u'(x^*))$ are the same. Finally $\hat{w'_1} = \hat{w}_1$, $\hat{w'_2} = \hat{w}_2$ and $\hat{w} = (x^*, w(x^*))$ and $\hat{w'_1 w'_2} = (x^*, w'_1 w'_2(x^*))$ are the same automorphism. The preimage of x^* by this automorphism is a unique input x satisfying $w(x) = w'_1 w'_2(x) = x^*$. So $w' = w''$ and as explained in the proof of the previous theorem it is provided by the execution trace of the algorithm running on x. □

The next hypothesis proposes a reasonable way to model the strategy. It is essentially the same than the third hypothesis in Gurevich's model(8) of a sequential algorithm by the so-called abstract state machines, namely the *bounded exploreness*[15].

Hypothesis 6. *There is a finite number of relations $\mathcal{R}_1, \ldots, \mathcal{R}_m$ of respective arities $a_1, \ldots, a_m$ on the domain and a finite number of terms $t_1, \ldots, t_p$ in the vocabulary, such that the decision of the strategy is based on (and exclusively on) the boolean values of the following k relational terms:*

$$\mathcal{R}_{i_1}(t_{i_1,1}, \ldots, t_{i_1,a_{i_1}}), \mathcal{R}_{i_2}(t_{i_2,1}, \ldots, t_{i_2,a_{i_2}}), \ldots, \mathcal{R}_{i_k}(t_{i_k,1}, \ldots, t_{i_k,a_{i_k}}),$$

where $j \in \{1, \ldots, k\}$, $i_j \in \{1, \ldots, m\}$ and $t_{i_j,h} \in \{t_1, \ldots, t_p\}$.

Equivalently Hypothesis 6 asserts that two coverings $(a) = (a_1, \ldots, a_n)$ and $(b) = (b_1, \ldots, b_n)$ coincide on $\mathcal{R}_{i_j}(t_{i_j,1}, \ldots, t_{i_1,a_{i_j}})$ for all $j \in \{1, \ldots, k\}$ if and only if the elementary transform determined for the next step of the algorithm by the strategy running on $(a) = (a_1, \ldots, a_n)$ and on $(b) = (b_1, \ldots, b_n)$ are the same.
For example, assume that we are working with Abelian free groups of rank 2, presented by two elements, the domain is $\mathbb{R}^3$ and there is only one binary relation $\mathcal{R}$ on the domain which is the classical preorder related to the Euclidean norm of vectors: For x and y two vectors of $\mathbb{R}^3$,

$$\mathcal{R}(x, y) \Leftrightarrow \|x\| \leq \|y\|.$$

The input set is $(\mathbb{R}^3)^2$. Assume there are four elementary transforms, namely $Id := (b_1, b_2) \to (b_1, b_2)$, $S := (b_1, b_2) \to (b_2, b_1)$, $T := (b_1, b_2) \to (b_1, b_2 + b_1)$ and $T^{-1} := (b_1, b_2) \to (b_1, b_2 + (-b_1))$. Now let us consider the following strategy:
If $\mathcal{R}(b_2, b_1)$ and not($\mathcal{R}(b_1, b_2)$) then return S
else if $\mathcal{R}(b_2 + b_1, b_2)$ and not($\mathcal{R}(b_2, b_2 + b_1)$) return T
else if $\mathcal{R}(b_2 - b_1, b_2)$ and not($\mathcal{R}(b_2, b_2 - b_1)$) return T^{-1}
else return Id.

Fact 16. *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two instances of the presented algebra. Under Hypothesis 6, the following assertions are equivalent:*

(*i*) *$\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class.*

(*ii*) *There are two coverings $(a) = (a_1, \ldots, a_n)$ of $\mathcal{I}_1$ and $(b) = (b_1, \ldots, b_n)$ of $\mathcal{I}_2$ such that for all n-tuple of terms $T = (T_1, \ldots, T_n)$, with $(T_1(a), \ldots, T_n(a))$ a covering of $\mathcal{I}_1$, and for all $j \in \{1, \ldots, k\}$,*

$$\mathcal{R}_{i_j}(t_{i_j,1}((T(a))), \ldots, t_{i_1,a_{i_j}}(T(a))) = \mathcal{R}_{i_j}(t_{i_j,1}(T(b)), \ldots, t_{i_1,a_{i_j}}(T(b)))$$

(*iii*) *There are two coverings $(a) = (a_1, \ldots, a_n)$ of $\mathcal{I}_1$ and $(b) = (b_1, \ldots, b_n)$ of $\mathcal{I}_2$ such that for all $\omega \in F^*$, and for all $j \in \{1, \ldots, k\}$,*

$$\mathcal{R}_{i_j}(t_{i_j,1}(\omega(a)), \ldots, t_{i_1,a_{i_j}}(\omega(a))) = \mathcal{R}_{i_j}(t_{i_j,1}(\omega(b)), \ldots, t_{i_1,a_{i_j}}(\omega(b)))$$

[15] Bounded changeness is already required by our Hypothesis 4

Proof. First notice that (*ii*) and (*iii*) are equivalent since elementary transforms are precisely given by terms and they generate the automorphism group of the presented algebra. Now assume that (*i*) is true. Then by Definition 12, there is a strategy–preserving isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$. This isomorphism can be given by two coverings $(a) = (a_1, \ldots, a_n)$ of $\mathcal{I}_1$ and $(b) = (b_1, \ldots, b_n)$ of $\mathcal{I}_2$. Definition 12 requires that for any covering of $\mathcal{I}_1$, i.e. for all n-tuple of terms $T = (T_1, \ldots, T_n)$, with $(T_1(a), \ldots, T_n(a))$ a covering of $\mathcal{I}_1$, the elementary transform output by the strategy is the same for $(T_1(a), \ldots, T_n(a))$ and its image by the extension of the strategy–preserving isomorphism to coverings, which is $(T_1(b), \ldots, T_n(b))$. For the reciprocal, just notice that (a, b) is a strategy preserving isomorphism. □

Definition 13. *Let $\mathcal{R}_1, \ldots, \mathcal{R}_m$ be m relations of respective arities $a_1, \ldots, a_m$ defined on the domain. Let $t_1, \ldots, t_p$ be p terms of the vocabulary and let consider the k' relational terms,*

$$\mathcal{R}_{i_1}(t_{i_1,1}, \ldots, t_{i_1,a_{i_1}}), \mathcal{R}_{i_2}(t_{i_2,1}, \ldots, t_{i_2,a_{i_2}}), \ldots, \mathcal{R}_{i_{k'}}(t_{i_{k'},1}, \ldots, t_{i_k,a_{i_{k'}}})$$

For any $k \in \{1, \ldots k'\}$, the equivalence relation $\equiv_{k,\mathcal{R}}$ on the set of outputs is defined as following. Two output coverings c and c' satisfy $c \equiv_{k,\mathcal{R}} c'$ if and only if for all $j \in \{1, \ldots k\}$,

$$\mathcal{R}_{i_j}(t_{i_1,1}(c), \ldots, t_{i_1,a_{i_j}}(c)) = \mathcal{R}_{i_j}(t_{i_1,1}(c'), \ldots, t_{i_1,a_{i_j}}(c')).$$

So to the k relational term defined by Hypothesis 6, one associates the equivalence relation $\equiv_{k,\mathcal{R}}$ that defines a partition on the set of outputs. The next corollary is an alternative statement of Theorem 1, under Hypothesis 6. Indeed in the statement of the previous fact, the coverings a and b can be taken as outputs.

Corollary 4. *There is a partition in the set of outputs such that if c_1^* and c_2^* are two outputs in the same class, then the isomorphism (eventually the automorphism) defined by (c_1^*, c_2^*) is strategy–preserving, i.e. for all $\omega \in F^*$ and for all $j \in \{1, \ldots k\}$,*

$$\mathcal{R}_{i_j}(t_{i_j,1}(\omega(c_1^*)), \ldots, t_{i_1,a_{i_j}}(\omega(c_1^*)) = \mathcal{R}_{i_j}(t_{i_j,1}(\omega(c_2^*)), \ldots, t_{i_1,a_{i_j}}(\omega(c_2^*)).$$

Usually the input set X is infinite and the number of input classes is finite. The next hypothesis proposes a reasonable way to insure the finiteness of the number of input classes.

Hypothesis 7. *There is a finite number k' of relational terms*

$$\mathcal{R}_{i_1}(t_{i_1,1}, \ldots, t_{i_1,a_{i_1}}), \mathcal{R}_{i_2}(t_{i_2,1}, \ldots, t_{i_2,a_{i_2}}), \ldots, \mathcal{R}_{i_{k'}}(t_{i_{k'},1}, \ldots, t_{i_k,a_{i_{k'}}}),$$

containing the relationnal terms in Hypothesis 6 ($k' \geq k$ and the k first relational terms are those defined by Hypothesis 6) such that the equivalence relations $\equiv_{k,\mathcal{R}}$ and $\equiv_{k',\mathcal{R}}$ defined on the set of outputs satisfy the following property. For all pairs of outputs c and c',

$$c \equiv_{k',\mathcal{R}} c' \quad \Longleftrightarrow \quad \forall \omega \in F^*, \omega(c) \equiv_{k,\mathcal{R}} \omega(c')$$

Fact 17. *Under Hyptheses $1, \ldots, 7$ the number of input classes and the number of rewrite systems associated with an algorithm by Theorem 1 are finite.*

Proof. □

Let us finish this section by two classical particular cases (the two following corollaries) when the domain is ordered and the "algorithm order" is deduced from the order on the domain, i.e. there is a known (pre-)order on the coverings, which comes from a preorder on the domain and the strategy is based only on the (pre)order relation among the elements of the instance. In the following cases the isomorphism Φ is clearly strategy preserving. Then the input classes are set of instances that are equivalent modulo an increasing isomorphism.

Fact 18. *Assume that Hypothesis 6 is satisfied and there is an isomorphism Φ between two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of the presented algebra satisfying: For all $i \in \{1, \ldots, m\}$ and for all $x_1, \ldots, x_{a_i}$ elements of $\mathcal{I}_1$,*

$$\mathcal{R}_i(x_1, \ldots, x_{a_i}) = \mathcal{R}_i(\Phi(x_1), \ldots, \Phi(x_{a_i})).$$

Then $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class.

Corollary 5. *Suppose there is a (pre)order $\leq$ on the domain such that the strategy of the algorithm is based only on the (pre)order among the elements of the instance. More precisely the strategy involves computations in the structure, equality tests and comparing elements. Assume that there is an isomorphism Φ between $\mathcal{I}_1$ and $\mathcal{I}_2$ such that both Φ and Φ^{-1} are increasing with respect to the (pre)order i.e. such that for all dinstinct elements $(x, y) \in \mathcal{I}_1^2$, with $x \neq y$, $x \leq y \Leftrightarrow \Phi(x) \leq \Phi(y)$. Then $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class.*

Fact 19. *Assume that Hypothesis 6 is satisfied and there is an isomorphism Φ between two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ such that the extentions $\tilde{\Phi}$ and $\tilde{\Phi}^{-1}$ to the sets $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ of coverings satisfy the following property. For all $j \in \{1, \ldots, k\}$ and for all covering $c \in \tilde{\mathcal{I}}_1$,*

$$\mathcal{R}_{i_j}(t_{i_j,1}(c), \ldots, t_{i_j,a_{i_j}}(c)) = \mathcal{R}_{i_j}(t_{i_j,1}(\tilde{\Phi}(c), \ldots, t_{i_j,a_{i_j}}\tilde{\Phi}(c)).$$

Then $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class.

Corollary 6. *Suppose there is a (pre)order on the input set such that the strategy of the algorithm is based only on it. More precisely the strategy involves computation of coverings by means of elementary transforms, equality tests and comparing coverings of one instance. Assume that there is an isomorphism Φ between $\mathcal{I}_1$ and $\mathcal{I}_2$ such that the extentions $\tilde{\Phi}$ and $\tilde{\Phi}^{-1}$ to the sets $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ of coverings are both increasing with respect to the (pre)order i.e. such that for all distinct elements $(x, y) \in \tilde{\mathcal{I}}_1^2$, with $x \neq y$, $x \leq y \Leftrightarrow \tilde{\Phi}(x) \leq \tilde{\Phi}(y)$. Then $\mathcal{I}_1$ and $\mathcal{I}_2$ are in the same input class.*

6. Comparing with Gurevich's ASM

For a nice introduction to Abstract State Machines (ASMs), see for example (8; 7).
In (8), Gurevich defines a sequential algorithm to be an object that satisfies three natural postulates: the so-called Sequential Time, Abstract State and Bounded Exploration postulates.
The Sequential Time Postulate associates any algorithm A with a set of *states* $S(A)$ (some of them $I(A) \subset S(A)$ are initial) and a mapping $\tau_A : S(A) \to S(A)$ which modifies a state to another at each step during an execution.
The Abstract State Postulate defines states to be (first order) structures (the vocabulary of a structure may contain function names and relation names) with the same vocabulary and with

the same base set during any execution. Moreover it requires $S(A)$ (and $I(A)$) to be closed under isomorphisms.
The Bounded Exploration Postulate requires the existence of a finite set T of terms (in the common vocabulary of the states) called *the bounded exploration witness* such that if values of two states are the same for $t \in T$ (i.e. if two states coincide over T), then the two states are updated (modified) in the same manner. The elements of T are called critical terms.
An *Abstract Sate Machine* (ASM) of vocabulary Ψ is a state transition system in which at any step a finite number of values of functions is modified. Modifications of a state at each step is done with *a set of updates rules* associated with the state. This consists in finitely many guarded assignment:

$$\text{if } p \text{ then } t := u$$

where t and u are terms over Ψ and p is a conjunction of equalities and disequalities between terms (here a term may be also relational).
In (8), Gurevich showed that for every sequential algorithm (i.e. every process satisfying the three above postulates), there is ASM that step-for-step simulates it. (i.e. there is an ASM with exactly the same runs).
Basic functions whose values are the same in all the states of an ASM are called *static*; the other basic functions are called *dynamic*. Observe that if a basic function is static, it never appears in the left side of an update rule (unless with a trivial update that can be avoided). But the reciprocal is false since the values of a basic function may be different in two different initial states.
In Section 3 we defined two equivalent algorithms to have isomorphic effective execution graphs (Definition 10). Here we will define two equivalent ASM in the same manner: Roughly speaking two ASMs *equivalent* iff they have the "same" runs. More precisely we associate with any ASM, a directed colored graph, called the effective execution graph, where the vertices are the states of the ASM and there is a directed edge from a state A to a sate A', iff the state A is updated to the state A', in some run of the ASM. Moreover one color is associated with any set of parallel update rules used for updating a state of an ASM. We know that the number of colors associate to an ASM is finite (This is a consequence of the Gurevich's bounded exploreness postulate (see (8)). Any edge of the effective execution graph is then colored with the color of the set of parallel update rules it represents. Two ASMs are *equivalent* if and only if there is an isomorphism of directed colored graphs between their effective execution graphs.
There is interesting discussion in (3) about the equality between two algorithms and the difficulty of defining such notion. In (8) two ASMs are called behaviourly equivalent iff they have the same set of states, the same sets of initial states and the same transition function. Our definition of equivalence between ASMs is slightly weaker, since it requires only bijective sets of states, bijective sets of initial states, and the same transition function modulo the bijection between states. In particular, two ASM with different vocabulary may be equivalent.
We call an ASM *always halting* iff all its runs are finite. We call an ASM *standard* if in all its update rules, the left side is a nullary function name of the vocabulary, i.e. any update rule is of the form $c_i = T_i$ where each c_i is a nullary function name of the vocabulary and T_i is any term constructed inductively as usual with the vocabulary of the ASM. Moreover we require all states of a standard ASM to be input states.
By modifying slightly the vocabulary of an ASM, one can easily make it standard[16]:

[16]Most of examples of ASM in (8) are already standard. When an ASM is not standard, the modified equivalent standard ASM is essentially the same. The modification corresponds to introducing some additional redundant variable names in the algorithm simulated by the ASM.

Fact 20. *Any ASM is equivalent to a standard ASM.*

Proof. Consider an ASM A and the bounded exploration witness T for A. We will define a slightly modified equivalent ASM A', i.e. having exactly the same runs. Define a new nullary function name to the vocabulary, for any update rule where the left side is not already a nullary function name: If $f(a_1, \ldots, a_r) = a_0$ is an update rule of A, define a new nullary function name c_i to the vocabulary, for $f(a_1, \ldots, a_r)$. The vocabulary of A' is the union of the vocabulary of A and the new redundant names. The base set of A', (i.e. the base set of all its states) is the same than the base set of the states of A. Any state of A' is a slight modification of a state of A: The interpretation of a term which is also a term in the vocabulary of A, other than a term which is the left side of an assignment and is not an nullary function name, remains unchanged in a state of A'. For any new nullary function symbol c_i introduced as a left side of an update rule $f(a_1, \ldots, a_r) = a_0$, set for c_i the same interpretation than for $f(a_1, \ldots, a_r)$ in the associate state of A. We do not care about the interpretation of $f(a_1, \ldots, a_r)$ in the states of A', since they become useless (For example, the interpretation $f(a_1, \ldots, a_r)$ is `undef`.) Finally update rules of A' are those of A, where newly introduced nullary function names replace all occurences of the terms they are associated with (in the left sides of the assignments, but also in the guards and in the right sides if they are present). Clearly the new witness remains finite. □

Remark 12. *The set of update rules of a state (i.e. a set of parallel atomic update rules of the state) of a standard ASM is of the form:*

$$c_1^{new} = T_1(c_1^{old}, \ldots, c_n^{old}), \ldots, c_n^{new} = T_n(c_1^{old}, \ldots, c_n^{old}), \tag{5}$$

where $c_1, \ldots, c_n$ are the sequence of dynamic nullary function names of the vocabulary of the ASM (an arbitrary fixed order on the nullary function names is fixed), and $T_1, \ldots, T_n$ are n elements of the term algebra $\mathbf{T}(X)$ (with $|X| = n$ variables) constructed with the vocabulary of the ASM.

Let call D the (inalterable) base set of an ASM A. Now we will deal with ASMs such that there is no loss of information during any run: the elements of D that are accessible in a state Σ, (i.e. those that are value of a term in Σ) are exactly those that are accessible in an initial state Σ_0 associated with the state Σ (i.e. those that are value of a term in Σ_0). We will call such an ASM *reversible*:

Definition 14. *Let A be a standard ASM of type Ψ and let n be the number of dynamic nullary function names of A and $c_1, \ldots, c_n$ be a fixed sequence of dynamic nullary functions names of Ψ. Concider the following two conditions.*

(i) For any set of update rules associated with a state of A, defined by (5), there are n elements $T'_1, \ldots, T'_n$ of the term algebra $\mathbf{T}(X)$ such that in any state Γ of A, for all $i \in \{1, \ldots, n\}$,

$$val(T'_i\left(T_j\left((c_k)_{1\leq k\leq n}\right)_{1\leq j\leq n}\right), \Gamma) = val(T_i\left(T'_j\left((c_k)_{1\leq k\leq n}\right)_{1\leq j\leq n}\right), \Gamma) = val(c_i, \Gamma).$$

(ii) Any set of update rules defined by (5) (not only those used by the ASM) and satisfying (i) can be decomposed in terms of the updates rules of the ASM A.

A is called standard reversible *if and only if it satisfies (i). It is called* standard reversible fully acting *if and only if it satisfies (i) and (ii).*
An ASM is called reversible (respectively reversible fully acting) *if and only if it is equivalent to a standard reversible (respectively to a standard reversible fully acting) ASM.*
A sequential algorithm is reversible *if there is a reversible ASM that simulates it step–for–step.*

Now let us a give an equivalent definition of a standard reversible fully acting ASM.

Lemma 2. *Let A be a standard ASM of type Ψ. It is reversible fully acting if and only if there exits a variety $\mathcal{V}$ of algebra of type Ψ and a presentation (X, S) relatively to $\mathcal{V}$ such that the following assertions are satisfied.*

(i) $|X| = n$ *is the number of dynamic nullary function names of* Ψ.

(ii) Any state of A satisfies all the identities of the variety $\mathcal{V}$.

(iii) Let $X = (x_1, \ldots, x_n)$ be a sequence of variables. There is a fixed sequence $(c_1, \ldots, c_n)$ of nullary functions names such that in any state of A, when for all $i \in \{1, \ldots, n\}$, x_i takes the value of c_i, then all the equalities in S are satisfied by the state.

(iv) For any state of A, the set of update rules defines an automorphism of $\mathbf{F}_{\mathcal{V},S}(X)$.

(v) The set of automorphisms defined by the update rules of A is a generating set of the automorphisms group of $\mathbf{F}_{\mathcal{V},S}(X)$.

Proof. Let us deduce Definition 2 from Definition 14. The vocabulary Ψ is the vocabulary of the standard ASM A. The variety is defined by the set (eventually empty) of identities satisfied in all the states of A.
Assertion (i) of Definition 14 introduces equalities between values of terms that are not necessarily consequences of the identities of the variety. (Any side of such an equality is a term inductively defined from the nullary function names). Any set of equalities between terms that generates the closure (for equality) of all these additional equality relations can be taken as the set S of equality relations of the presentation (X, S). From (i) of Definition 14, the equalities in S are satisfied in any state of A. Now complete any set of update rule of a state to a set of rules that updates all the nullary dynamic functions (by eventually adding trivial updating for some of the dynamic nullary functions). Then by fixing a set of variables $(x_1, \ldots, x_n)$, (an (X, S)-covering) any set of update rules moves it to another (X, S)-covering and so defines an automorphism of the presented algebra $\mathbf{F}_{\mathcal{V},S}(X)$, i.e. an automorphism of the variety.
Moreover, any automorphism of the presented algebra $\mathbf{F}_{\mathcal{V},S}(X)$, can be written as the n-uplet of terms expressing images of dynamic nullary functions. It defines then a set of update rules (not necessarily used in any run of the ASM) satisfying Assertion (i) of Definition 14. From Assertion (ii) of Definition 14, it can be decomposed in terms of update rules of the ASM, which shows Assertion (v) above.
Reciprocally, Definition 14 is a trivial consequence of Definition 2.

□

Remark 13. *Any set of update rules associated with a state of A defines a bijection on D^n. All the sets of update rules associated with all states of the ASM generates a group G that is acting on D^n. Whenever the elements of D^n are considered as (X, S)-coverings*[17]*, this is a regular action. So by the orbit-stabilizer theorem different orbits are isomorphic G-sets. Inside an orbit , when one initial (X, S)-covering is fixed, then any set of update rules is uniquely identified with an automorphism of the presented algebra. Of course changing the initial (X, S)-covering, provides a new identification of updates rules with the automorphisms of the presented algebra.*

[17]This means that in the algebra generated by $(a_1, \ldots, a_n) \in D^n$, the ASM considers only equality relations that are consequences of either the identities of the variety or the equality relations of the presentation (X, S), i.e. additional equalities are ignored by the ASM.

Lemma 3. *Any standard reversible ASM is a standard reversible fully acting ASM.*

Proof. As mentioned in the proof of the previous lemma, any set of update rules of a standard ASM A (defined by (5)) satisfying Assertion (i) of Definition 14 defines an automorphism of the presented algebra $\mathbf{F}_{\mathcal{V},S}(X)$, i.e. an automorphism of the presented algebra defined by the ASM A. All the states of an ASM share algebras of a same type and with a same base set. Fixe an (X, S) covering of the new presented algebra in each orbite to associate the sets of update rules with automorphisms. Now let call G the subgroup of automorphisms generated by all the sets of update rules associated with all the states of A[18].
For any initial state Σ_0, where $(c_1^0, \ldots, c_n^0)$ are the values of dynamic nullary functions, define an n-ary relation R on the presented algebra by:

$$R(c_1, \ldots, c_n) = \texttt{true} \quad \Leftrightarrow$$

$(c_1, \ldots, c_n)$ is in the same orbite than $(c_1^0, \ldots, c_n^0)$ under the action of G.

Now consider the presented algebra associated with the standard ASM by the previous lemma with the new n-ary relation R. We claim that by construction G is exactly the group of automorphisms of the presented algebra conserving R, i.e. $h \in G$ iff h is an automorphism of the presented algebra and if for all $(a_1, \ldots, a_n)$ in the algebra,

$$R(a_1, \ldots, a_n) = \texttt{true} \;\Leftrightarrow\; R(h(a_1), \ldots, h(a_n)) = \texttt{true}. \tag{6}$$

Clearly if $h \in G$ then (6) holds.
Let g be an automorphism of the presented algebra (not necessarily in G) that is compatible with the relation R. We will show that $g \in G$. Consider an n-uplet $(a_1, \ldots, a_n)$ in the same orbit than $(c_1^0, \ldots, c_n^0)$. This means that there is $f \in G$, such that $(a_1, \ldots, a_n) = (f(c_1^0), \ldots, f(c_n^0)$. Then $(g(a_1), \ldots, g(a_n))$ is in the same orbit than $(c_1^0, \ldots, c_n^0)$ iff there is $\ell \in G$ such that $(g(a_1), \ldots, g(a_n)) = (\ell(c_1^0), \ldots, \ell(c_n^0)$, i.e. $(g(f(c_1^0)), \ldots, g(f(c_n^0))) = (\ell(c_1^0), \ldots, \ell(c_n^0)$. Since $(c_1^0, \ldots, c_n^0)$ is an (X, S)-covering, $gf = \ell$ and $g = \ell f^{-1} \in G$.
At this point, to continue manipulating algebras, it is convenient to view relations as special functions, marked as relational (as it is done in the work of Gurevich). One requires any algebra type to contain the equality sign, and nullary names `true`, `false` and `undef` and unary name Boole, and the names of the usual Boolean operations. With the exception of `undef` , all these logic names are relational. So the relation R defined above can be seen as a basic function.

□

Remark 14.

- *If a standard ASM A is reversible, then the subalgebra generated by the values of nullary function names remains inalterable during any run of A. Now let A be an ASM with a set of critical terms that is closed under subterms (this can always be assumed without loss of generality). If the ASM A is reversible then the subalgebra generated by the values of critical terms remains inalterable during any run of A.*

- *The variety and the presentation are not uniquely defined in the previous definition. For example one may always consider the variety of all algebras of type Ψ and give the presentation (X, S) relatively to this algebra. Doing so, the set of equalities S may often be infinite.*

[18] in the case when Assertion (ii) of Definition 14 is satisfied, G is the group of all automorphisms of the variety.

- *A usual important particular case is when $S = \emptyset$. In this case, the point (iii) in the previous definition is of course obsolete.*

Theorem 2. *Any reversible, always halting ASM A solves a generalized sorting problem. Any reversible algorithm that halts on all its inputs is solving a generalized sorting problem.*

Proof. Consider a standard ASM A and let us define the associated problem. There is a standard ASM A' equivalent to A. The vocabulary or the type of the problem is the vocabulary of A'. From Definition 2, there is a variety $\mathcal{V}$ and a presentation (X, S) such that the claims $(i),\ldots,(v)$ of the definition are satisfied. In particular, the number of dynamic nullary function names is the rank of the presented algebra i.e. the cardinal of the set of variables X in the presentation (X, S).
Each set of update rules associated with a state of A' defines an automorphism of $\mathbf{F}_{\mathcal{V},S}(X)$ and the set of update rules generates the group of automorphisms $\mathbf{F}_{\mathcal{V},S}(X)$.
The desired representatives are those who are fixed points of the transition function ASM. From the definition of an ASM, being a fixed point of the transition function is expressed with relational terms in the the finite witness of the ASM.

□

Remark 15. *Consider D the base set of an ASM A and the family $\mathcal{K}$ of the states of A, all of them of the same type Ψ. One defines then $\mathbf{F}_{\mathcal{K}}(X)$ the $\mathcal{K}$-free algebra over X as it is classically done*[19] *(and recalled in the preliminairies).*
If a standard ASM is reversible then the variety $\mathcal{V}$ and the presentation (X, S) defined by the definition of reversibility are such that $\mathbf{F}_{\mathcal{K}}(X)$ is a factor of $\mathbf{F}_{\mathcal{V},S}(X)$.

We showed in this section that any sequential algorithm that is step–for–step simulated by a reversible, always halting ASM is solving a generalized sorting problem. Now consider any algorithm in the model defined in the introduction of the paper. Assume moreover that is solving a generalized sorting problem, i.e. satisfying Hyptheses 1 , 2 and 3. If it satisfies Hypothesis 4, then it is "bounded change" but not necessarily "bounded explore". In particular it may be not bounded memory: the strategy to decide which elementary transform to use at a moment may depend on an unbounded history of the algorithm. If one restricts our model to bounded explore algorithms (Hypothesis 6), then one has exactly sequential algorithms that are step–for–step simulated by a reversible, always halting ASM. When Hypothesis 6 is satisfied, our motivation to maintain the strategy depend on a bounded history as expressed in the introduction is to make the algebra related to the algorithm simpler (Otherwise the algebra is often multisort). This is particluarly useful when studying algorithms associated with classical algebras as sets or groups as in the next section.

7. Examples

In the sequel we will illustrate our purpose with examples. We do not give explicitly the rewrite systems in this paper. Most of these rewrite systems are available in (1). Sorting ordered finite sets or multisets by algorithms based on comparing items are here a first example to illustrate our purpose. It is not surprising that when sorting n (totally ordered) items using an algorithm based

[19] At each state X of A, the equality relation between terms induces an equivalence relation on the term algebra, called $\mathcal{R}_X$. We define $\mathcal{R}$ to be the intersection of all $\mathcal{R}_X$'s, with X a state of A. So states of A satisfy all equality relations of $\mathcal{R}$. The free algebra $\mathbf{F}_{\mathcal{V},S}(X)$ is defined by $\mathbf{T}(X)/\mathcal{R}$.

on comparing items, the number of input classes is the number of the underlying order types of the outputs, that is 2^{n-1}. One of these classes concerns inputs that are lists of n items chosen in a linearly ordered domain and such that in a list, no item is chosen more than once. This is the generic case (sorting sets) i.e. when the underlying order type of the output is a linear order. It is of course well–known that when sorting n distinct (totally ordered) items, one considers as inputs the permutations on n elements. But as far as we know there is no result (unless (1)) identifying a sorting algorithm with a rewrite system on the group of permutations. The cases when some items are present more than once in an input (sorting multisets) are the degenerate cases (see Section 2.4). There are $2^{n-1} - 1$ degenerate cases corresponding to the $2^{n-1} - 1$ underlying order types of the outputs.
One other example is the Euclidean algorithm and its generalization to reduce two-dimensional Euclidean lattice bases: The so–called Gaussian algorithm that finds shortest vectors in a two–dimensional lattice. Here the number of input classes is 5. We show that there are four types of lattice with respect to the Gaussian algorithms. When the Gaussian algorithm is running on the basis of a two dimensional lattice of one given type, there is a rewrite system over $GL_2(\mathbb{Z})$ such that the executions are exactly normal forms of this rewrite system. This is the generic case. The Euclidean algorithm is the degenerate case of the Gaussian algorithm. Further development of our method and its applications to other structures will follow soon.

7.1. Sorting sets and multisets

In this section we deal with "the generic case" (any equality in the associated presented algebra is either a consequence of identities of the variety or the equalities of the presentation; here the variety has no identity and there is no equality in the presentation): we consider sorting algorithms running on sequences without repetition. We deal with those algorithms having a strategy exclusively based on comparing elements of the sequence. All the executions of such an algorithm is characterized as normal forms of one rewriting system on the group of permutations of n items.

1. The vocabulary (or type) $\Psi = \emptyset$. The variety of algebra is the variety of sets. The fixed presentation is (X, S) where X is a set of n variables and $S = \emptyset$. So a presented algebra is here a set of cardinality n given with is a sequence of the n elements.
2. An isomorphism between two sets of the same cardinal is just a bijection between the sets. An automorphism is a permutation on a set and the automorphism group of the inputs is just the group of permutations on a set.
3. A covering is an enumeration of the elements. For a set of cardinal n, there are $n!$ coverings.
4. The domain where the sets are chosen is a known and possibly infinite set like $\mathbb{R}$ or $\mathbb{Q}$ or $\mathbb{Z}$. There is a known linear total order in the domain.
5. The problem is : Given a sequence of elements, obtain the generating sequence whose elements are in an increasing order[20].
6. We deal with sorting algorithms whose elementary transforms are a set of permutations and whose strategy for determining the right elementary transform is exclusively based on comparing elements of the instance and eventually on the previously determined elementary transforms. For example, in the bubble sort or the insertion sort, an elementary transform

[20]When working with sequences with no repetition of element, for any input the expected output is unique. Otherwise, i.e. when sorting multisets, the desired output is not unique.

is a swap of two consecutive elements of a sequence. For the selection sort an elementary transform is any swap of two elements of a sequence. Insertion sort is memory-less whereas bubble sort and selection sort are not.

7.2. Sorting sets.

Here the inputs are sequences with no repetition of element and for any input the expected output is unique. Now we notice that Hypothesis 1, 2, 3 and 4 are satisfied. Hypothesis 5 is not relevant here since any instance has a unique output. We also observe that there is an isomorphism between any two instances that is increasing and whose reciprocal isomorphism is also increasing: The unique isomorphism defined by the outputs of two instances. So,

Fact 21. *Consider a sorting algorithm. Assume that elementary transforms of the algorithm are a set of permutations and the strategy for determining the right elementary transform at one step is exclusively based on comparing elements and eventually on the previously determined elementary transforms. Assume that the algorithm is correct, i.e. when running on all sequences of n elements without repetition taken in a linearly ordered set, it outputs the ordered sequence after a finite number of steps. Then the elementary transforms form a generating set of the permutation group on n elements and all the executions of the algorithm are exactly normal forms of one rewrite system on the group of permutations.*

Now let us explicit some examples of algorithm orders. There is one order associated with each sorting algorithm. All these orders have the same minimal element. Notice that there may be different ways to express the order related to an algorithm. The algorithm order may be enriched by additional relations, such that the strategy becomes a greedy strategy: the strategy is to choose at each step the elementary transform that modifies the current generating sequence into the minimal one. If there is several such optimal choice, then one has to add additionnal criterions to determine the elementary transform, for instance dependance to the past of the algorithm.
Let c be a generating sequence of a set of cardinal n. The element of index i in c is denoted $c[i]$. The elementary transform that swaps the elements $c[i]$ and $c[j]$ is denoted by $t_{i,j}$ and t_i denotes the swap of ith elment $c[i]$ and $(i+1)$th element $c[i+1]$ of the generating sequence.

7.3. Insertion sort and bubble sort

First let us define a relation $\mathcal{R}$ among generating sequences:

$$c_1 \mathcal{R} c_0 \quad \Leftrightarrow \quad \exists i \in \mathbb{N} \text{ such that } c_1 = t_i(c_0) \text{ such that } c_1[i] \leq c_1[i+1] \tag{7}$$

Let c be a generating sequence. By considering the total number $n(c)$ of pairs (i, j), $i < j$ such that $c[i] > c[j]$ and observing that if $c_1 \mathcal{R} c_0$ then $n(c_1) \leq n(c_0) - 1$, one knows that there is no cycle in the graph of the relation R. So the reflexive and transitive closure of R is an order and this is the order among generating sequences that we consider.

Moreover since the family $(t_i)_{1 \leq i \leq n-1}$ generates the group $\mathcal{S}_n$ of permutations on n elements, any two elements of $\mathcal{S}_n$ have an upper and a lower bound. This order is an enrichment of both insertion sort order and bubble sort order (the two orders are closed to each other).
In the case of the insertion sort, at any moment during an execution the algorithm determines the elementary transform to apply to its current data c by just computing the length, say i, of the longest increasing prefix of c. While i is less than the cardinal the input list, the current data c is transformed by $t_i(c)$. The algorithm is clearly memory-less.

In the case of the bubble sort, let k be the index of t_k the most recent elementary transform used by the algorithm at a moment during the execution ($k := 0$, if none has been used, i.e. in the beginning). The adequate swap t_j is determined such that $j := k + i$ and $i := \min\{r \in \mathbb{N}, r > 0 : t_{k+r \bmod n-1}(c) < c\}$. To compute the elementary transform to use at a moment, the bubble sort strategy needs the index k of the last elementary transform used. Contrarily to the insertion sort, the bubble sort is not memory-less. (The insertion sort strategy is to take always $k = 0$ in the previous bubble sort strategy, i.e. by removing the memory)

Remark 16. *(1) Both previous strategies are greedy. At each step, they compute inside the set F of elementary transforms, the elementary transform $f \in F$ such that $f(c)$ is a minimum of the set $F(c) := \{g(c) : g \in F\}$.*
(2)There are other generalizations of insertion and bubble sort. Indeed rather than determining t_j as in the previous strategy such that

$$j := k + i \text{ and } i := \min\{r \in \mathbb{N}^* : t_{(k+r) \bmod n}(c) < c\},$$

the index i can be determined to be the second minimum (the third minimum, etc.) of the same set.

7.4. Sorting multisets

In the case of multisets, one has to consider the set structure where some of the elements of the set are "indistinguishable" with respect to the order. For example the multiset $(1, 2, 2, 3)$ may be considered as the set of four elements $\{1, 2_1, 2_2, 3\}$ where 2_1 and 2_2 are two different elements that are indistinguishable with respect to the order : $1 \leq 2_1 \leq 2_2 \leq 3$ and $1 \leq 2_2 \leq 2_1 \leq 3$. Since antisymmetry is no more fulfilled, the new set is then preorderd. The accepted output is not unique: Exactly two datas $(1, 2_1, 2_2, 3)$ and $(1, 2_2, 2_1, 3)$ are acceptable as outputs.
Clearly Hypothesis $1, 2, 3$ and 4 are still satisfied. Hypothesis 5 is also satisfied here since all accepted outputs play a symmetric role. There are exactly as many strategy-preserving isomorphisms as the number of outputs. In our example, between the multisets $\{1, 2_1, 2_2, 3\}$ and $\{10, 20_1, 20_2, 30\}$, there are exactly two strategy-preserving isomorphisms ϕ_1 and ϕ_2, with $\phi_1(1) = \phi_2(1) = 10$, $\phi_1(3) = \phi_2(3) = 30$, $\phi_1(2_1) = 20_1$ and $\phi_1(2_2) = 20_2$ whereas $\phi_1(2_1) = 20_1$ and $\phi_2(2_2) = 20_1$. The isomorphisms ϕ_1 and ϕ_2 are the only ones to be increasing and whose reciprocal isomorphism are also increasing.

Fact 22. *Consider a sorting algorithm. Assume that elementary transforms of the algorithm are a set of permutations and the strategy for determining the right elementary transform at one step is exclusively based on comparing elements of the coverings and eventually on the previously determined elementary transforms. Assume that the algorithm is correct, i.e. when running on all sequences taken in a linearly ordered set, the algorithm outputs one ordered sequence after a finite number of steps. Then the elementary transforms form a generating set of the permutation group and all the executions of the algorithm are exactly normal forms of rewriting systems on the group of permutations. Two multisets are in the same input class if and only if there is an increasing bijection that relates one multiset to the other. The input class are exactly the equivalence class for being related with an increasing bijection.*

8. The Gaussian and Euclidean algorithms

Gaussian algorithm is clearly memory-less but it is much less trivial to exhibit the number of input classes. We will show that in the generic case (reducing a two doimensionnal lattice whose

vectors are in $\mathbb{R}^n$)) there are 4 input classes and there is one additional class for the degenerate case (The Euclidean algorithm).
As mentioned in the last introductory section,

1. we consider here the Abelian free group structure presented by an (ordered) enumeration (without any repetition) of n generators. Such a group is called *a lattice*, the number n is the *dimension* or the rank of the lattice and the sequence of generators is called *a basis*.
2. For a given lattice a generating sequence is then simply a basis of the lattice and it is well-known that there are are infinitely many generating sequences.
   An isomorphism between two such groups is just a bijection transforming a system of generators (basis) of the first lattice into a basis of the second. An automorphism is just changing of a basis of a lattice. The automorphism group of the input is then the linear group $GL_n(\mathbb{Z})$.
3. The domain where lattices are chosen are $\mathbb{R}^p$ or $\mathbb{Q}^p$ or $\mathbb{Z}^p$, with the classical Euclidean structure.
4. The problem is : Given a lattice with an arbitrary basis, obtain a minimal basis, i.e. such that the first vector is a vector of the lattice whose Euclidean norm is minimal, the second is such that its norm is minimal among all vectors of the lattice that can complete the first vector into a basis, etc.

The Gaussian and Euclidean algorithms solve the problem when the dimension equals 2, i.e. when the inputs are two vectors of $\mathbb{R}^p$. In the case of the Gaussian algorithm the two vectors are linearly independent, i.e. any generating family has at least two elements, while in the case of Euclidean algorithm there exists a generating family of cardinal one, i.e. there is an integer relation between the two vectors of the input. The automorphism group is $GL_2(\mathbb{Z})$ that is the 2 by 2 matrices with integer entries and with a determinant equal to 1 or -1. Now let us define two such matrices S and T and I(the identity matrix) by

$$S := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad , \quad T := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad , \quad I := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

In the general exposition of our approach the set of elementary transforms of an algorithm is finite. Here we will make an exception and apply our approach in the case when the set F of elementary transforms that is not finite[21] of the Gaussian and Euclidean algorithms is defined by

$$F := \{T^r S, T^r : r \in \mathbb{Z}\}. \tag{8}$$

Now we have to show that the algorithms are satisfying our hypothesis and to determine the number of input class (4 in the case of the Gaussian algorithm, 1 in the case of the Euclidean algorithm).
First let us exhibit one possible order among the bases.

Definition 15. *Let $b = (b_1, b_2)$ and $b' = (b'_1, b'_2)$ be two sequences of vectors of $\mathbb{R}^p$ and the relation $\mathcal{R}$ isdefined by*

$$(b_1, b_2)\mathcal{R}(b'_1, b'_2) \Leftrightarrow \Big(b' \in F(b)\Big) \quad \textit{and} \quad \Big(\Big(\|b_1\| < \|b'_1\|\Big) \textit{ or } \Big(\|b_1\| = \|b'_1\| \textit{ and } \|b_2\| \leq \|b'_2\|\Big)\Big).$$

[21] One can also consider the easier case of the substractive version of the Gaussian and Euclidean algorithm. Then $F := \{T, S, T^{-1}\}$ is finite; Corollary 7 is no more needed and the strategy is straightforward: It is given in Section 5, after Hypothesis 6, as an example.

The transitive closure of the relation $\mathcal{R}$ is a partial preorder denoted $<$ among the set of sequences of two elements of $\mathbb{R}^p$. In the sequel, we call it lexicographic Euclidean order.

We will show that between two instances (two lattices) L and L' of one input class any isomorphism given by a pair of minimal bases of L and L' is strategy–preserving, i.e. all outputs inside a class play a symmetric role and Hypothesis 5 is statisfied. Let $b^* = (b_1^*, b_2^*)$ be a minimal basis of L and $c^* = (c_1^*, c_2^*)$ a minimal basis of L'. In the sequel we call Φ the isomorphism defined by (b^*, c^*), i.e.

$$\Phi(b_1^*) = c_1^* \quad \text{and} \quad \Phi(b_2^*) = c_2^*. \tag{9}$$

So to an element u of L, $u := xb_1^* + yb_2^*$, with x, y integers, the isomorphism Φ associates an element $\Phi(u)$ defined by $\Phi(u) = xc_1^* + yc_2^*$.

8.1. The Gaussian algorithm

It is well-known and easy to show that a minimal basis b^* for the previous order is a basis satisfying the two following conditions:

$$\|b_1^*\| \leq \|b_2^*\| \tag{10}$$

$$-1/2 \leq (b_1^*, b_2^*)/\|b_1^*\|^2 \leq 1/2. \tag{11}$$

Observe that if (10) is not fulfilled swapping b_1^* and b_2^* gives a new basis that is smaller fot the lexicographic Euclidean order than the original one. If (11) is not fulfilled then there is an integer translation of b_2^* in the direction of b_1^* such that the obtained basis is smaller than the original one. According to the strictness or not of the two previous inequalities, there are different number of acceptable outputs (and/or different relations between them) for the instances of the problem:

1. If both inequalities are strict, then there are 4 possible outputs: $(\varepsilon_1 b_1^*, \varepsilon_2 b_2^*)$ with $(\varepsilon_1, \varepsilon_2) \in \{-1, 1\}^2$
2. If the first inequality is strict and the second is an equality then there are 8 possible outputs: $(\varepsilon_1 b_1^*, \varepsilon_2 b_2^*)$ and $(\varepsilon_1 b_1^*, \varepsilon_2 (b_2^* + b_1^*))$ with $(\varepsilon_1, \varepsilon_2) \in \{-1, 1\}^2$.
3. If the first inequality is an equality and the second one is strict then there are 8 possible outputs: $(\varepsilon_1 b_1^*, \varepsilon_2 b_2^*)$ and $(\varepsilon_1 b_2^*, \varepsilon_2 b_1^*)$ with $(\varepsilon_1, \varepsilon_2) \in \{-1, 1\}^2$.
4. If both inequalities are equalities then there are 24 possible outputs: $(\varepsilon_1 b_1^*, \varepsilon_2 b_2^*)$, $(\varepsilon_1 b_2^*, \varepsilon_2 b_1^*)$, $(\varepsilon_1 b_1^*, \varepsilon_2 (b_2^* + b_1^*))$, $(\varepsilon_2 (b_2^* + b_1^*), \varepsilon_1 b_1^*)$,$(\varepsilon_1 b_2^*, \varepsilon_2 (b_2^* + b_1^*))$, $(\varepsilon_2 (b_2^* + b_1^*), \varepsilon_1 b_2^*)$ with $(\varepsilon_1, \varepsilon_2) \in \{-1, 1\}^2$.

The previous remark shows that there are a priori 4 class of inputs for the Gaussian algorithm. It remains to show that these candidates are indeed input class, i.e. for any two instances inside one of these 4 class, the isomorphism Φ is strategy–preserving. More precisely, we have to prove that whenever two lattices L and L' have minimal bases b^* and c^* both satisfying in the same manner inequalities (10) and (11) (i.e. whether they are equalities or not), then the isomorphism Φ defined by (9) satisfies Definition (12).
Let us define $b = (b_1, b_2)$ and $c := \Phi(b) = (\Phi(b_1), \Phi(b_2)) = (c_1, c_2)$ as arbitrary bases of the lattices L and L'. So there are integers x, y, z and t satisfying

$$|xt - zy| = 1, \quad \begin{cases} b_1 = x\, b_1^* + y\, b_2^* \\ b_2 = z\, b_1^* + t\, b_2^* \end{cases} \quad \text{and} \quad \begin{cases} c_1 = x\, c_1^* + y\, c_2^* \\ c_2 = z\, c_1^* + t\, c_2^* \end{cases} \tag{12}$$

It is easy to see that if $|xt - zy| \neq 1$ then b^* and b do not generate the same lattice. We first give a useful Lemma whose prof is available in appendix.

Lemma 4. *Let $b^* = (b_1^*, b_2^*)$ and $c^* = (c_1^*, c_2^*)$ be two families of linearly independent vectors of $\mathbb{R}^p$, satisfying (10(and (11). Let $(x, y, z, t) \in \mathbb{Z}^4$ be four integers satisfying $|xt - zy| = 1$ and let $b = (b_1, b_2)$ and $c = (c_1, c_2)$ be defined from b^* and c^* by Relation (12). Then $(\|b_2\| - \|b_1\|)(\|c_2\| - \|c_1\|) \geq 0$. Moreover,*

$$(\|b_2\| = \|b_1\| \Leftrightarrow \|c_2\| = \|c_1\|)$$

if and only if

$$(\|b_2^*\| = \|b_1^*\| \Leftrightarrow \|c_2^*\| = \|c_1^*\|) \text{ and } \left(|(b_1^*, b_2^*)|/\|b_1^*\|^2| = 1/2 \Leftrightarrow |(c_1^*, c_2^*)/\|c_1^*\|^2| = 1/2\right).$$

The previous lemma asserts that the sign (non negative, non positive) of the quantity $(\|b_2\| - \|b_1\|)$ depends only on the integers (x, y, z, t) and not on the basis b^*. Moreover for all bases b^*, satisfying (10(and (11) in the same manner, i.e. whether (10) and (11) are equalities or inequalities, the "precise" sign (zero, negative, positive) of the quantity $(\|b_2\| - \|b_1\|)$ depends only on the integers (x, y, z, t) and not on the basis b^* and this is no more true if one of two bases do not satisfy Relations (10) and (11) in the same manner, i.e. (10) and (11) is an equality for one basis, while it is not for the other.

Corollary 7. *Let b^* and c^* be two linearly independent system of $\mathbb{R}^p$ both satisfying in the same manner inequalities (10) and (11) (i.e. with respect to the fact that the inequalities are strict or they are equalities). For all $(x, y, z, t) \in \mathbb{Z}^4$, such that $|xt - zy| = 1$, let $b = (b_1, b_2)$ and $c = (c_1, c_2)$ be defined from b^* and c^* by Relation (12). Then*

$$c < b \Leftrightarrow \tilde{\Phi}(c) < \tilde{\Phi}(b). \tag{13}$$

Proof. The lexicographic Euclidean order is defined by (15) as the transitive closure of the relation $\mathcal{R}$. So, by transitivity, one has just to show (13) when $c \in F(b)$, i.e. we have to show that

$$\forall (r, \varepsilon) \in \mathbb{Z}^2 \times \{0, 1\}^2, \quad T^r S^\varepsilon(b) < b \Leftrightarrow \tilde{\Phi}(T^r S^\varepsilon(b)) < \tilde{\Phi}(b). \tag{14}$$

Clearly $\forall (r, \varepsilon) \in \mathbb{Z} \times \{0, 1\}$, $\tilde{\Phi}(T^r S^\varepsilon(b)) = T^r S^\varepsilon(\tilde{\Phi}(b))$.
Consider the case $\varepsilon = 1$. The basis $b' := T^r S^\varepsilon(b)$ is defined by $b'_1 = b_2$ and $b'_2 = b_1 + rb_2$. It is expressed in the basis b^* by

$$\begin{cases} b'_1 = x' b_1^* + y' b_2^* \\ b'_2 = z' b_1^* + t' b_2^* \end{cases} \quad \text{with} \quad \begin{cases} x' = z \ , \ y' = t, \\ z' = x + rz \ , \ t' = y + rt. \end{cases} \tag{15}$$

Clearly $|x't' - z'y'| = |xt - zy| = 1$. Since $T^r S^\varepsilon(b) < b$, one has $\|b_2\| < \|b_1\|$ or together $\|b_2\| = \|b_1\|$ and $\|b_1 + rb_2\| \leq \|b_2\|$. Lemma 4 asserts that $\|b_2\| < \|b_1\| \Leftrightarrow \|\Phi(b_2)\| < \|\Phi(b_1)\|$. Similarly $\|b_2\| = \|b_1\|$ and $\|b'_2\| \leq \|b'_1\| \Leftrightarrow \|\Phi(b_2)\| = \|\Phi(b_1)\|$ and $\|\Phi(b'_2)\| \leq \|\Phi(b'_1)\|$, provided that b^* and $c^* = \tilde{\Phi}(b^*)$ are both satisfying in the same manner inequalities (10(and (11).
Assume now that $\varepsilon = 0$. Then if $r = 1$ or -1, Relation (14) is deduced from Lemma 4 by changing in Relation (12), the variables x, y, z, t by $x' := z, y' := t, z' := x \pm z, t' := y \pm t$ and observing that $|x't' - z'y'| = |xt - zy| = 1$. If $r > 1$ (the case when $r < -1$ is similar) then from (14), one gets

$$T^{i+1}(b) < T^i(b), \quad \forall i \in \{1, \dots r - 1\},$$

and clearly,

$$T^{i+1}(b) < T^i(b) \Leftrightarrow T(\hat{b}) < (\hat{b}), \text{ where } \hat{b} := T^i(b).$$

Finally by Lemma 4, since b^* and c^* are satisfying (10) and (11) in the same manner, $\|\Phi(\hat{b}_2 + \hat{b}_1)\| - \|\Phi(\hat{b}_2)\|$ and $\|\hat{b}_2 + \hat{b}_1\| - \|\hat{b}_2\|$ have the same sign (positive, negartive, zero).

□

The strategy used by the Gaussian algorithm is the following. If $\|b_2\| < \|b_1\|$ then swap b_1 and b_2. Once $\|b_1\| \leq \|b_2\|$, compute the integer r nearest to $(b_1, b_2)/\|b_1\|^2$ and replace b_2 by $b_2 - rb_1$. So the determined elementary transform is T^{-r} or $T^{-r}S$. It can be easily shown that the obtained b_2 is the smallest (for the Euclidean order) one among $\{b_2 - kb_1 : k \in \mathbb{Z}\}$. Clearly the Gaussian algorithm has a greedy strategy : the adequate elementary transform f is the one such that $f(b)$ is a minimal element (for the lexicographic Euclidean order) of the set $F(b) := \{g(b) : g \in F\}$ where F is the set of elementary transforms of the algorithm. In case of several minimal elements in $F(b)$ the algorithm has a "lazy" strategy: it prefers T^k to T^kS, T^kS to $T^{k'}S$ if $|k| < |k'|$ and it prefers I to any other elementary transform.
Since the Gaussian algorithm's strategy is exclusively based on the lexicographic Euclidean order, the last corollary and Corollary 6 show that whenever two instances of lattices $\mathcal{I}$ and $\mathcal{I}'$ have two minimal bases b^* and c^* satisfying (10) and (11) in the same manner, i.e. (10) (respectively (11)) is an equality for b^* if and only if it is an equality for c^*, then $\mathcal{I}$ and $\mathcal{I}'$ are in the same input class. So the Gaussian algorithm has no more than 4 input classes.
In addition, observe that the strategy of the Gaussian algorithm is memory-less and can be simulated in the following way:

Strategy $\mathcal{S}$:
Input: (b_1, b_2).
Output: $r \in \mathbb{Z}$ and $\varepsilon \in \{0, 1\}$ (the elementary transform will be T^rS^ε)
Initialization: $r := 0; \varepsilon := 0$
If $\|b_1\| > \|b_2\|$ then swap b_1 and b_2; $\varepsilon := 1$;
While $\|b_2 + b_1\| < \|b_2\|$ **do** $b_2 := b_1 + b_2$; $r := r + 1$;
While $\|b_2 - b_1\| < \|b_2\|$ **do** $b_2 := b_2 - b_1$; $r := r - 1$;

By Lemma 4, $(\|\Phi(b_1)\| - \|\Phi(b_2)\|)(\|b_1\| - \|b_2\|) \geq 0$. Moreover $\Phi(b_1 \pm b_2) = \Phi(b_1) \pm \Phi(b_2)$ and the quantities $(\|\Phi(b_2) \pm \Phi(b_1)\| - \|\Phi(b_2)\|)(\|b_2 \pm b_1\| - \|b_2\|) \geq 0$.
Lemma 4 shows also that if two minimal bases b^* and c^* satisfy (10) and (11) but not in the same manner inequalities (i.e. (10) or (11) is a strict inequality for b^* whereas it is an equality for c^*) then there integers x, y, z, t satisfying $|xt - yz| = 1$ and there are bases b and $c = \Phi(b)$ defined with (12), such that $\|b_1\| = \|b_2\|$ but $\|\Phi(b_1)\| \neq \|\Phi(b_2)\|$. In this case clearly the strategy will not be the same running on b and on $c = \Phi(b)$, i.e. the isomorphism Φ defined by (9) is not strategy–preserving.

Corollary 8. *The Gaussian algorithm, running on linearly independent pairs of vectors of* $\mathbb{R}^p$ *(*$p \geq 2$*), has exactly* 4 *input class. For any two instances inside one input class the isomorphisms* Φ *defined by (9) is strategy–preserving.*

So the executions of the Gaussian algorithm running on linearly independent pairs of vectors of $\mathbb{R}^p$ ($p \geq 2$) are normal forms of four rewrite systems over $GL_2(\mathbb{Z})$. These rewrite systems are given in (1).

8.2. The Euclidean algorithm

In the case of the Euclidean algorithm, since the lattice generated by the two input vectors (or more usually by the two input integers) is one dimensional, a minimal basis is always such that the first element is zero. So when the input is a pair of integers (a, b), the only minimal outputs are always $(0, \gcd(a, b))$ and $(0, -\gcd(a, b))$. The order compatible isomorphism between two instances generated by (a, b) and by (a', b') is the one that associates $(0, \gcd(a, b))$ to

$(0, \pm \gcd(a', b'))$. It can be easily shown that the Euclidean algorithm is instance independent and there is only one input class.
It is important to notice that everything is here exactly as in the case of the Gaussian algorithm. The Euclidean algorithm does never use the fact that the input integers a and b are $\mathbb{Z}$-linearly dependent. Moreover the executions of the Euclidean algorithm are, as the executions of the Gaussian algorithm, a set of paths on the Cayley diagram of $GL_2(\mathbb{Z})$, since the generators used by both algorithm are the same (defined by (8))
Finally the executions of the Euclidean algorithm running on pairs of integers are normal forms of only one rewrite system over $GL_2(\mathbb{Z})$.

9. Conclusion

In this paper we propose a new approach to canonically associate a set of rewrite systems to an algorithm, where the executions of the algorithms are normal forms of some rewrite systems. This is a beginning step. Several developments of our approach are works in progress:

- Providing the rewrite rules. They are of course associated with circuits in the Cayley graph of the automorphism group of the presented algebra.
- Considering the case of an algorithm solving the problem of finding a representative for coverings of a fixed presented algebras of any rank (In this paper we dealt with the case of a presented algebra of a fixed rank.)
- Exhibiting relationships between usual properties of an algorithm and classical properties of the associated rewrite systems.
- Generalizing the approach to a wider class of algorithms.
- Dealing with more examples.

Our approach considers an algorithm as a symbolic dynamical system. Observe that the analysis of arithmetic algorithms and string algorithms by using dynamical systems in a random context have already been achieved in the work of Vallée (see for example (5; 12)).

Appendix A. Proof of Lemma 4

Let $b^* = (b_1^*, b_2^*)$ and $c^* = (c_1^*, c_2^*)$ be two families of linearly independent vectors of $\mathbb{R}^p$, satisfying (10) and (11). Let $(x, y, z, t) \in \mathbb{Z}^4$ be four integers satisfying $|xt - zy| = 1$ and let $b = (b_1, b_2)$ and $c = (c_1, c_2)$ be defined from b^* and c^* by Relation (12). Then $(\|b_2\| - \|b_1\|)(\|c_2\| - \|c_1\|) \geq 0$. Moreover,

$$(\|b_2\| = \|b_1\| \Leftrightarrow \|c_2\| = \|c_1\|)$$

if and only if

$$(\|b_2^*\| = \|b_1^*\| \Leftrightarrow \|c_2^*\| = \|c_1^*\|) \text{ and } \left(|(b_1^*, b_2^*)|/\|b_1^*\|^2| = 1/2 \Leftrightarrow |(c_1^*, c_2^*)/\|c_1^*\|^2| = 1/2\right).$$

Proof. From the definition of he basis b (Definition (12)),

$$||b_2||^2 - ||b_1||^2 = (z^2 - x^2)||b_1^*||^2 + (t^2 - y^2)||b_2^*||^2 + 2(zt - xy)(b_1^*, b_2^*).$$

First observe that if x, y, z, t are four integers satisfying $|xt - zy| = 1$ then

$$|x| > |z| \Rightarrow |y| \geq |t| \text{ or } |y| = |z| = 0.$$

Suppose that $|x| > |z|$ and $|t| > |y|$. This is equivalent to $|x| \geq |z| + 1$ and $|t| \geq |y| + 1$, and leads to $|xt| - |zy| \geq |z| + |y| + 1$. So $1 = |xt - zy| \geq ||xt| - |zy|| \geq |xt| - |zy| \geq |y| + |z| + 1$ and finally $|y| = |z| = 0$.
So if $|x| > |z|$ and either $|y| \neq 0$ or $|z| \neq 0$, then by (A.1), $|y| \geq |t|$ and $(z^2 - x^2)(t^2 - y^2) \geq 0$.
Similarly if $|x| < |z|$ and either $|x| \neq 0$ or $|t| \neq 0$, then $|y| \leq |t|$ and $(z^2 - x^2)(t^2 - y^2) \geq 0$. Finally,

$$(y, z) \neq (0, 0) \text{ and } (x, t) \neq (0, 0) \text{ and } |xt - zy| = 1 \Rightarrow (z^2 - x^2)(t^2 - y^2) \geq 0. \tag{A.1}$$

If $y = z = 0$ then $|x| = |t| = 1$ and $||b_2||^2 - ||b_1||^2 = ||b_2^*||^2 - ||b_1^*||^2$ which is either positive or zero, provided that the basis (b_1^*, b_2^*) is minimal (see Relation (10)). The case $x = t = 0$ is similar.
Now by inequality (10),

$$\left|(z^2 - x^2)||b_1^*||^2 + (t^2 - y^2)||b_2^*||^2\right| \geq |z^2 - x^2 + t^2 - y^2|||b_1^*||^2,$$

and this is a strict inequality when (10) is a strict one. On the other hand, thanks to (11),

$$2(zt - xy)(b_1^*, b_2^*) = 2(zt - xy)\frac{(b_1^*, b_2^*)}{||b_1^*||^2}||b_1^*||^2 \leq |zt - xy|||b_1^*||^2,$$

and this is once more a strict inequality when (11) is a strict one.
Now let us show that $|z^2 - x^2 + t^2 - y^2| \geq \sqrt{2}|zt - xy|$. Indeed considering the squares of the both sides and after standard computation, we have the following equivalent assertion

$$x^4 + y^4 + z^4 + t^4 - 2(xt - zy)^2 - 2x^2z^2 - 2y^2t^2 \geq 0.$$

Since (b_1, b_2) is a basis (see (12)), one has $(xt - zy)^2 = 1$ and finally the following equivalent inequality:

$$(x^2 - z^2)^2 + (y^2 - t^2)^2 - 2 \geq 0. \tag{A.2}$$

So if $|x| \neq |z|$ and $xz \neq 0$, then $(x^2 - z^2)^2 > 4$ since it is a square integer different from 0 and from 1. So Inequality (A.2) is strict and the sign of $||b_2||^2 - ||b_1||^2$ is the sign of $(z^2 - x^2)||b_1^*||^2 + (t^2 - y^2)||b_2^*||^2$. So, by (A.1), $||b_2||^2 \neq ||b_1||^2$. More precisely,

$$|x| < |z| \Rightarrow ||b_2||^2 - ||b_1||^2 > 0$$
$$|x| > |z| \Rightarrow ||b_2||^2 - ||b_1||^2 < 0$$

Now observe that if $|x| = |z|$ then $|y| \neq |t|$, otherwise $|xt| = |zy|$ and $|xt - zy|$ equals 0 or $2|xt|$, which contradicts the fact that x, y, z, t are integers satisfying $|xt - zy| = 1$.
So if $|x| = |z|$, $(y^2 - t^2)^2 \geq 1$. Now $(y - t)^2(y + t)^2 = 1$ leads to $(|y|, |t|) \in \{(1, 0), (0, 1)\}$ and $|x| = |z| = 1$, and finally to $|z^2 - x^2 + t^2 - y^2| = 1 = |zt - xy|$. If $(|x|, |z||y|, |t|) \notin \{(1, 1, 1, 0), (1, 1, 0, 1)\}$, the quantity $(y^2 - t^2)^2$ is strictly greater than 2 since it is a square integer strictly greater than 1.

If $|x| = |y| = |z| = 1$ and $t = 0$, say for example $x = y = z = 1$ then

$$||b_2||^2 - ||b_1||^2 = ||b_1^*||^2 - ||b_1^* + b_2^*||^2 \leq 0.$$

The last inequality is an equality if and only if $||b_2^*||^2 = ||b_1^*||^2$ and $(b_1^*, b_2^*)/||b_1^*||^2 = -1/2$.
If $|x| = |z| = |t| = 1$ and $y = 0$, say for example $x = z = t = 1$ then

$$||b_2||^2 - ||b_1||^2 = ||b_1^* + b_2^*||^2 - ||b_1^*||^2 \geq 0.$$

The last inequality is an equality if and only if $||b_2^*||^2 = ||b_1^*||^2$ and $(b_1^*, b_2^*)/||b_1^*||^2 = -1/2$.
If $x = 0$, then $|y| = |z| = 1$. If $|t| \geq 2$, once more Inequality (A.2) is strict and the sign of $||b_2||^2 - ||b_1||^2$ is the sign of $(z^2 - x^2)||b_1^*||^2 + (t^2 - y^2)||b_2^*||^2$, which is positive in this case.
Finally if $x = 0$, and $|y| = |z| = |t| = 1$, say for example $y = z = t = 1$ then

$$||b_2||^2 - ||b_1||^2 = ||b_2^* + b_1^*||^2 - ||b_2^*||^2 \geq 0.$$

The last inequality is an equality if and only if $(b_1^*, b_2^*)/||b_1^*||^2 = -1/2$.
If $z = 0$, then $|x| = |t| = 1$. If $|y| \geq 2$, Inequality (A.2) is strict and the sign of $||b_2||^2 - ||b_1||^2$ is the sign of $(z^2 - x^2)||b_1^*||^2 + (t^2 - y^2)||b_2^*||^2$, which is negative in this case.
Finally if $z = 0$, and $|x| = |y| = |t| = 1$, say for example $x = y = t = 1$ then

$$||b_2||^2 - ||b_1||^2 = ||b_2^*||^2 - ||b_1^* + b_2^*||^2 \leq 0.$$

The last inequality is an equality if and only if $(b_1^*, b_2^*)/||b_1^*||^2 = -1/2$.

Remark 17. *The only cases when $||b_2||^2 = ||b_1||^2$ are the following:*

- $y = z = 0$ *and* $|x| = |t| = 1$ *and* $||b_2^*||^2 = ||b_1^*||^2$
- $x = t = 0$ *and* $|y| = |z| = 1$ *and* $||b_2^*||^2 = ||b_1^*||^2$
- $|x| = |z| = |y| = 1$ *and* $t = 0$ *and* $||b_2^*||^2 = ||b_1^*||^2$ *and* $(b_1^*, b_2^*)/||b_1^*||^2 = \pm 1/2$
- $|x| = |z| = |t| = 1$ *and* $y = 0$ *and* $||b_2^*||^2 = ||b_1^*||^2$ *and* $(b_1^*, b_2^*)/||b_1^*||^2 = \pm 1/2$
- $|y| = |z| = |t| = 1$ *and* $x = 0$ *and* $(b_1^*, b_2^*)/||b_1^*||^2 = \pm 1/2$
- $|x| = |y| = |t| = 1$ *and* $z = 0$ *and* $(b_1^*, b_2^*)/||b_1^*||^2 = \pm 1/2$

This shows that if $||b_2||^2 = ||b_1||^2$ the basis (b_1, b_2) is very close to a minimal one.

Remark 18. *To show that the sign of $|b_2 + b_1|^2 - |b_2|^2$ does not depend on the minimal basis b^*, just change in Relation (12), the variables x, y, z, t by $x' := z, y' := t, z' := x + z, t' := y + t$ and observe that $|x't' - z'y'| = 1$ since $|xt - zy| = 1$.*

□

[1] A. Akhavi and C. Moreira. Another view of the Gaussian algorithm. In *Proceedings of LATIN'2004 - Buenos Aires*. LNCS 2976, pp 474–487. `http://arxiv.org/pdf/0707.0644.pdf <hal-00159666>`

[2] G. Birkhoff On the structure of abstract algebras. In *Proceedings of the Cambridge Philosophical Society* 31 (4), pp. 433-454, 1935.

[3] A. Blass and N. Dershowitz and Y. Gurevich. When are two algorithms the same? In *Bulletin of Symbolic Logic* 15 :2(2009), pp. 145–168.
[4] S. Burris and H.P. Sankappanavar. A course in Universal Algebra. Graduate text in mathematics, Springer, 1981. The Millennuim Edition
[5] H. Daudé and P. Flajolet and B. Vallée. An average-case analysis of the Gaussian algorithm for lattice reduction. In *Combinatorics, Probability and Computing* 6 (4), pp. 397-433, 117.
[6] M. Dauchet. Simulation of Turing machines by a regular rewrite rule. *Theoretical Computer Science*, 2 (103), pp 409–420, 1992.
[7] N. Derschowitz and Y. Gurevich. A natural axiomatisation of computability and proof of Church's thesis. *The Bulletin of Symbolic Logic*, 14 (3), pp 299–350, Sept. 2008.
[8] Y. Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on computational logic*, 1(1), pp 77–111, 2000.
[9] S. Lakshmirvaran and J.-S. Jwo and S.K. Dhall Symmetry in interconnection networks based on Cayley graphs of permutation groups: a survey. *Parallel Computing*, 19, pp 361–407, 1993.
[10] D. Lind and B. Markus. An introduction to symbolic dynamics and coding. Cambridge university press, 1995.
[11] W. Magnus and A. Karass and D. Solitar Combinatorial group theory. Second edition revisited, Dover, 1976. First edition, Interscience Publishers, 1966.
[12] B. Vallée. Dynamical analysis of a class of Euclidean algorithms. *Theoretical Computer Science*, 1-3(297), pp 447–486, 2003.