Turing Machines with Atoms, Constraint Satisfaction Problems, and Descriptive Complexity

Bartek Klin * Sławomir Lasota* Joanna Ochremiak † Szymon Toruńczyk*

University of Warsaw

{klin,sl,ochremiak,szymtor}@mimuw.edu.pl

Abstract

We study deterministic computability over sets with atoms. We characterize those alphabets for which Turing machines with atoms determinize. To this end, the determinization problem is expressed as a Constraint Satisfaction Problem, and a characterization is obtained from deep results in CSP theory. As an application to Descriptive Complexity Theory, within a substantial class of relational structures including Cai-Fürer-Immerman graphs, we precisely characterize those subclasses where the logic IFP+C captures order-invariant polynomial time computation.

Categories and Subject Descriptors F.1.1 [*Models of Computation*]: Turing machines; F.4.1 [*Mathematical Logic*]: Logic and constraint programming; F.2.2 [*Nonnumerical Algorithms and Problems*]: Computations on discrete structures

Keywords Sets with atoms, Turing machines, Constraint Satisfaction Problems, Descriptive Complexity Theory

1. Introduction

Imagine Turing machines which can manipulate not only binary digits, but also *atoms* which come from an infinite, countable set. Moreover, input letters can be finite structures built of atoms. Typical letters include ordered quadruples of atoms, unordered sets of eight atoms, or graphs with ten atoms as nodes. Such machines are called *Turing machines with atoms* [4], or *TMAs* for short.

A TMA is allowed to read and write letters on a tape and store them as parts of its internal state, but it is required to be invariant with respect to bijective atom renaming. For example, if a machine in a state that stores a set of two atoms $\{a, b\}$, upon reading a letter $\{b, c\}$ produces the letter $\{a, c\}$, then in a similar state storing some other set $\{d, e\}$, upon reading $\{e, f\}$, it must produce $\{d, f\}$. Intuitively, atoms have no discernible structure except equality, and a machine may base its actions on comparisons between atoms, but not on the identity of particular atoms. It follows that the language accepted by a TMA is always closed under bijective atom renaming. For example, over the alphabet of unordered pairs of atoms, there is a deterministic TMA recognizing the language of those words which, understood as lists of edges of an undirected graph, describe connected graphs.

In contrast to classical Turing machines, some TMAs do not determinize. In [4], a certain language, over a particular alphabet with each letter built of six atoms, was proved to be recognizable by a nondeterministic TMA (in polynomial time), but not recognizable by any deterministic one. An alphabet for which such a language exists is called *nonstandard*. On the other hand, alphabets such as tuples or finite sets of atoms, are standard: every TMA over them does determinize.

This raises a few questions: how to check whether an alphabet is standard? Is it a decidable property of alphabets? Is the six-atom alphabet of [4] the simplest nonstandard one? In this paper we tackle these questions, and in the process we reveal connections of computation with atoms to some well-studied areas of Theoretical Computer Science, in particular to the theory of Constraint Satisfaction Problems (CSP) [8].

First, we recall from [4] that an alphabet A is standard if and only if a deterministic TMA, given words v and w over A, can decide whether v can be obtained from w by a bijective atom renaming. Then we show that if any deterministic TMA decides this problem then it may be decided by a specific algorithm akin to consistency algorithms studied in CSP theory.

Exploring this connection further, we show how to encode a given alphabet as a finite relational template in the sense of CSP, so that the alphabet is standard if and only if the template is "easy" in a certain well-known sense (specifically, if it admits a majority polymorphism). The latter property is clearly decidable, which gives an effective characterization of standard alphabets. As a direct application, we show that all alphabets with letters built of up to five atoms are standard, so the nonstandard alphabet of [4] is indeed a minimal one. (However, other six-atom nonstandard alphabets do exist.)

The nonstandard alphabet and language defined in [4] resemble the well-known CFI graphs, introduced in [6] to show a limited power of a logic IFP+C over unordered structures (more precisely, an equivalent logic LFP+C was used). We explain this connection by showing that for an alphabet A with atoms, any word defines a relational structure, and over the class of structures obtained in this way, the logic IFP+C captures exactly polynomial time computations by deterministic TMAs. Our results yield a characterization of those classes of structures obtained from words with atoms, over which IFP+C captures polynomial time computations, in the

* Supported by ERC Starting Grant "Sosna".

† Supported by the Polish National Science Centre (NCN) grant 2012/07/B/ST6/01497.

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

http://dx.doi.org/10.1145/2603088.2603135

sense used in Descriptive Complexity. This result can be seen as a common generalization of the Immerman-Vardi and the Cai-Fürer-Immerman theorems. In Section 7 we discuss relations with a paper of Atserias, Bulatov and Dawar [1] and also with the graph isomorphism problem for bounded color classes.

2. Turing Machines with Atoms

We begin by recalling some basic notions of sets with atoms [3, 4], also known as Fraenkel-Mostowski sets [10] or nominal sets [14].

Sets with atoms. Fix a countably infinite set $\mathbb{A}$ of *atoms*. A *set with atoms* is any set that can contain atoms or other sets with atoms, in a well-founded way. Formally, sets with atoms are defined by ordinal induction: the empty set is the only set at level 0, and sets at level α either are atoms (which contain no elements) or contain sets at levels smaller than α.

Examples of sets with atoms include:

- any classical set without atoms,
- the set $\mathbb{A}$ itself,
- for any $n \in \mathbb{N}$, the set $\mathbb{A}^n$ of all n-tuples and $\mathbb{A}^{(n)}$ of non-repeating n-tuples of atoms (tuples may be encoded by usual set-theoretic constructions),
- the set $\binom{\mathbb{A}}{n}$ of sets of atoms of size n,
- the set $\mathbb{A}^*$ of finite words over $\mathbb{A}$,
- the set $\mathcal{P}_{\text{fin}}\mathbb{A}$ of all finite subsets of $\mathbb{A}$, etc.

Bijective atom renaming acts on sets with atoms in a canonical way; for instance, it acts coordinatewise on $\mathbb{A}^{(n)}$. For a set with atoms X and a bijection $\pi : \mathbb{A} \to \mathbb{A}$, by $\pi(X)$ we denote the set obtained by consistently replacing atoms in X and in its elements according to π (formally, this is again defined by ordinal induction). We say that a set $S \subseteq \mathbb{A}$ *supports* X if $X = \pi(X)$ for every π which is the identity on S. For example, a tuple $(a, b, c) \in \mathbb{A}^3$ is supported by the set $\{a, b, c\} \subseteq \mathbb{A}$, but also by any larger set.

A set with atoms is *hereditarily finitely supported* if it has some finite support, each of its elements has some finite support, and so on recursively. In this paper we only consider hereditarily finitely supported sets, and so in the following we omit this qualification.

It is not difficult to prove (see e.g. [10, Proposition 3.4]) that every set with atoms has the least finite support with respect to inclusion. By *the* support of a set with atoms X we will mean the least finite support; we denote it by $\sup(X)$. If X is a finite set of atoms, a finite set of such sets or so on, then $\sup(X)$ is the set of those atoms that appear in X.

Equivariance. A set with atoms is *equivariant* if its support is empty (note that its elements need not have the empty support). The example sets listed above are all equivariant.

A relation $R \subseteq X \times Y$ between two equivariant sets with atoms can be seen as a set with atoms itself. It is equivariant if and only if it is closed under the action of atom renaming, i.e., if for any $x \in X$ and $y \in Y$,

$$(x, y) \in R \text{ implies } (\pi(x), \pi(y)) \in R$$

for any bijection $\pi : \mathbb{A} \to \mathbb{A}$. If the relation is (the graph of) a function $f : X \to Y$, this translates to:

$$f(\pi(x)) = \pi(f(x))$$

for any bijection $\pi : \mathbb{A} \to \mathbb{A}$ and any $x \in X$; i.e., equivariant functions are those that commute with atom renaming. It follows that for any $x \in X$ the support of $f(x)$ is contained in the support of x, since a permutation π which fixes x will also fix $f(x)$. The notion of an equivariant function formalizes the intuition of a function that only cares about atom equality, and does not depend on any other structure of the atoms.

For example, the only equivariant function from $\mathbb{A}$ to $\mathbb{A}$ is the identity, the only equvariant functions from $\mathbb{A}^{(n)}$ to $\mathbb{A}$ are the n projections, and the only equivariant function from $\mathbb{A}$ to $\mathbb{A}^n$ is the diagonal. There is no equivariant function from $\binom{\mathbb{A}}{2}$ to $\mathbb{A}$. Indeed, suppose that $f : \binom{\mathbb{A}}{2} \to \mathbb{A}$ is such that, say,

$$f(\{a, b\}) = a$$

for some $a \neq b \in \mathbb{A}$. Then f is not equivariant, since for a bijection π that swaps a and b:

$$f(\pi(\{a, b\})) = f(\{a, b\}) = a \neq b = \pi(a) = \pi(f(\{a, b\})).$$

Intuitively, there is no way of choosing one atom out of two when all one has is atom equality. However,

$$\big\{(\{a, b\}, a) \mid a, b \in \mathbb{A},\ a \neq b\big\}$$

is an equivariant *relation* between $\binom{\mathbb{A}}{2}$ and $\mathbb{A}$. Note that it relates $\{a, b\}$ both to a and b.

Notational convention. Most often we will make a pragmatic distinction between those sets that we consider as collections of interesting elements (for example, the sets $\mathbb{A}$, $\mathbb{A}^{(n)}$, etc.), and those that serve mostly as elements of other sets (such as particular atoms, tuples of atoms, etc.). The former will usually be equivariant, and will be denoted by capital letters X, Y, A, and referred to as *sets with atoms*. The latter will often have nonempty least support, and will be denoted by small letters x, y, a, b, etc., and referred to as *elements*.

Orbit-finite sets. Elements naturally fall into disjoint *orbits*: x and y are in the same orbit if $\pi(x) = y$ for some bijection $\pi : \mathbb{A} \to \mathbb{A}$. A set is equivariant if and only if it is a union of orbits. For example, $\mathbb{A}$, $\mathbb{A}^{(n)}$ and $\binom{\mathbb{A}}{n}$ comprise one orbit each. The set $\mathbb{A}^2$ decomposes into two orbits – the diagonal and its complement – and $\mathbb{A}^*$ and $\mathcal{P}_{\text{fin}}\mathbb{A}$ have infinitely many orbits. In sets with atoms, sets with finitely many orbits (or *orbit-finite sets*) play the role of finite sets. The *dimension* of a set A, denoted $\dim A$, is the maximal size of the support of any of its elements. Every orbit-finite set has a finite dimension.

For every element x there is a unique single-orbit set – called *the orbit* of x – which contains x, namely the set of all elements of the form $\pi(x)$, where π is an atom permutation. For any finite set of elements there is a smallest equivariant set containing it, which is orbit-finite. Every orbit-finite set arises in this way, so orbit-finite sets indeed are "finite up to atom renaming".

Automorphisms of elements. Let x be an element. A permutation π of $\sup x$ is an *automorphism* of x if x is fixed by some (equivalently, every) permutation of atoms which extends π. Automorphisms of x form a group of permutations of $\sup(x)$, denoted $\mathrm{Aut}(x)$. For example, if x is an unordered pair $\{a, b\}$ of distinct atoms, then $\mathrm{Aut}(x)$ is the symmetric group on $\{a, b\}$. On the other hand, if x is the *ordered* pair (a, b), then $\mathrm{Aut}(x)$ is the trivial group acting on $\{a, b\}$. If x is a finite relational structure with atoms as nodes, such as a graph, then $\mathrm{Aut}(x)$ is the classical automorphism group of x.

When x and y are in the same orbit then $\mathrm{Aut}(x)$ and $\mathrm{Aut}(y)$ are isomorphic as permutation groups. The converse almost holds: if $\mathrm{Aut}(x)$ and $\mathrm{Aut}(y)$ are isomorphic as permutation groups, then the orbit of x maps equivariantly and bijectively to the orbit of y. This means that orbit-finite sets can be presented (up to equivariant bijection) by finite collections of finite permutation groups. Moreover, properties of orbit-finite sets, such as the standardness of orbit-finite alphabets, can be considered as properties of finite permutation groups.

Turing machines. Following [4], a Turing machine with atoms (TMA) is defined exactly as an ordinary Turing machine, but with finite sets replaced by orbit-finite sets with atoms. Thus a TMA consists of an input alphabet A, a work alphabet $B \supseteq A$, and set of states Q with distinguished subsets of initial and accepting states, all these orbit-finite sets with atoms, and an equivariant transition relation

$$\delta \subseteq Q \times B \times Q \times B \times \{-1, 0, 1\}$$

where the last atomless component encodes possible moves of the machine head as usual. An input is a finite word $w \in A^*$, and the definitions of a machine configuration, transition between configurations, machine run, acceptance and the language recognized by a machine are as in the classical case. A machine is deterministic if the transition relation is a partial function and there is exactly one initial state.

Some examples of TMAs were given in [4]; we follow with a few more to illustrate TMA determinization issues.

Example 2.1. Consider the alphabet $\mathbb{A}^2$ of ordered pairs of atoms; a word over this alphabet may be seen as a finite directed (multi-)graph with atoms as vertices. TMAs can decide all standard graph-theoretic properties of such words in a uniform way. Indeed, let the working alphabet of a machine M additionally contain single atoms as letters. The machine, given an input word w, may begin by deterministically computing (and writing to its tape) an ordered list of all atoms in w, in the order of appearance. This is done by checking, for each letter (a, b) of the input, whether a (and, further, b) appear in the list constructed so far, and if not, by adding them to the list. For this, it is important that a deterministic TMA, given a letter (a, b), may compute the atoms a and b; indeed, both projections from $\mathbb{A}^2$ to $\mathbb{A}$ are equivariant functions.

Once an ordered list of all atoms in w is computed, the machine M may simulate any classical, atomless algorithm on finite directed (multi-)graphs, representing every atom in w by the number of its position in the list. If the simulated algorithm is deterministic, then so is M.

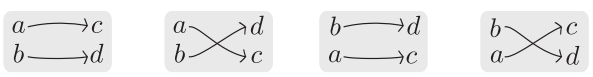
Example 2.2. Consider now the alphabet $\binom{\mathbb{A}}{2}$ of *unordered* pairs of atoms. By analogy to the previous example, a word may now be seen as a finite *undirected* (multi-)graph. The simple approach sketched above does not work as it is. Indeed, as we explained before, there is no equivariant function that, given a letter $\{a, b\}$, returns the atom a. As a result, a deterministic TMA cannot, in general, compute a total order of atoms that appear in a word over $\binom{\mathbb{A}}{2}$.

Fortunately, in this case the problem may be overcome rather easily. Note that taking the intersection, or the difference, of two sets of atoms is an equivariant function. Therefore, if some atom appears in one letter but not in another, then a deterministic TMA can detect this, and output this atom at the end of the tape. This way, the machine outputs all vertices of all non-isolated edges, in some order. Based on this order, a machine may again simulate any classical algorithm on finite undirected (multi-)graphs as in Example 2.1, with the only difference that it must remember that the input graph has a certain number of isolated edges that are not represented in the computed list of atoms.

Example 2.3. Consider the alphabet A of unordered pairs of disjoint, ordered pairs of atoms. Its letters are of the form

$$l = \{(a, c), (b, d)\}$$

for distinct $a, b, c, d \in \mathbb{A}$, and we may draw them as graphs:



Above, the same letter is depicted in four different ways, depending on the ordering of atoms chosen in the picture. It has a four-element support and two automorphisms:

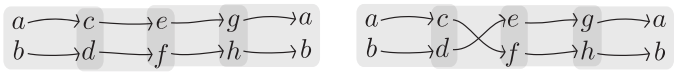
$$\sup(l) = \{a, b, c, d\} \qquad \mathrm{Aut}(l) = \{(),\ (a\ b)(c\ d)\}.$$

Here and in the following, we use the usual cycle decomposition notation to describe finite permutations.

Given a letter as above, we shall call the set $\{a, b\}$ its *left bag* and $\{c, d\}$ its *right bag*. Consider the language $L \subseteq A^*$ of words $l_1 l_2 \cdots l_n$ such that:

(i) for $1 \leq i < n$, the right bag of l_i equals the left bag of l_{i+1},
(ii) the right bag of l_n equals the left bag of l_1,
(iii) otherwise, bags are pairwise disjoint, and
(iv) it is possible to choose one pair of atoms (an "edge") from each l_i so that the chosen edges form a directed cycle of length n.

For example, the four-letter word on the left belongs to L, and the one on the right does not, as it fails the condition (iv):



An equivalent (slightly informal) phrasing of condition (iv) is that the atoms can be arranged in such a way, that the letters have no crossings.

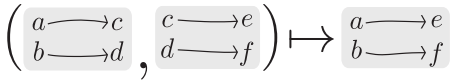
To investigate the recognizability of L, note that functions that return the left and the right bag of a given letter:

$$\{a, b\} \hookleftarrow \{(a, c), (b, d)\} \mapsto \{c, d\}$$

are equivariant, and therefore computable in a single step by a deterministic TMA. Since comparing two bags for equality (and checking whether they are disjoint) is also deterministically computable, it is clear that a deteministic TMA can check conditions (i)-(iii) in the definition of L. These conditions ensure that the input word has the shape of a circular band; the remaining condition (iv) says that it is a simple band, and not a "Möbius strip".

A *non*deterministic TMA can check the condition (iv) easily, guessing one edge from each letter of the input, writing them on the tape, and then deterministically checking that they form a directed cycle. For this, the work alphabet should be extended to include single edges, i.e., ordered pairs of atoms.

Although a deterministic TMA is unable to guess an edge from a letter of the alphabet A, condition (iv) can still be checked deterministically. To this end, note that two letters of A that share a bag, may be deterministically composed with an equivariant function that acts as follows:



A deterministic machine can sequentially compose the input letters from the input in this way, storing intermediate results in its state, and finally check that the structure obtained at the end is of the form:

.

The above deterministic construction relies on the fact that non-local dependencies on bags in the input word may be encoded as small structures of atoms (i.e. intermediate results of the composition process). In [4], an alphabet was proposed where this fortunate property fails, and as a consequence, TMAs do not determinize. We briefly recall that example now.

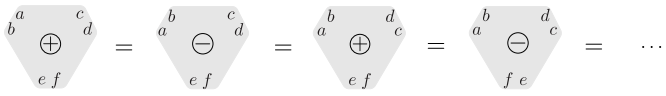
Example 2.4. Generalizing Example 2.3, consider an alphabet A whose letters are four-element sets of atom triples of the following shape, containing six atoms altogether:

$$l = \{(a, c, e), (a, d, f), (b, c, f), (b, d, e)\}.$$

It is straightforward to check that l has four automorphisms:

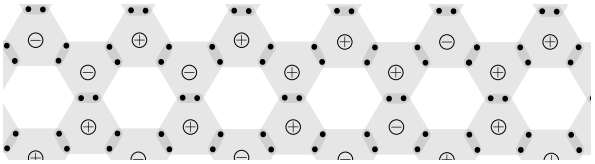
$$\sup(l) = \{a, b, c, d, e, f\}$$
$$\mathrm{Aut}(l) = \{(),\ (a\ b)(c\ d),\ (a\ b)(e\ f),\ (c\ d)(e\ f)\}.$$

A letter like this may be depicted in eight different ways, as a hyperedge on six vertices, with a positive or negative sign:



The rule is that each time a pair of atoms at some corner exchanges positions, the sign changes. This is analogous to Example 2.3, where each time a pair exchanges positions, then a crossing in the graph either appears or disappears.

In the alphabet of Example 2.3, an automorphism of a letter could swap a pair of atoms in a bag if and only if it swapped the remaining atoms as well. Here, the situation is a little more complicated: there are three exchangeable pairs of atoms in a letter, and an automorphism can perform a swap in any two (but not all three) of them. This enables much more complicated dependencies between bags. In [4], letters of A were used to cover a surface (specifically, a torus) as depicted below (atoms are represented by dots):

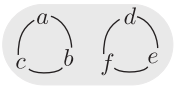


One then considers the language of words whose letters form such a torus (where the correct shape is ensured by conditions analogous to (i)-(iii) in Example 2.3) and satisfy an additional requirement analogous to (iv) in Example 2.3: that one can swap the atoms in such a way that each letter gets a positive sign. Note that swapping a pair of atoms changes the signs of both letters which contain this pair. A nondeterministic TMA can recognize this language similarly to Example 2.3, but, as is proved with a geometric argument, a deterministic TMA cannot recognize it. More details on this can be found in [4].

Example 2.5. Another way of arranging six atoms in a letter is a two-element set of disjoint three-element sets:

$$l = \big\{\{a, b, c\}, \{d, e, f\}\big\}$$

which may be presented as an undirected graph:



A letter like this has support $\{a, b, c, d, e, f\}$ and $3! \cdot 3! \cdot 2 = 72$ automorphisms. As we will show in Section 5, TMAs over this alphabet A do not determinize. On the other hand, as will also easily follow from our results, machines over *ordered pairs* of disjoint three-element sets of atoms, or over two-element sets of disjoint *ordered triples* of atoms, do determinize.

An alphabet will be called *standard* if all nondeterministically recognizable languages over it are also deterministically decidable; otherwise it is *nonstandard*. Our aim is to provide an effective characterization of these properties.

3. Word isomorphism and (k, l)-consistency

Recall that for any element x, an *automorphism* of x is a bijection on its support which extends to a bijection of atoms that fixes x. More generally, an *isomorphism* from x to y is a bijection from $\sup(x)$ to $\sup(y)$ which extends to an atom bijection that maps x to y. Note that x and y are isomorphic in this sense if and only if they are in the same orbit.

The *word isomorphism problem* for an orbit-finite alphabet A is the language

$$\mathit{Iso}_A = \{vw \in A^* \mid v \text{ is isomorphic to } w\}.$$

We recall the following theorem from [4]:

Theorem 3.1. *An alphabet A is standard if and only if the language Iso_A is decidable by a deterministic TMA.*

We will now study the word isomorphism problem in more detail. As a first step to test isomorphism, a TMA may determine whether two words are *similar*, as described below.

3.1 Word similarity

Bags. We say that two atoms *coappear* in a word $w \in A^*$ if they belong to its support and belong to the supports of exactly the same letters of w. Coappearance is an equivalence relation on $\sup(w)$, and its equivalence classes will be called *bags* of w, a notion that appeared in Example 2.3. A bag is either disjoint from or contained in the support of any letter of w, moreover it is determined by the sequence of letters which contain it. The set of all bags in w will be denoted by $\mathrm{Bags}(w)$. It inherits a total ordering from the lexicographic ordering on subsequences of w.

Similar words. Suppose that two words v and w over A have the same length and the same number of bags. We say that a letter of v *corresponds* to a letter of w if they are on the same positions in those words. Similarly a bag B in v *corresponds* to a bag $\tilde{B}$ in w if for every letter of v which contains B, the corresponding letter in w contains $\tilde{B}$, and vice versa.

We say that the words v and w are *similar* if each bag of v has a corresponding bag of the same size in w, and vice versa. Similar words induce a bijective correspondence between their bags. When v and w are understood from the context, we denote by $\tilde{B}$ the bag in w corresponding to a bag B in v.

Lemma 3.2. *There exists a deterministic TMA which determines whether two input words $w, v \in A^*$ are similar.*

Note that similar words are not necessarily isomorphic, as witnessed by the two four-letter words in Example 2.3.

3.2 Consistency

We now go beyond local interactions between intersecting letters and proceed to a more refined analysis of the structure of the words v and w. In this section, fix two similar words v and w over A.

Translations. An isomorphism σ from v to w decomposes into a family of bijections – one bijection σ_B from B to $\tilde{B}$ per each bag in v. In this section, we study families of bijections between corresponding bags which are "candidates" for forming a global isomorphism from v to w.

Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be some family of bags of v. A *translation* on *domain* $\mathcal{B}$ is a family of bijections $\sigma = (\sigma_B)_{B \in \mathcal{B}}$, where σ_B is a bijection from B to $\tilde{B}$. The *size* of σ is the size of its domain. A translation σ *covers* a letter l of the word v, if its domain contains all the bags of l.

Local consistency. A translation σ is *locally consistent* if it induces an isomorphism of every letter of v which it covers to the corresponding letter in w. Note that an isomorphism between v and w induces a locally consistent translation on the domain of all bags of v. Conversely, any locally consistent translation on that domain gives an isomorphism from v to w.

Example 3.1. Consider the pair of words v, w from Example 2.3. Both have four bags: $\{a, b\}, \{c, d\}, \{e, f\}, \{g, h\}$. For $i = 1, 2, 3, 4$, let σ_i map the ith bag of v identically to the ith bag of w. The translation (σ_1, σ_2) is locally consistent: it covers the first letter of v, and maps it isomorphically to the corresponding letter of w. However, the translation $(\sigma_1, \sigma_2, \sigma_3)$ is not locally consistent, as it fails to map the second letter of v to the second letter of w.

A consistency algorithm. We now sketch a variant of the (k, l)-consistency algorithm (see e.g. [2, 8]), adjusted to finding word isomorphisms. The relationship to the standard (k, l)-consistency algorithm will become clear in Sec. 4.

The algorithm has two natural numbers $k < l$ as parameters, and takes a pair of similar words v and w as input.

Let $\mathcal{F}$ be a collection of locally consistent translations of size at most l. We say that $\mathcal{F}$ is (k, l)-*consistent* if:

(1) $\mathcal{F}$ is *downward-closed*: if $\sigma \in \mathcal{F}$ is a translation and $\mathcal{B}$ is a subset of its domain, then the restriction $(\sigma_B)_{B \in \mathcal{B}}$ is in $\mathcal{F}$.

(2) $\mathcal{F}$ is *weakly upward-closed*: if $\sigma \in \mathcal{F}$ is a translation of size at most k and $\mathcal{B}$ is a family of at most l bags of v that contains the domain of σ, then there is a translation $\bar{\sigma} \in \mathcal{F}$ with domain $\mathcal{B}$ which extends σ.

Given two input words v, w, the (k, l)-*consistency algorithm* computes the largest (k, l)-consistent collection of locally consistent translations. The algorithm starts with the collection of all locally consistent translations of size at most l, and repeatedly removes all translations σ that falsify condition (1) or (2), until a fixpoint is reached. The result of this procedure is denoted $\mathit{cons}_{k,l}(vw)$.

The algorithm for computing $\mathit{cons}_{k,l}(vw)$ can be carried out by a deterministic TMA, where letters of the work alphabet include families of consistent translations over a common domain of size at most l. This work alphabet is (contained in) an orbit-finite set, and admits the operations needed for the fixpoint computation of $\mathit{cons}_{k,l}(vw)$.

If v and w are isomorphic then $\mathit{cons}_{k,l}(vw) \neq \emptyset$. Indeed, an isomorphism from v to w induces a locally consistent translation on the domain of all bags, and all its subfamilies of size at most l form a (k, l)-consistent collection.

We say that the alphabet A has *width* (k, l) if the other implication holds, i.e., if for any pair of similar words $v, w \in A^*$, $\mathit{cons}_{k,l}(vw) \neq \emptyset$ if and only if v and w are isomorphic.

The following theorem says that the (k, l)-consistency algorithms are in a sense "universal" for recognizing Iso_A.

Theorem 3.3. *For any alphabet A, the language Iso_A is recognized by a deterministic TMA if and only if there exist numbers k, l such that A has width (k, l).*

One implication is easy: if A has width (k, l) then the language Iso_A is deterministically recognizable (in polynomial time) by the TMA which computes $\mathit{cons}_{k,l}(vw)$ and tests whether the result is nonempty. For the other implication, suppose that Iso_A is recognized by a deterministic TMA M and that the family $\mathcal{F} = \mathit{cons}_{k,l}(vw)$ is nonempty (for sufficiently large k, l, depending on M). Roughly, one then shows that an accepting run of M over the word vv can be translated, using the family $\mathcal{F}$, into an accepting run over vw, implying that $vw \in \mathit{Iso}_A$.

Together with Theorem 3.1, this gives a characterization of standard alphabets in terms of width. However, it may not be clear how the existence of a finite width might be decided. In the next section, we encode the existence of an isomorphism between similar words as a constraint satisfaction problem, to draw on the rich body of results known about those.

4. Constraint Satisfaction Problems

An *instance* of a Constraint Satisfaction Problem (CSP) [8] consists of a set of *variables*, a set of *values*, called its *domain*, and a family of *constraints*. Each constraint is of the form $(\bar{v}, R)$, where $\bar{v} = (v_1, \ldots, v_n)$ is a tuple of variables, and R is an n-ary relation over the domain; we say that the tuple $\bar{v}$ is *constrained* to R. For a given instance, a *partial assignment* is a partial mapping of the variables to the domain. A partial assignment f is called a *partial solution* if it satisfies all the constraints, i.e., if $\bar{v}$ is constrained to R and f is defined over $\bar{v}$, then $(f(v_1), \ldots, f(v_n)) \in R$. We drop the qualifier *partial* if the mapping is total.

We will be interested in the case where all the above sets are finite.

4.1 Instances

Fix an orbit-finite alphabet A. The aim is to reduce the word isomorphism problem over A to a CSP. More precisely, given two similar words v and w over A, we will construct an instance which has a solution if and only if v and w are isomorphic. The idea is that each bag in v is a variable, and an assignment will assign to it a bijection to the corresponding bag in w. In order to describe bijections between bags using elements of a finite domain, we need to introduce canonical representations of various bags. The precise definition follows.

Maps and atlases. For a bag B, define $[B] = [n] = \{1, 2, \ldots, n\}$ where n is the size of B. A *map* of a bag B is any bijection from B to $[B]$, or equivalently, an ordering of the elements of B. An *atlas* α on a word v over A is a family of maps: one map α_B per each bag B of v.

Fix two similar words v and w over A, and let α and β be their atlases. Then $[B] = \left[\tilde{B}\right]$ for any bag B in v, and any permutation τ_B of $[B]$ induces a bijection σ_B from B to $\tilde{B}$:

$$\sigma_B = \alpha_B \cdot \tau_B \cdot (\beta_{\tilde{B}})^{-1} \tag{1}$$

(we use left-to-right function composition here). The correspondence of σ_B and τ_B is bijective. Our aim is to define a CSP instance whose solutions are families $(\tau_B)_{B \in \mathrm{Bags}(v)}$, where each τ_B is a permutation of $[B]$, such that the corresponding translation $(\sigma_B)_{B \in \mathrm{Bags}(v)}$ is locally consistent, and so induces an isomorphism from v to w.

The instance. For fixed atlases α and β, we define the following instance, denoted $I^{v,\alpha}_{w,\beta}$. Its variables are all bags of v. Its domain is the disjoint union:

$$D_A = \mathcal{S}_{[1]} + \mathcal{S}_{[2]} + \mathcal{S}_{[3]} + \cdots + \mathcal{S}_{[\dim A]} \tag{2}$$

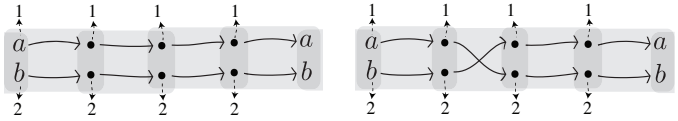
where $\mathcal{S}_{[n]}$ is the group of all permutations of $[n]$. Note that $\dim A$ is the maximal possible size of a bag in a word over A.

Most importantly, there are the constraints. They correspond to letters of v as follows. For a letter l of v, let $B_1, \ldots, B_n$ be its bags in increasing order. The tuple $(B_1, \ldots, B_n)$ is constrained to the set R_l of tuples $(\tau_1, \ldots, \tau_n) \in \mathcal{S}_{[B_1]} \times \cdots \times \mathcal{S}_{[B_n]}$ such that the translation $(\sigma_1, \ldots, \sigma_n)$ induced via (1) is locally consistent, i.e., forms an isomorphism from l to the corresponding letter of w. Note that R_l can be seen as an n-ary relation over D_A. Such are the constraints of the instance $I^{v,\alpha}_{w,\beta}$.

The construction implies that partial assignments for $I^{v,\alpha}_{w,\beta}$ correspond bijectively to translations from v to w. Moreover, a partial assignment is a partial solution if and only if the corresponding translation is locally consistent. The correspondence preserves inclusions of partial assignments and translations.

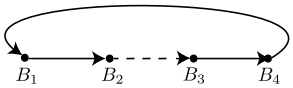
Example 4.1. Over the alphabet consisting of letters $\{(a, c), (b, d)\}$, consider the two words v, w from Example 2.3, depicted below

with most atoms represented by dots. Some atlases α, β of v, w are given; they assign numbers to atoms.



The instance has four variables B_1, B_2, B_3, B_4 corresponding to the four bags of v. There are four constraints, corresponding to the letters of v. The first letter constrains the pair (B_1, B_2) to the set of pairs $R = \{(id, id), (\sigma, \sigma)\} \subseteq \mathcal{S}_{[2]} \times \mathcal{S}_{[2]}$ (where σ denotes the transposition on $[2]$) because only these correspond to isomorphisms from the first letter of v to the first letter of w. The pair (B_2, B_3), however, is constrained to the set of pairs $R' = \{(id, \sigma), (\sigma, id)\}$.

The resulting instance $I^{v,\alpha}_{w,\beta}$ is drawn below. Solid edges represent the constraint R, the dashed edge – the constraint R'.



For different atlases on the same v and w the instance may look different, but it will always have one or three dashed edges.

4.2 Templates

If $T = (D, R_1, R_2, \ldots, R_n)$ is a relational structure, then we say that an instance is *over the template* T if its domain is D and each relation occurring in a constraint is one of the relations $R_1, \ldots, R_n$.

Fix an orbit-finite alphabet A. We define the template T_A as the set D_A from (2) together with all relations on D_A which appear in the constraints of all instances of the form $I^{v,\alpha}_{w,\beta}$. There are only finitely many such relations because each of them has arity at most $\dim A$. Moreover, each relation in T_A has a special structure: it is a coset.

If G is a group, then a *coset* in G is any subset of the form $H \cdot g = \{h \cdot g : h \in H\}$, for a subgroup H of G and $g \in G$.

Lemma 4.1. *Let v, w be similar words over A and α, β their atlases. For any letter l of v with bags $B_1, \ldots, B_n$, the relation R_l is a coset in the group $\mathcal{S}_{[B_1]} \times \cdots \times \mathcal{S}_{[B_n]}$, and a subgroup if $v = w$ and $\alpha = \beta$.*

Example 4.2. Let A be the alphabet from the previous example. Then $D_A = \mathcal{S}_{[1]} + \mathcal{S}_{[2]} + \mathcal{S}_{[3]} + \mathcal{S}_{[4]}$. Some of the relations of T_A include the binary relations $R, R' \subseteq \mathcal{S}_{[2]} \times \mathcal{S}_{[2]}$ which appeared in Example 4.1. Note that R is a subgroup and R' is its coset in $\mathcal{S}_{[2]} \times \mathcal{S}_{[2]}$. Other relations in T_A include a unary relation U which is a subgroup of $\mathcal{S}_{[4]}$ isomorphic to $\mathrm{Aut}(l)$, where l is a letter of A. The relation U arises in the instance $I^{l,\alpha}_{l,\alpha}$, where α is some map on l.

4.3 Templates of bounded width

Given an instance I, the well-known (k, l)-consistency algorithm (see e.g. [2, 8]) is completely analogous to the one in Sec. 3.2, but it computes partial solutions instead of locally consistent translations. We say that a template T has *width* (k, l) if for any instance I over T, $\mathit{cons}_{k,l}(I) \neq \emptyset$ if and only if I has a solution. A template of *bounded width* is a template of width (k, l), for some natural numbers k, l.

By construction of the instance $I^{v,\alpha}_{w,\beta}$, its partial solutions correspond to locally consistent translations, so we have:

Fact 4.2. *For any words v, w over A and their atlases α, β, $\mathit{cons}_{k,l}(vw) \neq \emptyset$ if and only if $\mathit{cons}_{k,l}(I^{v,\alpha}_{w,\beta}) \neq \emptyset$.*

It is also not difficult to prove the following.

Proposition 4.3. *Fix an alphabet A, and let k, l be natural numbers. Then the following conditions are equivalent:*

1. *The alphabet A has width (k, l),*
2. *The template T_A has width (k, l).*

The implication from (2) to (1) is immediate from Fact 4.2, since any pair of words v, w induces an instance $I^{v,\alpha}_{w,\beta}$ over T_A, and the consistency algorithm over $I^{v,\alpha}_{w,\beta}$ simulates the algorithm over vw. The converse implication is similar, and uses the the fact that an arbitrary instance I over T_A can be converted into a homomorphically equivalent instance of the form $I^{v,\alpha}_{w,\beta}$.

4.4 Majority polymorphisms

The bounded width property is decidable for any template [2, 13], but for templates T_A it can be analyzed further using the following fundamental notions of CSP theory.

Consider a template T over a domain D and a function $f : D^n \to D$. A relation R in T is *compatible with* f if by applying f (coordinatewise) to n tuples from R, we get a tuple in R. We say that f is a *polymorphism* of T if all the relations in T are compatible with f. A *majority* polymorphism m is a ternary polymorphism such that for all $x, y \in D$ we have

$$m(x, x, y) = m(x, y, x) = m(y, x, x) = x. \tag{3}$$

Another example of a ternary polymorphism is a *Maltsev* polymorphism M which for all $x, y \in D$ satisfies

$$M(x, x, y) = M(y, x, x) = y. \tag{4}$$

For every alphabet A, the template T_A has a Maltsev polymorphism M defined by:

$$M(x, y, z) = xy^{-1}z, \quad \text{if } x, y, z \in \mathcal{S}_{[n]} \text{ for some } n,$$

and for other arguments defined arbitrarily but satisfying (4). This is a polymorphism since, by Lemma 4.1, every relation in T_A is a coset.

A relational structure is a *core* if every endomorphism of it is a bijection. Every relational structure T has an induced substructure C which is a core and which is a retract of T, i.e. there is a homomorphism from T onto C which is the identity on C. We call C *the core* of T (a core is unique up to isomorphism).

The key technical result of CSP theory that we need is:

Theorem 4.4 ([7])**.** *If a core template has a Maltsev polymorphism and has bounded width, then it has a majority polymorphism.*

The following lemma does not hold for arbitrary relational templates, but it does for templates T_A:

Lemma 4.5. *For any alphabet A, the template T_A has a majority polymorphism if and only if its core has a majority polymorphism.*

From this we deduce:

Lemma 4.6. *For any alphabet A, the template T_A has bounded width if and only if T_A has a majority polymorphism.*

Proof. The right-to-left implication is classical [8] and holds for any template T. To prove the left-to-right implication, let C be the core of T_A, and let r be a homomorphism of T_A onto C which is the identity on C. Then C has a Maltsev polymorphism $r \circ M$, where M is a Maltsev polymorphism of T_A. It is well known (and easy to see) that a template has bounded width if and only if its core has; therefore, C has bounded width. By Theorem 4.4, C has a majority polymorphism. Therefore, by Lemma 4.5, T_A has a majority polymorphism. □

We arrive at the main result of this paper:

Theorem 4.7. *An alphabet A is standard if and only if its template T_A has a majority polymorphism.*

Proof. Follows from Theorems 3.1 and 3.3, Proposition 4.3, and Lemma 4.6. □

5. Applications

Theorem 4.7 gives a decidable (in the classical sense) characterization of standard alphabets. Indeed, given A represented as a finite collection of finite permutation group, the template T_A can be computed, since all relations in it arise from words of bounded size. Then one can try every ternary function on the domain of T_A for the polymorphism property and majority equations. (Note that there are also more efficient, polynomial time procedures of testing whether a template has bounded width [2, 13].) Rather than illustrate this tedious procedure, we show on examples how the existence of a majority polymorphism, or the lack thereof, may be deduced by studying the structure of a template T_A.

5.1 Alphabets of dimension up to five are standard

We will show that any alphabet A of dimension (at most) five is standard. The domain of the template T_A of such an alphabet has (at most) five components:

$$D_A = \mathcal{S}_{[1]} + \mathcal{S}_{[2]} + \mathcal{S}_{[3]} + \mathcal{S}_{[4]} + \mathcal{S}_{[5]}.$$

By definition of T_A, every relation in it is of the form:

$$R \subseteq \mathcal{S}_{[n_1]} \times \mathcal{S}_{[n_2]} \times \cdots \times \mathcal{S}_{[n_k]} \tag{5}$$

where $\sum_{i=1}^{k} n_i \leq 5$. Moreover, R is always a coset in the product group on the right.

For a function $m : D_A^3 \to D_A$ to be a polymorphism, it is required that every relation R in T_A is compatible with m, i.e., that whenever the first three rows below are tuples in R, the resulting tuple also is in R, where $\rho_i = m(\sigma_i, \tau_i, \nu_i)$:

$$m\begin{array}{|cccccc|}\hline \sigma_1 & \sigma_2 & \sigma_3 & \cdots & \sigma_k & \in R \\ \tau_1 & \tau_2 & \tau_3 & \cdots & \tau_k & \in R \\ \nu_1 & \nu_2 & \nu_3 & \cdots & \nu_k & \in R \\ \hline \rho_1 & \rho_2 & \rho_3 & \cdots & \rho_k & \in R. \\ \hline \end{array}$$

We will show that there exists a majority function for which *all* coset relations R as in (5) are compatible, thereby proving that every alphabet of dimension A is standard.

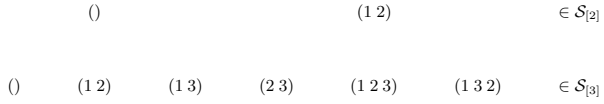
According to (5), every relation R determines a tuple of components $S_{[n_i]}$ so that ith element of a tuple in R belongs to the ith component. As a result, to define a polymorphism m on T_A, it is enough to define $m(x, y, z)$ for x, y and z in the same $\mathcal{S}_{[n]}$. On the remaining triples m may be defined arbitrarily, without compromising the polymorphism property.

We shall now study the structure of coset relations R as in (5) in some more detail. First, consider a relation R where some n_i in (5) equals 1, and let R_{-i} be the projection of R to all coordinates except i. Clearly, R_{-i} is a coset relation. It is easy to see that R is compatible with a given majority function m if and only if R_{-i} is. As a result, such relations R may safely be excluded when checking the polymorphism property on all coset relations.

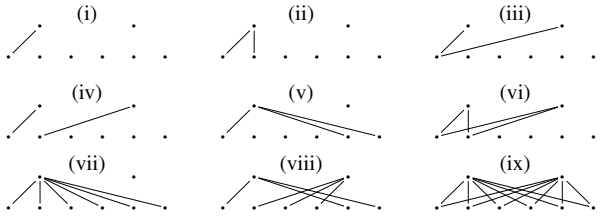
On the other hand, assume R is unary. Call a ternary function m *conservative* if $m(\sigma, \tau, \nu) \in \{\sigma, \tau, \nu\}$ for all arguments σ, τ and ν. It is easy to see that all unary relations are compatible with every conservative function. Therefore, for any conservative m, we may disregard all unary relations in T_A as well.

Now consider binary relations $R \subseteq \mathcal{S}_{[2]} \times \mathcal{S}_{[2]}$ in T_A. $\mathcal{S}_{[2]}$ has two elements, so there is only one way to define a majority function on this component: the majority equations determine it fully. It is easy to check that every binary relation on a two-element set is compatible with the unique majority function on it, so such relations may also be disregarded.

The only remaining case is binary relations $R \subseteq \mathcal{S}_{[2]} \times \mathcal{S}_{[3]}$. Arranging the elements of $\mathcal{S}_{[2]}$ and $\mathcal{S}_{[3]}$ in a diagram:



we may draw any R as a bipartite graph. Recall that any R in T_A is a subgroup or a coset in the group $\mathcal{S}_{[2]} \times \mathcal{S}_{[3]}$. Let us consider subgroups first; all are depicted here (dots correspond to permutations as drawn above)



with the exception of two conjugate subgroups per each of (ii), (iv) and (vi), which are isomorphic to them as bipartite graphs. Recall that there is only one way to define a majority function on $\mathcal{S}_{[2]}$; the question is whether we can extend it with a majority function m on $\mathcal{S}_{[3]}$ so that all relations above are compatible with it. It is easy to see that, as far as such exentions are concerned:

- (i)-(iv) and (vi) are compatible with any majority function,
- (v) is compatible with any conservative function,
- (vii) and (ix) are compatible with any function.

The only remaining case is (viii). The group $\mathcal{S}_{[3]}$ contains three even permutations (), (1 2 3) and (1 3 2), and three odd ones (1 2), (1 3), and (2 3). We say that a function $m : \mathcal{S}_{[3]}^3 \to \mathcal{S}_{[3]}$ is *odd-even-majority* if $m(\pi_1, \pi_2, \pi_3)$ is even whenever at least two of π_1, π_2, π_3 are even, and odd otherwise. It is easy to check that:

- (viii) is compatible with any odd-even-majority function.

Cosets in $\mathcal{S}_{[2]} \times \mathcal{S}_{[3]}$ do not pose additional problems. Indeed, all cosets of (i)-(vii) are isomorphic to their respective subgroups as bipartite graphs, and they are compatible with functions listed above for their respective subgroups. The full group (ix) has no cosets, and the unique coset of (viii) is its complement as a bipartite graph, and it is compatible with every odd-even-majority function.

Altogether we have proved that every ternary function on D_A that is conservative, a majority, and an odd-even-majority when restricted to $\mathcal{S}_{[3]}^3$, is a polymorphism on T_A. It is easy to define a function with all these properties, and as a result, T_A has a majority polymorphism, so A is standard.

5.2 Some nonstandard alphabets of dimension six

As one could expect, there are standard alphabets of dimensions greater than five. For instance, the alphabet $\binom{\mathbb{A}}{k}$ of k-element sets of atoms is standard for any k. It is easy to see this by looking at their templates, but there is an even simpler argument: over such an alphabet, two words are isomorphic if and only if they are similar.

However, there are nonstandard alphabets of dimension six. Consider the alphabet from Example 2.4. We depict a fragment of a word v over this alphabet (with atoms anonymized as dots as in Example 4.1), where the letter in the center intersects three other letters, splitting it into three bags:

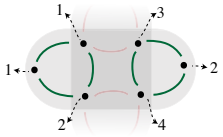
Consider the instance $I^{v,\alpha}_{v,\alpha}$, where α is any atlas on v (its choice will not affect the following). The triple of bags is constrained to a ternary relation R on $\mathcal{S}_{[2]}$ that consists of those triples of permutations where the swap $\sigma = (1\ 2) \in \mathcal{S}_{[2]}$ appears zero or two times.

As $\mathcal{S}_{[2]}$ has only two elements, there is exactly one majority function m on $\mathcal{S}^3_{[2]}$. As it turns out, it is not compatible with R:

$$m\begin{array}{cccc} () & \sigma & \sigma & \in R \\ \sigma & () & \sigma & \in R \\ \sigma & \sigma & () & \in R \\ \hline \sigma & \sigma & \sigma & \notin R, \end{array}$$

hence m is not a polymorphim of T_A and the alphabet from Example 2.4 is nonstandard. The reader should enjoy comparing this short argument with the pain suffered in [4] to prove the same result directly, without using CSP theory.

Now consider Example 2.5, and a fragment of a word v with a letter split into two bags of size two and four (the left and right atoms are in one bag), together with an atlas α as below:



In the instance $I^{v,\alpha}_{v,\alpha}$, the two bags of this letter are constrained to a binary relation $R_{\{1,2\}} \subseteq \mathcal{S}_{[4]} \times \mathcal{S}_{[2]}$ which is the graph of the partial function $f_{\{1,2\}} : \mathcal{S}_{[4]} \to \mathcal{S}_{[2]}$ such that

$$f_{\{1,2\}}(\pi) = \begin{cases} () & \text{iff } \pi \text{ preserves the set } \{1,2\}, \\ \sigma & \text{iff } \pi \text{ maps } \{1,2\} \text{ to } \{3,4\}. \end{cases}$$

For the same word but different atlases other relations $R_{\{1,3\}}$ and $R_{\{1,4\}}$, which are graphs of partial functions $f_{\{1,3\}}$ and $f_{\{1,4\}}$ defined analogously to $f_{\{1,2\}}$, can be obtained.

Assume any majority function m on the domain of T_A. For the following permutations in $\mathcal{S}_{[4]}$:

$$\pi_1 = (1\ 2)(3\ 4) \qquad \pi_2 = (1\ 3)(2\ 4) \qquad \pi_3 = (1\ 4)(2\ 3)$$

consider the value $\chi = m(\pi_1, \pi_2, \pi_3)$. Since $f_{\{1,2\}}(\pi_1) = ()$, $f_{\{1,2\}}(\pi_2) = \sigma$, and $f_{\{1,2\}}(\pi_3) = \sigma$, if m where a polymorphism then we would have $f_{\{1,2\}}(\chi) = m((), \sigma, \sigma) = \sigma$. Analogously we can prove that $f_{\{1,3\}}(\chi) = f_{\{1,4\}}(\chi) = \sigma$. This means that the permutation χ maps each of sets $\{1,2\}, \{1,3\}, \{1,4\}$ to their complements, which is impossible. As a result, m cannot be a polymorphism, hence the alphabet of Example 2.5 is nonstandard.

6. Descriptive Complexity

We relate our results concerning TMAs to logics over relational structures. To this end, we first describe a class of finite relational structures which correspond to words with atoms. We then describe how TMAs over such words correspond, in terms of expressive power, to certain logics over these relational structures.

In this section we consider both classical Turing machines and TMAs, but only deterministic ones.

6.1 Linearly patched structures

Below we consider finite relational structures over a finite vocabulary, assuming without loss of generality that the domains of the structures are subsets of the set $\mathbb{A}$ of atoms.

A *patched* structure is a relational structure $\mathfrak{M}$ together with a collection of substructures of $\mathfrak{M}$ (not necessarily induced substructures) which we call *patches* of $\mathfrak{M}$, such that $\mathfrak{M}$ is covered by its patches, i.e. the domain and relations of $\mathfrak{M}$ are the set-theoretical unions (not necessarily disjoint) of the domains and relations of its patches, respectively. We say that $\mathfrak{M}$ is *linearly patched* if a linear order on its patches is provided. For a finite family $\mathcal{P}$ of relational structures, we say that $\mathfrak{M}$ is *linearly $\mathcal{P}$-patched*, if every patch of $\mathfrak{M}$ is isomorphic to some structure in $\mathcal{P}$.

In order to evaluate logical formulas on a linearly patched structure, we add the set of patches to its domain; additionally, we include a binary relation witch connects each patch with all its vertices, and also for each relation R of arity n we add a relation $\hat{R}$ which contains all tuples $(\mathfrak{p}, v_1, \ldots, v_n)$ consisting of a patch $\mathfrak{p}$ and n vertices, such that $R(v_1, \ldots, v_n)$ holds in $\mathfrak{p}$. This allows formulas to quantify both over vertices and patches. Finally, we allow formulas to refer to the linear order on patches.

Example 6.1. Let $\mathcal{P}$ be the singleton family containing the graph with two vertices and one undirected edge. Then a linearly $\mathcal{P}$-patched structure is the same thing as an undirected graph without isolated vertices, together with a linear order on its edges. As a purely relational structure, this is represented as the union of the set of vertices and the set of edges, together with a linear order on the set of edges, and a ternary relation which says that two given vertices are endpoints of a given edge.

Example 6.2. The CFI graphs [6] can be constructed from three-regular graphs, by replacing each vertex by a certain gadget $\mathfrak{G}$ with ten nodes, which correspond to the six atoms and four triples of the letter $\{(a,c,e),(a,d,f),(b,c,f),(b,d,e)\}$ from Example 2.4, and with edges connecting each triple with its elements. Neighboring gadgets are then connected by identifying two exchangeable pairs of atoms. If the original three-regular graph is ordered, then the resulting structure is a linearly $\mathcal{P}$-patched structure, where $\mathcal{P} = \{\mathfrak{G}\}$.

6.2 Linearly patched structures and words with atoms

For a relational structure $\mathfrak{M}$ consider the set $A_{\mathfrak{M}}$ of all relational structures isomorphic to $\mathfrak{M}$. Then $A_{\mathfrak{M}}$ is a single-orbit set.

Fact 6.1. *Every single-orbit set is related by an equivariant bijection to $A_{\mathfrak{M}}$, for some finite relational structure $\mathfrak{M}$.*

For a finite collection $\mathcal{P}$ of finite relational structures, let $A_{\mathcal{P}}$ denote the disjoint union of the alphabets $A_{\mathfrak{M}}$, for $\mathfrak{M} \in \mathcal{P}$.

Linearly $\mathcal{P}$-patched structures correspond to words over the alphabet $A_{\mathcal{P}}$. Indeed, a word w induces a linearly patched structure with $\sup(w)$ as the domain and the letters of w as patches. Conversely, a linearly $\mathcal{P}$-patched structure defines a word whose letters are the patches. This allows us to move freely between structures and words, and to view TMAs over $A_{\mathcal{P}}$ as recognisers of linearly $\mathcal{P}$-patched structures. For instance, by Fact 6.1, Theorem 3.1 may be reformulated as:

Lemma 6.2. *For every fixed finite collection $\mathcal{P}$, isomorphism of linearly $\mathcal{P}$-patched structures is recognizable by a deterministic TMA if and only if the alphabet $A_{\mathcal{P}}$ is standard.*

6.3 Order-invariant classical polynomial time

Let us come back to the world of classical (atomless) computations. Any relational structure has a *description*, i.e., a list of identifiers (one unique binary string per each element of the universe) and for each relation, a list of its tuples, referred via the identifiers. Note that a fixed structure has many descriptions, depending on the chosen identifiers and orderings. In this non-unique way, relational structures are presented as inputs for classical Turing machines. We assume that these descriptions use a fixed finite alphabet, say $\{0,1\}$.

A classical Turing machine is *order-invariant* if given on input a description of a relational structure, the output of the machine does not depend on the chosen description. For example, a Turing machine which checks whether a graph is connected is order-

invariant, whereas a Turing machine which checks whether the first two vertices are adjacent is not.

Order-invariant Turing machines can be therefore viewed as recognizers of relational structures; however, it is undecidable whether a given Turing machine is order-invariant. One of the main open questions in Finite Model Theory, and in Descriptive Complexity Theory in particular, is whether there is a *logic which captures* order-invariant polynomial time. Such a logic is a class of formulas with decidable syntax, such that every formula can be effectively translated into a polynomial time order-invariant Turing machine accepting the same structures, and conversely: every property recognized by a polynomial time order-invariant Turing machine is definable by a formula. For example, first order logic satisfies only half of the requirements: a formula, such as $\forall x \exists y. E(x, y)$, gives rise to a polynomial time algorithm, which is order-invariant; however, graph connectedness is not definable by a first order formula.

One can relativize the above definitions to a class of structures $\mathcal{C}$, to say that a logic captures order-invariant polynomial time Turing machines *over* $\mathcal{C}$.

Observe that we can use two types of Turing machines to accept linearly $\mathcal{P}$-patched structures: deterministic TMAs, and classical Turing machines, which are order-invariant. The latter ones are – a priori – more powerful, as they have access to a linear ordering of the elements of the structure, which comes from the chosen description of the structure. On the other hand TMAs only have access to a linear ordering of the patches. Our results allow to characterize those families $\mathcal{P}$ for which the two types of machines are equally expressive. Furthermore, we relate these machines to the IFP logic.

6.4 IFP and IFP+C

We roughly describe the logics IFP and IFP+C. They are extensions of first order logic, which capture order-invariant polynomial time over some classes of structures, but not over all structures. For precise definitions and overviews of the cited results, see e.g. [11].

Instead of providing the precise definition of IFP, we give an illustrative example, and then roughly sketch the general form of the syntax.

Example 6.3. Consider structures over a relational language with one binary symbol E; they can be seen as directed graphs (possibly with self-loops). The following formula, with free variables x, y, says that there is a directed path from x to y:

$$\Big(\text{IFP}_{R,z}\big[(z = x) \vee \exists v : R(v) \wedge E(v, z)\big]\Big)(y) \qquad (\diamond)$$

The semantics is as follows. Let x, y be vertices of a graph. Start with R being the empty set of vertices, treated as a unary relation. Repeat indefinitely the following *inflationary* step:

> Add to R those vertices z which satisfy the subformula of ($\diamond$) delimited by the brackets [ ].

The formula ($\diamond$) holds for the pair of vertices (x, y) if y belongs to R in some step of the loop. Observe that the property described by ($\diamond$) is not first-order definable.

The general form of the construct is $\text{IFP}_{R,\bar{z}}[\phi(\bar{x}, \bar{z})](\bar{y})$. Comparing to ($\diamond$), there can be tuples of variables $\bar{x}, \bar{y}, \bar{z}$ instead of single variables x, y, z, under the restriction that $\bar{y}$ and $\bar{z}$ have the same length, say n. Then R is interpreted as an n-ary relation which is computed in an inflationary manner, starting from the empty relation.

The logic IFP+C further extends IFP by a construct $\#_{\bar{x}}\phi(\bar{x}, \bar{y})$ which allows to count (using a special counting sort) the number of assignments for $\bar{x}$ satisfying $\phi(\bar{x}, \bar{y})$.

It is not difficult to prove that a formula of IFP+C can be effectively translated into an equivalent polynomial time Turing machine, which is automatically order-invariant. The polynomial bound is a consequence of the fact that for finite models, due to the inflationary mode of computation, stabilization is reached after a polynomial number of steps.

The Immerman-Vardi theorem states that IFP (and, in consequence, IFP+C) captures order-invariant polynomial time over linearly ordered structures [12, 15]. On the other hand, IFP+C does not capture order-invariant polynomial time over all graphs, as proved in [6] using the CFI graphs. Our aim is to generalize both these results, using TMAs. How can TMAs be useful for studying the logic IFP?

IFP **over linearly $\mathcal{P}$-patched structures.** In the Immerman-Vardi theorem, instead of linearly ordered structures, one could equivalently use graphs (without isolated vertices) whose set of edges is linearly ordered. This is more consistent with linearly patched structures (see Example 6.1). The following proposition is then a generalization of the Immerman-Vardi theorem to linearly $\mathcal{P}$-patched structures instead of graphs with ordered edges, and TMAs instead of Turing machines.

Proposition 6.3. *The logics* IFP *and* IFP+C *capture polynomial time TMAs over linearly $\mathcal{P}$-patched structures.*

6.5 Main result

The main result of this section is:

Theorem 6.4. *The logic* IFP *captures order-invariant polynomial time over linearly $\mathcal{P}$-patched structures if and only if the alphabet $A_{\mathcal{P}}$ is standard.*

Observe that IFP can be replaced by IFP+C in the above theorem, since those logics are equivalent over linearly $\mathcal{P}$-patched structures by Proposition 6.3. Also, this proposition allows us to use polynomial time TMAs instead of the logic IFP in the proof.

Proof. Fix a finite family $\mathcal{P}$ of structures, and let $A_{\mathcal{P}}$ be the corresponding alphabet with atoms.

We first show the left-to-right implication. Assume that polynomial time deterministic TMAs capture classical order-invariant Turing machines over linearly $\mathcal{P}$-patched structures. Then, in particular, deterministic TMAs can express the isomorphism problem of linearly $\mathcal{P}$-patched structures, according to Lemma 6.5 below. Lemma 6.2 then implies that the alphabet $A_{\mathcal{P}}$ is standard, proving the left-to-right implication.

Lemma 6.5. *Isomorphism of linearly $\mathcal{P}$-patched structures is decidable by an order-invariant polynomial time Turing machine.*

Proof. As shown in Section 4, the isomorphism problem for linearly $\mathcal{P}$-patched structures reduces to the CSP problem over the template $T_{A_{\mathcal{P}}}$. For a fixed alphabet $A_{\mathcal{P}}$, the reduction is polynomial. It is known [5] that for a template with a Maltsev polymorphism, the induced CSP is in polynomial time. □

For the proof of the other implication, we show that if $A_{\mathcal{P}}$ is standard, then polynomial time TMAs are equally expressive as classical order-invariant polynomial time Turing machines.

Consider the language $D_{\mathcal{P}}$ containing all words of the form $w\#desc$, where $w \in A_{\mathcal{P}}^*$ and $desc \in \{0, 1\}^*$ is a description (cf. Section 6.3) of the linearly $\mathcal{P}$-patched relational structure corresponding to w. Reusing the (k, l)-consistency algorithm for recognizing the language $D_{\mathcal{P}}$, we get the following:

Lemma 6.6. *If the alphabet $A_{\mathcal{P}}$ is standard, then the language $D_{\mathcal{P}}$ is recognized by a polynomial time TMA.*

The following lemma shows that if $A_{\mathcal{P}}$ is standard, then not only descriptions can be recognized, but also produced in polynomial time (i.e., there is a polynomial time *canonisation* algorithm).

Lemma 6.7. *Using a machine for the language $D_{\mathcal{P}}$ as blackbox, a polynomial time TMA can produce, for an input word $w \in A_{\mathcal{P}}^*$, a description desc $\in \{0,1\}^*$ of the structure corresponding to w.*

Proof. This can be done by finding descriptions of longer and longer prefixes of w. There is no need of backtracking, since any description of a prefix can be extended to a description of the whole word. Moreover, to extend the description of a prefix by one letter, the machine needs to check only constantly many possible candidate strings over $\{0,1\}$ and for each candidate, run the machine for $D_{\mathcal{P}}$ (the constant depends on the alphabet $A_{\mathcal{P}}$ but not on the length of the input). □

Suppose that $A_{\mathcal{P}}$ is standard and let L be a property of linearly $\mathcal{P}$-patched structures, recognized by a classical polynomial time Turing machine M. Then a TMA over $A_{\mathcal{P}}$ can decide, for an input word w, whether the structure corresponding to w has property L, in the following two steps. In the first step, the machine produces a description of the structure corresponding to the word w, according to Lemmas 6.6 and 6.7. In the second step the machine simulates, on that description, the order-invariant polynomial time Turing machine M recognizing L.

This proves the right-to-left implication of Theorem 6.4. □

Many results of the previous sections can be translated into results about logic. For example, from Section 5.1 and Theorem 6.4, we get:

Corollary 6.8. *If no structure in $\mathcal{P}$ has more than five elements, then* IFP *captures order-invariant polynomial time computations over linearly $\mathcal{P}$-patched structures.*

As a byproduct of the proof of Theorem 6.4 we obtain a "logic" that captures order-invariant polynomial time over linearly $\mathcal{P}$-patched structures (even if $A_{\mathcal{P}}$ is nonstandard), namely TMAs with an oracle for the language $D_{\mathcal{P}}$. This can be converted to a more reasonable logic (as in Proposition 6.3), namely, an extension of IFP+C by a construct which tests whether two linearly $\mathcal{P}$-patched structures are isomorphic.

7. Related work

The paper [1] also studies a relationship between the logic IFP+C and templates of bounded width. The emphasis there is to determine for which templates T there exists a formula of IFP+C which holds in a given structure (instance) I over the signature of T if and only if I maps homomorphically to T. It follows from that paper (see Corollary 23) and from later deep results from algebraic CSP theory [2] that this happens precisely when T has bounded width. In our paper, a similar, but weaker result is implicit (see Theorem 3.3, Proposition 4.3 and Proposition 6.3): it concerns only templates of the form T_A, where A is an alphabet. In particular, those templates have a Maltsev polymorphism.

Graph Isomorphism problem. The graph isomorphism problem with bounded color classes is the problem of deciding whether there is a color-preserving isomorphism between two given colored graphs G, H, where the coloring is such that at most k vertices get the same color (where k is a fixed parameter). This problem is the first restricted version of the graph isomorphism problem to be shown in PTime using group theoretic methods [9].

Colored graphs whose color classes have size at most k can be seen as patched structures, where the patches are the subgraphs induced by any pair of colors. Therefore, each patch is a graph with at most $2k$ vertices. Let $\mathcal{P}$ be the family of all graphs on vertices $\{1, \ldots, 2k\}$. Then the isomorphism problem of such colored graphs reduces to the isomorphism problem for linearly $\mathcal{P}$-patched structures – the patches are linearly ordered according to an arbitrary linear ordering of the set of pairs of colors. It therefore follows from Lemma 6.5 that this can be done in polynomial time (and in logarithmic space if the alphabet $A_{\mathcal{P}}$ is standard, using [7]). Our proof does not rely on group theory, but instead uses the polynomial algorithm for solving CSPs over a template admitting a Maltsev polymorphism [5].

Acknowledgments

We are grateful to Marcin Kozik for guiding us in the land of Constraint Satisfaction Problems, to Mikołaj Bojańczyk for inspiring discussions on sets with atoms, and to anonymous reviewers for insightful comments.

References

[1] A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18): 1666 – 1683, 2009.

[2] L. Barto and M. Kozik. Constraint satisfaction problems of bounded width. In *Procs. FOCS'09*, pages 595–603. IEEE Computer Society, 2009.

[3] M. Bojańczyk, B. Klin, and S. Lasota. Automata with group actions. In *Proc. LICS'11*, pages 355–364, 2011.

[4] M. Bojańczyk, B. Klin, S. Lasota, and S. Toruńczyk. Turing machines with atoms. In *Proc. LICS'13*, pages 183–192, 2013.

[5] A. A. Bulatov and V. Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.

[6] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4): 389–410, 1992.

[7] V. Dalmau and B. Larose. Maltsev + Datalog –> symmetric Datalog. In *Procs. LICS*, pages 297–306. IEEE Computer Society, 2008.

[8] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[9] M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 36–41. IEEE Computer Society, 1980.

[10] M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[11] M. Grohe. Descriptive complexity, canonisation, and definable graph structure theory, December 2013. URL `http://www.automata.rwth-aachen.de/~grohe/cap/index.en`.

[12] N. Immerman. Upper and lower bounds for first order expressibility. *J. Comput. Syst. Sci.*, 25(1):76–98, 1982.

[13] B. Larose, M. Valeriote, and L. Zádori. Omitting types, bounded width and the ability to count. *Int. J. Algebra and Computation*, 19(5):647–668, 2009.

[14] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[15] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.