Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Łukasz Wołochowski
Nr albumu: 277626

Classification of alphabets with atoms

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr. Szymona Toruńczyka
Instytut Informatyki

Październik 2014

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

This thesis concerns Turing machines with atoms. There exist alphabets with atoms over which Turing machines do not determinize and a method of alphabet classification is known. In this thesis we show and implement an improved version of this algorithm. The improvement is made by using advanced algorithms from the theory of Constraint Satisfaction Problems, as well as algebraic methods to reduce the size of a problem. As a result, we obtain a classification of all alphabets of dimension 8.

Słowa kluczowe

Turing machines with atoms, sets with atoms, Constraint Satisfaction Problems, relational structures, majority polymorphism, Singleton Arc Consistency

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

F. Theory of Computation
F.1. Computation by Abstract Devices
F.1.1 Models of Computation

Tytuł pracy w języku polskim

Klasyfikacja alfabetów z atomami

Contents

Introduction

Turing machines with atoms (TMAs), introduced in [BKLT13], are, roughly speaking, Turing machines that can operate not only on binary digits but also on infinite alphabets whose letters are finite structures built of *atoms* — elements of an infinite countable set that can be tested only for equality. An example of an alphabet with atoms is the family of all unordered pairs of atoms. A TMA can, for instance, detect if a word contains two letters that have exactly one atom in common.

This computational model has an interesting property — Turing machines with atoms, unlike the classical ones, do not determinize [BKLT13]. This means that there are some nondeterministic TMAs that recognize languages which cannot be recognized by any deterministic TMA. However, this phenomenon does not occur for every alphabet, i.e. there are alphabets such that every TMA over it determinizes — we call them *standard alphabets* — and there are alphabets for which some TMAs do not determinize — we call them *nonstandard alphabets*. Unordered pairs of atoms are an example of a standard alphabet. For other examples of standard and nonstandard alphabets, see [BKLT13] and [KLOT14].

An alphabet with atoms can be encoded in the finite way using its algebraic properties and can be an input to a classical Turing machine. Deciding whether an alphabet is standard is an interesting topic and has relations with Finite Model Theory and Constraint Satisfaction Problems (CSPs) [KLOT14]. A method of alphabet classification has been proposed in [KLOT14]. It is based on the theory of Constraint Satisfaction Problems and reduces this problem to a problem in CSP theory — whether some relational structure called a *template* has, so called, *majority polymorphism*. However, this method has large complexity and is not useful in practice.

The main goal of this thesis is to present and implement an improved version of the algorithm shown in [KLOT14]. The improvement has been achieved by applying modern and efficient algorithms from CSP theory (e.g. SAC3 algorithm [LC05]), as well as using some algebraic properties of the template in order to reduce its size.

This improved algorithm has been implemented as a proof of concept but also as a research tool. The program is written in Haskell. We also implemented generic functions that compute some CSP algorithms. Using these algorithms, we classified all alphabets of dimension 8.

This thesis is organized as follows. In Chapter 1 we provide a definition of Turing machines with atoms and standard and nonstandard alphabets. Chapter 2 is devoted to showing the method of alphabet classification from [KLOT14]. It consists of two parts — the first part shows the construction of the template and shows the equivalence of an alphabet being standard and the template built from this alphabet having some algebraic property (having a majority polymorphism). This part is solely based on [KLOT14]. The second part shows an algorithm that decides whether a structure has a majority polymorphism. This polynomial time algorithm is nontrivial and was presented is [BCH$^+$13].

In Chapter 3 we show the main result of this thesis — an improved algorithm for alphabet classification. This chapter also consists of two parts — in the first part we show the

SAC3 algorithm which allows us to enhance the majority detection algorithm presented in Chapter 2. The second part uses some algebraic properties of the template to reduce the size of a problem. Chapter 4 discusses some implementation details and shows some results obtained with the implemented program. Appendix A contains a complete classification of all single-orbit alphabets of dimension 8.

Chapter 1

Turing machines with atoms

In this section we give a precise definition of Turing machines with atoms. We use the notion of sets with atoms, also known as nominal sets [Pit13].

1.1. Alphabets with atoms

Sets with atoms Let us fix an infinite countable set $\mathfrak{A}$. We will call the elements of $\mathfrak{A}$ *atoms*. We define *sets with atoms* by transfinite induction: the only set with atoms at level 0 is the empty set and, for any ordinal κ, sets with atoms at level κ are the sets consisting of atoms and sets at level smaller than κ. A set with atoms is a set at an arbitrary level.

Examples of sets with atoms include:

- classical sets (without atoms),
- the set of ordered pairs of atoms: $\mathfrak{A}^2 = \{(a, b) : a, b \in \mathfrak{A}\}$, where $(a, b) = \{a, \{a, b\}\}$,
- the family of all finite subsets of $\mathbb{A}$: $\mathcal{P}_{fin}(\mathfrak{A})$.

Support For any bijection of atoms $\pi : \mathfrak{A} \to \mathfrak{A}$ and any set with atoms X, $\pi(X)$ denotes a set obtained by renaming every atom in X according to π and applying π recursively to all sets that are members of X. A set of atoms $S \subseteq \mathfrak{A}$ is called a *support of* X if X is invariant under any bijection π which is an identity on S (i.e. if $\pi(a) = a$ for any $a \in S$, then $\pi(X) = X$).

A set with atoms is called *hereditarily finitely supported* if it has a finite support and each of its element is either an atom or a hereditarily finitely supported set (transfinite inductive definition). In this thesis we will only consider hereditarily finitely supported sets with atoms and we will call them simply *sets with atoms*.

Examples of hereditarily finitely supported sets with atoms include classical sets without atoms, the set of ordered pairs of atoms, and $\mathcal{P}_{fin}(\mathfrak{A})$. On the other hand, the family of all subsets of $\mathfrak{A}$ is not hereditarily finitely supported – it is finitely supported (by the empty set) but each set $A \in \mathcal{P}(\mathfrak{A})$ that is neither finite nor cofinite does not have a finite support.

If a set with atoms is finitely supported, then it has the smallest support with respect to inclusion [GP02, Prop.3.4]. We denote the smallest support of X by $\sup(X)$. A set with atoms X is *equivariant* if $\sup(X) = \emptyset$, which means that it is invariant under every bijection of atoms. The *dimension* of an alphabet A, denoted by $\dim(A)$, is the maximal value of $\sup(x)$ of any element $x \in A$.

Relations and functions Basing on the new notion of a set, we define other standard notions. In particular, if A, B are sets with atoms and $x \in A, y \in B$, then we define the pair of elements as $(x, y) = \{x, \{x, y\}\}$ and the Cartesian product as $A \times B = \{(x, y) : x \in A, y \in B\}$. A *relation with atoms* is a relation on a set with atoms which is hereditarily finitely supported. It is clear that $\sup((x, y)) \subseteq \sup(x) \cup \sup(y)$. Since all elements of a relation are hereditarily finitely supported, it is sufficient that the relation itself has a finite support. A *function with atoms* is a function, whose domain and codomain are sets with atoms, and whose graph is a relation with atoms.

Moreover, it is easy to notice that a function with atoms $f : X \to Y$ is equivariant iff for any bijection $\pi : \mathfrak{A} \to \mathfrak{A}$ and any $x \in X$, $f(\pi(x)) = \pi(f(x))$. A relation with atoms $r \subseteq X \times Y$ is equivariant iff for any bijection $\pi : \mathfrak{A} \to \mathfrak{A}$ and any $x \in X$, $y \in Y$, $(x, y) \in r$ iff $(\pi(x), \pi(y)) \in r$.

Orbits Let A be a set with atoms. An *orbit* of an element $x \in A$ is the set

$$o(x) = \{\pi(x) : \pi - \text{bijection of atoms}\}.$$

It is clear that A is equivariant iff it is the union of orbits of all of its elements. An *orbit-finite set* is a set with atoms that has finitely many orbits. Every orbit-finite set has a finite dimension.

Automorphisms Fix an element $x \in A$. A permutation π of $\sup x$ is called an *automorphism of* x if x is fixed by any permutation of atoms that extends π. The set of all automorphisms of x is denoted by $Aut(x)$. For example, if a, b are atoms, then $Aut(\{a, b\})$ is the set of all permutation of $\{a, b\}$ but $Aut((a, b))$ consists only of the identity permutation.

Recall that the family of all permutations of a set forms a group, where the identity permutation is the neutral element and the composition of permutations is the group operation. It is easy to check that $Aut(x)$ is a subgroup of the group of all permutations of $\sup x$.

Two alphabets A, B are *isomorphic* iff there exists an equivariant bijection $f : A \to B$. The following lemma follows easily from [BKL14].

Lemma 1.1. *Let A, B be two single-orbit alphabets and $a \in A$, $b \in B$. Then, A and B are isomorphic iff $Aut(a)$ and $Aut(b)$ are isomorphic as permutation groups.*

Moreover, many properties of orbit-finite sets, such as the standardness of orbit-finite alphabets, can be considered as properties of finite permutation groups [KLOT14]. This means that orbit-finite alphabets can be finitely represented as inputs to classical Turing machines (using automorphism groups of a representative of each orbit of the alphabet).

1.2. Turing machines with atoms

Now we are ready to define Turing machines with atoms (TMAs). The definition is analogous to the definition of a regular Turing machine, only that we use orbit-finite sets with atoms.

A *Turing machine with atoms* is a tuple $(A, B, Q, Q_{init}, Q_{fin}, \delta)$ of orbit-finite sets with atoms, where A is the *input alphabet*, $B \supseteq A$ is the *work alphabet*, Q is the *set of states*, $Q_{init}, Q_{fin} \subseteq Q$ are the *sets of initial and final states*, and $\delta \subseteq Q \times B \times Q \times B \times \{-1, 0, 1\}$ is the equivariant *transition relation*. We define the machine state, the machine run, machine accepting a word and language recognized by a machine exactly as in the standard definition.

We say that a TMA, as described above, is *deterministic* if δ is a partial function $Q \times B \to Q \times B \times \{-1, 0, 1\}$ and Q_{init} consists of exactly one element.

1.3. Standard and nonstandard alphabets

As we mentioned previously, there exist languages recognizable by nondeterministic Turing machines with atoms that cannot be recognized by any deterministic TMA. A *nonstandard alphabet* is an alphabet with atoms for which this phenomenon occurs, i.e. there exists a language over this alphabet recognized by some nondeterministic TMA but not recognizable by any deterministic TMA. Conversely, an alphabet for which this property does not hold is called a *standard alphabet*. Examples of standard and nonstandard alphabets can be found if [KLOT14]. All single-orbit alphabets of dimension 8 are classified in Appendix A.

The main goal of this paper is to find an efficient algorithm (in the classical sense) that classifies alphabets. One method was proposed in [KLOT14] and we present it in Chapter 2. It reduces the problem of classification of an alphabet to a problem in the theory of Constraint Satisfaction Problems (whose basics we present in Section 2.1). With the help of techniques from CSP theory, we present in Chapter 3 a more efficient classification algorithm. By implementing this algorithm, we classified all alphabets of dimension 8. The classification of all single-orbit alphabets of dimension 8 is presented in Appendix A.

Chapter 2

Classification of alphabets

The goal of this chapter is to show an algorithm that determines whether an alphabet is standard. In [KLOT14] this problem was proved to be equivalent to a problem in Constraint Solving Problem theory – we will show this reduction (without a proof) in Section 2.2. Then, in Section 2.3 we will show an algorithm that solves this problem.

First, we need to introduce some basic facts and notions about Constraint Satisfaction Problems.

2.1. Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs) are decision problems traditionally defined as problems of determining the existence of an assignment of values to a given set of variables, subject to a given set of constraints. Examples of such problems include the boolean satisfaction problem and graph coloring problems. A more mathematical, equivalent definition of CSP uses algebra and relational structures. This approach allows us to use algebraic techniques to work on complexity of some CSP instances.

Throughout this thesis we will use the algebraic definition of CSP, but first we need to recall some basic algebraic notion.

Definition 2.1. A *relational signature* is a finite set of relational symbols $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$, each of which has an associated arity (denoted by $\mathrm{ar}(R)$).

An $\mathcal{R}$-*structure* is a tuple $\mathbb{A} = (A, R_1^{\mathbb{A}}, R_2^{\mathbb{A}}, \ldots, R_m^{\mathbb{A}})$, where A is the set of elements of the structure and, for each $i \in \{1, 2, \ldots, m\}$, $R_i^{\mathbb{A}} \subseteq A^n$ is a relation over A of arity $n = \mathrm{ar}(R_i)$.

We will often abuse the notation and, instead of writing $a \in A$, we will just write $a \in \mathbb{A}$. The set A is called the *universe* of $\mathbb{A}$. The *size* of a structure is the number of elements in the universe.

Definition 2.2. Let $\mathbb{A}$ and $\mathbb{B}$ be $\mathcal{R}$-structures. A function $h : \mathbb{A} \to \mathbb{B}$ is called a *homomorphism* if for every $R \in \mathcal{R}$ and every tuple $(a_1, a_2, \ldots, a_n) \in R^{\mathbb{A}}$, where n is the arity of R, we have

$$(h(a_1), h(a_2), \ldots, h(a_n)) \in R^{\mathbb{B}}.$$

Let us fix a signature $\mathcal{R}$ and an $\mathcal{R}$-structure $\mathbb{T}$. By $CSP(\mathbb{T})$, we will denote the following decision problem:

Input	$\mathbb{A}$ – an $\mathcal{R}$-structure
Question	Does there exist a homomorphism $h : \mathbb{A} \to \mathbb{T}$?

The structure $\mathbb{T}$ will be often fixed and called a *template*. We encode the input (an $\mathcal{R}$-structure) in the following way: we enumerate all of its elements and then, for every relation R, we enumerate all of the tuples in R. The size of such encoding is at least the number of elements of $\mathbb{A}$ and its polynomially bounded ($O(n^r)$, where n is the size of the structure and r is the largest arity of a relation in the signature).

An important construction of relational structures is the Cartesian power.

Definition 2.3. Let $\mathcal{R}$ be a signature and $\mathbb{A} = (A, R_1^{\mathbb{A}}, R_2^{\mathbb{A}}, \ldots, R_m^{\mathbb{A}})$ be an $\mathcal{R}$-structure. For any $n \in \mathbb{N}$ we define *Cartesian power* $\mathbb{A}^n = (A^n, R_1^{\mathbb{A}^n}, R_2^{\mathbb{A}^n}, \ldots, R_m^{\mathbb{A}^n})$, where, for every $i \in \{1, 2, \ldots, m\}$ and $k = \mathrm{ar}(R_i)$, we have

$$R_i^{\mathbb{A}^n} = \{(\tau_1, \tau_2, \ldots, \tau_k) : \tau_1, \tau_2, \ldots, \tau_k \in A^n, \forall j \in \{1, 2, \ldots n\}\ (\tau_1[j], \tau_2[j], \ldots \tau_k[j]) \in R_i^{\mathbb{A}}\}.$$

By $\tau[j]$ we denote j-th coordinate in the tuple τ.

A homomorphism $p : \mathbb{A}^n \to \mathbb{A}$ is called a *polymorphism* of the structure $\mathbb{A}$. An important class of polymorphisms are *majority polymorphisms*.

Definition 2.4. A polymorphism $m : \mathbb{A}^3 \to \mathbb{A}$ is called a *majority polymorphism* if for every $a, b \in \mathbb{A}$, we have

$$m(a, a, b) = m(a, b, a) = m(b, a, a) = a.$$

Majority polymorphisms are an important tool in CSP theory. It has been proved that if a template $\mathbb{T}$ has a majority polymorphism, then $CSP(\mathbb{T})$ is in P (see [FV99] and [CDG13]). In Section 2.3 we will show an example of a generic polynomial algorithm that solves $CSP(\mathbb{T})$ if $\mathbb{T}$ has a majority polymorphism.

Examples Let us show an example of CSP —*k-coloring of a graph*. The classical definition of this problem is following: Fix a constant k. Given an undirected graph, decide whether its vertices can be colored with k colors such that no two adjacent vertices have the same color.

It is known that for $k \leq 2$ this problem is in P and for $k \geq 3$ it is NP-complete.

Let us formulate this problem in the language of CSP. An undirected graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V^2$ is the symmetric and irreflexive adjacency relation, is a relational structure. The signature of such a structure consists of one binary relation E, the universe is the set of all vertices. Let $\mathbb{T}_k$ be a k-element clique (without self-loops), whose vertices are $\{1, 2, \ldots, k\}$. It is easy to see that a graph $\mathbb{G}$ is k-colorable iff there exists a homomorphism $h : \mathbb{G} \to \mathbb{T}_k$. This follows from the fact that the necessary and sufficient condition for h to be a homomorphism is the following:

$$\forall x, y\ (x, y) \in E^{\mathbb{G}} \implies (h(x), h(y)) \in E^{\mathbb{T}_k}$$

and, since $\mathbb{T}_k$ is a clique (without self-loops), it is equivalent to

$$\forall x, y\ (x, y) \in E^{\mathbb{G}} \implies h(x) \neq h(y).$$

Hence h defines the coloring.

Let us show that $\mathbb{T}_2$ has a majority polymorphism. Consider the following function:

$$m(x, y, z) = \begin{cases} y & \text{if } y = z \\ x & \text{otherwise} \end{cases}$$

Clearly, for any a, b, we have $m(a, a, b) = m(a, b, a) = m(b, a, a) = a$. We will show that this function is a polymorphism. Take any $((x_1, y_1, z_1), (x_2, y_2, z_2)) \in E^{\mathbb{T}_2^3}$. By the definition

of Cartesian power, we have that $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in E^{\mathbb{T}_2}$ and, since $\mathbb{T}_2$ is a clique: $x_1 \neq x_2$, $y_1 \neq y_2$, $z_1 \neq z_2$. Recall that $\mathbb{T}_2 = \{1, 2\}$. Let us consider two cases.

Suppose $y_1 = z_1$. This means that $y_2 = z_2$, as the universe has only two elements. Then $(m(x_1, y_1, z_1), m(x_2, y_2, z_2)) = (y_1, y_2)$, so

$$m \ \frac{\begin{array}{c}(x_1, x_2) \in E^{\mathbb{T}_2} \\ (y_1, y_2) \in E^{\mathbb{T}_2} \\ (z_1, z_2) \in E^{\mathbb{T}_2}\end{array}}{(y_1, y_2) \in E^{\mathbb{T}_2}.}$$

Now suppose $y_1 \neq z_1$. This means that $y_2 \neq z_2$ and

$$m \ \frac{\begin{array}{c}(x_1, x_2) \in E^{\mathbb{T}_2} \\ (y_1, y_2) \in E^{\mathbb{T}_2} \\ (z_1, z_2) \in E^{\mathbb{T}_2}\end{array}}{(x_1, x_2) \in E^{\mathbb{T}_2}.}$$

This implies that $m : \mathbb{T}_2^3 \to \mathbb{T}_2$ is a majority polymorphism.

On the other hand, let us show that $\mathbb{T}_3$ does not have a majority polymorphism. Suppose that $m : \mathbb{T}_3^3 \to \mathbb{T}_3$ is a majority polymorphism. It is clear that $m(1, 2, 3) \in \{1, 2, 3\}$. Assume that $m(1, 2, 3) = 1$. Then,

$$m \ \frac{\begin{array}{c}(1, 2) \in E^{\mathbb{T}_3} \\ (2, 1) \in E^{\mathbb{T}_3} \\ (3, 1) \in E^{\mathbb{T}_3}\end{array}}{(1, 1) \notin E^{\mathbb{T}_3}.}$$

and we get a contradiction. Analogously, we can show that cases $m(1, 2, 3) = 2$ and $m(1, 2, 3) = 3$ also imply a contradiction. This means that $\mathbb{T}_3$ does not have a majority polymorphism.

2.2. Template

In this section we will show the construction of a template associated to an alphabet. It is a relational structure which, by Theorem 2.5, has a majority polymorphism if and only if the alphabet it is based on is standard.

The construction of the template presented in this section consists mainly of dry facts and might be found unintuitive. For a much more detailed description, that also explains the origins of the construction and shows examples, see [KLOT14]. The reason for this approach is the fact that the construction of the template from an alphabet is not the main topic of this thesis — in fact, in the remainder of the thesis, we will only use the algebraic properties of such templates presented at the end of this section (Facts 2.6-2.9).

Let A be an alphabet and $x \in A$ a letter. A *bag* is a set of atoms. A partition of $\sup x$ into bags is a tuple of bags $\mathbb{B}^x = (B_1^x, B_2^x, \ldots, B_n^x)$ such that every atom in $\sup x$ is in exactly one bag. A *map* of a bag B_i^x is any bijection $\mu_i^x : B_i^x \to \{1, 2, \ldots, |B_i^x|\}$ and an *atlas* is a family of maps – one map α_B for every bag $B \in \mathbb{B}^x$. We say that two atlases α, β *have the same type*, if the atlas α is on some letter x, the atlas β is on some letter y, the letters x and y are in the same orbit, the atlases α, β correspond to the same number of bags, and $|B_i^x| = |B_i^y|$, for any $i \in \{1, 2, \ldots, n\}$ (we will then denote the size of a bag by $|B_i|$).

Let S_n denote the family of all permutations on the set $\{1, 2, \ldots, n\}$. It is a group (called a *symmetric group*), where the neutral element is the identity permutation and the group operation is the composition of permutations.

Notice that for letters x, y in the same orbit and atlases α and β that have the same type, a tuple of permutations $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in S_{|B_1|} \times S_{|B_2|} \times \ldots \times S_{|B_n|}$ corresponds to some bijection $\rho : \sup x \to \sup y$ in the following way:

$$\rho(a) = (\mu_i^y)^{-1}(\sigma_i(\mu_i^x(a))) \text{ where } a \in B_i^x$$

and

$$\sigma_i(m) = \mu_i^y(\rho((\mu_i^x)^{-1}(m))).$$

We will say that this tuple of permutations is locally consistent (with respect to α and β) if ρ can be extended to a permutation of atoms π in a way that $\pi(x) = y$.

For any alphabet A, we define a relational structure $\mathbb{T}_A$ (called the *template* of A) in the following way. The domain of the structure is $T_A = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_{\dim A}$. For every orbit o in A, we take two representatives $x, y \in o$. We take any atlases of the same type α, β, on x, y respectively, that are based on bags of sizes $k_1, k_2, \ldots, k_n$ and we define

$$R_{o,\alpha,\beta} = \{(\sigma_1, \sigma_2, \ldots, \sigma_n) \in S_{k_1} \times S_{k_2} \times \ldots \times S_{k_n} : \\ : (\sigma_1, \sigma_2, \ldots, \sigma_n) \text{ are locally consistent with respect to } \alpha \text{ and } \beta\}.$$

The relations in $\mathbb{T}_A$ are $R_{o,\alpha,\beta}$ for every orbit o and every pair of atlases α, β that have the same type. Notice that there is a finite number of distinct relations of this type since the arity of a relation is bounded by the dimension of an alphabet.

The size of the template is equal to

$$\sum_{k=1}^{\dim A} k!.$$

The main theorem in [KLOT14], which we will use as the tool for the classification of alphabets, is the following.

Theorem 2.5. *An alphabet A is standard if and only if its template $\mathbb{T}_A$ has a majority polymorphism.*

Algebraic properties of the template There are some important algebraic properties of the template that we will use to optimize the classification algorithm. First, notice that the universe of $\mathbb{T}_A$ is $T_A = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_{\dim A}$, so every element $x \in T_A$ is a member of exactly one of the sets $S_1, S_2, \ldots, S_{\dim A}$. We will call this set a *type of* x. If two elements x, y have the same type, then they are, in fact, permutations of the same set and we can compose them (which is a group operation). Moreover, the composition $x \cdot y$ has the same type as x and y.

A relation in $\mathbb{T}_A$ is called *typed* if its projection onto any coordinate consists of elements of the same type.

Notice the following fact, which comes directly from the definition of the template:

Fact 2.6. *All relations in $\mathbb{T}_A$ are typed.*

This comes from the fact that any relation R in $\mathbb{T}_A$ is a subset of $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$.

Fact 2.7. *If a relation R is a subset of $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$, then $n_1 + n_2 + \ldots + n_r = \dim A$.*

In particular, this means that

$$|R^{\mathbb{T}}| \leq \prod_{i=1}^{r} |S_{n_i}| = \prod_{i=1}^{r} n_i! \leq \dim A!.$$

For a group G, its subgroup H and an element $a \in \mathbb{G}$, we define a *left coset*

$$aH = \{ab : b \in H\}.$$

In this thesis we will simply call it a *coset*. Let us note that the set $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$ is a group and the following fact was shown in [KLOT14]:

Fact 2.8. *Any relation in $\mathbb{T}_A$ is a coset.*

Moreover the family of relations in $\mathbb{T}_A$ has an interesting algebraic structure.

Fact 2.9. *Let R be an r-ary relation in $\mathbb{T}_A$ and $R \subseteq S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$. For any tuples $\mathbf{a}, \mathbf{b} \in S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$, $\mathbf{a} = (a_1, a_2, \ldots a_r)$, $\mathbf{b} = (b_1, b_2, \ldots b_r)$, we have that*

$$R' = \mathbf{a} \cdot R \cdot \mathbf{b} = \{(a_1 \cdot x_1 \cdot b_1, a_2 \cdot x_2 \cdot b_2, \ldots, a_r \cdot x_r \cdot b_r) : (x_1, x_2, \ldots, x_r) \in R\}$$

is also a relation in $\mathbb{T}_A$.

This fact holds because, if $R = R_{o,\alpha,\beta}$, for some orbit o and atlases α, β, then it is easy to check that $R' = R_{o,a\circ\alpha,b\circ\beta}$.

2.3. Algorithm

By Theorem 2.5, there is an algorithm that classifies alphabets with atoms and it consists of two parts — it creates the template of an alphabet and decides whether the template has a majority polymorphism. Since the first part is quite straightforward, this chapter will be devoted to the second part. Fix a signature $\mathcal{R}$. Our goal is to solve the following problem:

Input	an $\mathcal{R}$-structure $\mathbb{T}$
Question	Decide whether there exists a majority polymorphism $m : \mathbb{T}^3 \to \mathbb{T}$

An algorithm solving this problem was shown in [BCH$^+$13] and it is based on a theorem from [CDG13], which says that the polynomial time algorithm Singleton Arc Consistency can solve a CSP instance if the template has a majority polymorphism.

Singleton Arc Consistency It is known that there are templates $\mathbb{T}$ for which the problem $CSP(\mathbb{T})$ can be decided in polynomial time[1]. Interestingly, some algebraic properties of the structure $\mathbb{T}$ imply that $CSP(\mathbb{T})$ is tractable. We will say that an algorithm *solves* $CSP(\mathbb{T})$ if, for a given structure $\mathbb{A}$, it gives the answer YES iff there exists a homomorphism $\mathbb{A} \to \mathbb{T}$. There is a class of generic polynomial algorithms that solve $CSP(\mathbb{T})$ if some algebraic properties of $\mathbb{T}$ hold. In this section we will show two such algorithms: *Arc Consistency* and *Singleton Arc Consistency*.

By $[\mathbb{A}, R_1, R_2, \ldots, R_n]$, where $\mathbb{A}$ is a relational structure and $R_1, \ldots, R_n$ are relations over $\mathbb{A}$, we denote a structure obtained from $\mathbb{A}$ by adding the relations $R_1, \ldots, R_n$. If $\mathbb{A}, \mathbb{B}$ are

[1] For example 2-coloring of a graph is in P and corresponds to the template $\mathbb{T}_2$ (see *Examples* in Section 2.1)

structures over the same signature, $R_1, \ldots, R_n$ are relations over $\mathbb{A}$, $R'_1, \ldots, R'_n$ are relations over $\mathbb{B}$, and, for every i, R_i has the same arity as R'_i, then we will say that the signatures of the structures $[\mathbb{A}, R_1, R_2, \ldots, R_n]$ and $[\mathbb{B}, R'_1, R'_2, \ldots, R'_n]$ are the same.

The first algorithm we present is Arc Consistency[2]. This algorithm has polynomial time complexity if we assume a fixed maximal arity of a relation.

Algorithm 2.1: Arc Consistency

Data: a pair of $\mathcal{R}$-structures: $(\mathbb{A}, \mathbb{T})$
begin
  forall $a \in \mathbb{A}$ **do**
    $D_a := \mathbb{T}$;
  end
  repeat
    forall *relations* $R \in \mathcal{R}$ **do**
      forall *tuples* $(a_1, \ldots, a_k) \in R^{\mathbb{A}}$ **do**
        forall $i \in \{1, \ldots, k\}$ **do**
          $D_{a_i} := \pi_i(R^{\mathbb{T}} \cap (D_{a_1} \times \ldots \times D_{a_k}))$;
        end
      end
    end
  until *no set* D_a *is changed*;
  if *there exists* $a \in \mathbb{A}$ *such that* $D_a = \emptyset$ **then**
    return $\perp$
  else
    return $(D_a)_{a \in \mathbb{A}}$
  end
end

Arc Consistency has time complexity $O(rea^{r+1}b^{r+1})$, where a is the size of the structure $\mathbb{A}$, b is the size of the template $\mathbb{T}$, e is the number of all relations, and r is the maximal arity of a relation. The space complexity of Arc Consistency is $O(ab)$.

The second algorithm we present is Singleton Arc Consistency, which uses Arc Consistency internally.

[2] In the literature this algorithm is often called *General Arc Consistency*, as *Arc Consistency* refers to its equivalent version for binary structures.

Algorithm 2.2: Singleton Arc Consistency

Data: a pair of $\mathcal{R}$-structures: $(\mathbb{A}, \mathbb{T})$
begin
  forall $a \in \mathbb{A}$ **do**
    $D_a := \mathbb{T}$;
  end
  denote $\{a_1, \ldots, a_n\}$ elements of $\mathbb{A}$
  repeat
    forall $a \in \mathbb{A}$, $b \in D_a$ **do**
      if $AC([\mathbb{A}, \{a_1\}, \ldots, \{a_n\}, \{a\}], [\mathbb{T}, D_{a_1}, \ldots, D_{a_n}, \{b\}]) = \perp$ **then**
        remove b from D_a
      end
    end
  until *no set* D_a *is changed*;
  if *there exists* $a \in \mathbb{A}$ *such that* $D_a = \emptyset$ **then**
    return $\perp$
  else
    return $(D_a)_{a \in \mathbb{A}}$
  end
end

Singleton Arc Consistency has time complexity $O(rea^{r+3}b^{r+3})$ and space complexity $O(ab)$, where a, b, e are the same as for Arc Consistency.

Both of the algorithms above have an algebraic property associated with them, such that the algorithm solves $CSP(\mathbb{T})$ if and only if this property holds for $\mathbb{T}$ (*see* [CDG13]). What is interesting for us is the following theorem.

Theorem 2.10 ([CDG13])**.** *If* $\mathbb{T}$ *is a structure that has a majority polymorphism, then Singleton Arc Consistency solves* $CSP(\mathbb{T})$*.*

For any template $\mathbb{T}$, if AC solves $CSP(\mathbb{T})$, then clearly SAC solves $CSP(\mathbb{T})$. The converse, however, does not hold. In particular, Theorem 2.10, does not hold for AC – there are templates with a majority polymorphism for which the CSP cannot be solved by AC. An example of such a template is a two-element clique $\mathbb{T}_2$ which corresponds to the problem of coloring a graph with two colors (see *Examples* is Section 2.1). It has a majority polymorphism. However, imagine how the AC algorithm will work on a pair $(\mathbb{T}_3, \mathbb{T}_2)$, where $\mathbb{T}_3$ is the three-element clique. It is easy to see that no domain will be changed, so the AC algorithm will give the answer YES. But $\mathbb{T}_3$ is not 2-colorable so there is no homomorphism $\mathbb{T}_3 \to \mathbb{T}_2$.

Backtrack-free algorithm It is worth noticing that Theorem 2.10 provides a way of constructing, in polynomial time, a homomorphism $h : \mathbb{A} \to \mathbb{T}$, where $\mathbb{A} \in CSP(\mathbb{T})$ and $\mathbb{T}$ has a majority polymorphism. Consider the following algorithm:

Algorithm 2.3: Backtrack-free

Data: a pair of $\mathcal{R}$-structures: $(\mathbb{A}, \mathbb{T})$
begin
  if $SAC(\mathbb{A}, \mathbb{T}) = \perp$ **then**
    return $\perp$;
  end
  denote $\{a_1, \ldots, a_n\}$ elements of $\mathbb{A}$;
  forall $i \in \{1, \ldots, n\}$ **do**
    $b_i := NIL$;
    forall $b \in \mathbb{T}$ **do**
      if $SAC([\mathbb{A}, \{a_1\}, \ldots, \{a_{i-1}\}, \{a_i\}], [\mathbb{T}, \{b_1\}, \ldots, \{b_{i-1}\}, \{b\}]) \neq \perp$ **then**
        $b_i := b$;
      end
    end
    if $b_i = NIL$ **then**
      return $\perp$;
    end
  end
  return $((a_1, b_1), \ldots, (a_n, b_n))$
end

The correctness of this algorithm is quite straightforward and can be proved by induction. If there exists a homomorphism $h : [\mathbb{A}, \{a_1\}, \ldots, \{a_n\}] \to [\mathbb{T}, \{b_1\}, \ldots, \{b_n\}]$ and $a \in \mathbb{A} \setminus \{a_1, \ldots, a_n\}$, then there is at least one element $b \in \mathbb{T} \setminus \{b_1, \ldots, b_n\}$ such that there is a homomorphism $[\mathbb{A}, \{a_1\}, \ldots, \{a_n\}, \{a\}] \to [\mathbb{T}, \{b_1\}, \ldots, \{b_n\}, \{b\}]$ – namely $b = h(a)$. Since a majority polymorphism preserves any new unary one-element relations added to a structure (as $m(x, x, x) = x$), then $[\mathbb{T}, \{b_1\}, \ldots, \{b_n\}]$ has a majority polymorphism and SAC algorithm works correctly for it. This proves the correctness of the backtrack-free algorithm.

The backtrack-free algorithm runs SAC algorithm ab times (in the worst case scenario), so its time complexity is $O(rea^{r+4}b^{r+4})$. The space complexity of this algorithm is $O(ab)$.

Detecting majority Let us fix an $\mathcal{R}$-structure $\mathbb{T}$. We will show that the question whether $\mathbb{T}$ has a majority polymorphism is, in fact, an instance of CSP.

Let $\{x_1, x_2, \ldots, x_n\}$ denote the universe of $\mathbb{T}$. Let

$$\mathbb{T}_U = [\mathbb{T}, U_{x_1}^{\mathbb{T}_U}, U_{x_2}^{\mathbb{T}_U}, \ldots, U_{x_n}^{\mathbb{T}_U}]$$

and

$$\mathbb{T}_U^3 = [\mathbb{T}^3, U_{x_1}^{\mathbb{T}_U^3}, U_{x_2}^{\mathbb{T}_U^3}, \ldots, U_{x_n}^{\mathbb{T}_U^3}],$$

where, for every $x \in \mathbb{T}$,

$$U_x^{\mathbb{T}_U} = \{x\}$$

and

$$U_x^{\mathbb{T}_U^3} = \{(x, x, a), (x, a, x), (a, x, x) : a \in \mathbb{T}\}.$$

Consider two simple propositions.

Proposition 2.11. *$\mathbb{T}$ has a majority polymorphism iff there exists a homomorphism*

$$h : \mathbb{T}_U^3 \to \mathbb{T}_U.$$

Proof. Let T be the universe of $\mathbb{T}$ (and of $\mathbb{T}_U$ as well, by definition). Notice that a function $h : T^3 \to T$ preserves the relations U_x, for any x, iff it is a majority function, i.e. $h(x,x,y) = h(x,y,x) = h(y,x,x) = x$. This means that a polymorphism $h : \mathbb{T}_U^3 \to \mathbb{T}_U$ is a majority function, so it is a majority polymorphism of $\mathbb{T}$ and, conversely, a majority polymorphism $m : \mathbb{T}^3 \to \mathbb{T}$ preserves the relations U_x, for any x, so it is a homomorphism $\mathbb{T}_U^3 \to \mathbb{T}_U$. □

Proposition 2.12. *$\mathbb{T}$ has a majority polymorphism if and only if $\mathbb{T}_U$ has one.*

Proof. A majority polymorphism of $\mathbb{T}_U$ is clearly a majority polymorphism of $\mathbb{T}$ — it preserves all the relations. Conversely, let $h : \mathbb{T}^3 \to \mathbb{T}$ be a majority polymorphism. Note that for any x, $U_x^{(\mathbb{T}_U)^3} = \{(x,x,x)\}$ and $h(x,x,x) = x$, so h preserves all the relations U_x. This means that it is a majority polymorphism of $\mathbb{T}_U$. □

Now consider the following algorithm:

Algorithm 2.4: Detect majority

```
Data: an R-structure: 𝕋
begin
    create 𝕋_U and 𝕋_U^3 as described above;
    if BacktrackFree (𝕋_U, 𝕋_U^3) =⊥ then
        return NO;
    else
        return YES;
    end
end
```

Let us show that this algorithm produces the answer YES if and only if $\mathbb{T}$ has a majority polymorphism. If the answer is YES, then there is a homomorphism $\mathbb{T}_U^3 \to \mathbb{T}_U$, so $\mathbb{T}$ has a majority polymorphism. Conversely, if $\mathbb{T}$ has a majority polymorphism then there is a homomorphism $\mathbb{T}_U^3 \to \mathbb{T}_U$ and $\mathbb{T}_U$ has a majority polymorphism. By Theorem 2.10, SAC algorithm is correct for $\mathbb{T}_U$, so the algorithm will produce answer YES.

Complexity of the algorithm The complexity of the backtrack-free algorithm run on a pair $(\mathbb{A}, \mathbb{B})$ is $O(rea^{r+4}b^{r+4})$, where a is the size of the structure $\mathbb{A}$, b is the size of the template $\mathbb{B}$, e is the number of all relations, and r is the maximal arity of a relation. The size of $\mathbb{T}_A$ is $\sum_{k=1}^{d} k! = O(d!)$ and the maximal arity of a relation is d. Using this formula to compute the complexity of Algorithm 2.4, we get

$$O(ed(d!)^{4d+16}).$$

However, it easy to notice that, if τ_a is the maximal size of a relation in $\mathbb{A}$ and τ_b is the maximal size of a relation in $\mathbb{B}$, then $AC(\mathbb{A}, \mathbb{B})$ has time complexity $O(rea\tau_a b\tau_b)$. Then, the complexity of the backtrack-free algorithm run on a pair $(\mathbb{A}, \mathbb{B})$ is $O(rea^4b^4\tau_a\tau_b)$.

We showed that the maximal size of a relation in the template is $d!$. This means that $\tau_b = d!$, $\tau_a = (d!)^3$, and the time complexity of Algorithm 2.4 is

$$O(ed(d!)^{20}).$$

Chapter 3

Optimizations

We can optimize the algorithm in two ways. First, we can use faster algorithms to compute singleton arc consistency – such algorithms have been shown e.g. in [BE04], [BD08], [LC05] and we will present one of them in Section 3.1. The second approach is to use the algebraic properties of the template in order to reduce the size of the CSP instance, i.e. find an CSP instance of smaller size which has a majority polymorphism if and only if the original one has. Such reductions will be shown in Section 3.2.

3.1. Algorithms for SAC

Many algorithms that solve singleton arc consistency (i.e. produce the same result as the SAC algorithm in Section 2.3) have been proposed. The original $SAC(\mathbb{A}, \mathbb{B})$ algorithm, as presented in Section 2.3, has time complexity $O(rea^{r+3}b^{r+3})$ and space complexity $O(ab)$, where a is the size of the structure $\mathbb{A}$, b is the size of the template $\mathbb{B}$, e is the number of all relations, and r is the maximal arity of a relation. The algorithm presented in [BD08] (called SAC-Opt) is considered to be optimal with respect to time complexity[1]. Its time complexity is $O(r^2ea^{r+1}b^{r+1})$. The main drawback of this algorithm is its large space complexity – $O(era^{r+1}b^2)$. In this thesis we will use the SAC3 algorithm, presented in [LC05]. Its time complexity is $O(r^2ea^{r+2}b^{r+2})$ and space complexity – $O(ea^rb)$.

For the remainder of the section let us fix a signature $\mathcal{R}$ and $\mathcal{R}$-structures $\mathbb{A}$ and $\mathbb{B}$. Let $\{x_1, x_2, \ldots, x_n\}$ be the universe of $\mathbb{A}$. Let $\mathcal{R}_{un}$ and $\mathcal{R}_{nonun}$ denote the family of, respectively, unary and nonunary relations in $\mathcal{R}$. We define

$$D_i = \{a \in \mathbb{B} : \forall R \in \mathcal{R}_{un}\ x_i \in R^{\mathbb{A}} \implies a \in R^{\mathbb{B}}\},$$

$$C = \{((x_{i_1}, x_{i_2}, \ldots, x_{i_m}), R) : R \in \mathcal{R}_{nonun}, (x_{i_1}, x_{i_2}, \ldots, x_{i_m}) \in R^{\mathbb{A}}\}.$$

Elements of C will be called *constraints*. For every constraint $c = ((x_{i_1}, x_{i_2}, \ldots, x_{i_m}), R) \in C$, denote $var(c) = (x_{i_1}, x_{i_2}, \ldots, x_{i_m})$. Let $\mathbb{B}^{var(c)}$ be the set of all functions $\tau : var(c) \to \mathbb{B}$ and

$$rel(c) = \{\tau \in \mathbb{B}^{var(c)} : (\tau[x_{i_1}], \ldots, \tau[x_{i_m}]) \in \mathcal{R}^{\mathbb{B}}\}.$$

We assume a linear ordering on the elements of $\mathbb{A}$. We order the elements of $rel(c)$ in the lexicographical order. Let $succ\,(\tau, rel(c)|x_j \mapsto a)$, for $\tau \in \mathbb{B}^{var(c)}$, denote the smallest element in $rel(c)$ that is greater than τ and whose value at x_j is equal to a. We will assume that

[1] The complexity of a version of SAC-Opt for binary structures (i.e. having only unary and binary relations) is $O(ea^3b^3)$ and it is the lower bound for time complexity of a SAC algorithm under some assumption about the computational model [BD08].

the value of $succ(\tau, rel(c)|x_j \mapsto a)$ is undefined if such an element does not exists and for $\tau = nil$, $succ(\tau, rel(c)|x_j \mapsto a)$ is the smallest element in $rel(c)$ whose value at x_j is equal to a. This function, as used in the function Revise presented below, can be computed in amortized constant time by using a simple auxiliary data structure.

In SAC3 we will use variables br and M whose values are subsets of $\mathbb{A} \times \mathbb{B}$. Let $dom(br)$ and $dom(M)$ denote the projection of, respectively, br and M onto the first coordinate.

Let us present the algorithm AC2001, introduced in [BRYZ05], and the algorithm SAC3 introduced in [LC05], which uses AC2001 internally. The function Revise presented below differs slightly from the original – instead of iterating over the domain $D_{j_1} \times \ldots \times D_{j_r}$, we iterate over $rel(c)$. This modification does not affect the outcome of the algorithm but allows us to give better complexity bounds in terms of the maximal size of a relation.

Algorithm 3.1: AC2001

```
Data: C, D as defined above
begin
    Q := {(x_i, c) : c ∈ C, x_i ∈ var(c)};
    while Q ≠ ∅ do
        select and delete any (x_i, c) from Q;
        if Revise(x_i, c) then
            Q := Q ∪ {(x_j, c') : i ≠ j, c ≠ c', x_i, x_j ∈ var(c')};
        end
    end
    if ∀i D_i ≠ ∅ then
        return D
    else
        return ⊥
    end
end
```

Function Revise(x_i, c)

```
begin
    DELETE := false;
    forall a ∈ D_i do
        τ := Last[(x_i, a), c];
        if ∃j_k τ[x_{j_k}] ∉ D_{j_k} then
            τ := succ(τ, rel(c)|x_i ↦ a);
            while τ ≠ nil and ∃j_k τ[x_{j_k}] ∉ D_{j_k} do
                τ := succ(τ, rel(c)|x_i ↦ a);
            end
            if τ ≠ nil then
                Last[(x_i, a), c] := τ;
            else
                remove a from D_i;
                DELETE := true
            end
        end
    end
    return DELETE
end
```

Algorithm 3.2: SAC3

```
Data: C, D as defined above
begin
    D := AC2001(C, D);
    repeat
        D^before := D;
        M := {(x_i, a) : x_i ∈ 𝔸 ∧ a ∈ D_i};
        while M ≠ ∅ do
            buildBranch()
        end
    until D = D^before;
    if ∀i D_i ≠ ∅ then
        return D
    else
        return ⊥
    end
end
```

Function buildBranch

```
begin
    br := ∅;
    D^before := D;
    consistent := true;
    repeat
        pick and delete (x_i, a) ∈ M such that x_i ∉ dom(br);
        D := AC2001(C, (D_1, ..., D_{i-1}, {a}, D_{i+1}, ..., D_n));
        if D ≠⊥ then
            add (x_i, a) to br
        else
            consistent := false;
            if br ≠ ∅ then
                add (x_i, a) to M
            end
        end
    until ¬consistent ∨ dom(M) − dom(br) = ∅;
    D := D^before;
    if br = ∅ then
        remove a from D_i;
        D := AC(C, D);
        M := M − {(x_j, b) : b ∈ D_j^before ∧ b ∉ D_j};
    end
end
```

The algorithm AC2001 is a correct algorithm for computing arc consistency [BRYZ05] and SAC3 is a correct algorithm for computing singleton arc consistency [LC05]. The time complexity of the algorithm AC2001 is $O(r^2ea^rb^r)$ and the space complexity $O(ea^rb)$ [BRYZ05]. The SAC3 algorithm uses AC algorithm internally and its complexity depends on complexity of the AC algorithm used – its space complexity is the same as for the AC algorithm and

the time complexity is β times larger, where β is the number of branches built by SAC3 [LC05]. This means that, since $\beta = O(a^2b^2)$, the SAC3 algorithm that uses AC2001 has $O(r^2ea^{r+2}b^{r+2})$ time complexity and $O(ea^rb)$ space complexity.

However, we showed that the maximal size of a relation in the template defined in Section 2.2 is relatively small (at most the size of the template). If τ_a, τ_b are the maximal sizes of relations in $\mathbb{A}$, $\mathbb{B}$ respectively, then we can observe that the time complexity of the AC2001 algorithm is $O(r^2e\tau_a\tau_b)$ and the space complexity is $O(e\tau_ab)$. This means that the time complexity of SAC3 is $O(r^2ea^2\tau_ab^2\tau_b)$ and the space complexity is $O(e\tau_ab)$.

Recall that we use Algorithm 2.4 to detect the existence of a majority polymorphism. It uses the backtrack-free algorithm (Algorithm 2.3), which runs SAC $O(ab)$ times in order to build the polymorphism. This means that the algorithm that detects the existence of a majority polymorphism using SAC3 has $O(r^2ea^{r+3}b^{r+3})$ time complexity and $O(ea^rb)$ space complexity. Using the maximal size of a relation, we have that the backtrack-free algorithm has time complexity $O(r^2ea^3\tau_ab^3\tau_b)$ and space complexity $O(e\tau_ab)$, and Algorithm 2.4 has $O(ed^2(d!)^{16})$ time complexity and $O(e(d!)^4)$ space complexity, where d is the dimension of the alphabet.

3.2. Reducing size of the problem

Another approach to reducing the complexity is to find an equivalent instance of smaller size. We will use the algebraic properties of templates described in Chapter 1 in order to reduce the problem to a smaller one.

For the entire section, let us fix an alphabet $\mathbb{A}$ and let $\mathbb{T} := \mathbb{T}_A$ be the template defined in Chapter 1. Recall that Algorithm 2.4 determines whether $\mathbb{T}$ has a majority polymorphism by running the backtrack-free algorithm (Algorithm 2.3) on $(\mathbb{T}^3_U, \mathbb{T}_U)$. Notice the following proposition.

Proposition 3.1. *Let $\mathbb{A}$ and $\mathbb{B}$ be relational structures over the same signature. Assume that they have the following properties:*

- *$\mathbb{T}$ has a majority polymorphism if and only if there exists a homomorphism $h : \mathbb{A} \to \mathbb{B}$,*
- *if $\mathbb{T}$ has a majority polymorphism, then $\mathbb{B}$ also has a majority polymorphism.*

Then, the backtrack-free algorithm run on the pair $(\mathbb{A}, \mathbb{B})$ finds a solution if and only if $\mathbb{T}$ has a majority polymorphism.

Proof. If $\mathbb{T}$ has a majority polymorphism, then so does $\mathbb{B}$ and there exists a homomorphism $h : \mathbb{A} \to \mathbb{B}$. So, by Theorem 2.10, the backtrack-free will find a solution. Conversely, if the backtrack-free algorithm finds a solution, then there is a homomorphism $h : \mathbb{A} \to \mathbb{B}$ and $\mathbb{T}$ has a majority polymorphism. □

An example of such structures is the pair $(\mathbb{T}^3_U, \mathbb{T}_U)$ (see Proposition 2.11 and Proposition 2.12). Our goal in this section is to find as small as possible structures $\mathbb{A}, \mathbb{B}$ that meet these requirements.

Algebraic properties of the template Recall that the universe of $\mathbb{T}_A$ is $T_A = S_1 \sqcup S_2 \sqcup \ldots \sqcup S_{\dim A}$ and we say that $x, y \in T_A$ have the same type if $x, y \in S_i$ for some i. In Chapter 1 the following properties of relations in $\mathbb{T}$ were stated (Fact 2.6, Fact 2.7, Fact 2.8, and Fact 2.9):

- All relations in $\mathbb{T}$ are typed (a projection onto any coordinate is a subset of S_i for some i).
- Every r-ary relation R is a coset of a group $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$, for some $n_1, n_2, \ldots, n_r$, and $n_1 + n_2 + \ldots + n_r = \dim A$.
- For a relation R such that $R^{\mathbb{T}} \subseteq S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$ and $\mathbf{a}, \mathbf{b} \in S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$, $\mathbf{a} = (a_1, a_2, \ldots a_r)$, $\mathbf{b} = (b_1, b_2, \ldots b_r)$, there is a relation R' such that

$$R'^{\mathbb{T}} = \mathbf{a} \cdot R^{\mathbb{T}} \cdot \mathbf{b} = \{(a_1 \cdot x_1 \cdot b_1, a_2 \cdot x_2 \cdot b_2, \ldots, a_r \cdot x_r \cdot b_r) : (x_1, x_2, \ldots, x_r) \in R^{\mathbb{T}}\}.$$

These are the only properties of the template that we will use to reduce the CSP instance to a smaller problem.

Majority and structural relations Let us recall the construction of the structures $\mathbb{T}_U^3$ and $\mathbb{T}_U$ from Section 2.3. $\mathbb{T}_U$ is a structure with additional unary relations U_x for every $x \in \mathbb{T}$ and $U_x^{\mathbb{T}_U} = \{x\}$. We will call these additional relations *majority relations*, while the original relations from structure $\mathbb{T}$ will be called *structural relations*. The structure $\mathbb{T}_U^3$ is obtained from the structure $\mathbb{T}^3$ by adding majority relations (so $\mathbb{T}_U$ and $\mathbb{T}_U^3$ are over the same signature) and

$$U_x^{\mathbb{T}_U^3} = \{(x, x, a), (x, a, x), (a, x, x) : a \in \mathbb{T}\}.$$

Let T denote the universe of $\mathbb{T}_U$. For the remainder of the chapter, it is useful to define the following function $m : T^3 \to T$:

$$m(x, y, z) = \begin{cases} y & \text{if } y = z \\ x & \text{otherwise} \end{cases}$$

It is clear that the function m preserves all majority relations and for all x, y, z we have that $m(x, y, z) \in \{x, y, z\}$.

Reductions to smaller problems First, let us recall that for an alphabet of dimension d, the size of $\mathbb{T}_U$ is $\sum_{k=1}^{d} k!$, the size of T_U^3 is $\left(\sum_{k=1}^{d} k!\right)^3$, and the maximal arity of a relation is d. The maximal size of a relation in $\mathbb{T}_U$ is $d!$ and the maximal size of a relation in $\mathbb{T}_U^3$ is $(d!)^3$.

However, not all elements in $\mathbb{T}_U^3$ are necessary.

Proposition 3.2. *Let $\mathbb{M} \subseteq \mathbb{T}_U^3$ be a substructure of $\mathbb{T}_U^3$ whose universe is the set of all triples consisting of elements of the same type (i.e. $\{(x, y, z) \in \mathbb{T}_U^3 : \exists i(x, y, z \in S_i)\}$). Then, there exists a homomorphism $h : \mathbb{T}_U^3 \to \mathbb{T}_U$ if and only if there exists a homomorphism $h' : \mathbb{M} \to \mathbb{T}_U$.*

Proof. If $h : \mathbb{T}_U^3 \to \mathbb{T}_U$ then $h' = h_{|\mathbb{M}} : \mathbb{M} \to \mathbb{T}_U$ is a homomorphism. Conversely, assume that there is a homomorphism $h' : \mathbb{M} \to \mathbb{T}_U$. Let us define $h : \mathbb{T}_U^3 \to \mathbb{T}_U$:

$$h(x, y, z) = \begin{cases} h'(x, y, z) & \text{if } (x, y, z) \in M \\ m(x, y, z) & \text{otherwise} \end{cases}$$

We will show that the function h is a homomorphism. It preserves all majority relations because the functions h' and m do so. We need to show that h preserves the structural relations.

Let R be a structural relation of arity r. Let us take any tuple

$$((x_1, y_1, z_1), \ldots, (x_r, y_r, z_r)) \in R^{\mathbb{T}_U^3}.$$

Denote

$$\mathbf{x} = (x_1, \ldots, x_r).$$

We will use analogous notation for other variables. Moreover let us denote

$$\bar{h}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (h(x_1, y_1, z_1), \ldots, h(x_r, y_r, z_r)).$$

By the definition of $\mathbb{T}_U^3$, we have that

$$\mathbf{x}, \mathbf{y}, \mathbf{z} \in R^{\mathbb{T}_U}.$$

Since structural relations in $\mathbb{T}_U$ are typed, we have that, for any $i \in \{1, \ldots, r\}$, the elements x_i, y_i, z_i have the same type, so $(x_i, y_i, z_i) \in \mathbb{M}$ and $h(x_i, y_i, z_i) = h'(x_i, y_i, z_i)$. This means that

$$\bar{h}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \bar{h}'(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in R^{\mathbb{M}},$$

so h preserves the relation R. □

This proposition implies the following corollary.

Corollary 3.3. *The structure $\mathbb{T}$ has a majority polymorphism if and only if there exists a homomorphism $\mathbb{M} \to \mathbb{T}_U$.*

The size of $\mathbb{M}$ is equal to

$$\sum_{k=1}^{d} (k!)^3.$$

The maximal size of a relation in $\mathbb{M}$ is the same as for $\mathbb{T}_U^3$ and is equal to $(d!)^3$.

Since, for any $(x, y, z) \in \mathbb{M}$, x, y, z, have the same type, we will say that the type of the tuple (x, y, z) is the same as the type of the elements of this tuple. It is clear that all relations in $\mathbb{M}$ are typed and for every relation R, $R^{\mathbb{T}_U}$ and $R^{\mathbb{M}}$ have the same types on corresponding coordinates. Moreover, let us show that, without loss of generality, we can assume that if there exists a homomorphism, then it preserves the types.

Proposition 3.4. *Let $\mathbb{A}$ be a substructure of $\mathbb{M}$ and $h : \mathbb{A} \to \mathbb{T}_U$ be a homomorphism. Then there exists a homomorphism $h' : \mathbb{A} \to \mathbb{T}_U$ such that for all $m \in \mathbb{A}$, m and $h'(m)$ have the same type.*

Proof. Define $C = \{(x, y, z) \in \mathbb{A} : (x, y, z)$ has different type than $h(x, y, z)\}$. We define $h' : \mathbb{A} \to \mathbb{T}_U$ as follows:

$$h'(x, y, z) = \begin{cases} m(x, y, z) & \text{if } (x, y, z) \in C \\ h(x, y, z) & \text{otherwise} \end{cases}$$

Clearly h' preserves all majority relations and for any $(x, y, z) \in \mathbb{A}$, the tuple (x, y, z) has the same type as $h'(x, y, z)$ (h' preserves types). Moreover, let us notice that no $(x, y, z) \in C$ can be a member of any tuple in any structural relation $R^{\mathbb{A}}$. This comes from the fact that h preserves the relation R and R is typed – this means that (x, y, z) and $h(x, y, z)$ would need to have the same type. Therefore h' preserves structural relations. □

The next reduction omits the elements of $S_{\dim A}$ and $S_{\dim A-1}$.

Proposition 3.5. *Let $\mathbb{V} \subseteq \mathbb{T}_U$, $\mathbb{M}' \subseteq \mathbb{M}$ be substructures such that*

$$\mathbb{V} = \{x \in \mathbb{T}_U : x \in S_1 \sqcup S_2 \sqcup \ldots \sqcup S_{\dim A-2}\},$$

$$\mathbb{M}' = \{(x, y, z) \in \mathbb{M} : x, y, z \in S_1 \sqcup S_2 \sqcup \ldots \sqcup S_{\dim A-2}\}.$$

Then, there exists a homomorphism $h : \mathbb{M} \to \mathbb{T}_U$ if and only if there exists a homomorphism $h' : \mathbb{M}' \to \mathbb{V}$.

Proof. If $h : \mathbb{M} \to \mathbb{T}_U$ is a homomorphism, then $h_{|\mathbb{M}'}$ is also a homomorphism. Moreover, by Proposition 3.4, we can assume without loss of generality that h preserves types, so for all $(x, y, z) \in \mathbb{M}'$, $h(x, y, z) \in \mathbb{V}$. Hence $h_{|\mathbb{M}'} : \mathbb{M}' \to \mathbb{V}$.

Conversely, assume that there exists $h' : \mathbb{M}' \to \mathbb{V}$. We define $h : \mathbb{M} \to \mathbb{T}_U$ in the following way:

$$h(x, y, z) = \begin{cases} h'(x, y, z) & \text{if } (x, y, z) \in \mathbb{M}' \\ m(x, y, z) & \text{otherwise} \end{cases}$$

Clearly h preserves all majority relations; let us show that it preserves all structural relations.

Let R be a structural relation. By Fact 2.7, $R^{\mathbb{T}_U} \subseteq S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$ and $n_1 + n_2 + \ldots + n_r = \dim A$. If, for all i, $n_i \leq \dim A - 2$, then, by the definition of $\mathbb{M}'$, all members of all tuples in $R^{\mathbb{M}}$ are in $\mathbb{M}'$. This means that h preserves R because h' does so.

On the other hand, if $n_i \geq \dim A - 1$ for some i, we have two cases. Without loss of generality, assume $n_1 \geq \dim A - 1$. Then, either $n_1 = \dim A$ and then $r = 1$ and R is unary, or $n_1 = \dim A - 1$ and then $r = 2$ and $n_2 = 1$.

In the first case, when R is unary, for any tuple $(x, y, z) \in R^{\mathbb{M}}$, we have that $x, y, z \in R^{\mathbb{T}_U}$ and $(x, y, z) \in S_{\dim A}$, so $(x, y, z) \notin \mathbb{M}'$. This means that

$$h(x, y, z) = m(x, y, z) \in \{x, y, z\} \subseteq R^{\mathbb{T}_U}$$

and h preserves R.

In the second case, when $n_1 = \dim A - 1$ and $n_2 = 1$, for any $((x, y, z), (x', y', z')) \in R^{\mathbb{M}}$, we have that $x', y', z' \in S_1$. This means that $x' = y' = z' = e$, where e is the neutral (and the only) element of S_1. Moreover $(x, e), (y, e), (z, e) \in R^{\mathbb{T}_U}$ and $(x, y, z) \in S_{\dim A-1}$, so $(x, y, z) \notin \mathbb{M}'$. This means that

$$(h(x, y, z), h(e, e, e)) = (m(x, y, z), h'(e, e, e)) = (m(x, y, z), e) \in \{(x, e), (y, e), (z, e)\} \subseteq R^{\mathbb{T}_U}.$$

Hence, h preserves R.

This means that all structural relations are preserved by h and h is a homomorphism. □

The following corollary is a direct consequence of Proposition 3.5 and Corollary 3.3.

Corollary 3.6. *$\mathbb{T}$ has a majority polymorphism if and only if there exists a homomorphism $\mathbb{M}' \to \mathbb{V}$.*

Notice a simple fact.

Fact 3.7. *If $\mathbb{T}$ has a majority polymorphism, then $\mathbb{V}$ has a majority polymorphism.*

This fact follows Proposition 2.12. If $\mathbb{T}$ has a majority polymorphism, then $\mathbb{T}_U$ also has a majority polymorphism $h : (\mathbb{T}_U)^3 \to \mathbb{T}$ and we can assume without loss of generality that it preserves types. Then it is easy to see that $h_{|\mathbb{V}^3}$ is a majority polymorphism of $\mathbb{V}$.

It is easy to notice that the size of $\mathbb{V}$ is equal to

$$\sum_{k=1}^{d-2} k!$$

and the size of $\mathbb{M}'$ is

$$\sum_{k=1}^{d-2} (k!)^3.$$

However, the maximal size of a relation is the same as for $\mathbb{T}_U^3$ and $\mathbb{T}_U$ and is equal to $(d!)^3$ and $d!$ for $\mathbb{M}'$ and $\mathbb{V}$ respectively.

Let us consider the substructure $\mathbb{M}'' \subseteq \mathbb{M}'$ whose universe is the set of all tuples from $\mathbb{M}'$ which have a neutral element of S_n on the first coordinate. We will prove the following proposition.

Proposition 3.8. *Every homomorphism $h' : \mathbb{M}'' \to \mathbb{V}$ can be extended to a homomorphism $h : \mathbb{M}' \to \mathbb{V}$.*

Proof. By Proposition 3.4, we can assume without loss of generality that, for every $m \in \mathbb{M}''$, m and $h'(m)$ have the same type. We will show that the extension defined as

$$h(x, y, z) = x \cdot h'(e, x^{-1} \cdot y, x^{-1} \cdot z),$$

where e is the neutral element of the group which is a type of x, y, z, is a homomorphism. We need to show that h preserves all relations. Let $(x, y, z) \in U_a^{\mathbb{M}'}$. It is easy to see that $(e, x^{-1} \cdot y, x^{-1} \cdot z) \in U_{x^{-1} \cdot a}^{\mathbb{M}''}$, so $h'(e, x^{-1} \cdot y, x^{-1} \cdot z) = x^{-1} \cdot a$ and

$$h(x, y, z) = x \cdot x^{-1} \cdot a = a.$$

Let R be an structural r-ary relation. Then, by Fact 2.8, $R^{\mathbb{V}} = R^{\mathbb{T}} \subseteq S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$ is a coset. This means that there exists $\mathbf{a} = (a_1, a_2, \ldots, a_r) \in S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$ such that $R^{\mathbb{T}} = \mathbf{a} \cdot Z$, where Z is a subgroup of $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_r}$. By Fact 2.9, Z is also a relation in $\mathbb{T}$, so there is $R' \in \mathcal{R}$ such that $R'^{\mathbb{T}} = Z$.

Let $((x_1, y_1, z_1), \ldots, (x_r, y_r, z_r)) \in R^{\mathbb{M}'}$ be any tuple. Then,

$$\mathbf{x}, \mathbf{y}, \mathbf{z} \in R^{\mathbb{T}}.$$

By the definition of R',

$$\mathbf{a}^{-1}\mathbf{x}, \mathbf{a}^{-1}\mathbf{y}, \mathbf{a}^{-1}\mathbf{z} \in R'^{\mathbb{T}}$$

and, since $R'^{\mathbb{T}}$ is a group, we have that $\mathbf{e} \in R'^{\mathbb{T}}$ and

$$\mathbf{x}^{-1} \cdot \mathbf{y} = (\mathbf{a}^{-1}\mathbf{x})^{-1} \cdot (\mathbf{a}^{-1}\mathbf{y}) \in R'^{\mathbb{T}},$$

$$\mathbf{x}^{-1} \cdot \mathbf{z} = (\mathbf{a}^{-1}\mathbf{x})^{-1} \cdot (\mathbf{a}^{-1}\mathbf{z}) \in R'^{\mathbb{T}}.$$

This means that

$$\bar{h}'(\mathbf{e}, \mathbf{x}^{-1} \cdot \mathbf{y}, \mathbf{x}^{-1} \cdot \mathbf{z}) \in R'^{\mathbb{T}}$$

and, since $\mathbf{x} \in R^{\mathbb{T}}$,

$$\bar{h}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{x} \cdot \bar{h}'(\mathbf{e}, \mathbf{x}^{-1} \cdot \mathbf{y}, \mathbf{x}^{-1} \cdot \mathbf{z}) \in R^{\mathbb{T}} = R^{\mathbb{V}}.$$

Hence $h : \mathbb{M}' \to \mathbb{V}$ is a homomorphism. □

Corollary 3.9. $\mathbb{T}$ *has a majority polymorphism if and only if there exists a homomorphism* $\mathbb{M}'' \to \mathbb{V}$.

Proof. If $\mathbb{T}$ has a majority polymorphism, then, by Corollary 3.3, there exists a homomorphism $h : \mathbb{M}' \to \mathbb{V}$ and $h_{|\mathbb{M}''} : \mathbb{M}'' \to \mathbb{V}$ is a homomorphism. Conversely, if there exists a homomorphism $h' : \mathbb{M}'' \to \mathbb{V}$, then by Proposition 3.8, it can be extended to a homomorphism $h : \mathbb{M}' \to \mathbb{V}$, which, by Corollary 3.3 means that $\mathbb{T}$ has a majority polymorphism. □

The size of $\mathbb{M}''$ is

$$\sum_{k=1}^{d-2}(k!)^2.$$

The maximal size of a relation is $(d!)^2$.

Notice a simple but useful fact.

Fact 3.10. *Let R be a structural relation such that $R^{\mathbb{V}}$ is not a group. Then, $R^{\mathbb{M}''} = \emptyset$.*

This statement comes from the fact that all elements in $\mathbb{M}''$ have a neutral element on their first coordinate and if there is a tuple in $R^{\mathbb{M}''}$, then $(e_1, e_2, \ldots, e_r) \in R^{\mathbb{V}}$, where $e_1, e_2, \ldots, e_r$ are neutral elements. This means that $R^{\mathbb{V}}$ must be a group (since, by Fact 2.8, it is a coset).

This fact implies that every function $h : \mathbb{M}'' \to \mathbb{V}$ preserves all structural relations that are not groups in a trivial way. This means that:

Corollary 3.11. *If a function $h : \mathbb{M}'' \to \mathbb{V}$ preserves all majority relations and all structural relations that are groups, then h is a homomorphism.*

Therefore, throughout the remainder of thesis, we remove from the signature all structural relations that are not groups and we can assume that all structural relations are groups.

Let $\mathbb{D}$ be an $\mathcal{R}$-structure such that the universe of $\mathbb{D}$ are all pairs $(x, y) \in \mathbb{V}^2$ for which x, y have the same type, $U_a^{\mathbb{D}} = \{(a, a)\}$ for every a that is not a neutral element of any group, $U_e^{\mathbb{D}} = \{(e, x), (x, e) : x - \text{an element with the same type as } e\}$, where e is a neutral element of some group. For every structural relation R, we have $R^{\mathbb{D}} = R^{\mathbb{V}^2}$.

The reason why we want to use $\mathbb{D}$ is to simplify the notation. Notice the following proposition.

Proposition 3.12. $\mathbb{M}''$ *is isomorphic to* $\mathbb{D}$.

Proof. Let us just show the isomorphism. Let $h : \mathbb{D} \to \mathbb{M}''$, $h(x, y) = (e, x, y)$, where e is the neutral element of the group that is the type of x and y. Let $g : \mathbb{M}'' \to \mathbb{D}$, $g(e, x, y) = (x, y)$. Clearly $g \circ h$ and $h \circ g$ are identity function. Also it is straightforward that h and g are homomorphisms. □

Clearly the size of $\mathbb{D}$ is the same as the size of $\mathbb{M}''$, which is $\sum_{k=1}^{d-2}(k!)^2$.

Corollary 3.13. $\mathbb{T}$ *has a majority polymorphism if and only if there exists a homomorphism* $\mathbb{D} \to \mathbb{V}$.

We can reduce the size of the CSP instance even further. Let us recall the definition of a conjugacy class.

Definition 3.14. Let G be a group and $g \in G$ be any element. The *conjugacy class of g* is the set

$$[g] = \{x \cdot g \cdot x^{-1} : x \in G\}.$$

It is clear that any two conjugacy classes are either equal or distinct and each group can be presented as a sum of distinct conjugacy classes. Moreover, if e is the neutral element, then $[e] = \{e\}$.

For every S_i and every conjugacy class $[x] \subseteq S_i$, we choose an arbitrary representative of this class $r_{[x]} \in [x]$. Let *Rep* be the set of all such representatives. Let $\mathbb{D}_O \subseteq \mathbb{D}$ be a substructure generated by all pairs in $\mathbb{D}$ that have a member of *Rep* on their first coordinate. We will show the following proposition, which is very similar to Proposition 3.8.

Proposition 3.15. *Every homomorphism $h' : \mathbb{D}_O \to \mathbb{V}$ can be extended to a homomorphism $h : \mathbb{D} \to \mathbb{V}$.*

Proof. We will show how to construct such an extension. For any $x \in \mathbb{V}$ we arbitrarily choose an element $g_x \in \mathbb{V}$ such that

$$x = g_x^{-1} \cdot r_{[x]} \cdot g_x.$$

Without loss of generality, we can assume that if $x \in Rep$ (which is equivalent to $x = r_{[x]}$), then $g_x = e$, where e is the neutral element of the same type as x. Let us show that the following extension of $h' : \mathbb{D}_O \to \mathbb{V}$:

$$h(x, y) = g_x^{-1} h'(r_{[x]}, g_x y g_x^{-1}) g_x$$

is a homomorphism. As previously, we need to check that it preserves all relations. Let U_x be a majority relation where x is not a neutral element. Then $U_x^{\mathbb{D}} = \{(x, x)\}$ and

$$h(x, x) = g_x^{-1} h'(r_{[x]}, g_x x g_x^{-1}) g_x = g_x^{-1} h'(r_{[x]}, r_{[x]}) g_x.$$

As $(r_{[x]}, r_{[x]}) \in U_{r_{[x]}}^{\mathbb{D}}$ and $U_{r_{[x]}}^{\mathbb{V}} = \{r_{[x]}\}$, we have $h'(r_{[x]}, r_{[x]}) = r_{[x]}$ and

$$h(x, x) = g_x^{-1} r_{[x]} g_x = x \in U_x^{\mathbb{V}}.$$

Let us show that h preserves U_e, where e is a neutral element. Since $[e] = \{e\}$, we have that $r_{[e]} = e$. Recall that we have

$$U_e^{\mathbb{D}} = \{(e, a), (a, e) : a \text{ an element with the same type as } e\}.$$

Let a be an element with the same type as e. Then

$$h(e, a) = g_e^{-1} h'(e, g_e a g_e^{-1}) g_e = g_e^{-1} e g_e = e.$$

And

$$h(a, e) = g_a^{-1} h'(r_{[a]}, g_a e g_a^{-1}) g_a = g_a^{-1} h'(r_{[a]}, e) g_a = g_a^{-1} e g_a = e.$$

This means that h preserves the relation U_e.

Let R be a n-ary structural relation. Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ where a_i is an element of the type of i-th coordinate of R and let $R_{\mathbf{a}}$ be a relation such that

$$R_{\mathbf{a}}^{\mathbb{V}} = \{\mathbf{a} \cdot \mathbf{x} \cdot \mathbf{a}^{-1} : \mathbf{x} \in R^{\mathbb{V}}\}.$$

By Fact 2.9 such a relation exists for any $\mathbf{a}$. Let $((x_1, y_1), \ldots, (x_n, y_n)) \in R^{\mathbb{D}}$. Then

$$\mathbf{x}, \mathbf{y} \in R^{\mathbb{V}}.$$

Let us denote

$$\mathbf{r_x} = (r_{[x_1]}, \ldots, r_{[x_n]}),$$

$$\mathbf{g_x} = (g_{x_1}, \ldots, g_{x_n}).$$

Then, we have that

$$\mathbf{g_x} \cdot \mathbf{x} \cdot \mathbf{g_x}^{-1} = \mathbf{r_x} \in R^{\mathbb{V}}_{\mathbf{g_x}},$$

$$\mathbf{g_x} \cdot \mathbf{y} \cdot \mathbf{g_x}^{-1} \in R^{\mathbb{V}}_{\mathbf{g_x}},$$

so

$$\bar{h}'(\mathbf{r_x}, \mathbf{g_x} \cdot \mathbf{y} \cdot \mathbf{g_x}^{-1}) \in R^{\mathbb{V}}_{\mathbf{g_x}}.$$

This means that

$$\bar{h}(\mathbf{x}, \mathbf{y}) = \mathbf{g_x}^{-1} \bar{h}'(\mathbf{r_x}, \mathbf{g_x} \cdot \mathbf{y} \cdot \mathbf{g_x}^{-1}) \mathbf{g_x} \in R^{\mathbb{V}}$$

and h preserves all relations, hence is a homomorphism. □

Corollary 3.16. *$\mathbb{T}$ has a majority polymorphism if and only if there exists a homomorphism $\mathbb{D}_O \to \mathbb{V}$.*

Recall that Fact 3.7 says that, if $\mathbb{T}$ has a majority polymorphism, then $\mathbb{V}$ has a majority polymorphism. This fact, combined with Corollary 3.16 and Proposition 3.1, means that we can run the backtrack-free algorithm on $(\mathbb{D}_O, \mathbb{V})$ in order to check whether $\mathbb{T}$ has a majority polymorphism.

We will try to approximate the size of $\mathbb{D}_O$. Consider the following definition:

Definition 3.17. The *partition function* $p(n)$ gives the number of partitions of a nonnegative integer n into positive integers.

It is a well-known fact that:

Fact 3.18. *The number of conjugacy classes in the symmetric group S_n is equal to $p(n)$.*

The value of $p(n)$ is significantly smaller than $n!$ as the asymptotic formula for the partition function is

$$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}},$$

while the factorial function has an asymptotic formula

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

It is clear, by the definition of $\mathbb{D}_O$, that the size of $\mathbb{D}_O$ is

$$\sum_{k=1}^{d-2} p(k) \cdot k!.$$

Estimating the maximal size of a relation in $\mathbb{D}_O$ is a little bit more tricky. For any tuple of natural numbers $\mu = (\mu_1, \mu_2, \ldots, \mu_k)$, we define

$$p(\mu) = \prod_{i=1}^{k} p(\mu_i).$$

For any natural number n, we define the *maximal partition value of n* as

$$\mathrm{maxp}(n) = \max\left(p(\mu) : \sum \mu = n\right).$$

Denote $Rep_i = S_i \cap Rep$. Let us notice that any relation in $\mathbb{D}_O$ is a subset of

$$(Rep_{n_1} \times S_{n_1}) \times \ldots \times (Rep_{n_r} \times S_{n_r}),$$

where $n_1 + n_2 + \ldots + n_r = d$. Let $\mathbf{n} = (n_1, n_2, \ldots, n_r)$. The upper bound on the size of a relation in $\mathbb{D}_O$ is

$$|Rep_{n_1}| \cdot |S_{n_1}| \cdot \ldots \cdot |Rep_{n_r}| \cdot |S_{n_r}| = p(n_1) \cdot n_1! \cdot \ldots \cdot p(n_r) \cdot n_r!.$$

We have that

$$n_1! \cdot n_2! \cdot \ldots \cdot n_r! \leq d!$$

and

$$p(n_1) \cdot p(n_2) \cdot \ldots \cdot p(n_r) = p(\mathbf{n}) \leq \mathrm{maxp}(d).$$

This means that the maximal size of a relation in $\mathbb{D}_O$ is $\mathrm{maxp}(d) \cdot d!$. In [BO14] a closed formula of $\mathrm{maxp}(n)$ for $n > 8$ was found:

$$\mathrm{maxp}(n) = \begin{cases} 5^{\frac{n}{4}} & \text{if } n \equiv 0 \pmod 4 \\ 7 \cdot 5^{\frac{n-5}{4}} & \text{if } n \equiv 1 \pmod 4 \\ 11 \cdot 5^{\frac{n-6}{4}} & \text{if } n \equiv 2 \pmod 4 \\ 77 \cdot 5^{\frac{n-11}{4}} & \text{if } n \equiv 3 \pmod 4 \end{cases}$$

This means that $\mathrm{maxp}(n) = O(5^{\frac{n}{4}})$ and the maximal size of a relation is $O(5^{\frac{d}{4}} d!)$.

Table 3.1 summarized all the reduction we did.

Size comparison Let us compare sizes of the structures after each reduction. Table 3.3 and Table 3.2 show the sizes of, respectively, instances and templates after each reduction. To give better overview of order of those values, we show example values for $d = 5, 6, 7, 8, 9$.

Table 3.4 shows the time and space complexity of backtrack-free algorithm (Algorithm 2.3), run on each pair of structures. Recall, from Section 3.1, that the time complexity of the backtrack-free algorithm run on pair of structures $(\mathbb{A}, \mathbb{B})$, that uses internally SAC3 algorithm, is $O(r^2 e a^3 \tau_a b^3 \tau_b)$, where a is the size of $\mathbb{A}$, b is the size of $\mathbb{B}$ and r is the maximal arity of a relation, e is the number of relations and τ_a, τ_b are the maximal sizes of relations in $\mathbb{A}$, $\mathbb{B}$ respectively. The space complexity is bounded by $O(e\tau_a b)$. Recall also that the maximal size of a relation in $\mathbb{T}_U$ and $\mathbb{V}$ is $d!$, in $\mathbb{T}_U^3$, $\mathbb{M}$ and $\mathbb{M}'$ is $(d!)^3$, in $\mathbb{D}$ is $(d!)^2$, and in $\mathbb{D}_O$ is $\mathrm{maxp}(d) \cdot d! = O(5^{d/4} d!)$.

The most efficient algorithm for alphabet classification that we found has time complexity $O(ed^2((d-2)!)^6(p(d-2))^3(d!)^2 5^{d/4})$ and space complexity $O(e5^{d/4} d!(d-2)!)$.

Structures	Reduction
$(\mathbb{T}^3_U, \mathbb{T}_U)$	—
$(\mathbb{M}, \mathbb{T}_U)$	remove tuples of elements of different types
$(\mathbb{M}', \mathbb{V})$	remove elements from $S_{\dim A}$ and $S_{\dim A-1}$
$(\mathbb{D}, \mathbb{V}) \simeq (\mathbb{M}'', \mathbb{V})$	restrict to triples of M' of the form (e, x, y)
$(\mathbb{D}_O, \mathbb{V})$	take only representatives of conjugacy classes on the first element

Table 3.1: Reductions

Structure	Size	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$
$\mathbb{T}_U$	$\sum_k^d k!$	153	873	5,913	46,233	409,113
$\mathbb{V}$	$\sum_k^{d-2} k!$	9	33	153	873	5,913

Table 3.2: Sizes of templates

Structure	Size	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$
$\mathbb{T}^3_U$	$(\sum_k^d k!)^3$	3.6×10^6	6.7×10^8	2.1×10^{11}	9.9×10^{13}	6.8×10^{16}
$\mathbb{M}$	$\sum_k^d (k!)^3$	1.7×10^6	3.7×10^8	1.3×10^{11}	6.6×10^{13}	4.7×10^{16}
$\mathbb{M}'$	$\sum_k^{d-2} (k!)^3$	225	14,049	1.7×10^6	3.7×10^8	1.3×10^{11}
$\mathbb{D}$	$\sum_k^{d-2} (k!)^2$	41	617	15,017	533,417	2.6×10^7
$\mathbb{D}_O$	$\sum_k^{d-2} p(k) \cdot k!$	23	143	983	8,903	84,503

Table 3.3: Sizes of instances

Structures	Time complexity	Space complexity
$(\mathbb{T}^3_U, \mathbb{T}_U)$	$O(ed^2(d!)^{16})$	$O(e(d!)^4)$
$(\mathbb{M}, \mathbb{T}_U)$	$O(ed^2(d!)^{16})$	$O(e(d!)^4)$
$(\mathbb{M}', \mathbb{V})$	$O(ed^2((d-2)!)^{12}(d!)^4)$	$O(e(d!)^3(d-2)!)$
$(\mathbb{D}, \mathbb{V})$	$O(ed^2((d-2)!)^9(d!)^3)$	$O(e(d!)^2(d-2)!)$
$(\mathbb{D}_O, \mathbb{V})$	$O(ed^2((d-2)!)^6(p(d-2))^3(d!)^2 5^{d/4})$	$O(e5^{d/4}d!(d-2)!)$

Table 3.4: Complexity

Chapter 4

Conclusion

4.1. Technical issues

Implementation The algorithm for alphabet classification was implemented as a proof of concept but also as a research tool for testing hypotheses or finding nontrivial examples of nonstandard alphabets. It was implemented in Haskell mostly because of algebraic data types and pattern matching. Algebraic data types are, as we think, the most natural way of representing abstract mathematical objects, such as relational structures. Moreover, operations on those structure, such as building them and performing the reductions described in Chapter 3, are much nicer with higher order function (e.g. *map* or *filter*), which can be used natively in Haskell.

We implemented a small generic CSP library that provides some useful functions that operate on relational structures. We also implemented Arc Consistency and Singleton Arc Consistency algorithms — SAC3 and AC2001 — and the algorithm for detecting the existence of a majority polymorphism.

Results We classified all alphabets up to dimension 8. This was achieved as follows. First, we classified all single-orbit alphabets. According to Lemma 1.1, single-orbit alphabets of dimension d correspond to permutation groups on d elements. Permutation groups on d elements (up to isomorphism) correspond to subgroups of the symmetric group S_d (up to conjugacy). Using GAP (a system for computational discrete algebra), we generated all subgroups of S_8 (up to conjugacy). There are 296 such subgroups (see `http://groupprops.subwiki.org/wiki/Subgroup_structure_of_symmetric_group:S8`). This means that, up to isomorphism, there are 296 single-orbit alphabets of dimension 8. For each of those alphabets we determined whether it is standard or not, by first computing the corresponding pair $(\mathbb{D}_O, \mathbb{V})$, and then running the backtrack-free algorithm (Algorithm 2.3) on this pair. Out of the 296 single-orbit alphabets, 163 turned out to be standard, and the remaining 133 non-standard. This gives a classification of all single-orbit alphabets of dimension 8 (listed in Appendix A).

To classify all (not necessarily single-orbit) alphabets of dimension 8, we considered the alphabet which is the union of all 163 standard alphabets of dimension 8, and tested whether the resulting alphabet is standard. The result turned out to be positive. This gives us the following theorem.

Theorem 4.1. *An alphabet of dimension up to* 8 *is standard if and only if each of its orbits is standard.*

The analogous problem for higher-dimension alphabets remains open.

d	Number of alphabets	Average time	Maximal time
5	19	38 ms	375 ms
6	56	1.18 s	11.5 s
7	96	3.6 min	224.3 min

Table 4.1: Running times

Efficiency The complexity of the improved algorithm, even though significantly smaller than the one in [KLOT14], is still large — $O(ed^2((d-2)!)^6(p(d-2))^3(d!)^2 5^{d/4})$, where d is the dimension of an alphabet and e is the number of relations in the template. Table 4.1 shows the time needed to classify all single-orbit alphabets of a given dimension on a regular one-core personal computer. Classifying all alphabets of dimension 8 is out of reach on a personal computer — this took about a week on a multi-core high-end machine.

The running time of the algorithm varies among alphabets of the same dimension. It might be partially due to the structure of the algorithm and the use of SAC3 algorithm. When an alphabet is nonstandard, the inconsistency can be found on any stage of the program execution. On the other hand, when an alphabet is standard, SAC3 may, at any time, accidentally "guess" a homomorphism. Also the time complexity of SAC3 varies among different pairs of structures.

4.2. Summary

We showed and implemented an improved version of the algorithm from [KLOT14] which determines whether a given alphabet with atoms is standard. The improvement is significant and was achieved by using algebraic properties of a template, defined in [KLOT14], to reduce the size of the problem, as well as by using modern algorithm from the theory of CSP — SAC3 and AC2001. We implemented the algorithm in Haskell, mainly as a proof of concept, but it can also serve as a research tool in finding examples of interesting nonstandard alphabets and testing hypotheses.

Appendix A

Classification of alphabets of dimension 8

In this appendix we present the classification of all single-orbit alphabets (up to isomorphism) of dimension 8. This result was obtained as described in Section 4.1. Alphabets are represented as subgroups of S_8, as described in Lemma 1.1. A subgroup $G \subseteq S_8$ corresponds to an alphabet A_G defined, as follows [BKL14]: $A_G = \mathfrak{A}^8/G$, where G acts on tuples of $\mathfrak{A}^8$ by permuting its coordinates (i.e. for any $\pi \in G$, $\tau \in \mathfrak{A}^8$, and $i \in \{1, 2, \ldots, 8\}$, $(\pi \cdot \tau)[i] = \tau[\pi(i)]$, where $\tau[i]$ is the i-th coordinate of τ).

Let $\langle \pi_1, \pi_2, \ldots, \pi_k \rangle$ denote the group generated by permutations $\pi_1, \pi_2, \ldots, \pi_k$. We present below all single-orbit alphabets (up to isomorphism) of dimension 8.

Standard alphabets

1. $\langle () \rangle$
2. $\langle (7\ 8) \rangle$
3. $\langle (6\ 7\ 8) \rangle$
4. $\langle (5\ 6)(7\ 8) \rangle$
5. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8) \rangle$
6. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8) \rangle$
7. $\langle (7\ 8), (5\ 6) \rangle$
8. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7\ 6\ 8) \rangle$
9. $\langle (3\ 4)(5\ 6)(7\ 8) \rangle$
10. $\langle (7\ 8), (3\ 4)(5\ 6) \rangle$
11. $\langle (6\ 7\ 8), (7\ 8) \rangle$
12. $\langle (4\ 5\ 6\ 7\ 8) \rangle$
13. $\langle (7\ 8), (4\ 5\ 6) \rangle$
14. $\langle (6\ 7\ 8), (4\ 5)(7\ 8) \rangle$
15. $\langle (3\ 4\ 5)(6\ 7\ 8) \rangle$
16. $\langle (3\ 4\ 5)(6\ 7\ 8), (3\ 6)(4\ 8)(5\ 7) \rangle$
17. $\langle (6\ 7\ 8), (2\ 3)(4\ 5) \rangle$
18. $\langle (6\ 7\ 8), (2\ 3)(4\ 5)(7\ 8) \rangle$
19. $\langle (3\ 4)(5\ 6)(7\ 8), (3\ 5\ 7)(4\ 6\ 8) \rangle$
20. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4)(5\ 7\ 6\ 8) \rangle$
21. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4) \rangle$
22. $\langle (3\ 4\ 5)(6\ 7\ 8), (4\ 5)(7\ 8) \rangle$
23. $\langle (2\ 3\ 4\ 5\ 6\ 7\ 8) \rangle$

24. $\langle (3\ 4\ 5)(6\ 7\ 8), (1\ 2)(4\ 5)(7\ 8)\rangle$
25. $\langle (7\ 8), (1\ 2)(3\ 4)(5\ 6)\rangle$
26. $\langle (3\ 4\ 5)(6\ 7\ 8), (1\ 2)(3\ 6)(4\ 8)(5\ 7)\rangle$
27. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
28. $\langle (7\ 8), (5\ 6), (5\ 7)(6\ 8)\rangle$
29. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8)\rangle$
30. $\langle (7\ 8), (1\ 2\ 3)(4\ 5\ 6)\rangle$
31. $\langle (7\ 8), (5\ 6), (3\ 4)\rangle$
32. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5)(4\ 6)\rangle$
33. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8)\rangle$
34. $\langle (3\ 4\ 5)(6\ 7\ 8), (1\ 2)(3\ 6)(4\ 7)(5\ 8)\rangle$
35. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
36. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4)\rangle$
37. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8)\rangle$
38. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4)\rangle$
39. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (1\ 5\ 2\ 6)(3\ 8\ 4\ 7)\rangle$
40. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4)\rangle$
41. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5\ 4\ 6)\rangle$
42. $\langle (6\ 7\ 8), (3\ 4\ 5)\rangle$
43. $\langle (4\ 5\ 6\ 7\ 8), (5\ 8)(6\ 7)\rangle$
44. $\langle (7\ 8), (2\ 3\ 4\ 5\ 6)\rangle$
45. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8)\rangle$
46. $\langle (4\ 5\ 6\ 7\ 8), (2\ 3)(5\ 8)(6\ 7)\rangle$
47. $\langle (7\ 8), (4\ 5\ 6), (5\ 6)\rangle$
48. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5)\rangle$
49. $\langle (7\ 8), (3\ 4)(5\ 6), (1\ 2)(3\ 5\ 4\ 6)\rangle$
50. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8)\rangle$
51. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4\ 3\ 5)(7\ 8)\rangle$
52. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4\ 3\ 5)\rangle$
53. $\langle (6\ 7\ 8), (7\ 8), (2\ 3)(4\ 5)\rangle$
54. $\langle (7\ 8), (5\ 6), (2\ 3\ 4)\rangle$
55. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5\ 2\ 6)(3\ 7\ 4\ 8)\rangle$
56. $\langle (7\ 8), (4\ 5\ 6), (2\ 3)(5\ 6)\rangle$
57. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 8)(4\ 7)\rangle$
58. $\langle (2\ 3\ 4\ 5\ 6\ 7\ 8), (3\ 8)(4\ 7)(5\ 6)\rangle$
59. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
60. $\langle (7\ 8), (1\ 2\ 3)(4\ 5\ 6), (1\ 4)(2\ 6)(3\ 5)\rangle$
61. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8)\rangle$
62. $\langle (7\ 8), (1\ 2)(3\ 4)(5\ 6), (1\ 3\ 5)(2\ 4\ 6)\rangle$
63. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)\rangle$
64. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2)\rangle$
65. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (1\ 5\ 3\ 7\ 2\ 6\ 4\ 8)\rangle$
66. $\langle (7\ 8), (1\ 2\ 3)(4\ 5\ 6), (2\ 3)(5\ 6)\rangle$
67. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4)\rangle$
68. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)\rangle$
69. $\langle (6\ 7\ 8), (1\ 2\ 3\ 4\ 5)\rangle$
70. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)\rangle$
71. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (1\ 5\ 3\ 7\ 2\ 6\ 4\ 8)\rangle$
72. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (3\ 4)\rangle$

73. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3\ 2\ 4) \rangle$
74. $\langle (6\ 7\ 8), (3\ 4\ 5), (4\ 5)(7\ 8) \rangle$
75. $\langle (6\ 7\ 8), (3\ 4\ 5), (3\ 6)(4\ 7)(5\ 8) \rangle$
76. $\langle (6\ 7\ 8), (7\ 8), (3\ 4\ 5) \rangle$
77. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4) \rangle$
78. $\langle (4\ 5\ 6\ 7\ 8), (5\ 8)(6\ 7), (5\ 6\ 8\ 7) \rangle$
79. $\langle (4\ 5\ 6\ 7\ 8), (5\ 8)(6\ 7), (2\ 3)(5\ 6\ 8\ 7) \rangle$
80. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
81. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8) \rangle$
82. $\langle (7\ 8), (2\ 3\ 4\ 5\ 6), (3\ 6)(4\ 5) \rangle$
83. $\langle (2\ 3\ 4\ 5\ 6\ 7\ 8), (3\ 4\ 6)(5\ 8\ 7) \rangle$
84. $\langle (6\ 7\ 8), (7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5) \rangle$
85. $\langle (7\ 8), (5\ 6), (3\ 4), (3\ 5\ 7)(4\ 6\ 8) \rangle$
86. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5)(4\ 6), (4\ 5\ 6) \rangle$
87. $\langle (7\ 8), (4\ 5\ 6), (1\ 2\ 3) \rangle$
88. $\langle (7\ 8), (5\ 6), (2\ 3\ 4), (3\ 4) \rangle$
89. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (2\ 3\ 4) \rangle$
90. $\langle (6\ 7\ 8), (7\ 8), (2\ 3)(4\ 5), (2\ 4\ 3\ 5) \rangle$
91. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (1\ 2\ 3) \rangle$
92. $\langle (6\ 7\ 8), (3\ 4\ 5), (1\ 2)(3\ 6)(4\ 7)(5\ 8) \rangle$
93. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4) \rangle$
94. $\langle (6\ 7\ 8), (3\ 4\ 5), (1\ 2)(4\ 5)(7\ 8) \rangle$
95. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8), (1\ 5)(2\ 6)(3\ 8)(4\ 7) \rangle$
96. $\langle (6\ 7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4)(7\ 8) \rangle$
97. $\langle (6\ 7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4) \rangle$
98. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4) \rangle$
99. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (1\ 5\ 2\ 6)(3\ 8\ 4\ 7), (3\ 5\ 7)(4\ 6\ 8) \rangle$
100. $\langle (6\ 7\ 8), (7\ 8), (1\ 2\ 3\ 4\ 5) \rangle$
101. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8), (3\ 4)(7\ 8) \rangle$
102. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
103. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5), (3\ 4\ 5) \rangle$
104. $\langle (6\ 7\ 8), (3\ 4\ 5), (4\ 5)(7\ 8), (3\ 6)(4\ 7\ 5\ 8) \rangle$
105. $\langle (6\ 7\ 8), (7\ 8), (3\ 4\ 5), (4\ 5) \rangle$
106. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (3\ 4), (1\ 2) \rangle$
107. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4) \rangle$
108. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
109. $\langle (7\ 8), (4\ 5\ 6), (1\ 2\ 3), (2\ 3)(5\ 6) \rangle$
110. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
111. $\langle (2\ 3\ 4\ 5\ 6\ 7\ 8), (3\ 4\ 6)(5\ 8\ 7), (3\ 8)(4\ 7)(5\ 6) \rangle$
112. $\langle (7\ 8), (2\ 3\ 4\ 5\ 6), (3\ 6)(4\ 5), (3\ 4\ 6\ 5) \rangle$
113. $\langle (6\ 7\ 8), (7\ 8), (3\ 4\ 5), (1\ 2)(4\ 5) \rangle$
114. $\langle (6\ 7\ 8), (3\ 4\ 5), (4\ 5)(7\ 8), (1\ 2)(3\ 6)(4\ 7\ 5\ 8) \rangle$
115. $\langle (7\ 8), (4\ 5\ 6), (5\ 6), (1\ 2\ 3) \rangle$
116. $\langle (7\ 8), (4\ 5\ 6), (1\ 2\ 3), (1\ 4)(2\ 5)(3\ 6) \rangle$
117. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4)(6\ 7\ 8) \rangle$
118. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4) \rangle$
119. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5)(4\ 6), (4\ 5\ 6), (5\ 6) \rangle$
120. $\langle (3\ 4), (7\ 8), (5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8) \rangle$
121. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4) \rangle$

122. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (2\ 3\ 4), (3\ 4) \rangle$
123. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5) \rangle$
124. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4) \rangle$
125. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8), (1\ 2)(3\ 4) \rangle$
126. $\langle (6\ 7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4), (2\ 3\ 5\ 4) \rangle$
127. $\langle (6\ 7\ 8), (7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4) \rangle$
128. $\langle (6\ 7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4), (2\ 3\ 5\ 4)(7\ 8) \rangle$
129. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (2\ 3\ 5\ 4\ 7\ 8\ 6) \rangle$
130. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5), (3\ 4\ 5), (4\ 5) \rangle$
131. $\langle (3\ 4), (7\ 8), (5\ 6), (3\ 5\ 7)(4\ 6\ 8), (1\ 2) \rangle$
132. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (3\ 4), (1\ 2), (1\ 3)(2\ 4) \rangle$
133. $\langle (6\ 7\ 8), (7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5), (3\ 4\ 5) \rangle$
134. $\langle (7\ 8), (4\ 5\ 6), (5\ 6), (1\ 2\ 3), (2\ 3) \rangle$
135. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (4\ 5) \rangle$
136. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4) \rangle$
137. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4) \rangle$
138. $\langle (7\ 8), (4\ 5\ 6), (1\ 2\ 3), (2\ 3)(5\ 6), (1\ 4)(2\ 5\ 3\ 6) \rangle$
139. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4) \rangle$
140. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4)(6\ 7\ 8), (1\ 5)(2\ 6)(3\ 8)(4\ 7) \rangle$
141. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (7\ 8) \rangle$
142. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4)(6\ 7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
143. $\langle (6\ 7\ 8), (7\ 8), (1\ 2\ 3\ 4\ 5), (2\ 5)(3\ 4), (2\ 3\ 5\ 4) \rangle$
144. $\langle (6\ 7\ 8), (7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5), (3\ 4\ 5), (4\ 5) \rangle$
145. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4) \rangle$
146. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4) \rangle$
147. $\langle (1\ 2\ 3\ 4\ 5\ 6\ 8), (1\ 2\ 4)(3\ 6\ 5), (1\ 6)(2\ 3)(4\ 5)(7\ 8) \rangle$
148. $\langle (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (1\ 4)(2\ 3)(5\ 8)(6\ 7), (2\ 3\ 5\ 4\ 7\ 8\ 6), (3\ 5\ 7)(4\ 6\ 8) \rangle$
149. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (6\ 7\ 8) \rangle$
150. $\langle (1\ 2\ 3\ 4\ 5), (4\ 5\ 6), (5\ 6) \rangle$
151. $\langle (5\ 6), (7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4) \rangle$
152. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4) \rangle$
153. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (5\ 7\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8) \rangle$
154. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (6\ 7\ 8), (7\ 8) \rangle$
155. $\langle (3\ 4), (7\ 8), (5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8), (1\ 2) \rangle$
156. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (7\ 8), (4\ 5) \rangle$
157. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8) \rangle$
158. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4) \rangle$
159. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (6\ 7\ 8), (4\ 5) \rangle$
160. $\langle (1\ 7\ 6\ 5\ 4\ 2\ 3), (2\ 3\ 4), (6\ 7) \rangle$
161. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (6\ 7\ 8), (7\ 8), (4\ 5) \rangle$
162. $\langle (1\ 2\ 3\ 4\ 5), (4\ 5\ 6), (7\ 8), (5\ 6) \rangle$
163. $\langle (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8), (1\ 2) \rangle$

Nonstandard alphabets

1. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8) \rangle$
2. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7)(6\ 8) \rangle$
3. $\langle (3\ 4)(5\ 6)(7\ 8), (1\ 2)(5\ 7)(6\ 8) \rangle$
4. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4)(5\ 7)(6\ 8) \rangle$

5. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4)(7\ 8)\rangle$
6. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8)\rangle$
7. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
8. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7\ 6\ 8), (1\ 2)(5\ 7\ 6\ 8)\rangle$
9. $\langle (7\ 8), (3\ 4)(5\ 6), (1\ 2)(5\ 6)\rangle$
10. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
11. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (3\ 4)(7\ 8)\rangle$
12. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4)(5\ 7)(6\ 8)\rangle$
13. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4)(7\ 8)\rangle$
14. $\langle (7\ 8), (5\ 6), (3\ 4)(5\ 7)(6\ 8)\rangle$
15. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4)(7\ 8)\rangle$
16. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(5\ 7\ 6\ 8)\rangle$
17. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (3\ 4)(7\ 8)\rangle$
18. $\langle (3\ 4)(5\ 6)(7\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(6\ 7)\rangle$
19. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
20. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (3\ 5\ 7)(4\ 6\ 8)\rangle$
21. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5)(7\ 8)\rangle$
22. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (2\ 3)(7\ 8)\rangle$
23. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(5\ 7\ 6\ 8)\rangle$
24. $\langle (7\ 8), (3\ 4)(5\ 6), (1\ 2)(3\ 5)(4\ 6)\rangle$
25. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7\ 6\ 8)\rangle$
26. $\langle (3\ 4)(5\ 6)(7\ 8), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8)\rangle$
27. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
28. $\langle (3\ 4\ 5)(6\ 7\ 8), (4\ 5)(7\ 8), (1\ 2)(3\ 6)(4\ 7)(5\ 8)\rangle$
29. $\langle (3\ 4)(5\ 6)(7\ 8), (3\ 5\ 7)(4\ 6\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
30. $\langle (3\ 5\ 4)(6\ 8\ 7), (3\ 6)(4\ 8)(5\ 7), (1\ 2)(4\ 5)(7\ 8)\rangle$
31. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(7\ 8)\rangle$
32. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8)\rangle$
33. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7\ 6\ 8), (1\ 2)(5\ 7\ 6\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
34. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(5\ 7)(6\ 8)\rangle$
35. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
36. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(7\ 8)\rangle$
37. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7\ 6\ 8)\rangle$
38. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(7\ 8)\rangle$
39. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8)\rangle$
40. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(7\ 8)\rangle$
41. $\langle (7\ 8), (5\ 6), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
42. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
43. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2)(5\ 7)(6\ 8)\rangle$
44. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(7\ 8)\rangle$
45. $\langle (5\ 6)(7\ 8), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(5\ 7\ 6\ 8)\rangle$
46. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
47. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7\ 6\ 8)\rangle$
48. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5)(4\ 6), (1\ 2)(5\ 6)\rangle$
49. $\langle (5\ 6)(7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5\ 3\ 7)(2\ 6\ 4\ 8)\rangle$
50. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (3\ 4)(7\ 8)\rangle$
51. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5\ 4\ 6), (1\ 2)(5\ 6)\rangle$
52. $\langle (3\ 4)(5\ 6)(7\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(6\ 7), (1\ 5\ 2\ 8)(3\ 6\ 4\ 7)\rangle$
53. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8)\rangle$

54. $\langle (3\ 4)(5\ 6)(7\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(6\ 7), (1\ 5)(2\ 8)(3\ 6)(4\ 7)\rangle$
55. $\langle (7\ 8), (5\ 6), (2\ 3\ 4), (3\ 4)(5\ 7)(6\ 8)\rangle$
56. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (2\ 3)(7\ 8), (2\ 4)(3\ 5)(7\ 8)\rangle$
57. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (2\ 3)(7\ 8), (2\ 4)(3\ 5)\rangle$
58. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8), (3\ 4)(7\ 8)\rangle$
59. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4)(7\ 8)\rangle$
60. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
61. $\langle (7\ 8), (3\ 4)(5\ 6), (1\ 2)(5\ 6), (1\ 3\ 5)(2\ 4\ 6)\rangle$
62. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (1\ 2)(5\ 7\ 6\ 8)\rangle$
63. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (1\ 2)(7\ 8)\rangle$
64. $\langle (7\ 8), (1\ 2)(3\ 4)(5\ 6), (1\ 3\ 5)(2\ 4\ 6), (3\ 5)(4\ 6)\rangle$
65. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
66. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)\rangle$
67. $\langle (5\ 6)(7\ 8), (5\ 7\ 6\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(7\ 8)\rangle$
68. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
69. $\langle (7\ 8), (5\ 6), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)\rangle$
70. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7\ 4\ 8)\rangle$
71. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5\ 3\ 7\ 2\ 6\ 4\ 8)\rangle$
72. $\langle (7\ 8), (5\ 6), (3\ 4)(5\ 7)(6\ 8), (1\ 2)(5\ 7)(6\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8)\rangle$
73. $\langle (6\ 7\ 8), (3\ 4\ 5), (4\ 5)(7\ 8), (3\ 6)(4\ 7)(5\ 8)\rangle$
74. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5\ 3\ 7)(2\ 6\ 4\ 8)\rangle$
75. $\langle (6\ 7\ 8), (3\ 4\ 5), (4\ 5)(7\ 8), (1\ 2)(3\ 6)(4\ 7)(5\ 8)\rangle$
76. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(7\ 8)\rangle$
77. $\langle (3\ 4\ 5), (6\ 7\ 8), (3\ 6)(4\ 7)(5\ 8), (1\ 2)(4\ 5)(7\ 8)\rangle$
78. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (1\ 2\ 3), (2\ 3)(7\ 8)\rangle$
79. $\langle (5\ 6)(7\ 8), (1\ 2)(5\ 8\ 6\ 7), (3\ 4)(5\ 8\ 6\ 7), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
80. $\langle (5\ 6)(7\ 8), (1\ 2)(5\ 8)(6\ 7), (3\ 4)(5\ 8)(6\ 7), (1\ 3)(2\ 4)(7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
81. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4)(7\ 8)\rangle$
82. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (1\ 2)(3\ 4), (1\ 3\ 2\ 4)(7\ 8)\rangle$
83. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8)\rangle$
84. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7\ 6\ 8), (1\ 2)(7\ 8)\rangle$
85. $\langle (1\ 2\ 3\ 4\ 6), (1\ 4)(5\ 6)\rangle$
86. $\langle (3\ 4)(7\ 8), (3\ 4)(5\ 6), (3\ 5\ 7)(4\ 6\ 8), (5\ 7)(6\ 8), (1\ 2)(7\ 8)\rangle$
87. $\langle (3\ 4), (7\ 8), (5\ 6), (3\ 5\ 7)(4\ 6\ 8), (1\ 2)(5\ 7)(6\ 8)\rangle$
88. $\langle (7\ 8), (3\ 4)(5\ 6), (3\ 5)(4\ 6), (4\ 5\ 6), (1\ 2)(5\ 6)\rangle$
89. $\langle (7\ 8), (1\ 2)(5\ 6), (1\ 2)(3\ 4), (1\ 3\ 5)(2\ 4\ 6), (3\ 5\ 4\ 6)\rangle$
90. $\langle (7\ 8), (1\ 2)(5\ 6), (1\ 2)(3\ 4), (1\ 3\ 5)(2\ 4\ 6), (3\ 5)(4\ 6)\rangle$
91. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 7\ 2\ 8)(3\ 5\ 4\ 6), (1\ 3\ 2\ 4)(5\ 7\ 6\ 8), (3\ 5\ 7)(4\ 6\ 8), (3\ 4)(5\ 8)(6\ 7)\rangle$
92. $\langle (1\ 2)(3\ 4)(5\ 6)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (2\ 3\ 4)(6\ 7\ 8), (3\ 4)(7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
93. $\langle (6\ 7\ 8), (7\ 8), (3\ 4\ 5), (4\ 5), (3\ 6)(4\ 7)(5\ 8)\rangle$
94. $\langle (6\ 7\ 8), (2\ 3)(4\ 5), (2\ 4)(3\ 5), (3\ 4\ 5), (4\ 5)(7\ 8)\rangle$
95. $\langle (6\ 7\ 8), (7\ 8), (3\ 4\ 5), (4\ 5), (1\ 2)(3\ 6)(4\ 7)(5\ 8)\rangle$
96. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
97. $\langle (5\ 6)(7\ 8), (5\ 8\ 6\ 7), (1\ 2)(5\ 6), (3\ 4)(5\ 6), (1\ 3)(2\ 4)(5\ 6), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
98. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (1\ 2\ 3), (2\ 3)(7\ 8), (1\ 6)(2\ 7\ 3\ 8)\rangle$
99. $\langle (6\ 7\ 8), (4\ 5)(7\ 8), (1\ 2\ 3), (2\ 3)(7\ 8), (1\ 6)(2\ 7)(3\ 8)\rangle$
100. $\langle (7\ 8), (4\ 5\ 6), (1\ 2\ 3), (2\ 3)(5\ 6), (1\ 4)(2\ 5)(3\ 6)\rangle$
101. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5\ 3\ 7)(2\ 6\ 4\ 8)\rangle$
102. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(5\ 6), (3\ 4)(5\ 6), (1\ 3)(2\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$

103. $\langle (5\ 6)(7\ 8), (5\ 8\ 6\ 7), (1\ 2)(5\ 6), (3\ 4)(5\ 6), (1\ 3)(2\ 4)(5\ 6), (1\ 5)(2\ 6)(3\ 7\ 4\ 8)\rangle$
104. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(5\ 6), (3\ 4)(5\ 6), (1\ 3)(2\ 4), (1\ 5)(2\ 6)(3\ 7\ 4\ 8)\rangle$
105. $\langle (7\ 8), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4)(5\ 7)(6\ 8)\rangle$
106. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)\rangle$
107. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (6\ 7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(7\ 8)\rangle$
108. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4)(6\ 7\ 8), (3\ 4)(7\ 8)\rangle$
109. $\langle (1\ 2\ 3\ 4\ 6), (1\ 4)(5\ 6), (3\ 4\ 6\ 5)\rangle$
110. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (4\ 5)(7\ 8)\rangle$
111. $\langle (1\ 2\ 3\ 4\ 6), (1\ 4)(5\ 6), (7\ 8)\rangle$
112. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (3\ 5\ 7)(4\ 6\ 8)\rangle$
113. $\langle (1\ 2\ 3\ 4\ 5\ 6\ 7), (1\ 2)(3\ 6)\rangle$
114. $\langle (1\ 2\ 3\ 4\ 6), (1\ 4)(5\ 6), (3\ 4\ 6\ 5)(7\ 8)\rangle$
115. $\langle (7\ 8), (4\ 5\ 6), (5\ 6), (1\ 2\ 3), (2\ 3), (1\ 4)(2\ 5)(3\ 6)\rangle$
116. $\langle (1\ 2\ 3\ 4\ 6), (1\ 4)(5\ 6), (7\ 8), (3\ 4\ 6\ 5)\rangle$
117. $\langle (7\ 8), (5\ 6), (5\ 7)(6\ 8), (1\ 2), (3\ 4), (1\ 3)(2\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
118. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 4)(2\ 3)(5\ 8)(6\ 7), (1\ 6)(2\ 5)(3\ 8)(4\ 7), (3\ 5\ 8)(4\ 6\ 7), (5\ 7)(6\ 8)\rangle$
119. $\langle (1\ 2\ 3\ 4\ 5), (4\ 5\ 6)\rangle$
120. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (5\ 7\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4)(7\ 8)\rangle$
121. $\langle (5\ 6)(7\ 8), (3\ 4)(7\ 8), (1\ 2)(7\ 8), (1\ 4)(2\ 3)(5\ 8)(6\ 7), (1\ 6)(2\ 5)(3\ 8)(4\ 7), (3\ 5\ 8)(4\ 6\ 7), (5\ 7\ 6\ 8)\rangle$
122. $\langle (7\ 8), (5\ 6), (3\ 4), (1\ 2), (1\ 3)(2\ 4)(5\ 7)(6\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (3\ 5\ 7)(4\ 6\ 8)\rangle$
123. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4)(6\ 7\ 8), (3\ 4)(7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
124. $\langle (1\ 2\ 3\ 4\ 5), (3\ 4\ 5), (6\ 7\ 8), (4\ 5)(7\ 8)\rangle$
125. $\langle (1\ 7\ 6\ 3\ 2\ 5\ 4), (1\ 7\ 3)(2\ 6\ 5), (1\ 5)(2\ 3)(4\ 8)(6\ 7), (3\ 4)(5\ 7)(6\ 8)\rangle$
126. $\langle (1\ 2\ 3\ 4\ 5), (4\ 5\ 6), (7\ 8)\rangle$
127. $\langle (1\ 2\ 3\ 4\ 5), (4\ 5\ 6), (5\ 6)(7\ 8)\rangle$
128. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (5\ 7\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4)(7\ 8), (1\ 5)(2\ 6)(3\ 7\ 4\ 8)\rangle$
129. $\langle (1\ 2\ 3\ 4\ 5\ 6\ 7), (5\ 6\ 7)\rangle$
130. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (5\ 7\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4)(7\ 8), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
131. $\langle (1\ 8)(2\ 3)(4\ 5)(6\ 7), (1\ 3)(2\ 8)(4\ 6)(5\ 7), (1\ 5)(2\ 6)(3\ 7)(4\ 8), (1\ 2\ 6\ 3\ 4\ 5\ 7), (1\ 2\ 3)(4\ 6\ 5), (1\ 2)(5\ 6)\rangle$
132. $\langle (5\ 6)(7\ 8), (5\ 7)(6\ 8), (5\ 7\ 6), (5\ 6), (1\ 2)(3\ 4), (1\ 3)(2\ 4), (2\ 3\ 4), (3\ 4), (1\ 5)(2\ 6)(3\ 7)(4\ 8)\rangle$
133. $\langle (1\ 2\ 3\ 4\ 5\ 6\ 7), (6\ 7\ 8)\rangle$

Bibliography

[BCH+13] C. Bessiere, C. Carbonnel, E. Hebrard, G. Katsirelos, and T. Walsh. Detecting and exploiting subproblem tractability. In *Proc. IJCAI'13*, pages 468–474, 2013.

[BD08] C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artificial Intelligence*, 172(1):29–41, 2008.

[BE04] R. Barták and R. Erben. A new algorithm for singleton arc consistency. In *Proc. FLAIRS'04*, pages 257–262, 2004.

[BKL14] M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014.

[BKLT13] M. Bojańczyk, B. Klin, S. Lasota, and S. Toruńczyk. Turing machines with atoms. In *Proc. LICS'13*, pages 183–192, 2013.

[BO14] C. Bessenrodt and K. Ono. Maximal multiplicative properties of partitions. *ArXiv e-prints*, March 2014, arXiv:1403.3352v2.

[BRYZ05] C. Bessiere, J. C. Régin, R. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[CDG13] H. Chen, V. Dalmau, and B. Grußien. Arc consistency and friends. *J. Logic Computation*, 23(1):87–108, 2013.

[FV99] 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. Computing*, 28(1):57–104, 1999.

[GP02] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[KLOT14] B. Klin, S. Lasota, J. Ochremiak, and S. Toruńczyk. Turing machines with atoms, constraint satisfaction problems, and descriptive complexity. In *Proc. CSL-LICS'14*, pages 58:1–58:10, 2014.

[LC05] C. Lecoutre and S. Cardon. A greedy approach to establish singleton arc consistency. In *Proc. IJCAI'05*, pages 199–204, 2005.

[Pit13] A.M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.