Computing finite models using free Boolean generators

Žarko Mijajlović and Aleksandar Pejović

Abstract. A parallel method for computing Boolean expressions based on the properties of finite free Boolean algebras is presented. We also show how various finite combinatorial objects can be codded in the formalism of Boolean algebras and counted by this procedure. Particularly, using a translation of first order predicate formulas to propositional formulas, we give a method for constructing and counting finite models of the first order theories. An implementation of the method that can be run on multi-core CPUs as well as on highly parallel GPUs is outlined.

1. Introduction

Even ordinary personal computers are capable for specific massive parallel computations. Examples of this kind are logical operations which can be computed bitwise, i.e., by use of all register bits in one processor cycle. Based on this idea, we propose a method for computing Boolean expressions using the parallel structure of standard computer processors. The mathematical background of our approach is based on the properties of finite free Boolean algebras. The idea of parallelization of computing logical operations in this way is indicated in [**1**]. The basic idea is as follows.

Let $f(x_1, x_2, \ldots, x_n)$ be a Boolean expression in n variables $x_1, x_2, \ldots, x_n$. We give a construction of n Boolean vectors $b_1, b_2, \ldots, b_n$ of size 2^n with the following property:

($\mathscr{P}$) $f(b_1, b_2, \ldots, b_n)$ is a Boolean vector that codes the full DNF of f.

It appears that vectors $b_1, b_2, \ldots, b_n$ are exactly free generators of a free Boolean algebra having n free generators.

Using a translation procedure from the first order predicate formulas to propositional formulas, we give a method for constructing and counting various combinatorial objects. This idea is formally developed in [**2**], but it was used there in the study of problems in the infinitary combinatorics, particularly in finding their

2010 *Mathematics Subject Classification.* 03B35; 03C13; 03G27.

Key words and phrases. free boolean algebras, finite model theory, computing.

Partially supported by Serbian Ministry of Science, Grant III 44006.

complexity in the Borel hierarchy. Related combinatorial problems are considered, for example the number of automorphisms of finite structures and various partition problems over finite sets. We also give an implementation of the method that can be run on multi-core CPUs as well as on highly parallel GPUs (Graphics processing units).

Standard notation and terminology from model theory is assumed as in [4] and [9]. Also, for notions from universal algebras we shall refer to [1]. Models of a first order language L are denoted by bold capital letters $\mathbf{A}$, $\mathbf{B}$, etc, while their domains respectively by A, B and so on. By a domain we mean any nonempty set. The letter L will be used to denote a first-order language. The first order logic is denoted by $L_{\omega\omega}$ and the propositional calculus with a set $\mathcal{P}$ of propositional variables by $L_\omega^{\mathcal{P}}$, or simply L_ω. The set of natural numbers $\{0, 1, 2, \ldots\}$ is denoted by N. We also take $2 = \{0, 1\}$. By $\mathbf{2}$ we denote the two-element Boolean algebra and then $\mathbf{2}^I$ is the power of $\mathbf{2}$, while $\mathbf{0}$ and $\mathbf{1}$ are respectively the smallest and the greatest element of $\mathbf{2}^I$. Occasionally elements of $\mathbf{2}^I$ are called Boolean vectors. Whenever is needed to distinguish the formal equality sign from identity, for the first one we shall keep $=$, while $\equiv$ denotes identity.

2. Variables

In this section we develop and explain the logical and algebraic background for our computing method. The power of a model $\mathbf{A}$, the product $\prod_{i\in I}\mathbf{A}$, is denoted by $\mathbf{A}^I$.

2.1. Interpretation of variables. By a set of variables we mean any nonempty set V so that no $v \in V$ is a finite sequence of other elements from V. This assumption secures the unique readability of terms and formulas. Particularly we shall consider finite and countable sets of variables V, e.g. $V = \{v_0, v_1, \ldots\}$. A valuation of a domain A is any map from V to A. Let I denote the set of all valuations from domain A, i.e., $I = A^V$. In this section, the letter I will be reserved for the set of valuation of a domain A. Sometimes we shall assume that elements from I will have finite supports.

Definition 2.1. (*Interpretation of variables*). Let v be a variable from V. The interpretation of variable v in domain A is the map $\hat{v}\colon I \to A$ defined by $\hat{v}(\mu) = \mu(v)$, $\mu \in I$.

The set of interpretations of variables from V into domain A is denoted by $\hat{V}_A$. Therefore, $\hat{V}_A = \{\hat{v}\colon v \in V\}$.

Let $\varphi(v_1, \ldots, v_n)$ be a formula of a language L having free variables $v_1, \ldots, v_n$ and $\mathbf{A}$ a model of L. The map $\hat{\varphi}^{\mathbf{A}}(\hat{v}_1, \ldots, \hat{v}_n)$, abbreviated by $\hat{\varphi}^{\mathbf{A}}$, is $\hat{\varphi}^{\mathbf{A}}\colon I \to 2$ defined by $\hat{\varphi}^{\mathbf{A}}(\mu) = 1$ if $\mathbf{A} \models \varphi[\mu]$, otherwise $\hat{\varphi}^{\mathbf{A}}(\mu) = 0$, $\mu \in I$. Hence $\hat{\varphi}^{\mathbf{A}} \in 2^I$.

Proposition 2.1. *Let φ be an identity $s = t$, where s and t are terms of L. Then the following are equivalent:*

$$1^\circ\ \mathbf{A}^I \models \varphi[\hat{v}_1, \ldots, \hat{v}_n], \quad 2^\circ\ \hat{\varphi}^{\mathbf{A}}(\hat{v}_1, \ldots, \hat{v}_n) = \mathbf{1}, \quad 3^\circ\ \mathbf{A} \models \varphi[\mu],\ \mu \in I. \tag{2.1}$$

Proof. The equivalence of 2° and 3° follows immediately by definition 2.1. From 3° follows 1° since identities are preserved under products of models. Finally, assume 1°. Then

$$s^{\mathbf{A}^I}(\hat{v}_1,\dots,\hat{v}_n)=t^{\mathbf{A}^I}(\hat{v}_1,\dots,\hat{v}_n). \tag{2.2}$$

Let $\pi_\mu\colon \mathbf{A}^I\to\mathbf{A}$ be a projection, $\mu\in I$. Since π_μ is a homomorphism we have

$$\begin{aligned}\pi_\mu(s^{\mathbf{A}^I}(\hat{v}_1,\dots,\hat{v}_n)) &= s^{\mathbf{A}}(\pi_\mu\hat{v}_1,\dots,\pi_\mu\hat{v}_n)\\ &= s^{\mathbf{A}}(\hat{v}_1(\mu),\dots,\hat{v}_n(\mu))\\ &= s^{\mathbf{A}}(\mu(v_1),\dots,\mu(v_n))=s^{\mathbf{A}}[\mu].\end{aligned} \tag{2.3}$$

Hence, 3° follows by 2.2. □

For an algebra $\mathbf{A}$ of L let $\mathcal{J}(\mathbf{A})$ be the set of all identities that are true in $\mathbf{A}$. Similarly, $\mathcal{J}(\mathcal{K})$ denotes the set of all identities that are true in all algebras of a class $\mathcal{K}$ of algebras of L. If $\mathcal{J}(\mathbf{A})=\mathcal{J}(\mathbf{B})$, $\mathbf{A}$ and $\mathbf{B}$ are algebras of L, we hall also write $\mathbf{A}\equiv_{\mathcal{J}}\mathbf{B}$.

The notion of interpretation of variables will play the fundamental role in our analysis and program implementation. But they can be useful in other cases, too. For example, for so introduced notions it is easy to prove the Birkhoff HSP theorem and other related theorems. Here we prove a theorem on the existence of free algebras. The novelty of these proof is that it does not use the notion of a term algebra (absolutely free algebra). For the simplicity of exposition, we shall assume that L is countable.

Theorem 2.1. *(G. Birkhoff) Let $\mathcal{K}$ be a nontrivial abstract*[1] *class of algebras of L, closed under subalgebras and products. Then $\mathcal{K}$ has a free algebra over every nonempty set.*

Proof. It is easy to see, for example by use of the downward Skolem-Löwenheim theorem, that for each algebra $\mathbf{A}\in\mathcal{K}$ there is at most countable subalgebra $\mathbf{A}'$ of $\mathbf{A}$ so that $\mathbf{A}'\equiv_{\mathcal{J}}\mathbf{A}$. The algebra $\mathbf{A}'$ is obviously isomorphic to an algebra of which the domain is a subset of N. Hence, there is a set $\mathcal{K}'=\{\mathbf{A}_s\colon s\in S\}$ of at most countable algebras such that $\mathcal{K}'\subseteq\mathcal{K}$ and $\mathcal{J}(\mathcal{K})=\mathcal{J}(\mathcal{K}')$.

Let $\mathbf{A}=\prod_s\mathbf{A}_s$ be the product of all algebras from $\mathcal{K}'$. Since $\mathcal{K}$ is closed under products, it follows $\mathbf{A}\in\mathcal{K}$, hence $\mathcal{J}(\mathcal{K})\subseteq\mathcal{K}(\mathbf{A})$. On the other hand, for each $s\in S$, $\mathbf{A}_s$ is a homomorphic image of $\mathbf{A}$, as $\mathbf{A}_s=\pi_s\mathbf{A}$. Hence each identity φ of L which holds on $\mathbf{A}$ is also true in all algebras from $\mathcal{K}'$ and therefore in all algebras from $\mathcal{K}$. So we proved

$$\mathcal{J}(\mathcal{K})=\mathcal{J}(\mathbf{A}). \tag{2.4}$$

Since $\mathcal{K}$ is nontrivial, it must be $|A|\geqslant 2$. Let X be any non empty set. For our purpose we may identify X with $\hat{V}_A$ for some set of variables V. Let $\mathbf{\Omega}$ be subalgebra of $\mathbf{A}^I$ generated by $\hat{V}_A$. Since $\mathcal{K}$ is closed under subalgebras, it follows $\mathbf{\Omega}\in\mathcal{K}$. Now we prove that $\mathbf{\Omega}$ is a free algebra over $\hat{V}_A$ for class $\mathcal{K}$. Let $\mathbf{B}\in\mathcal{K}$ be an arbitrary algebra and $g\colon\hat{V}_A\to B$. Each element $a\in\Omega$ is of the form

[1] closed for isomorphic images

$a = s^{\mathbf{\Omega}}(\hat{v}_1, \dots, \hat{v}_n)$ for some L-term s and some (different) variables $v_1, \dots, v_n \in V$. We extend g to $f \colon \mathbf{\Omega} \to \mathbf{B}$ taking

$$f(a) = s^{\mathbf{B}}(g\hat{v}_1, \dots, g\hat{v}_n). \tag{2.5}$$

The map f is well defined. Indeed, suppose that for some other term t of L, $a = t^{\mathbf{\Omega}}(\hat{v}_1, \dots, \hat{v}_n)$. Let φ denote the identity $s(v_1, \dots, v_n) = t(v_1, \dots, v_n)$. Then $s^{\mathbf{\Omega}}(\hat{v}_1, \dots, \hat{v}_n) = t^{\mathbf{\Omega}}(\hat{v}_1, \dots, \hat{v}_n)$ and as $\mathbf{\Omega} \subseteq \mathbf{A}^I$ it follows $\mathbf{A}^I \models \varphi[\hat{v}_1, \dots, \hat{v}_n]$. By Proposition 2.1 it follows that the identity φ holds on $\mathbf{A}$. By 2.4 then φ is true in all algebras from $\mathcal{K}$. Hence

$$s^{\mathbf{B}}(g\hat{v}_1, \dots, g\hat{v}_n) = t^{\mathbf{B}}(g\hat{v}_1, \dots, g\hat{v}_n), \tag{2.6}$$

and thus we proved that the f is well-defined.

In a similar manner we prove that f is a homomorphism. For simplicity, suppose $*$ is a binary operation of L. We denote the interpretations of $*$ in $\mathbf{\Omega}$ and $\mathbf{B}$ by $\cdot$. Take $a, b \in \Omega$ and let s and t be terms of L so that

$$a = s^{\mathbf{\Omega}}(g\hat{v}_1, \dots, g\hat{v}_n), \quad b = t^{\mathbf{\Omega}}(g\hat{v}_1, \dots, g\hat{v}_n) \tag{2.7}$$

and let w be the combined term $w = s * t$. Then

$$f(a \cdot b) = f(w^{\mathbf{\Omega}}(\hat{v}_1, \dots, \hat{v}_n)) = w^{\mathbf{B}}(g\hat{v}_1, \dots, g\hat{v}_n) = g(a) \cdot g(b). \tag{2.8}$$

Thus, f is a homomorphism from $\mathbf{\Omega}$ to $\mathbf{B}$ which extends g. □

Suppose $\mathcal{K}$ is the class of algebras to which refer the previous theorem. We note the following.

Note 2.1.1 It is easy now to prove the Birkhoff HSP theorem. Assume $\mathcal{K}$ is also closed under homomorphic images and let $T = \mathcal{J}(\mathcal{K})$. Let $\mathbf{\Omega}$ be a free algebra of $\mathcal{K}$ with infinitely many free generators. Then $\mathcal{J}(\mathbf{\Omega}) = \mathcal{J}(\mathcal{K}) = T$. Suppose $\mathbf{B}$ is a model of T and let Ω be a free algebra for class $\mathcal{K}$ and X is a set of free generators of $\mathbf{\Omega}$ such that $|X| \geqslant |B|$. Let $g \colon X \to B$ so that $g(X) = B$. Then by the same construction as in the previous proof g extends to some homomorphism $f \colon \mathbf{\Omega} \to \mathbf{B}$, thus $\mathbf{B}$ is a homomorphic image of $\mathbf{\Omega}$. Hence $\mathbf{B}$ belongs to $\mathcal{K}$.

Note 2.1.2 Assume $\mathbf{A} \in \mathcal{K}$ is an arbitrary algebra which satisfies condition 2.4. Such an algebra $\mathbf{A}$ will be called the characteristic algebra for the class $\mathcal{K}$. By close inspection of the proof of Theorem 2.1, we see that this condition suffices to construct a free algebra for $\mathcal{K}$ from $\mathbf{A}$ as we did in the proof of 2.1. This idea is indicated to some extent in [1], (Part II, chapter 11, particularly see problem 11.5, p. 77) but under stronger and amended assumptions and without referring to variable interpretations.

2.2. Free Boolean vectors. It is well known that finite free Boolean algebras with n free generators are the algebras $\mathbf{2}^{2^n}$. We remark that this immediately follows by note 2.1.2, since $\mathbf{2}$ is the characteristic algebra for the class of all Boolean algebras. The structure and properties of free Boolean vectors of $\mathbf{\Omega}_n = \mathbf{2}^{2^n}$ are discussed in [7] in details.

We remind that a collection $\{b_1, \dots, b_n\}$ of elements of a Boolean algebra $\mathbf{B}$ is independent if $b_1^{\alpha_1} \wedge \dots \wedge b_n^{\alpha_n} \neq 0$, where $b^1 = b$ and $b^0 = b'$. A similar definition of independence is for families of subsets of a given set. A collection $\{b_1, \dots, b_n\}$

generates the free subalgebra of $\mathbf{B}$ if and only if it is independent, cf. [12]. The number of free generating sets of $\mathbf{\Omega}_n$ is found in [7]. In fact, the following holds.

Theorem 2.2. *Let $S = \{1, 2, \ldots, 2^n\}$ and a_n, b_n, c_n be the sequences defined as follows.*

(1) *a_n = number of labeled Boolean algebras with domain S (number of different Boolean algebras with domain S).*

(2) *b_n = number of independent collections $\{P_1, \ldots, P_n\}$ of subsets of S.*

(3) *c_n = number of free generating sets $\{b_1, \ldots, b_n\}$ of $\mathbf{\Omega}_n$.*

Then $a_n = b_n = c_n = (2^n)!/n!$.

Proof. The number of labelings of a finite model $\mathbf{A}$ of size m is equal to $m!/|\mathrm{Aut}(\mathbf{A})|$. As $\mathrm{Aut}(\mathbf{2}^n)$ is isomorphic to the permutation group S_n, it follows $a_n = (2^n)!/n!$.

Let $\mathbf{B} = \mathbf{2}^n$ and $\mathbf{B}^l$ a labeled algebra obtained from $\mathbf{B}$. Algebra $\mathbf{B}$ has exactly n ultrafilters and so has $\mathbf{B}^l$. Let $U(\mathbf{B})$ be the set of all ultrafilters of $\mathbf{B}$. By Theorem 2.2.7 in [7], $U(\mathbf{B})$ is an independent collection of subsets of S. The map U which assigns $U(\mathbf{B}^l)$ to $\mathbf{B}^l$ is $1-1$. Indeed, let us for $S_1, \ldots, S_n \subseteq S$ and $\alpha \in 2^n$ define

$$S^\alpha = S^{\alpha_1} \cap \ldots \cap S^{\alpha_n}. \tag{2.9}$$

For $a \in S$ let $P_1, \ldots, P_k, P_{k+1}, \ldots, P_n \in U(\mathbf{B}^l)$ be such that $a \in P_1, \ldots, P_k$ and $a \notin P_{k+1}, \ldots, P_n$. Then $P_1 \cap \ldots, \cap P_k \cap P_{k+1}^c \ldots \cap P_n^c = \{a\}$.

Therefore, we proved that for each $a \in B^l$ there is a unique $\alpha \in 2^n$ such that $P^\alpha = \{a\}$, $P_1, \ldots, P_n \in U(\mathbf{B}^l)$. Let $\wedge^l$ and $'^l$ be Boolean operations of $\mathbf{B}^l$. Then for $a, b \in B^l$ and corresponding $\alpha, \beta \in 2^n$ we have

$$P^{\alpha'} = \{a^{'l}\}, \quad P^{\alpha\wedge\beta} = \{a \wedge^l b\} \tag{2.10}$$

where α', $\alpha \wedge \beta$ are computed in $\mathbf{2}^n$. Thus, we proved that $U(\mathbf{B}^l)$ uniquely determines $\mathbf{B}^l$, hence $a_n \leqslant b_n$.

Suppose $P = \{P_1, \ldots, P_n\}$ is an independent collection of subsets of S. Then P can serve as $U(\mathbf{B}^l)$ for certain labeled Boolean algebra $\mathbf{B}^l$. To prove it, note that each P^α has at least one element and that $\bigcup_{\alpha\in 2^n} P^\alpha$ has at most 2^n elements. This shows that P^α is one-element set. Therefore, a Boolean algebra $\mathbf{B}^l$ with domain S is defined by 2.10 and it is easy to see that $P = U(\mathbf{B}^l)$. Hence $a_n = b_n$.

Finally, as noted, a collection $X = \{X_1, \ldots, X_n\}$ of subsets of S freely generates the power set algebra $P(S)$ if and only if X is independent. Hence, $c_n = b_n$. □

We will be dealing particularly with free generators of $\mathbf{\Omega}_n$ of the following form. Let a_i, $i = 0, 1, \ldots, 2^n - 1$, be binary expansions of integers i with zeros padded to the left up to the length n. Let M be the matrix whose columns are vectors a_i. As noted in [7], binary vectors b_i, $i = 1, 2 \ldots n$, formed by rows of M are free vectors of $\mathbf{\Omega}_n$. In the case n = 3, the matrix M and vectors b_i are

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}, \tag{2.11}$$

$b_1 = 00001111$, $b_2 = 00110011$, $b_3 = 01010101$.

2.3. Computing Boolean expressions. Let $t = t(v_1, \dots, v_n)$ be a Boolean expression in variables $v_1, \dots, v_n$ and $b_1, \dots, b_n$ free generators of $\mathbf{\Omega}_n$.

Proposition 2.2. *$t^{\mathbf{\Omega}_n}(b_1, \dots, b_n)$ codes the the full DNF of t.*

Proof. By our previous discussion, we may take $b_i = \hat{v}_i$ and $I = \{\hat{v}_1, \dots, \hat{v}_n\}$. Let π_μ be a projection from $\mathbf{\Omega}_n$ to $\mathbf{2}$, $\mu \in I$, and $d = t^{\mathbf{\Omega}_n}(b_1, \dots, b_n)$. Then

$$\pi_\mu d = \pi_\mu t^{\mathbf{\Omega}_n}(\hat{v}_1, \dots, \hat{v}_n) = t^{\mathbf{2}}(\mu(v_1), \dots, \mu(v_n)),$$

hence $t = \sum_{\pi_\mu d = 1} v_1^{\mu_1} \cdots v_n^{\mu_n}$, so d codes the full DNF of t. □

The parallel algorithm for computing $d = t^{\mathbf{\Omega}_n}(b_1, \dots, b_n)$ is described in details in [7], Section 2. We repeat in short this procedure. Suppose we have a 2^k-bit processor at our disposal, $k < n$. Each vector b_i is divided into 2^{n-k} consecutive sequences of equal size. Hence, b_i consists of 2^{n-k} blocks b_{ij}, each of size 2^k. To find d, blocks $d_j = t(b_{1j}, \dots, b_{nj})$ of size 2^k are computed bitwise for $j = 1, 2, \dots, 2^{n-k}$. Then the combined vector $d_1 d_2 \dots d_{2^{n-k}}$ is the output vector d. The total time for computing d approximately is $T = 2^{l+n-k}\delta$, where 2^l is the total number of nodes in the binary expression tree of the term t and δ is the time interval for computing bitwise one logical operation[2].

Suppose now that we have 2^r 2^k-bit processors. Computations of d_j is distributed among all processors and they compute $t(b_{1j}, \dots, b_{nj})$ in parallel. Actually, they are acting as a single 2^{k+r}-bit processor. Hence, the total time for computing d in this case is $T = 2^{l+n-k-r}\delta$.

We implemented this algorithm on a PC with two GPU's, each having 2^{11} 32-bit processors. Therefore, this installation is equivalent to a machine with one 2^{17} - bit processor, as $k = 5$ and $r = 12$. Our implementation at this moment is based on 30 free Boolean vectors, each with 2^{30} bits. This implementation theoretically computes a Boolean term t with 30 variables and 2^{17} nodes in it's Boolean expression in time $2^{30}\delta$ i.e., in about one second. Our experimental results are very close to this time.

We note that the number of free variables is limited by the size of internal memory and the size of the output vector d. The installation that we are using could admit the described computation with 35 free Boolean vectors. With further partition of the particular problems the computation can be done in real time with up to 50 Boolean variables. For the most powerful modern supercomputers, these numbers respectively are 50 and 70. It is interesting that these numbers were anticipated in [7], 15 years ago.

3. Computing finite models

Using a translation from $L_{\omega\omega}$ to L_ω, we are able to state and computationally solve various problems on finite structures. There are attempts of this kind. For example H. Zhang developed the system SATO for computing specific quasigroups,

[2] For modern computers, $\delta \approx 10^{-9}$ seconds

see [11]. There are many articles with the similar approach on games, puzzles and design of particular patterns. An example of this kind is Lewis article [5] on Sudoku.

3.1. Translation from $L_{\omega\omega}$ to L_ω. A method for coding some notions, mostly of the combinatorial nature and related to countable first-order structures, by theories of propositional calculus L_{ω_1} is presented in [8]. The primary goal there was to study the complexity of these notions in Borel hierarchy. The coding is given there by a map $*$. We reproduce this map adapted for our needs.

Let L be a finite first-order language and $L_A = L \cup \{\underline{a} | a \in A\}$, where A is a finite non-empty set. Here $\underline{a}$ is a new constant symbol, the name of the element a. We define the set $\mathcal{P}$ of propositional letters as follows

$$\begin{array}{rl} \mathcal{P} = & \{p_{Fa_1\ldots a_k b} |\, a_1, \ldots, a_k, b \in A, F \text{ is a } k\text{-ary function symbol of } L\} \cup \\ & \{q_{Ra_1\ldots a_k b} |\, a_1, \ldots, a_k, b \in A, R \text{ is a } k\text{-ary relation symbol of } L\} \end{array} \tag{3.1}$$

The map $*$ from the set Sent_{L_A} of all $L_{\omega\omega}$-sentences of L_A into the set of propositional formulas of $L_\omega^{\mathcal{P}}$ is defined recursively as follows.

$$\begin{array}{l} (F(\underline{a}_1, \ldots, \underline{a}_k) = \underline{b})^* \equiv p_{Fa_1\ldots a_k b}, \quad (R(a_1, \ldots, a_k))^* \equiv q_{Ra_1\ldots a_k}, \\ (F(\underline{a}_1, \ldots, \underline{a}_k) = F'(\underline{a}'_1, \ldots, \underline{a}'_k))^* \equiv \\ \qquad \bigwedge_{b\in A}(F(\underline{a}_1, \ldots, \underline{a}_k) = \underline{b})^* \Rightarrow (F'(\underline{a}'_1, \ldots, \underline{a}'_k) = \underline{b})^*), \\ (F(t_1(\underline{a}_{11}, \ldots, \underline{a}_{1m}), \ldots, t_k(\underline{a}_{k1}, \ldots, \underline{a}_{km})) = \underline{b})^* \equiv \\ \qquad \bigwedge_{(b_1,\ldots,b_k)\in A^k} \left(\bigwedge_{i=1}^k (t_i(\underline{a}_{i1}, \ldots, \underline{a}_{im}) = \underline{b}_i)^* \Rightarrow p_{Fb_1\ldots b_k b}\right), \\ (R(t_1(\underline{a}_{11}, \ldots, \underline{a}_{1m}), \ldots, t_k(\underline{a}_{k1}, \ldots, \underline{a}_{km})))^* \equiv \\ \qquad \bigwedge_{(b_1,\ldots,b_k)\in A^k} \left(\bigwedge_{i=1}^k (t_i(\underline{a}_{i1}, \ldots, \underline{a}_{im}) = \underline{b}_i)^* \Rightarrow q_{Rb_1\ldots b_k b}\right), \\ \quad (\neg\varphi)^* \equiv \neg\varphi^*, \quad (\varphi \wedge \psi)^* \equiv \varphi^* \wedge \psi^*, \quad (\varphi \vee \psi)^* \equiv \varphi^* \vee \psi^*, \\ \quad\quad (\forall x\varphi(x))^* \equiv \bigwedge_{a\in A} \varphi(\underline{a})^*, \quad (\exists x\varphi(x))^* \equiv \bigvee_{a\in A} \varphi(\underline{a})^*. \end{array} \tag{3.2}$$

The constants symbols from L are handled in this definition of $*$ as 0-placed function symbols. If L has only one function symbol F, then we shall write $p_{b_1\ldots b_k b}$ instead of $p_{Fb_1\ldots b_k b}$. The similar convention is assumed for a relation symbol R. For example, if φ is the sentence which states the associativity of the binary function symbol $\cdot$, it is easy to see that the $*$-transform of $i \cdot j = u$ is p_{iju} and that over domain $I_n = \{0, 1, \ldots, n-1\}$, φ^* is equivalent to

$$\bigwedge_{i,j,k,u,v,l<n} ((p_{iju} \wedge p_{jkv} \wedge p_{ukl}) \Rightarrow p_{ivl}) \tag{3.3}$$

If not stated otherwise, we assume that the domain of a finite model $\mathbf{A}$ having n elements is $I_n = \{0, 1, \ldots, n-1\}$. Observe that $\mathcal{P}$ is finite. If $\mathbf{A}$ is a model of L, note that the simple expansion $(\mathbf{A}, a)_{a\in A}$ is a model of L_A.

3.2. Correspondence between models of T and T^*. Using translation $*$, we give a method for constructing and counting finite models of first order theories for a finite language L. In the rest of the paper the notion of a labeled model will have the important role. Therefore we fix this and related concepts.

Let $\mathbf{A}$ be a finite model of L, $|A| = n$. Any one-to-one and onto map $\alpha\colon I_n \to A$ will be called the labeling of $\mathbf{A}$. We can transfer the structure of $\mathbf{A}$ to a model $\mathbf{A}_\alpha$ with the domain I_n in the usual way:

1. If $R \in L$ is is a k-placed relation symbol then we take
   $R^{\mathbf{A}_\alpha}(i_1, \ldots, i_k)$ iff $R^{\mathbf{A}}(\alpha(i_1), \ldots, \alpha(i_k))$, $i_1 \ldots, i_k \in I_n$.
2. If $F \in L$ is is a k-placed function symbol then we take
   $F^{\mathbf{A}_\alpha}(i_1, \ldots, i_k) = \alpha^{-1}(F(\alpha(i_1), \ldots, \alpha(i_k))$, $i_1 \ldots, i_k \in I_n$.
3. If $c \in L$ is a constant symbol then $c^{\mathbf{A}_\alpha} = \alpha^{-1}(c^A)$.

We see that $\alpha\colon \mathbf{A}_\alpha \cong \mathbf{A}$. We shall call $\mathbf{A}_\alpha$ a labeled model of $\mathbf{A}$. Let $c_0, \ldots, c_{n-1}$ be new constant symbols to L and $L' = L \cup \{c_0, \ldots, c_{n-1}\}$. The simple expansion $(\mathbf{A}, \alpha_0, \ldots, \alpha_{n-1})$ is a model of L' such that c_i is interpreted by $\alpha_i = \alpha(i)$, $0 \leqslant i < n$. Instead of $(\mathbf{A}, \alpha_0, \ldots, \alpha_{n-1})$ we shall write shortly $(\mathbf{A}, \alpha)$.

THEOREM 3.1. *Assume $\mathbf{A}$ is a finite model of L, $|A| = n$ and α, β are labelings of $\mathbf{A}$. Then the following are equivalent*

(1) $(\mathbf{A}, \alpha) \equiv (\mathbf{A}, \beta)$, *i.e.,* $(\mathbf{A}, \alpha)$ *and* $(\mathbf{A}, \beta)$ *are elementary equivalent models,*
(2) $(\mathbf{A}, \alpha) \cong (\mathbf{A}, \beta)$,
(3) $\mathbf{A}_\alpha = \mathbf{A}_\beta$,
(4) $\alpha \circ \beta^{-1} \in \mathrm{Aut}(\mathbf{A})$.

PROOF. It is well known that finite elementary equivalent models are isomorphic. Hence (1) is equivalent to (2).

Suppose $(\mathbf{A}, \alpha_0, \ldots, \alpha_{n-1}) \cong (\mathbf{A}, \beta_0, \ldots, \beta_{n-1})$. So there is $f \in \mathrm{Aut}(\mathbf{A})$ such that $f(\beta_i) = \alpha_i$, $0 \leqslant i < n$. Hence $f \circ \beta = \alpha$, so $\alpha \circ \beta^{-1} \in \mathrm{Aut}(\mathbf{A})$. Therefore (2) implies (4). Reversing this proof, it also follows that (4) implies (2).

Suppose $(\mathbf{A}, \alpha) \equiv (\mathbf{A}, \beta)$ and let $F \in L$ be a k-placed function symbol. Then for any choice of constant symbols $c_{i_1}, \ldots, c_{i_{k+1}}$, $(\mathbf{A}, \alpha) \models F(c_{i_1}, \ldots, c_{i_k}) = c_{i_{k+1}}$ if and only if $(\mathbf{A}, \beta) \models F(c_{i_1}, \ldots, c_{i_k}) = c_{i_{k+1}}$. Hence

$$F^{\mathbf{A}}(\alpha(i_1), \ldots, \alpha(i_k)) = \alpha(i_{k+1}) \quad \text{iff} \quad F^{\mathbf{A}}(\beta(i_1), \ldots, \beta(i_k)) = \beta(i_{k+1}), \tag{3.4}$$

therefore $\alpha^{-1}(F(\alpha(i_1), \ldots, \alpha(i_k)) = \beta^{-1}(F(\beta(i_1), \ldots, \beta(i_k))$ for all $i_1, \ldots, i_k \in I_n$. Thus we proved that $F^{(\mathbf{A},\alpha)} = F^{(\mathbf{A},\beta)}$. Similarly we can prove that $R^{(\mathbf{A},\alpha)} = R^{(\mathbf{A},\beta)}$ for each relation symbol $R \in L$. Hence we proved that $\mathbf{A}_\alpha = \mathbf{A}_\beta$ and so (1) implies (3). Similarly one can prove that (3) implies (1). □

Finite models of a first order theory T which have for domains sets I_n are called labeled models of T. By $\mathscr{L}_{T,n}$ we shall denote the set of all labeled models of T of size n. By T_n we denote the theory $T \cup \{\sigma_n\}$, where σ_n denotes the sentence there are exactly n elements. Therefore, $\mathscr{L}_{T,n}$ is the set of all labeled models of T_n.

By a finite theory we mean a first order theory T with finitely many axioms, i.e., T is a finite set of sentences of a finite language L. We can replace T with a single sentence, but in some cases we need to add or remove a sentence from T. In these cases, it is technically easier to work with a set of sentences then with a single sentence which replaces T.

Suppose T is a finite theory. Let $\mathcal{P}$ be the set of propositional letters defined by 3.1 over $A = I_n$ and the language L and let $T^* = \{\varphi^* | \, \varphi \in T\}$. Further, let

$\mathfrak{M}(T^*) \subseteq 2^{\mathcal{P}}$ denote the set of all models of T^*, i.e., valuations satisfying all propositional formulas in T^*. The following construction describes the correspondence between labeled models of T and models of T^*.

The function h which assigns to each $\mu \in \mathfrak{M}(T^*)$ a labeled model $h(\mu) = \mathbf{A}$ of T is defined as follows. Let $a_1, \ldots, a_k, b \in I_n$. Then

If $F \in L$ is an k-placed function symbol, then

$$F^{\mathbf{A}}(a_1, \ldots, a_k) = b \quad \text{iff} \quad \mu(p_{Fa_1\ldots a_k b}) = 1. \tag{3.5}$$

If $R \in L$ is an k-placed relation symbol, then

$$\mathbf{A} \models R[a_1, \ldots, a_k] \quad \text{iff} \quad \mu(q_{Ra_1\ldots a_k}) = 1. \tag{3.6}$$

By induction on the complexity of the formula φ, it is easy to prove that $\mathbf{A} \in \mathscr{L}_{T,n}$ and if $\mu \neq \nu$, then for the corresponding $\mathbf{A}_\mu$ and $\mathbf{A}_\nu$ we have $\mathbf{A}_\mu \neq \mathbf{A}_\nu$. Hence, map $h\colon \mathfrak{M}(T^*) \to \mathscr{L}_{T,n}$ is one-to-one. On the other hand, assume $\mathbf{A} \in \mathscr{L}_{T,n}$. We can use 3.5 and 3.6 now to define the valuation $\mu_{\mathbf{A}}$. Since $\mathbf{A}$ is a model of T, it follows that $\mu_{\mathbf{A}} \in \mathfrak{M}(T^*)$. Hence, h is onto. Therefore we proved:

Theorem 3.2. *The map h codes the models in $\mathscr{L}_{T,n}$ by models of T^*.*

This theorem is our starting point in finding finite models of T of size n. As T^* is finite, we can replace it with a single propositional formula $\theta = \bigwedge_{\psi \in T^*} \psi$. Obviously, we may consider θ as a Boolean term $t(v_1, \ldots, v_m)$. Computing $t^{\mathbf{\Omega}_m}(b_1, \ldots, b_m)$ in free Boolean algebra $\mathbf{\Omega}_m$ for free generators $b_1, \ldots, b_m$, we obtain the vector b which by Proposition 2.2 codes the full DNF of θ, hence all models of T^*. This gives us all labeled models of T of size n via the map h.

Let $l_{T,n}$ denote the cardinality of $\mathscr{L}_{T,n}$. Obviously, $l_{T,n}$ is equal to the number of bits in vector b which are equal to 1.

The mayor target in finite model theory is to count or to determine non-isomorphic models of T of size n. By $\mathfrak{M}(T)_n$ we denote a maximal set of non-isomorphic models of T with the domain I_n. Elements of this set are also called unlabeled models of T. By $\kappa_{T,n} = |\mathfrak{M}(T)_n|$ we denote the number of non-isomorphic (unlabeled) models of T of size n. If a theory T is fixed in our discussion, we often omit the subscript T in these symbols. In other words, we shall simply write $\mathscr{L}_n$, l_n, $\mathfrak{M}_n$ and κ_n. In our examples, the following theorem will be useful in finding numbers l_n and κ_n.

Theorem 3.3. (*Frobenius - Burnside counting lemma*) *Let $\mathbf{A}$ be a finite model, $|A| = n$. Then the number of models isomorphic to $\mathbf{A}$ which have the same domain A is equal to $n!/|\mathrm{Aut}(\mathbf{A})|$.*

If T is a theory of a finite language L with finite number of axioms, then

$$l_n = \sum_{\mathbf{A} \in \mathfrak{M}_n} \frac{n!}{|\mathrm{Aut}(\mathbf{A})|}. \tag{3.7}$$

Note that this theorem immediately follows from theorem 3.1 and direct application of Langrange's subgroup theorem on the symmetric group S_n of I_n.

It is said that a set of models $\mathcal{K}$ is adequate for n-models of T if $\mathfrak{M}_n \subseteq \mathcal{K} \subseteq \mathscr{L}_n$. Even for small n the set $\mathscr{L}_n$ can be very large. On the other hand, it is possible in some cases to generate easily all labeled models, or to determine l_n from $|\mathcal{K}|$ for an adequate family $\mathcal{K}$ of the reasonable size. Also, it is commonly hard to generate directly non-isomorphic models of T, or to compute κ_n. But for a well chosen adequate set of models these tasks can be done. Adequate families are usually generated by filtering $\mathscr{L}_n$, fixing some constants or definable subsets in models of T, or imposing extra properties, for example adding a new sentences to T. In our examples some instances of adequate families will be given.

3.3. Killing variables. Suppose a theory T describes a class of finite models. The set of propositional letters $\mathcal{P}$ defined by 3.1 and which appears in translation from T to T^* is large even for small n for domains $A = I_n$ from which $\mathcal{P}$ is generated. For example, if the language L consists of k unary operations, then $|\mathcal{P}| = kn$. If L has only one binary operation R, then $|\mathcal{P}| = n^2$. If L has only one binary operation F, then $|\mathcal{P}| = n^3$. Hence, even for small n, $\mathcal{P}$ can be enormously large. It can have hundreds, or even thousands of propositional variables. Hence, we need a way to eliminate some propositional variables appearing in T^*. Any procedure of elimination variables from $\mathcal{P}$ we shall call killing variables. As we have seen, the size of $\mathcal{P}$ which appears in T^* and is feasible for computing on small computers is bellow 50 and on supercomputers below 70. Let us denote by K this feasible number of variables[3]. The main goal of killing variables is to reduce T^* to a propositional theory T' having at most K variables. We note that killing variables in general produces an adequate set of structures, not the whole $\mathscr{L}_n$.

Killing variables is reduced in most cases by fixing the values of certain variables. For example, if p_{ijk} represents a binary operation $i \cdot j = k$, $i, j, k \in A$, and if it is known that for some $a, b, c \in A$, $p_{abc} = 1$, then for all $d \in A$, $d \neq c$, we may take $p_{abd} = 0$. The next consideration explains in many cases this kind of killing variables. It is related to the definability theory and for notions and terminology we shall refer to [4].

Suppose $\mathbf{A}$ is a model of L and $X \subseteq A$. We say that X is absolutely invariant in $\mathbf{A}$ if for all $f \in \mathrm{Aut}(\mathbf{A})$, $f(X) \subseteq X$. As usual, X is definable in $\mathbf{A}$ if there is a formula $\varphi(x)$ of L so that $X = \{a \in A \colon \mathbf{A} \models \varphi[a]\}$. The proof of the next theorem is based on the the Svenonius definability theorem, cf. [10], or Theorem 5.3.3 in [4].

THEOREM 3.4. *Let $\mathbf{A}$ be a finite model of L and $X \subseteq A$. Then X is absolutely invariant in $\mathbf{A}$ if and only if X is definable in $\mathbf{A}$.*

PROOF. Obviously, if X is definable then it is absolutely invariant. So we proceed to the proof of the other direction. In order to save on notation, we shall take $L = \{R\}$, R is a binary relation symbol. Suppose X is invariant under all automorphisms of $\mathbf{A}$. Let $\psi_1(U)$ be the following sentence of $L \cup \{U\}$, U is a new

[3] Hence $50 \leqslant K \leqslant 70$ for today's computers

unary predicate:

$$\begin{aligned}&\forall x_1 \ldots x_n \forall y_1 \ldots y_n((\bigwedge_{i<j} x_i \neq x_j \wedge \bigwedge_{i<j} y_i \neq y_j \wedge \bigwedge_{i,j}(R(x_i,x_j) \Leftrightarrow R(y_i,y_j)))\\ &\qquad \Rightarrow \bigwedge_i (U(x_i) \Rightarrow U(y_i))).\end{aligned} \tag{3.8}$$

The sentence $\psi_1(U)$ states that U is absolutely invariant in any model $\mathbf{B}$ of L which has n elements, i.e., if $(\mathbf{B}, Y) \models \psi_1(U)$ then Y is absolutely invariant in $\mathbf{B}$.

Let ψ_2 be the following sentence of L:

$$\exists x_1 \ldots x_n (\bigwedge_{i<j} x_i \neq x_j \wedge \forall x \bigvee_i x = x_i \wedge \bigwedge_{R^{\mathbf{A}}(i,j)} R(x_i, x_j) \wedge \bigwedge_{\neg R^{\mathbf{A}}(i,j)} \neg R(x_i, x_j)). \tag{3.9}$$

We see that the sentence ψ_2 codes the model $\mathbf{A}$, i.e., if $\mathbf{B}$ is a model of L and $\mathbf{B} \models \psi_2$ then $\mathbf{B} \cong \mathbf{A}$.

Let $\psi(U) = \psi_1(U) \wedge \psi_2$. Suppose $\mathbf{B}$ is any model of L, $(\mathbf{B}, Y)$ and $(\mathbf{B}, Y')$ are expansion of $\mathbf{B}$ to models of $\psi(U)$ and assume $(\mathbf{B}, Y) \cong (\mathbf{B}, Y')$. Then we see that $Y = Y'$. Therefore, by Svenonius theorem it follows that ψ defines U explicitly up to disjunction. In other words there are formulas $\varphi_1(x), \ldots, \varphi_m(x)$ of L such that

$$\psi(U) \models \bigvee_i \forall x (U(x) \Leftrightarrow \varphi_i(x)) \tag{3.10}$$

As $(\mathbf{A}, X) \models \psi(U)$, there is i so that $(\mathbf{A}, X) \models \forall x (U(x) \Leftrightarrow \varphi_i(x))$. Hence X is definable by $\varphi_i(x)$. □

The following corollaries follow by direct application of the last theorem to one-element absolutely invariant subsets.

Corollary 3.1. *Let* $\mathbf{A}$ *be a finite model of finite* L *and* $a \in A$. *If* a *is fixed by all automorphisms of* $\mathbf{A}$ *then* a *is definable in* $\mathbf{A}$ *by a formula* $\varphi(x)$ *of* L.

Corollary 3.2. *Let* $\mathbf{A}$ *be a finite model of finite* L. *Then* $\mathrm{Aut}(\mathbf{A}) = \{i_A\}$ *if and only if every element of* A *is definable in* $\mathbf{A}$.

Here are other examples of absolutely invariant, and hence definable subsets X in various types of finite structures $\mathbf{A}$. If $\sim$ is a relation of equivalence over A and $k \in N$, then $X =$ "the union of all classes of equivalences of size k" is absolutely invariant. Let $\mathbf{A} = (A, \leqslant)$ be a partial order. Then the set S of all minimal elements and the set T of all maximal elements of $\mathbf{A}$ are absolutely invariant. The same holds for the set of all minimal elements of $A \backslash S$. In groups, characteristic subgroups, such as the center and the commutator subgroup, are absolutely invariant.

In our examples we shall often use the following argument. Let T be a finite theory of L and assume $\varphi_0(x), \ldots, \varphi_{k-1}(x)$ are formulas of L for which T proves they are mutually disjoint, i.e., for $i \neq j$, $T \vdash \neg\exists x(\varphi_i(x) \wedge \varphi_j(x))$. Assume they define constants in T, in other words, for each i

$$T \vdash \exists x(\varphi(x) \wedge \forall y(\varphi_i(y) \Rightarrow \varphi_i(x)). \tag{3.11}$$

Let $\mathbf{B}$ be a model of T, $B = I_n$ and $\mathbf{B} \models T$. Then $\mathbf{B}$ has a unique expansion to $(\mathbf{B}, b_0, \dots, b_{k-1})$ which is a model of $T' = T \cup \{\varphi_0(c_0), \dots, \varphi_k(c_{k-1})\}$, $c_0, \dots, c_{k-1}$ are new symbols of constants to L. Since $b_i \neq b_j$ for $i \neq j$ we can define

$$f\colon I_k \to \{b_0, \dots, b_{k-1}\}, \quad f(i) = b_i, \quad i = 1, \dots, k. \tag{3.12}$$

It is easy to see that we can define labeled model $\mathbf{A}$ of L and that f extends to $h\colon (\mathbf{A}, 0, \dots, k-1) \cong (\mathbf{B}, b_0, \dots, b_{k-1})$. Hence, for an adequate set of n-models of T we can choose a set $\mathcal{K}$ of labeled models $\mathbf{A}$ of T such that $(\mathbf{A}, 0, \dots, k-1)$ is a model of T'. Therefore, models in K have the fixed labelings by $0, \dots, k-1$ of constants definable in T.

Obviously, we can take in 3.12 any $S \subseteq I_n$, $|S| = k$ instead of I_k. There are $\binom{n}{k}$ such choices of S. Let s denote a permutation $s_0 \dots s_{k-1}$ of S and $\mathcal{K}_s$ the corresponding adequate set for n-models of T: for models $\mathbf{A}$ in $\mathcal{K}_s$, $(\mathbf{A}, s_0, \dots, s_{k-1})$ is a model of T'. In other words, definable elements formerly labeled by $0, \dots, k-1$ in models of $\mathcal{K}$ they are labeled now in $\mathcal{K}_s$ by $s_0, \dots, s_{k-1}$. Suppose S and S' are k-subsets of I_n and s, s' permutations either of S or S', $s \neq s'$. Then $\mathcal{K}_s \cap \mathcal{K}_{s'} = \emptyset$ and $|\mathcal{K}_s| = |\mathcal{K}_{s'}|$. Hence $\mathscr{L}_{T,n} = \bigcup_s \mathcal{K}_s$ and so

$$l_{T,n} = \binom{n}{k} k!|\mathcal{K}| = n(n-1)\cdots(n-k+1)|\mathcal{K}|. \tag{3.13}$$

In many cases theory T determines the values of atomic formulas which contains some of the definable constants. Hence, the corresponding propositional letter from $\mathcal{P}$ has a definite value. For example, suppose R is a 2-placed relation symbol and that T proves $\forall x R(c_0, x)$. Then we can take $p_{0i} = 1$, $i = 0, \dots, n-1$. Hence, if $\mathcal{P}$ is generated over I_n, n propositional variables are killed in $\mathcal{P}$. The remaining number of variables is $n^2 - n$.

3.4. Definable partitions.

The presented idea with definable constants can be extended to definable subsets as well. For simplicity, we shall assume that $L = \{R\}$, where R is a binary relation symbol.

A sequence $\Delta = \theta_1(x), \dots, \theta_m(x)$ of formulas of L is called a definable partition for T_n if T_n proves:

1. $\forall x(\theta_1(x) \vee \dots \vee \theta_m(x))$.
2. $\neg\exists x(\theta_i(x) \wedge \theta_j(x))$, $\quad 1 \leqslant i \leqslant j \leqslant m$.

We shall say that Δ is a good definable partition if there are formulas $S_{ij}(x, y)$, $1 \leqslant i, j \leqslant m$, such that each $S_{ij}(x, y)$ is one of $R(x, y)$, $R(y, x)$, $\neg R(x, y)$, $\neg R(y, x)$, and T_n proves:

$$\forall xy((\theta_i(x) \wedge \theta_j(y)) \Rightarrow S_{ij}(x, y)), \quad 1 \leqslant i \leqslant j \leqslant m. \tag{3.14}$$

Example 3.1. It is easy to write first-order formula $\theta_k(x)$ which says that x has exactly k R-connections with other elements. In other words, $\theta_k(x)$ expresses that there are exactly k elements y such that $R(x, y)$. Assume T_n proves that R is an acyclic graph. Then $k \leqslant l$ implies $(\theta_k(x) \wedge \theta_l(y)) \Rightarrow \neg R(x, y)$. Hence, in this case definable partition $\theta_k(x)$ is good.

In any labeled model $\mathbf{A}$ of T_n, Δ determines sequence $\mathcal{X}$ of definable subsets $X_1, \ldots, X_m$. By a component we shall mean elements of $\mathcal{X}$. It may happen that some components are empty. The sequence of non-empty sets from $\mathcal{X} = (X_1, \ldots, X_m)$ form an ordered partition of A. A sequence $\mathcal{X}$ with this property will be called a c-partition.

Our idea for using a good definable partition Δ in generating labeled models $\mathbf{A}$ of T_n is as follows. We assume that the propositional letter p_{ij} represents $R^{\mathbf{A}}(i, j)$ as described by 3.6. We generate all c-partitions $\mathcal{X} = (X_1, \ldots, X_m)$ of I_n that are potentially components of $\mathbf{A}$, taking that X_i corresponds to θ_i. For each $\mathcal{X}$ we assign values to particular p_{ij} in the following way. If $S(x, y)$ is $R(x, y)$ then we set $p_{ij} = 1$ for $i \in X_k$ and $j \in X_l$, $k \leqslant l$ and if $S(x, y)$ is $\neg R(x, y)$, then we set $p_{ij} = 0$. We assign similarly values to p_{ij} if $S(x, y)$ is $R(y, x)$ or $\neg R(y, x)$. Therefore we obtained propositional theory $T_{\mathcal{X}} \subseteq T_n^*$ with the reduced number of unknowns from $\mathcal{P}$. Then set $\mathcal{K}_{\mathcal{X}}$ of labeled models corresponding to $T_{\mathcal{X}}$ in the sense of Subsection 3.2 is adequate for set $\mathscr{L}_{\mathcal{X}}$ of all labeled models of T_n in which Δ defines partition $\mathcal{X}$.

Obviously, every model of T_n is isomorphic to a model $\mathbf{A}$ with domain I_n with canonical components $\mathcal{X}$

$$\begin{gathered} X_1 = \{0, 1, \ldots, \alpha_1 - 1\},\, X_2 = \{\alpha_1, \alpha_1 + 1, \ldots, \alpha_1 + \alpha_2 - 1\}, \ldots, \\ X_m = \{\sum_{i<m} \alpha_i, \sum_{i<m} \alpha_i + 1, \ldots, \sum_{i \leqslant m} \alpha_i - 1\}. \end{gathered} \tag{3.15}$$

Let us denote by $\mathscr{P}$ the set of all c-partitions of I_n. Then every model $\mathbf{A} \in \mathscr{L}_{\mathcal{X}}$, $\mathcal{X} = (X_1, \ldots, X_n)$ is obtained from a model $\mathbf{B} \in \mathcal{K}_X$ choosing component X_1 from I_n, then X_2 from $I_n \backslash X_1$, X_3 from $I_n \backslash \{X_1 \cup X_2\}$ and so on, until all X_i from $\mathcal{X}$ are exhausted. Therefore

$$l_{T,n} = \sum_{\mathcal{X} \in \mathscr{P}} \binom{\beta_1}{\alpha_1} \cdots \binom{\beta_k}{\alpha_k} |\mathcal{K}_{\mathcal{X}}| \tag{3.16}$$

where $\mathcal{X} = (X_1, \ldots, X_k)$, $|X_i| = \alpha_i$ and

$$\beta_1 = n, \quad \beta_2 = \beta_1 - \alpha_1, \quad \ldots, \quad \beta_k = \beta_{k-1} - \alpha_{k-1}. \tag{3.17}$$

Note that if $\mathcal{X} \neq \mathcal{Y}$, $\mathcal{X}, \mathcal{Y} \in \mathscr{P}$, and if $\mathbf{A} \in \mathcal{K}_X$ and $\mathbf{B} \in \mathcal{K}_Y$, then $\mathbf{A}$ and $\mathbf{B}$ are non-isomorphic. Hence, if $\kappa_{\mathcal{X},n}$ is the number of non-isomorphic models in $\mathcal{K}_{\mathcal{X}}$, then

$$\kappa_n = \sum_{\mathcal{X} \in \mathscr{P}} k_{\mathcal{X},n}. \tag{3.18}$$

The following proposition is useful in estimation of the number of computing steps of $\mathcal{K}_{\mathcal{X}}$.

Proposition 3.1. *Assume $|A| = n$. Then there are*

$$c_{nm} = \sum_{k=1}^{m} \binom{m}{k} \binom{n-1}{k-1} \tag{3.19}$$

c-partitions $\mathcal{X} = (X_1, \ldots, X_m)$ of A.

Proof. Let $|X_i| = \alpha_i$. Therefore $\alpha_1, \ldots, \alpha_m$ is an integer solution of

$$n = x_1 + \ldots + x_m, \quad x_1, \ldots, x_m \geqslant 0. \tag{3.20}$$

Since the integer solutions of

$$n = x_1 + \ldots + x_k, \quad x_1, \ldots, x_k \geqslant 1. \tag{3.21}$$

are obtained from 3.20 by choosing k variables $x_i \neq 0$, $k \geqslant 1$, and 3.21 has $\binom{n-1}{k-1}$ solutions, there are $\binom{m}{k}\binom{n-1}{k-1}$ solutions of 3.20. Hence, there are $\binom{m}{k}\binom{n-1}{k-1}$ c-partitions $\mathcal{X}$ with exactly k nonempty sets X_i. Summing up for $k = 1, \ldots, m$, we obtain expression 3.19 for c_{nm}. □

Hence, for an adequate set of models of T_n we can take set $\mathcal{K}$ of labeled models $\mathbf{A}$ of T_n with the components 3.15. So our method for computing models of $\mathcal{K}$ is as follows. As usual, the propositional letter p_{ij} stands for $R(i, j)$.

Counting procedure TBA

1. Find good definable partition $\theta_1(x), \ldots, \theta_m(x)$ which satisfies condition 3.14.
2. Generate all c-partitions $\mathcal{X} = (X_1, \ldots, X_m)$ of I_n with arrangements 3.15.
3. *Killing variables*: For all $1 \leqslant k \leqslant l \leqslant n$ we fix the values of certain p_{ij} as follows. Take $p_{ij} = 1$ for $i \in X_k$ and $j \in X_l$ if $S(x, y)$ is $R(x, y)$. If $S(i, j)$ is $\neg R(i, j)$ then we take $p_{ij} = 0$. If $S(i, j)$ is $R(j, i)$, then $p_{ji} = 1$. If $S(i, j)$ is $R\neg R(j, i)$, then set $p_{ji} = 0$.
4. Reduce T_n^* to $T_{\mathcal{X}}$ with the reduced number of variables using assigned values to variables p_{ij} in the previous step.
5. Generate and count models of $\mathcal{K}_{\mathcal{X}}$ using $T_{\mathcal{X}}$ and free Boolean vectors by the procedure described in section 3.2.
6. Find $\kappa_{X,n}$ by enumerating elements of $\mathcal{K}_X$.
7. Repeat steps (5) and (6) until $\mathscr{P}$ is exhausted.
8. Compute $l_{T,n}$ by formula 3.16.
9. Compute $\kappa_{T,n}$ by 3.18.

4. Program implementation

We implemented the algorithms and ideas presented in the previous sections into a programming system which we shall call TBA. It is divided into two layers. The first one is implemented in OpenCL which we have chosen as a good framework for writing parallel programs that execute across heterogeneous platforms consisting of central processing units (CPUs) and graphics processing units (GPUs). This part of code manipulates with free Boolean vectors as described in subsection 2.3 and it is invisible to the general user of TBA. The second layer is developed in Python programming language and we used it to achieve two goals. The first-one is to manipulate Boolean expressions as described in subsections 3.1 and 3.3. The second aim was to define new constructs in Python mainly related to the predicate calculus. The general user may use them into scripts to solve combinatorial problems using techniques such as described in subsection 3.4. The main body of a script strictly follow the syntax of predicate calculus, but Python standard constructs can be embedded in the scripts as well. The user executes the scripts by TBA in the terminal mode.

4.1. TBA Core. The core of the system is a parallel computational engine that searches for models of a Boolean formula $\varphi(x_1, \ldots, x_n)$. This part of TBA is generated in OpenCL language which is then compiled to binaries and executed. In a sense, the core uses brute force search over a problem space, but utilizing all of the available bit level parallelism of the underlying hardware as described in Subsection 2.3. Whenever $\varphi(x_1, \ldots, x_n)$ is dispatched to the engine, it first partitions search space S. The table of S associated to $\varphi(x_1, \ldots, x_n)$ is of size $n \times 2^n$ and consists of free Boolean vectors. The partitioning of S is done by slicing this table into appropriate blocks and depends on the number of available processors and memory. Due to the simplicity of the representation, the slicing scheme is very scalable. This enables us to choose a partition such that all cores of all of available processing units are used in parallel in further computation.

In addition, the engine generates an efficient computing tree for $\varphi(x_1, \ldots, x_n)$, adapted to the actual parallel hardware and hardware architecture. The implementation is done for both, GPU's and CPU's and it is on the user which implementation will be used. While for GPU's the advantage is the number of computing cores, for CPU's this is the length of the vector units and the processor's speed. Modern GPU's have more than 2000 computing 32 bit cores, while, in contrast, CPU's have four cores, 256 bit registers and up to four time faster clock speed. The approximative formula for the ratio between the speeds of the execution of our code on a GPU and on a CPU is:

$$f = \frac{n_g b_g s_g}{n_c b_c s_c} \tag{4.1}$$

where n_g is the number of b_g-bit computing cores and s_g is the number of clock cycles of GPU, while n_c, b_c, s_c are the similar parameters for the CPU (b_c is the number of bits of the vector unit). Hence, for the above mentioned configuration ($n_g = 2^{11}$, $b_g = 2^5$, $n_c = 2^2$, $b_c = 2^8$ and $s_c/s_g = 4$), we have $f = 16$. Therefore, GPU's are superior to CPU's and our tests are in agreement with 4.1.

There are also other submodules. Submodule `Translate` translates predicate formulas into Boolean expressions according to the rules explained in Subsection 3.1. It also build the computing tree of so obtained Boolean term. Another important submodule is `Reduction` which reduces a Boolean term having constants 0 and 1 to the expression without these constants. We observe that a Boolean expression may have several hundreds of thousands of characters, but `Reduction` is limited not by the size of the expression, but only by the available computer's memory.

4.2. TBA scripts. TBA scripts are used to implement algorithms for generating and counting finite combinatorial structures such as specific graphs, orders, Latin squares, automorphisms of first-order structures, etc. The user writes TBA scripts as txt files and they follow Python syntax. In general their structure consist of three parts. The first part contains definitions of domains over which combinatorial objects are generated. The second one consists of definitions of combinatorial structures by axioms written in the syntax of the predicate calculus. Propositional calculus is embedded into Python, but we had to expand it with bounded quantifiers in order to express predicate formulas having in mind finite structures as the main (and only) semantics. The quantifier extension of Python we named Python-AE, since we denoted by A the universal quantifier and the existential quantifier by E. Finally, the third part is used for killing variables, as described in subsection 3.3. These parts are not strictly separated and they may overlap.

Here is a simple example of a TBA script, named `SO.txt`. It computes all partial orders over domain $S = \{0, 1, \ldots, n-1\}$ with a special element. The propositional letter p_{ij} stands for $i \leqslant j$. An element $a \in S$ is special if it is comparable with all elements of domain S.

```
n= 6
S= range(n)
S2= perm(range(n),2)
S3= perm(range(n),3)
f1= A[i,j:S2] (~p(i,j) | ~p(j,i))
f2= A[i,j,k:S3] (~(p(i,j)& p(j,k)) | p(i,k))
f3= E[i:S].A[j:S] (p(i,j) | p(j,i))
assumptions= {p(i,i): 1 for i in S}
```

First four lines define domain $S = \{0, 1, 2, 3, 4, 5\}$, set S_2 of ordered pairs of elements of S with distinct coordinates and S_3, the set of triplets. The next three lines define predicate formulas $\varphi_1, \varphi_2, \varphi_3$. Boolean operation signs are represented in the standard Python notation. Hence, the Python signs $\sim, \&, |, \hat{}$ stand respectively for $\neg, \wedge, \vee, +$, where $x + y = x\bar{y} \vee \bar{x}y$ (symmetric difference of x and y). The construct $A[i : S]$ stand for the bounded universal quantifier (in the manner of Polish logic school, eg [6]) $\bigwedge_{i \in S}$. Similarly, $E[i : S]$ denotes the bounded existential quantifier $\bigvee_{i \in S}$. Hence, f_1, f_2, f_3 are Python-AE transcripts of the following predicate formulas, if p_{ij} is read as $i \leqslant j$:

$$\begin{aligned}
\varphi_1 &= \bigwedge_{i,j \in S_2} (\neg p_{ij} \vee \neg p_{ji}) \\
\varphi_2 &= \bigwedge_{i,j,k \in S_3} (\neg(p(i,j) \wedge p(j,k)) \vee p(i,k)) \\
\varphi_3 &= \bigvee_{i \in S} \bigwedge_{j \in S} (p_{ij} \vee p_{ji}).
\end{aligned} \tag{4.2}$$

Obviously, φ_1 states that $\leqslant$ is antisymmetric ie, $\forall i, j \in S(i \leqslant j \wedge j \leqslant i \Rightarrow i = j)$. Further, φ_2 states that $\leqslant$ is transitive, $\forall i, j, k \in S(i \leqslant j \wedge j \leqslant k \Rightarrow i \leqslant k)$, assuming it is reflexive. The reflexivity is handled in the last line of the script. Finally, φ_3 states that the order has a special element, $\exists i \in S\, \forall j \in S(i \leqslant j \vee j \leqslant i)$.

The last line states that $\leqslant$ is reflexive. It also kills variables p_{ii}, $i \in S$. The last line can be replaced by $\bigwedge_{i \in S}$, but during the execution of `SO.txt` we would have then more free variables and the program would be less capable. Observe that there are all together n^2 variables p_{ij} and that n variables are killed. Hence, during the execution of the script, there are $n^2 - n$ free variables. Our current implementation solves on GPU's systems of the Boolean equations which have up to 30 unknowns and on CPU's with up to 32 unknowns. Hence the script can be run for $n \leqslant 6$. More sophisticated examples which could be executed for much larger n are explained in the next section.

The script is executed on a GPU (default case) by `solve.exe --all SO.txt` and on a CPU: `solve.exe --all --cpu SO.txt`.

Output file `out.txt` contains after execution all solutions of 4.2, ie, all models of propositional formulas which are $*$-transforms of formulas $\varphi_1, \varphi_2, \varphi_3$ in the sense of subsection 3.1. All partial orders $(S, \leqslant_\mu)$ with a special element are obtained then by choosing valuations (rows) μ from `out.txt` and setting $i \leqslant_\mu j$ iff $\mu(p_{ij}) = 1$.

Here are some general remarks and basic rules for Python-AE. Predicate formulas only with bounded quantifiers are allowed and must be in written in the prenex normal form. The quantifier-free part otherwise follows the Python syntax for Boolean expressions and must be parenthesized. Quantifiers are delimited from each others by the dot sign.

Killing variables means setting values for some variables appearing in formulas of a TBA script file. Construction implemented in Python for killing variables is called `assumptions`. Assumptions for killing variables are defined using Python dictionary structure. For example {`a:1, b:0`} defines a dictionary which sets values of two variables: $a = 1, b = 0$. In this way, listing values of variable, any dictionary for killing variables can be constructed. A dictionary can be constructed also in other ways using Python syntax.

Example (dictionary comprehension): `assumptions=` {`p(i):1 for i in S`}. In this way we defined `p(i)=1` for all i in S.

Dictionary which defines values of variables must be named `assumptions`. The above example demonstrates killing variables using incremental method applied on assumptions (dictionary): An already existing dictionary (assumptions) is updated by the command `assumptions.update`. If `assumptions.update` refers to already killed variables, their values are set to new values defined by this command. Hence, the order of updating is important.

A TBA script file `file.txt` is executed in the terminal mode by `solve.exe --all file.txt`. The result of the execution is placed in `out.txt`.

5. Examples

The portable codes for executing programs in our system, explanation how to use them and all examples described in this paper and some additional ones, can be found at the address http://www.mi.sanu.ac.rs/∼pejovica/tba. Most of our examples are tested against to the examples from the On-Line Encyclopedia of Integer Sequences (OEIS)[4]. In all cases, our results were in the agreement with the results which we found there.

[4] http://oeis.org

5.1. Solving Boolean equations. Solving Boolean equations is the simplest use of our software. Any system of Boolean equation should be written in our system in the following way:

$$e_1 = \varphi_1(x_1,\dots,x_n),\dots,e_k = \varphi_k(x_1,\dots,x_n). \tag{5.1}$$

The program finds all $(\alpha_1,\dots,\alpha_n) \in 2^n$ such that $\varphi_(\alpha_1,\dots,\alpha_n) \equiv 1$, $1 \leqslant i \leqslant k$.

Here is an example of two Boolean equations with unknowns x, y, z, u (example BAequ4_in.txt at the above address):

$$x + y + \bar{z} + u = 1, \quad x \vee yz = u \tag{5.2}$$

The second equation is equivalent to $\neg((x \vee yz) + u) = 1$. Hence, Python-AE file BAequ4_in.txt solves 5.2 and contains only two lines:

$$\begin{array}{rcl} e1 & = & x\,\hat{}\,y\,\hat{}\sim z\,\hat{}\,u \\ e2 & = & \sim((x \mid y \;\&\; z)\,\hat{}\,u) \end{array} \tag{5.3}$$

File BAequ4_in.txt is executed on a GPU (default case) by

```
solve.exe --all BAequ4_in.txt BAequ4_out.txt.
```

and on a CPU by

```
solve.exe --all --cpu BAequ4_in.txt BAequ4_out.txt.
```

Output file BAequ4_out.txt contains after execution all solutions of 5.2.

5.2. Ordered structures. Let T be the theory of partial orders of $L = \{\leqslant\}$ having at least 2 elements with extra axioms which state there are the least element and the greatest element x. Instead of T we can take the theory T_1 of partially ordered sets which are upward and downward directed. Theories T and T_1 are not equivalent, for example T_1 has an infinite model which is not a model of T. But T and T_1 have same finite models.

We see that $l_{T,n} = n(n-1)|\mathcal{K}|$, $n \geqslant 2$, where $\mathcal{K}$ is the set of all partial orders $\mathbf{A} = (A, \leqslant, 0, n-1)$, $A = I_n$, 0 is the least and $n-1$ is the greatest element in $\mathbf{A}$. Since p_{ij} states $i \leqslant j$ and $\leqslant$ is reflexive, we can also take $(n \geqslant 2)$

$$\begin{array}{l} p_{0i} = 1,\ p_{j0} = 0,\ p_{i1} = 1,\ p_{1k} = 0,\ p_{ii} = 1, \\ i = 0,\dots,n-1,\ j = 1,\dots,n-1,\ k = 0,\dots,n-2. \end{array} \tag{5.4}$$

Hence, $5n-6$ variables are killed and T^* is reduced to T' which has $v = n^2 - 5n + 6$ variables. If $n = 8$ then $v = 30$ and all partial orders having 8 elements are generated in one computer cycle in our computer installation. Simply adding to T some new axioms, we can generate models of the new theory in the same way and the same computing time. For example, in this way we can compute all lattices of order 8 just by adding to T only one axiom.

With small adjustments, this algorithm works on small computers in real time for $n \leqslant 12$. Namely, for larger n, the feasibility constant K, see the footnote (3), is exceeded. For larger n we have to use the previously described procedure based on components. In order to describe them, let us define recursively the following sequence of length n of the following formulas.

$$\theta_0(x) \equiv \forall y(x \leqslant y), \quad \theta_{k+1}(x) \equiv \forall y(\bigvee_{i\leqslant k} \theta_i(y) \vee x \leqslant y) \wedge \bigwedge_{i\leqslant k} \neg\theta_i(x). \tag{5.5}$$

If $\mathbf{A} = (A, \leqslant)$ is a partial order with domain I_n, we see that the associated components are: $X_0 = \{0\}$, 0 is the least element of $\mathbf{A}$, X_1 is the set of minimal elements of $A\backslash\{0\}$, X_2 is the set of minimal elements of $A\backslash(X_0 \cup X_1)$, and so on. Let us call an element of layer X_k, a k-minimal element. Since $X_{i+1} \neq \emptyset$ implies $X_i \neq \emptyset$, we see that $X_k = \emptyset$ for $k > m$ for some $m \leqslant n$. Hence,

$$\mathcal{X} = (X_1, \ldots, X_m, 0, \ldots, 0), \quad X_i \neq \emptyset, \tag{5.6}$$

is the associated c-partition of A.

Proposition 5.1. *Let* $\mathbf{A}$ *be a partial order of size* n *with the least element and the greatest element. Then the number of* c*-partitions* (5.6) *of* A *which consist from layers* X_k *of* k*-minimal elements is* $c_n = 2^{n-3}$.

Proof. Obviously, the least element and the greatest element can be omitted from A. Hence, we count c-partitions of $A' = \{1, 2 \ldots, n-2\}$. Let $|X_i| = \alpha_i$. Therefore $(\alpha_1, \ldots, \alpha_m)$ is an integer solution of

$$n - 2 = x_1 + \ldots + x_m, \quad x_1, \ldots, x_m \geqslant 1, \tag{5.7}$$

where $m \leqslant n-2$. Equation 5.7 has $\binom{n-3}{m-1}$ solutions, hence the total number of c-partitions 5.6 over domain A is

$$c_n = \sum_{m=1}^{n-2} \binom{n-3}{m-1} = 2^{n-3}. \tag{5.8}$$

□

If $i \in X_k$, $j \in X_l$, $l \leqslant k$ then $i \nleqslant j$. Hence, in addition to (5.4), for each c-partition more variables p_{ij} are killed:

$$p_{ij} = 0, \quad i \in X_k,\, j \in X_l,\, l \leqslant k. \tag{5.9}$$

For so introduced parameters, we can use the counting procedure TBA (Section 3.4) for finding and counting labeled and unlabeled partial orders of size n with the least element and the greatest element. According to Proposition 5.1, the procedure consists from 2^{n-3} loops. In each loop, a c-partition $\mathcal{X} = (X_1, \ldots, X_m, 0, \ldots, 0)$, $X_i \neq \emptyset$, is produced and adequate family $\mathcal{K}_{\mathcal{X}}$ from which labeled and unlabeled models are generated and counted by (3.16) and (3.18). A program implementation of this procedure in our system can be found at the given above address. Simply adding axioms for particular types of ordering, e.g. lattices, distributive lattices, etc. we construct and count labeled and unlabeled structures of this particular type as well.

5.3. Other examples. Semantics of our system lay in the first order predicate logic, hence in principle models of any class of finite structures described in this logic can be computed. The obvious limitation is the memory size and the hyper-exponential growth of the number of propositional variables appearing in the description of the related class of models. However, with a good choice of an adequate subclass of models and the ably reduction (killing) of variables we believe that new and interesting results in computational discrete mathematics can be obtained. Even if the aim of this paper is not to study the particular class of finite structures, we proposed a number of examples of this kind. These examples refer to ordered structures, automorphisms of structures and Latin squares (quasigroups). Examples of interest include computations of various types of lattices and a solution of Sudoku problem. In Sudoku problem appear 729 propositional variables, but our system solved it effortlessly by virtue of good elimination (killing) of variables. There are particular attempts for analysis and modeling classes of Latin squares in propositional calculus, eg [3], [5] and [11]. In contrast to our approach, their computation relies on Davis-Putnam algorithm. Our aim is to refine some of the ideas we have just outlined, particularly based on definability as presented in subsections 3.3, 3.4 and Example 5.2.

References

[1] S. Burris and H.P. Sankappanavar, *A course in Universal algebra*, Springer, 1981, 2012 Update.

[2] A. Dow, P. Nyikos, *Representing free Boolean algebras*, Fundamenta Mathematicae, **141**, (1992), 21–30.

[3] Maria Ercsey-Ravasz, Zoltan Toroczkai, *The Chaos Within Sudoku*, Scientific Reports 2, Article number: 725 doi:10.1038/srep00725, 2012.
http://www.nature.com/srep/2012/121011/srep00725/full/srep00725.html

[4] C. C. Chang, J. H. Keisler, *Model theory*, North Holland, (1990).

[5] Rhyd Lewis, *Metaheuristics can Solve Sudoku Puzzles*, Journal of Heuristics, Vol. 13, Issue 4, pp 387-401, 2007.

[6] K. Kuratowski, A. Mostowski, *Set Theory*, PWN, 1967.

[7] Ž. Mijajlović, *On free Boolean vectors*, Publ. Inst. Math, **64(78)**, 1998, 2–8.

[8] Ž. Mijajlović, D. Doder, A. Ilić-Stepić, *Borel sets and countable models*, Publ. Inst. Math, **90(104)**, (2011), 1–11.

[9] Ž. Mijajlović, *Model Theory*, Novi Sad, (1985).

[10] L. Svenonius *A theorem on permutations in models*, vol. 25, 173-178, 1959.

[11] Hantao. Zhang, Maria Paola Bonacina, Jieh Hsiang PSATO: *a Distributed Propositional Prover and its Application to Quasigroup Problems*, Jour. of Symbolic Computation, Vol. 21, Issues 46, 1996, 543560.

[12] R. Sikorski, *Boolean Algebras*, Springer-Verlag, Berlin, (1969).

Faculty of Mathematics, University of Belgrade, Belgrade, Serbia
E-mail address: zarkom@matf.bg.ac.rs

Institute of Mathematics, Serbian Academy of Science and Arts, Belgrade, Serbia
E-mail address: pejovica@mi.sanu.ac.rs