Acta Universitatis Sapientiae

Informatica

Volume 5, Number 2, 2013

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Contents

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 145–166

Many-valued logics—implications and semantic consequences

Katalin PÁSZTOR VARGA
Eötvös Loránd University
email: pkata@ludens.elte.hu

Gábor ALAGI*
Eötvös Loránd University
email: alagig@caesar.elte.hu

Magda VÁRTERÉSZ
Debrecen University
email:
varteresz.magda@inf.unideb.hu

Abstract.

In this paper an application of the well-known matrix method to an extension of the classical logic to many-valued logic is discussed: we consider an n-valued propositional logic as a propositional logic language with a logical matrix over n truth-values. The algebra of the logical matrix has operations expanding the operations of the classical propositional logic. Therefore we look over the Łukasiewicz, Post, Heyting and Rosser style expansions of the operations negation, conjunction, disjunction and with a special emphasis on implication.

In the frame of consequence operation, some notions of semantic consequence are examined. Then we continue with the decision problem and the logical calculi. We show that the cause of difficulties with the notions of semantic consequence is the weakness of the reviewed expansions of negation and implication. Finally, we introduce an approach to finding implications that preserve both the modus ponens and the deduction theorem with respect to our definitions of consequence.

Computing Classification System 1998: F.4.1
Mathematics Subject Classification 2010: 03B50
Key words and phrases: many-valued logic, extensions of implication, notions of consequence

*Current affiliation of the author: Max Planck Institute für Informatik, email: galagi@mpi-inf.mpg.de

1 Introduction

The construction of propositional logics can follow several methodological ways. The algebraic method is fundamental and also prior to any others. One such an algebraic tool for constructing a logic is the logical matrix method. We begin with a brief survey of related notions and notations. After defining the notion of consequence operation, we show that the semantic consequence for the classical propositional logic generates a consequence operation. Later, we outline the conventional axiomatic treatment of logic for a given logic language. Here, we prove that the usual derivation notion is also a consequence operation.

After this, we discuss the n-valued propositional logics ($n > 2$). It is desirable to obtain an algebraic structure close to a Boolean algebra by expansion of the classical logical matrix to n values. Here, we define two notions of semantic consequence, and prove that both are consequence operations. Because the usual expansion of conjunction is the minimum unanimously, and the expansion of disjunction is the maximum in the same way, we deal only with the Łukasiewicz, Post, Heyting and Rosser style expansions of implication.

Finally, we look for a suitable implication for a general consequence notion such that both the modus ponens and the deduction theorem remain valid.

2 Logical matrices

Let U be any nonempty set. A mapping $o : U^m \to U$, defined on the Cartesian product of m copies of U, with values in U, is called an m-argument (or an m-ary) operation in U (for $m = 0, 1, \ldots$). By an algebra we mean a pair $\langle U, (o_1, o_2, \ldots, o_k)\rangle$ ($k \geq 1$), where U is a (nonempty) set, called the universe of the algebra, and each o_j is an m_j-argument operation over U. A tuple $(m_1, m_2, \ldots, m_k)$ associated to the operations is called the signature of the algebra.

We consider an arbitrary logic language $L = \langle V, (c_1, c_2, \ldots, c_k), F\rangle$, where V is the set of propositional variables; $c_1, c_2, \ldots, c_k$ are logical connectives; F is the set of formulas generated by the variables and the connectives in the standard way. At the same time, the set F of the formulas can also be regarded as the universe of an algebra with concatenation operations induced by the connectives. If we can connect m_j formulas with the connective c_j, the induced operation has m_j arguments, and the signature of the algebra freely generated by V is $(m_1, m_2, \ldots, m_k)$. This algebra is a logic language algebra.

A logic system is semantically determined, if we have an interpretation no-

tion in the sense that every formula has some truth-value with respect to each such interpretation. A basic assumption in classical logics is the principle of compositionality: the truth-value of a compound formula is a function of the truth-values of its immediate subformulas (every formula represents a function into the set of truth-values). Hence the most essential semantical decision is the determination of the operations over the truth-value set which characterizes the connectives. Later, the algebraic structure of the truth-value set will play an important role.

Definition 1 [5] *By a logical matrix* M *for a logic language algebra* L *with a signature* $(m_1, m_2, \ldots, m_k)$ *we mean a triple*

$$\langle U, (o_1, o_2, \ldots, o_k), U^* \rangle,$$

where $\langle U, (o_1, o_2, \ldots, o_k)\rangle$ *is an algebra with the signature* $(m_1, m_2, \ldots, m_k)$*, and* U^* *is a nonempty subset of* U*.* U *is the set of truth-values, the elements of* U^* *are called designated truth-values.*

After this, we define the semantics as a correspondence between the set of connectives and operations using the signature. This is followed by an interpretation $I : V \to U$. The interpretation I can uniquely be extended to a homomorphism (called a valuation of formulas) from the set of formulas F to the universe U:

(a) $|v|_I = I(v)$ for $v \in V$;

(b) $|c_j(\alpha_1, \ldots, \alpha_{m_j})|_I = o_j(|\alpha_1|_I, \ldots, |\alpha_{m_j}|_I)$ for every m_j-ary connective c_j and for all $\alpha_1, \ldots, \alpha_{m_j} \in F$.

In every interpretation, a truth-value is assigned to a formula, depending on the truth-values assigned to the variables occurring in the formula. Thus, a formula expresses a truth-function $U^n \to U$ (n-variable operation over U). If we want to handle every potential truth-function with the logic language, then the set of operations in the logical matrix should be functionally complete. We say that a set of operations is functionally complete, when every truth-function $U^n \to U$ can be expressed by a formula using only the logical connectives corresponding to these operations.

Now, a notion of partial interpretation $Ip : V' \to U$ ($V' \subseteq V$) is convenient. If $V' = V$, the partial interpretation Ip is a (total) interpretation. And, if the domain of Ip contains all the variables occurring in a set X of formulas, then Ip is total with respect to X. Sometimes later, it is simpler to handle an

(partial) interpretation $\mathfrak{Ip}$ as a relation $\mathfrak{Ip} \subseteq V \times U$, where for all pair (v_1, u_1) and (v_2, u_2) in $\mathfrak{Ip}$, if $u_1 \neq u_2$, then $v_1 \neq v_2$. In this notation, we can formalize an extension of the partial interpetation $\mathfrak{Ip}$ to the variable $v \notin \mathrm{Dom}(\mathfrak{Ip})$ with $\mathfrak{Ip} \cup \{(v, u)\}$, where $u \in U$.

In order that a logic language and its matrix can become a logic system, the consequence notion and the decision problem are inevitable. In [7], Tarski developed an abstract theory of logical systems. He introduced a finitary closure operation on the sets of formulas, called consequence operation. Let $\mathcal{P}(F)$ denote the power set of F.

Definition 2 *The consequence operation* $\mathfrak{Cn} : \mathcal{P}(F) \to \mathcal{P}(F)$ *in* L *is an operation which satisfies the following conditions for any* $X, Y \subseteq F$ *and* $\alpha, \beta \in F$:

(1) $X \subseteq \mathfrak{Cn}(X) \subseteq F$;

(2) if $X \subseteq \mathfrak{Cn}(Y)$ *then* $\mathfrak{Cn}(X) \subseteq \mathfrak{Cn}(Y)$;

(3) if $\alpha \in \mathfrak{Cn}(X)$ *then there exists a finite set* Y *such that* $Y \subseteq X$ *and* $\alpha \in \mathfrak{Cn}(Y)$.

Note that $\mathfrak{Cn}(\mathfrak{Cn}(X)) \subseteq \mathfrak{Cn}(X)$ holds for every consequence operation, because $\mathfrak{Cn}(X) \subseteq \mathfrak{Cn}(X)$ and *(2)*.

Let α be a formula and let X be a set of formulas. The decision problem is to decide whether $\alpha \in \mathfrak{Cn}(X)$.

To sum it up, by a propositional logic we mean a quadruple

$$\langle L, M, \mathfrak{In}, \mathfrak{Pr} \rangle,$$

where L is a logic language algebra, M is a logical matrix for L, $\mathfrak{In}$ is the set of interpretations of L, $\mathfrak{Pr}$ is a consequence operation.

Example 3 *A classical two-valued propositional logic (CPL) is a quadruple*

$$\langle L, M, \mathfrak{In}, \mathfrak{Pr}' \rangle,$$

where

(a) L *is a language algebra* $\langle V, (\neg, \wedge, \vee), F \rangle$ *with signature* $(1, 2, 2)$.

(b) M *is a logical matrix* $\langle \{0, 1\}, (\neg', \wedge', \vee'), \{1\} \rangle$, *where the values* 0 *and* 1 *are truth-values, 1 stands for true, 0 stands for false. The operation* $\wedge'$ *is the classical conjunction (minimum),* $\vee'$ *is the classical disjunction*

(maximum), and $\neg'$ *is the classical negation. This operation set is functionally complete. (We remark, if we use the definition* $x \supset' y \rightleftharpoons \neg' x \vee' y$ *in* M*, the set* $\{\neg', \supset'\}$ *is also functionally complete.)*

The structure

$$\langle \{0,1\}, \{\neg', \wedge', \vee'\}, 0, 1 \rangle$$

yields a Boolean algebra. The set $\{0,1\}$ *is the universe of the Boolean algebra, the operations* $\wedge'$ *and* $\vee'$ *are lattice operations, the unary operation* $\neg'$ *is the complementation, and 1 is the unit, 0 is the zero element.*

(c) $\mathrm{In} = \{I \mid I : V \to \{0,1\}$ *is an interpretation of* $L\}$.

(d) Pr' *is the usual semantic consequence:* $\alpha \in \mathrm{Pr}'(X)$ *if and only if* $|\alpha|_I = 1$*, whenever* $|\beta|_I = 1$ *for every formula* β *in* X.

Next, we verify that Pr' is a consequence operation.

Proposition 4 Pr' *satisfies the conditions (1)-(3) in Definition 2.*

Proof.

(1) is obvious.

(2) Let In_X be the set of interpretations, where $|\beta|_I = 1$ for every formula β in X. If elements of X are consequences of Y, then $\mathrm{In}_Y \subseteq \mathrm{In}_X$. Whereas $\mathrm{In}_X \subseteq \mathrm{In}_{\mathrm{Pr}'(X)}$, thus $\mathrm{In}_Y \subseteq \mathrm{In}_{\mathrm{Pr}'(X)}$.

(3) If $\alpha \in \mathrm{Pr}'(X)$, then $\mathrm{In}_X \cap \mathrm{In}_{\neg\alpha} = \emptyset$. Because of compactness theorem in CPL, if $\mathrm{In}_X \cap \mathrm{In}_{\neg\alpha} = \emptyset$, then there exists a finite set Y such that $Y \subseteq X$ and $\mathrm{In}_Y \cap \mathrm{In}_{\neg\alpha} = \emptyset$ also. Thus, Y is a finite subset of X and $\alpha \in \mathrm{Pr}'(Y)$. □

3 Axiomatic treatment of logics

Another method to construct logics is the axiomatic (syntax-based) way. Let L be a logic language with the set F of formulas.

Definition 5 *A finite subset* A *of formulas is called an axiom system.*

Definition 6 *A rule over* F *is a nonempty relation*

$$r \subseteq \{(\alpha_1, \ldots, \alpha_m, \alpha) \mid \alpha_1, \ldots, \alpha_m, \alpha \in F\}.$$

Definition 7 *Let* A *be an axiom system,* R *a set of rules and* X *any set of formulas. A formula* α *is derived from* X *if there is a finite sequence of formulas* $\alpha_1, \ldots, \alpha_k$ *such that*

(1) $\alpha_k = \alpha$*, and*

(2) for each i $(1 \leq i \leq k)$*, either* $\alpha_i \in X \cup A$*, or there exist indices* $i_1, \ldots, i_l$ *smaller than* i *such that* $(\alpha_{i_1}, \ldots, \alpha_{i_l}, \alpha_i) \in r$ *for some rule* $r \in R$*.*

Proposition 8 $Pr^* : X \to \{\alpha \mid \alpha$ *is derived from* $X\}$ *satisfies conditions (1)-(3) in Definition 2.*

Proof.

(1) is obvious.

(3) can be seen easily. If $\alpha \in Pr^*(X)$, the derivation of α is a finite sequence of formulas. Let Y be the set of formulas of X occuring in this derivation. Clearly, α can be derived from Y, as well.

(2) If α can be derived from X, because of *(3)*, there is a finite $Z \subseteq X$ such that α can be derived from Z. But every element of Z can be derived from Y, i.e. from some finite subset of Y. If we concatenate the derivations of the elements of Z from Y and furthermore, we add the derivation of α from Z to it, then the result is a derivation of α from Y. Herewith, condition *(2)* holds. □

Informally, a propositional logic is axiomatically given by

$$\langle L, A, R, Pr^* \rangle ,$$

if its language algebra L is specified, an axiom system A is fixed, a finite set R of derivation rules is specified and Pr^* is the consequence operation.

An axiomatically given propositional logic (calculus) $\langle L, A, R, Pr^* \rangle$ is said to be (strongly) adequate for a propositional logic $\langle L, M, In, Pr \rangle$ if their consequence operations are the same.

Example 9 *By a classical propositional calculus we mean a quadruple*

$$\langle L^*, A, R, Pr^* \rangle ,$$

where

(a) L^* *is the free language algebra* $\langle V, (\neg, \supset), F\rangle$ *with signature* $(1, 2)$*;*

(b) the axiom system A *consists of the axioms*

$$\{\alpha \supset (\beta \supset \alpha),\ (\alpha \supset (\beta \supset \gamma)) \supset ((\alpha \supset \beta) \supset (\alpha \supset \gamma)),$$
$$(\neg\alpha \supset \beta) \supset ((\neg\alpha \supset \neg\beta) \supset \alpha)\};$$

(c) the set R *contains the single derivation rule*

$$\frac{\alpha,\ \alpha \supset \beta}{\beta};$$

(d) and $Pr^* : X \to \{\alpha \mid \alpha$ *is derived from* $X\}$ *is the consequence operation.*

The classical propositional calculus $\langle L^*, A, R, Pr^*\rangle$ *is adequate for the classical propositional logic* $\langle L^*, M^*, In, Pr'\rangle$*, where* M^* *is a logical matrix for* L^**.*

4 Propositional many-valued logics

By the literature [1], [2] and [3], a non-classical logic may be an extended logic and/or a deviant logic. "Extended logics expand classical logic by additional logical constructs. For example, in modal logic modal operators are added to classical logic to express modal notions. In contrast, deviant logics are rivals to classical logic that give up some classical principles. In many-valued logics, we allow for many truth-values instead of two truth-values (we give up the principle of bivalence)". This deviation leads to the extension of the operations of the classical two-valued logic. An operation is extended if, whenever the arguments are classical truth-values, the result has the same truth-value as it does in classical logic. "In this sense, classical logic can be thought of as a special case of many-valued logic."

Let U_n be a set of truth-values $\{0, 1, 2, \ldots, n-1\}$ $(n \geq 2)$. Formally, we can define a propositional many-valued logic (MVPL) as a quadruple

$$\langle L, M, In, Pr^n\rangle,$$

where

(a) $L = \langle V, Con, F\rangle$ is a language algebra with a signature σ.

(b) $M = \langle U_n, Op, U_n^*\rangle$ is a logical matrix for L, where $\langle U_n, Op\rangle$ is an algebra over U_n with the signature σ, as well. Moreover, let $S \in U_n$. Then $U_n^* = \{S+1, \ldots, n-1\}$ is the set of the designated truth-values, and $0, 1, \ldots, S$ are called non-designated ones.

(c) $\mathrm{In} = \{I \mid I : V \to U_n$ is an interpretation of $L\}$.

(d) $\mathrm{Pr}^n : \mathcal{P}(F) \to \mathcal{P}(F)$ should be a consequence operation.

Now, we look for a consequence operation.

Let $L = \langle V, Con, F\rangle$ be a language algebra and let $M = \langle U_n, Op, U_n^*\rangle$ be a logical matrix for the language L.

Definition 10 *A formula α is a weak semantic consequence of a set X of formulas, denoted as*

$$X \models_S \alpha,$$

if for any interpretation in which the truth-value of every formula $\beta \in X$ is designated, the truth-value of α is also designated. If X is the empty set, we have no constraint for the interpretations. Thus, α is said to be an S-tautology ($\emptyset \models_S \alpha$) if the truth-value of α is designated for every interpretation.

We can give a more rigorous notion of the consequence relation if we also take the extent of truth-values of formulas into consideration.

Definition 11 *A formula α is a strong semantic consequence of a set X of formulas, denoted as*

$$X \models_{S^*} \alpha,$$

if for any interpretation in which the truth-value of every formula $\beta \in X$ is designated, the truth-value of α is also designated with at least the same truth-value as the minimum of the truth-values of formulas in X in the underlying interpretation.

We need some further notions and a lemma to discuss simply the characteristic of the consequence relation.

Definition 12 *Let a partial interpretation* Ip *be a total interpretation with respect to the set $X \cup \{\alpha\}$ of formulas. X is appropriate for α in* Ip *if the truth-value of α is not less than the minimum of the truth-values of formulas in X, whenever this minimum is designated.*

Definition 13 *X is finitely bad for α with respect to a partial interpretation* Ip *if for all finite subsets Y of X there exists an extension of* Ip *in which Y is not appropriate for α.*

Lemma 14 *If* X *is finitely bad for* α *with respect to a partial interpretation* Ip *and the variable* ν *has no value in* Ip *yet, then there is some* $j \in U_n$ *such that* X *is also finitely bad for* α *with respect to the partial interpretation* $Ip \cup \{(\nu, j)\}$.

Proof. Otherwise, X is not finitely bad for α with respect to any partial interpretation $Ip \cup \{(\nu, i)\}$ $(i \in U_n)$. So for all i, a finite subset Y_i of X would exist such that Y_i would be appropriate for α in all of the total extensions of $Ip \cup \{(\nu, i)\}$. Thus, $\cup_{i=0}^{n-1} Y_i$ is a finite set and appropriate for α in all of the total extensions of Ip. It means that X is not finitely bad for α with respect to a partial interpretation Ip. It is a contradiction. □

Proposition 15 $Pr_S^n : X \to \{\alpha \mid X \models_S \alpha\}$ *and* $Pr_{S^*}^n : X \to \{\alpha \mid X \models_{S^*} \alpha\}$ *are consequence operations.*

Proof.

(1) is obvious.

(2) For every interpretation I', where $q = \min_{\alpha \in X} |\alpha|_{I'} > S$, $|\gamma|_{I'} \geq q$ holds for any $\gamma \in Pr_S^n(X)$. Now, let I be an interpretation, where $|\beta|_I > S$ for every formula β in Y, and let p be $\min_{\beta \in Y} |\beta|_I$. According to condition $X \subseteq Pr_S^n(Y)$, we get $|\alpha|_I \geq p > S$ for every $\alpha \in X$. Thus, I is an interpretation, where $|\gamma|_I \geq p$ holds for all $\gamma \in Pr_S^n(X)$. It means, we have $Pr_S^n(X) \subseteq Pr_S^n(Y)$.

(3) Now, let α be a strong consequence of X. Then α is also a weak consequence of X. Let us define a special kind of negation:

$$\neg x \rightleftharpoons \begin{cases} 0 & \text{if } x \in U_n^* \,, \\ (n-1) & \text{otherwise} \end{cases}$$

Moreover, let In_X contain all the interpretations in which every formula in X has a designated truth-value.

It is clear that $X \models_S \alpha$ if and only if $In_X \cap In_{\neg\alpha} = \emptyset$. Because of the compactness theorem in MVPL (see in [4]), if $In_X \cap In_{\neg\alpha} = \emptyset$, then there exists a finite set Y_0 such that $Y_0 \subseteq X$ and $In_{Y_0} \cap In_{\neg\alpha} = \emptyset$.

Thus, Y_0 is a finite subset of X and $Y_0 \models_S \alpha$. It means, that for any interpretation in which the truth-value of every formula in Y_0 is designated, the truth-value of α is also designated. At the same time, the truth-value of α may be less than the minimum of the truth-values of formulas in Y_0 in several (however finite number of) interpretations.

Now, let the partial interpretation Ip be total with respect to $Y_0 \cup \{\alpha\}$ such that Y_0 is not appropriate for α in Ip. We prove that there is a finite subset Y of X such that $Y_0 \cup Y$ is appropriate for α in all of the total extensions of Ip.

Let $\nu_1, \nu_2, \ldots$ be a list of variables not occuring in Ip. Assume that the opposite of what we are trying to prove is true: X is finitely bad for α with respect to Ip.

In view of Lemma 14, for all k there is some $j_k \in U_n$ such that X is also finitely bad for α with respect to $Ip_k = Ip_{k-1} \cup \{(\nu_k, j_k)\}$. In the total interpretation $\cup_{k=1}^{\infty} Ip_k$, there is an index k for all $\gamma \in X$ such that Ip_k is total with respect to γ. Since X is finitely bad for α with respect to Ip_k, $|\gamma|_{Ip_k} > |\alpha|_{Ip_k}$. It means that X is not appropriate for α in the interpretation $\cup_{k=1}^{\infty} Ip_k$, so α is not a strong consequence of X, a contradiction.

Our indirect assumption is false, so there is a finite subset Y of X such that $Y_0 \cup Y$ is appropriate for α in all extensions of Ip.

To sum it up, if we have some interpretations, in which the truth-value of α is less than the minimum of the truth-values of formulas in Y_0, when it is designated, then we have no more than finitely many such interpretations. For every such interpretation, there exists a finite subset Y of X such that $Y_0 \cup Y$ is appropriate for α in all extensions of the interpretation. Adding the union of finite number of the finite subsets to Y_0 we get a finite set and in every interpretation, if the minimum of the truth-values of formulas in this set is designated, the minimum is not greater than the truth-value of α. □

5 Problems with n-valued operations

In the classical logic, the consequence notion leads to the decision problem through the deduction theorem. The deduction theorem requires the classical syllogism, modus ponens.

In a many-valued logic with the weak consequence relation, the modus ponens is valid if we have an operation $\supset$ with $\alpha \supset \beta, \alpha \models_S \beta$, i.e. if $\alpha \supset \beta$ and α have designated values, then β has a designated value, too.

The Łukasiewicz implication is defined by

$$x_1 \supset_L x_2 \rightleftharpoons \begin{cases} n-1 & \text{if } x_1 \leq x_2, \\ (n-1) - x_1 + x_2 & \text{if } x_1 > x_2, \end{cases}$$

or by Table 1. Designated values are marked by an asterisk.

$\supset_L$	0	1	$\cdots$	S	$S+1^*$	$\cdots$	$n-3^*$	$n-2^*$	$n-1^*$
0	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
1	$n-2$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
2	$n-3$	$n-2$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$S-1$	$n-S$	$n-S+1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
S	$n-S-1$	$n-S$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$S+1^*$	$n-S-2$	$n-S-1$	$\cdots$	$n-2$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$n-3^*$	2	3	$\cdots$	$S+2$	$S+3$	$\cdots$	$n-1$	$n-1$	$n-1$
$n-2^*$	1	2	$\cdots$	$S+1$	$S+2$	$\cdots$	$n-2$	$n-1$	$n-1$
$n-1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$

Table 1: Łukasiewicz implication

We can see that if $S < n-2$, then $S+1 \supset_L S$ and $S+1$ are designated, but S is not. The modus ponens is not valid in such many-valued logics and moreover, it is not valid when the consequence relation is the second one.

The Post implication is defined by

$$x_1 \supset_P x_2 \rightleftharpoons \begin{cases} n-1 & \text{if } x_1 \leq x_2, \\ x_2 & \text{if } x_1 > x_2,\ x_1 > S, \\ (n-1) - x_1 + x_2 & \text{if } x_1 > x_2,\ x_1 \leq S, \end{cases}$$

or by Table 2.

The modus ponens is valid in the case of Post implication:

Proposition 16

$$\alpha \supset_P \beta, \alpha \models_{S^*} \beta.$$

Proof. If $|\alpha \supset_P \beta| > S$ and $|\alpha| > S$ in an interpretation, then either $|\alpha| > |\beta|$ or $|\alpha| \leq |\beta|$. In the first case, $S < |\alpha \supset_P \beta| = |\beta|$ and $\min(|\alpha|, |\alpha \supset_P \beta|) = |\beta|$. In the second case, $|\alpha \supset_P \beta| = n-1$, so $\min(|\alpha|, |\alpha \supset_P \beta|) = |\alpha| \leq |\beta|$. □

Now, we must verify whether the deduction theorem is valid. The deduction theorem would state that $X, \alpha \models_S \beta$ if and only if $X \models_S \alpha \supset_P \beta$. It is easy to realize, this theorem is not valid: if $X, \alpha \models_S \beta$, $X \models_S \alpha \supset_P \beta$ does not necessarily follow.

Actually, let $n-1 \leq 2S$ and $\gamma, \alpha \models_S \beta$. There is no constraint for the truth-value of β in the interpretations where α is not designated. So $|\gamma| =$

$\supset_P$	0	1	$\cdots$	S	$S+1^*$	$\cdots$	$n-3^*$	$n-2^*$	$n-1^*$
0	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
1	$n-2$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
2	$n-3$	$n-2$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\ldots$			$\ldots$			
$S-1$	$n-S$	$n-S+1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
S	$n-S-1$	$n-S$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$S+1^*$	0	1	$\cdots$	S	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\ldots$			$\ldots$			
$n-3^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-1$	$n-1$	$n-1$
$n-2^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-1$	$n-1$
$n-1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$

Table 2: Post implication

$n-1$, $|\alpha| = S$ and $|\beta| = 0$ might hold in an interpretation. In this case γ is designated, but $|\alpha \supset_P \beta| = (n-1) - S \leq S$ is not. Thus, $\gamma \models_S \alpha \supset_P \beta$ is not valid.

$\supset_H$	0	1	$\cdots$	S	$S+1^*$	$\cdots$	$n-3^*$	$n-2^*$	$n-1^*$
0	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
1	0	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
2	0	1	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\ldots$			$\ldots$			
$S-1$	0	1	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
S	0	1	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$S+1^*$	0	1	$\cdots$	S	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\ldots$			$\ldots$			
$n-3^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-1$	$n-1$	$n-1$
$n-2^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-1$	$n-1$
$n-1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$

Table 3: Heyting implication

The Heyting implication is often used in a many-valued logic: $x_1 \supset_H x_2$ is the greatest element in U_n such that $x_1 \wedge (x_1 \supset_H x_2) \leq x_2$, that is for every $x_1, x_2 \in U_n$,

$$x_1 \supset_H x_2 \rightleftharpoons \begin{cases} n-1 & \text{if } x_1 \leq x_2, \\ x_2 & \text{if } x_1 > x_2, \end{cases}$$

or see Table 3.

Proposition 17

$$\alpha \supset_H \beta, \alpha \models_{S^*} \beta.$$

Proof. If $|\alpha \supset_H \beta| > S$ and $|\alpha| > S$ hold in an interpretation, then by the definition of the Heyting implication

(1) if $|\alpha| > |\beta|$, $|\alpha \supset_H \beta| = |\beta|$, thus $|\beta| > S$ and $\min(|\alpha \supset_H \beta|, |\alpha|) = |\beta|$, and

(2) if $|\alpha| \leq |\beta|$, thus $|\beta| > S$ and, because $|\alpha \supset_H \beta| = n-1$, thus $\min(|\alpha \supset_H \beta|, |\alpha|) = |\alpha| \leq |\beta|$.

□

It is easy to see, the deduction theorem is not valid: if $X, \alpha \models_S \beta$, it does not necessarily follow that $X \models_S \alpha \supset_H \beta$.

Actually, let $\gamma, \alpha \models_S \beta$. There is no constraint for the truth-value of β in the interpretations where α is not designated. So it can happen that $|\gamma| = n-1$, $|\alpha| = S$ and $|\beta| = 0$ hold in an interpretation. In this case γ is designated, but $|\alpha \supset_H \beta| = 0$ is not. So, $\gamma \models_S \alpha \supset_H \beta$ is not valid.

$\supset_R$	0	1	$\cdots$	S	$S+1^*$	$\cdots$	$n-3^*$	$n-2^*$	$n-1^*$
0	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
1	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
2	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$S-1$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
S	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$S+1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$n-3^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$
$n-2^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$
$n-1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$

Table 4: Rosser implication

In [6], another implication has been used by Rosser:

$$x_1 \supset_R x_2 \rightleftharpoons \begin{cases} n-1 & \text{if } x_1 \leq S, \\ x_2 & \text{if } x_1 > S. \end{cases}$$

Table 4 shows the truth-table of this implication. In this paper we name this implication after Rosser.

Proposition 18

$$\alpha \supset_R \beta, \alpha \models_{S^*} \beta.$$

Proof. If $|\alpha \supset_R \beta| > S$ and $|\alpha| > S$ hold in an interpretation, then by the definition of the Rosser implication $|\alpha \supset_R \beta| = |\beta|$, thus $|\beta| > S$ and $\min(|\alpha \supset_R \beta|, |\alpha|) \leq |\beta|$. □

Proposition 19 *If* $X, \alpha \models_S \beta$, *then* $X \models_S \alpha \supset_R \beta$.

Proof. Suppose $X, \alpha \models_S \beta$. In every interpretation where every formula from X is designated, either α is also deignated or not. In the first case, according to the condition, β is designated, and because $|\alpha \supset_R \beta| = |\beta|$, $\alpha \supset_R \beta$ is designated, too. In the second case, according to the definition of the Rosser implication, we have $|\alpha \supset_R \beta| = n - 1$. This is a designated value. Therefore, $X \models \alpha \supset_R \beta$. □

We can not prove that if $\gamma, \alpha \models_{S^*} \beta$, then $\gamma \models_{S^*} \alpha \supset_R \beta$. If $|\alpha| = |\beta| = S + 1$ and $|\gamma| = n - 1$ in an interpretation, then $|\alpha \supset_R \beta| = |\beta| = S + 1$, thus $\alpha \supset_R \beta$ is designated, but if $S + 1 < n - 1$, then $|\gamma| > |\alpha \supset_R \beta|$.

Proposition 20 *If* $X \models_{S^*} \alpha \supset_R \beta$, *then* $X, \alpha \models_{S^*} \beta$.

Proof. Let I be an interpretation in which every formula from X and α are designated. According to the condition, $\alpha \supset_R \beta$ is designated with truth-value at least $\min_{\gamma \in X}\{|\gamma|\}$. If α is designated, $|\beta| = |\alpha \supset_R \beta|$, thus β is also designated with truth-value at least $\min_{\gamma \in X}\{|\gamma|\} \geq \min_{\gamma \in X}\{|\gamma|, |\alpha|\}$. □

Finally, let $f : U \times U \to U$ such that $f(x_1, x_2) \leq S$ for all $x_1, x_2 \in U$, when $x_1 > S$ and $x_2 \leq S$. Then the implication defined below admits the modus ponens and the deduction theorem:

$$x_1 \supset_*^f x_2 \rightleftharpoons \begin{cases} n - 1 & \text{if } x_1 \leq S \text{ or } x_1 \leq x_2, \\ x_2 & \text{if } x_1 > x_2 > S, \\ f(x_1, x_2) & \text{otherwise.} \end{cases}$$

Proposition 21

$$\alpha \supset_*^f \beta, \alpha \models_{S^*} \beta.$$

$\supset_*^f$	0	1	$\cdots$	S	$S+1^*$	$\cdots$	$n-3^*$	$n-2^*$	$n-1^*$
0	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
1	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
2	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$S-1$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
S	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$S+1^*$	0	1	$\cdots$	S	$n-1$	$\cdots$	$n-1$	$n-1$	$n-1$
$\vdots$			$\cdots$			$\cdots$			
$n-3^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-1$	$n-1$	$n-1$
$n-2^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-1$	$n-1$
$n-1^*$	0	1	$\cdots$	S	$S+1$	$\cdots$	$n-3$	$n-2$	$n-1$

Table 5: The implication with $f(x_1, x_2) = x_2$

Proof. If $|\alpha \supset_*^f \beta| > S$ and $|\alpha| > S$ in an interpretation, then by the definition of the new implication either $|\alpha \supset_*^f \beta| = n-1$, or $|\alpha \supset_*^f \beta| = |\beta|$. In the first case $|\beta| \geq |\alpha| > S$ and $\min(|\alpha \supset_*^f \beta|, |\alpha|) \leq |\beta|$. In the second case $|\alpha| > |\beta| > S$ and $\min(|\alpha \supset_*^f \beta|, |\alpha|) = |\beta|$. □

Proposition 22 *If* $X, \alpha \models_{S^*} \beta$, *then* $X \models_{S^*} \alpha \supset_*^f \beta$.

Proof. Suppose $X, \alpha \models_S \beta$. In every interpretation where every formula from X is designated, either α is also deignated or not. In the first case, according to the condition, β is designated, and

(1) either $S < |\alpha| \leq |\beta|$ and $|\alpha \supset_*^f \beta| = n-1$, so $\min_{\gamma \in X}\{|\gamma|\} \leq |\alpha \supset_*^f \beta|$;

(2) or $|\alpha| > |\beta| > S$ and $|\alpha \supset_*^f \beta| = |\beta|$, thus $\alpha \supset_*^f \beta$ is also designated moreover, $\min_{\gamma \in X}\{|\gamma|\} \leq |\beta| = |\alpha \supset_*^f \beta|$ because $\min_{\gamma \in X}\{|\gamma|, |\alpha|\} \leq |\beta| < |\alpha|$.

In the second case, $|\alpha \supset_*^f \beta| = n-1$, which is a designated value. Therefore, $X \models_{S^*} \alpha \supset_*^f \beta$. □

Proposition 23 *If* $X \models_{S^*} \alpha \supset_*^f \beta$, *then* $X, \alpha \models_{S^*} \beta$.

Proof. Let I be an interpretation in which every formula from X and α are designated. According to the condition, $\alpha \supset_*^f \beta$ is designated with truth-value

at least $\min_{\gamma\in X}\{|\gamma|\}$. Because α is designated, if $|\alpha| \leq |\beta|$, then $|\beta|$ is designated with truth-value at least $\min_{\gamma\in X}\{|\gamma|, |\alpha|\}$. In the case $|\alpha| > |\beta|$, $|\alpha \supset^{f}_{*} \beta| = |\beta|$, thus β is designated with truth-value at least

$$\min_{\gamma\in X}\{|\gamma|\} \geq \min_{\gamma\in X}\{|\gamma|, |\alpha|\},$$

as well. □

Finally, all what was proved about the examined implications at this section we summarized in the following table:

	$\supset_L$	$\supset_P$	$\supset_H$	$\supset_R$	$\supset^{f}_{*}$
modus ponens	-	+	+	+	+
deduction theorem with $\models_S$	-	-	-	+	+
deduction theorem with $\models_{S^*}$	-	-	-	-	+

6 Suitable implication for a given consequence

It is desirable that both the modus ponens and the deduction theorem hold with respect to the underlying consequence. Now, we look for a suitable implication for a generally given consequence notion such that both the modus ponens and the deduction theorem are valid.

Now, let $\psi : U \times U \to \{0, 1\}$ be an arbitrary classical truth-valued function with the following properties:

(a) $\psi(x, x) = 1$ for all $x \in U$,

(b) if $\psi(x, y) \wedge \psi(y, z) = 1$, then $\psi(x, z) = 1$ for all $x, y, z \in U$.

Then, define the consequence as below:

Definition 24 *A formula α is a formal semantic consequence of a set X of formulas, denoted as $X \models \alpha$, if*

$$\bigvee_{\gamma\in X} \psi(|\gamma|_I, |\alpha|_I) = 1 \text{ for any interpretation } I,$$

where $\bigvee_{\gamma\in X} \psi(|\gamma|_I, |\alpha|_I)$ denotes the supremum of $\{\psi(|\gamma|_I, |\alpha|_I) \mid \gamma \in X\}$.

Proposition 25 $\mathsf{Pr} : X \to \{\alpha \mid X \models \alpha\}$ *satisfies conditions (1)-(3) in Definition 2.*

Proof.

(1) Now to prove the condition (1), let $\alpha \in X$. Since in any interpretation $\psi(|\alpha|, |\alpha|) = 1$, therefore $\bigvee_{\gamma \in X} \psi(|\gamma|, |\alpha|) = 1$, so $X \models \alpha$. It means that $X \subseteq \mathsf{Pr}(X)$.

(2) Next, let $X \subseteq \mathsf{Pr}(Y)$ for some $X, Y \subseteq F$. We show that $\mathsf{Pr}(X) \subseteq \mathsf{Pr}(Y)$.

- For any $\alpha \in \mathsf{Pr}(X)$, since
$$\bigvee_{\gamma \in X} \psi(|\gamma|_I, |\alpha|_I) = 1, \text{ so } \bigvee_{\gamma \in \mathsf{Pr}(Y)} \psi(|\gamma|_I, |\alpha|_I) = 1.$$
It means $\mathsf{Pr}(X) \subseteq \mathsf{Pr}(\mathsf{Pr}(Y))$.
- Now, we show that $\mathsf{Pr}(\mathsf{Pr}(Y)) = \mathsf{Pr}(Y)$. Since $\mathsf{Pr}(Y) \subseteq \mathsf{Pr}(\mathsf{Pr}(Y))$ by the property (1), it is enough to prove, that $\alpha \in \mathsf{Pr}(Y)$ for all $\alpha \in \mathsf{Pr}(\mathsf{Pr}(Y))$.
Obviously $Y \subseteq \mathsf{Pr}(Y)$. Let Y' denote the set $\mathsf{Pr}(Y) \setminus Y$ and let $\alpha \in \mathsf{Pr}(\mathsf{Pr}(Y))$. Then
$$\bigvee_{\gamma \in \mathsf{Pr}(Y)} \psi(|\gamma|_I, |\alpha|_I) = \bigvee_{\gamma \in Y' \cup Y} \psi(|\gamma|_I, |\alpha|_I) = 1.$$
If
$$\bigvee_{\gamma \in Y'} \psi(|\gamma|_I, |\alpha|_I) = 0, \text{ then } \bigvee_{\gamma \in Y} \psi(|\gamma|_I, |\alpha|_I) = 1.$$
And if
$$\bigvee_{\gamma \in Y'} \psi(|\gamma|_I, |\alpha|_I) = 1,$$
then there exists a $\gamma' \in Y'$ for which $\psi(|\gamma'|_I, |\alpha|_I) = 1$. But $\gamma' \in \mathsf{Pr}(Y)$ also holds, thus
$$\bigvee_{\gamma \in Y} \psi(|\gamma|_I, |\gamma'|_I) = 1$$
must hold, i.e. there exists a $\gamma'' \in Y$ for which $\psi(|\gamma''|_I, |\gamma'|_I) = 1$. Using the property (b) of ψ we get $\psi(|\gamma''|_I, |\alpha|_I) = 1$, that is
$$\bigvee_{\gamma \in Y} \psi(|\gamma|_I, |\alpha|_I) = 1.$$
Thus in both cases, we get $\alpha \in \mathsf{Pr}(Y)$.

– Since $X \subseteq \mathrm{Pr}(Y)$, thus $\mathrm{Pr}(X) \subseteq \mathrm{Pr}(\mathrm{Pr}(Y))$, and thereby $\mathrm{Pr}(X) \subseteq \mathrm{Pr}(Y)$ must hold.

(3) We prove compactness by reducing the problem to the compactness of first-order logic with equality. First, let us define the language of our encoding:

– We have a single binary predicate symbol $\hat{\psi}$.
– For each many-valued operation o, we have a corresponding function symbol $\hat{o}$ with the same arity.
– For each variable v, we have a corresponding constant c_v.
– For each truth-value u, we have an additional constant $\hat{u}$.

Given this language, we might fix the interpretation of our symbols by defining a set Σ of the following axioms:

(i) $\forall x(x = \hat{u}_1 \vee x = \hat{u}_2 \vee \cdots \vee x = \hat{u}_n)$ if $U = \{u_1, u_2, \ldots, u_n\}$
(ii) $\hat{u} \neq \hat{u'}$ for each $u, u' \in U$ with $u \neq u'$
(iii) $\hat{\psi}(\hat{u}, \hat{u'})$ if $\psi(u, u') = 1$ and $u, u' \in U$
(iv) $\neg\hat{\psi}(\hat{u}, \hat{u'})$ if $\psi(u, u') = 0$ and $u, u' \in U$
(v) $\hat{o}(\hat{a_1}, \hat{a_2}, \ldots, \hat{a_k}) = \hat{u}$ if o is an operator with arity k, $a_1, a_2, \ldots, a_k, u \in U$, and $o(a_1, a_2, \ldots, a_k) = u$

Since U is finite, Σ is a finite set of first-order formulas as well. It is easy to see that if $\hat{I}$ is a first-order model of Σ, then there is a corresponding many-valued interpretation I which assigns the same values to variables as did $\hat{I}$ to the corresponding constants.

Let $\hat{\alpha}$ denote the encoding of a formula α in this language, i.e. the first-order formula we get from α by substituting each symbol with the corresponding first-order symbol. By our definitions, if I and $\hat{I}$ are corresponding many-valued and first-order interpretations, $|\alpha|_I = u$ if and only if $|\hat{\alpha}|_{\hat{I}} = \hat{u}$. Thus, for each α, β, we have $\psi(|\alpha|_I, |\beta|_I)$ holds if and only if $\hat{\psi}(\hat{\alpha}, \hat{\beta})$ holds in $\hat{I}$.

Now, by our assumptions, $X \models \alpha$ if and only if $\bigvee_{\gamma \in X} \psi(|\gamma|_I, |\alpha|_I)$ holds for all interpretation I. This, on the other hand, holds if and only if the set $\Gamma = \{\neg\, \psi(|\gamma|_I, |\alpha|_I) \mid \gamma \in X\}$ is not satisfied under any interpretation I. Consider the first-order set

$$\hat{\Gamma} = \Sigma \cup \{\neg\, \hat{\psi}(\hat{\gamma}, \hat{\alpha}) \mid \gamma \in X\}$$

From our considerations it follows that $\hat{\Gamma}$ is unsatisfiable if and only if the original Γ is unsatisfiable.

Then, by the compactness of first-order logic, we know that there is a finite $\hat{\Gamma}' \subseteq \hat{\Gamma}$ such that $\hat{\Gamma}'$ is unsatisfiable. Since Σ is finite, we might assume $\Sigma \subseteq \hat{\Gamma}'$. Now, let X' the finite set $\{\gamma \in X \mid \neg\hat{\psi}(\hat{\gamma}, \hat{\alpha}) \in \hat{\Gamma}'\}$.

We know that the corresponding set $\Gamma' = \{\neg\psi(|\gamma|_I, |\alpha|_I) \mid \gamma \in X'\}$ is not satisfied by any I either. Therefore, $X' \models \alpha$ must hold where X' is a finite subset of X. □

Proposition 26 *Let $\supset$ be an implication operation over U. If*

$$\psi(x_1, x_2) \vee \psi(y, x_2) = \psi(y, x_1 \supset x_2)$$

for all $x_1, x_2, y \in U$, then $\supset$ admits the modus ponens and the deduction theorem.

Proof. First, we prove the modus ponens, i.e. we show, that $\{\alpha, \alpha \supset \beta\} \models \beta$ holds for any formulas α, β. For all $\alpha, \beta \in F$ and for all $I \in \mathfrak{In}$ we get

$$\psi(|\alpha|_I, |\beta|_I) \vee \psi(|\alpha \supset \beta|_I, |\beta|_I).$$

For all $x_1, x_2 \in U$, by applying the proposed equality

$$\psi(x_1, x_2) \vee \psi(x_1 \supset x_2, x_2) = \psi(x_1 \supset x_2, x_1 \supset x_2) = 1.$$

To prove the deduction theorem, we have to show for any α, β, X

$$X, \alpha \models \beta \text{ if and only if } X \models \alpha \supset \beta.$$

Again, applying our assumptions to both sides, we get for all $I \in \mathfrak{In}$

$$\bigvee_{\gamma \in X} \psi(|\gamma|_I, |\beta|_I) \vee \psi(|\alpha|_I, |\beta|_I) = \bigvee_{\gamma \in X} (\psi(|\gamma|_I, |\beta|_I) \vee \psi(|\alpha|_I, |\beta|_I))$$

if and only if for all $I \in \mathfrak{In}$

$$\bigvee_{\gamma \in X} \psi(|\gamma|_I, |\alpha|_I \supset |\beta|_I).$$

From our assumption with $y = |\gamma|_I, x_1 = |\alpha|_I, x_2 = |\beta|_I$, we get

$$\psi(|\gamma|_I, |\beta|_I) \vee \psi(|\alpha|_I, |\beta|_I) = \psi(|\gamma|_I, |\alpha|_I \supset |\beta|_I),$$

from which the desired equivalence immediately follows. □

In the remaining part of the section we apply this proposition to the earlier defined semantic consequences.

Example 27 *By Definition 10,*

$$X \models_S \alpha \text{ if and only if } \min_{\gamma \in X}\{|\gamma|_I\} \leq S \vee S < |\alpha|_I \text{ for all } I \in In.$$

Thus, for this case we get $\psi(x,y) = (x \leq S \vee S < y)$. *To find a suitable implication, it is enough to satisfy*

$$(x_1 \leq S \vee S < x_2) \vee (y \leq S \vee S < x_2) \text{ if and only if } (y \leq S \vee S < x_1 \supset x_2)$$

for all $x_1, x_2, y \in U$. *Let* $f, h : U \times U \to U$ *such that for all* $x_1 > S$ *and* $x_2 \leq S$ $f(x_1, x_2) \leq S$ *and if* $x_1 \leq S$ *or* $x_2 > S$, *then* $h(x_1, x_2) > S$. *Then, as we have seen above, the implication defined below admits the modus ponens and the deduction theorem:*

$$x_1 \supset_*^{f,h} x_2 \rightleftharpoons \begin{cases} h(x_1, x_2) & \text{if } x_1 \leq S \text{ or } x_2 > S, \\ f(x_1, x_2) & \text{otherwise.} \end{cases}$$

Example 28 *By Definition 11,*

$$X \models_{S^*} \alpha \text{ if and only if } \min_{\gamma \in X}\{|\gamma|_I\} \leq S \vee \min_{\gamma \in X}\{|\gamma|_I\} \leq |\alpha|_I \text{ for all } I \in In.$$

For this case we get $\psi(x,y) = x \leq S \vee x \leq y$. *Thus to find a suitable implication, it is enough to satisfy*

$$(x_1 \leq S \vee x_1 \leq x_2) \vee (y \leq S \vee y \leq x_2) \text{ if and only if } (y \leq x_1 \supset x_2 \vee y \leq S)$$

for all $x_1, x_2, y \in U$. *The possible values for* $x_1 \supset x_2$ *might be deduced as follows:*

- $x_1 \leq S \vee x_1 \leq x_2$*: since the right side must also hold, even for* $y = n-1$, *we get* $x_1 \supset x_2 = n-1$, *which is indeed a good choice.*
- $x_1 \geq x_2 > S$*: for* $y = x_2$ *we get* $x_2 \leq x_1 \supset x_2$, *and for* $y = x_2 + 1$ $x_1 \supset x_2 < x_2 + 1$. *Thus only* $x_1 \supset x_2 = x_2$ *is possible, and it indeed satisfies the equality in this case.*
- $x_1 > S \geq x_2$*: for* $y > S$ *we get* $x_1 \supset x_2 \leq S$. *In this case any value smaller than* S *satisfies the equality.*

Let $f : U \times U \to U$ *be such that for all* $x_1 > S$ *and* $x_2 \leq S$ $f(x_1, x_2) \leq S$. *Then, as we have seen above, the implication defined below admits the modus ponens and the deduction theorem:*

$$x_1 \supset_*^{f} x_2 \rightleftharpoons \begin{cases} n-1 & \text{if } x_1 \leq S \text{ or } x_1 \leq x_2, \\ x_2 & \text{if } x_1 > x_2 > S, \\ f(x_1, x_2) & \text{otherwise.} \end{cases}$$

7 Summary

In this paper we demonstrated that both semantic and syntactic consequences of classical logic generate consequence operators. We proved similar propositions about the weak and strong semantic consequences in the many-valued logic. After this, we investigated the Lukasiewicz, Post, Heyting and Rosser style many-valued implications whether the modus ponens rule and the deduction theorem are valid beside of our consequence relations. By the strong consequence, the deduction theorem is not valid with none of them. However, the implication family $\supset_*^{\mathrm{f}}$ defined in our paper found to comply with the modus ponens and the deduction theorem by the strong consequence as well.

In the last section, we introduced a general formal consequence relation and showed, that it also leads to a consequence operator. The weak and strong consequence definitions are realizations of this general consequence notion. It would be profitable to consider what additional realizations are possible. By this general consequence, we also gave a suitable implication which admits the modus ponens and the deduction theorem as well.

The problem of a syntactic treatment of logical consequences in the many-valued logic could be an exciting topic of future research.

Acknowledgements

The publication is supported by the TÁMOP-4.2.2/B-10/1-2010-0024 project. The project is co-financed by the European Union and the European Social Fund.

References

[1] M. Bergmann, *An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems*, Cambridge University Press, 2008. ⇒151

[2] L. Bolc, P. Borowik, *Many-valued Logics. Vol.1. Theoretical Foundations*, Springer-Verlag, Berlin, 1992. ⇒151

[3] R. Hähnle, G. Escalada-Imaz, Deduction in many-valued logics: a survey, *Mathware and Soft Computing* **4,** 2 (1997) 69–97. ⇒151

[4] J.-L. Lee, On compactness theorem, in: *Taiwan Philosophical Association 2006 Annual Meeting*, (2006) pp. 1–11. ⇒153

[5] K. Pásztor Varga, M. Várterész, Many-valued logic, mappings, ICF graphs, normal forms, *Annales Univ. Sci. Budapest. de R. Eötvös Nom. Sect. Computatorica* **31** (2009) 185–202. ⇒147

[6] J. B. Rosser, A. R. Turquette, *Many-valued Logics. Studies in Logic and Foundations of Math.*, North-Holland Publishing Co., Amsterdam, 1952. ⇒157
[7] A. Tarski, On some fundamental concepts of metamathematics, in: *Logic, Semantics and Metamath.*, Clarendon Press, Oxford, 1956, pp. 30–38. ⇒148

Received: June 5, 2013 • Revised: September 7, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 167–183

Stackless programming in Miller*

Boldizsár NÉMETH
Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary
email: nboldi@caesar.elte.hu

Zoltán CSÖRNYEI
Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary
email: csz@inf.elte.hu

Abstract. Compilers generate from the procedure or function call instruction a code that builds up an "activation record" into the stack memory in run time. At execution of this code the formal parameters, local variables, data of visibility and the scope are pushed into the activation record, and in this record there are fields for the return address and the return value as well. In case of intensive recursive calls this is the reason of the frequent occurrences of the stack-overflow error messages. The classical technique for fixing such stack-overflows is to write programs in stackless programming style using tail recursive calls; the method is usually realised by Continuation Passing Style. This paper describes this style and gives an introduction to the new, special purpose stackless programming language *Miller*, which provides methods to avoid stack-overflow errors.

1 Introduction

"The modern operating systems we have operate with what I call the 'big stack model'. And that model is wrong, sometimes, and motivates the need for 'stackless' languages." (Ira Baxter, 2009 [1])

Computing Classification System 1998: D.3.2, D.3.3.
Mathematics Subject Classification 2010: 68-02, 68N15, 68N18
Key words and phrases: tail recursion, continuation passing style, stackless programming, programming language Miller

*Supported by Ericsson Hungary and EITKIC 12-1-2012-0001.

Using the big stack model in case of intensive recursive calls stack-overflow error messages may occur frequently. The classical method to fix such stack-overflows is to write programs in stackless programming style, when tail recursive procedures are used to eliminate the cases of running out of the available stack memory. This method is usually realised by Continuation Passing Style (CPS).

2 Stackless programming

2.1 Recursion and iteration

Hereinafter the usual definition of the factorial function is given. It is obvious that at all recursive call `fac i` it is needed to save information for the next operation, namely what operation has to be executed when the `fac i` is finished and its value is available.

$$\begin{aligned} \mathtt{fac} \equiv \quad & \lambda x \,.\, \mathtt{if} \quad (\mathtt{zero}\ x) \\ & \qquad\quad\ \ 1 \\ & \qquad\quad\ \ (*\ x(\mathtt{fac}\ (-\ x\ 1))) \end{aligned}$$

For example, the action of calculating the value of `fac 3` as follows.

$\mathtt{fac}\ 3 \rightarrow$
$*\ 3\ (\mathtt{fac}\ 2) \rightarrow$
$*\ 3\ (*\ 2(\mathtt{fac}\ 1)) \rightarrow$
$*\ 3\ (*\ 2(*\ 1(\mathtt{fac}\ 0))) \rightarrow$
$*\ 3\ (*\ 2(*\ 1\ 1)) \rightarrow$
$*\ 3\ (*\ 2\ 1) \rightarrow$
$*\ 3\ 2 \rightarrow$
6

It is obvious that if the value of the call `fac i` is calculated then it is needed to return to the caller process to execute multiplications. It means that activation records have to be used and thus a stack memory has to be applied for registration the calculating processes.

This is a so called "*recursive-controlled behaviour*", and it is obvious that using this method the stack-overflow error may appear in the case of intensive, multiple recursive calls.

There is a simple method to avoid stack-overflow errors, it has the name

"*iterative-controlled behaviour*". It is a simple iteration, where there is no need to preserve long series of operations, the size of occupied memory is not increased in the course of execution of recursive calls, and the most important property is that all of the calls are on the same level. This method uses an accumulator for storing intermediate results [4].

The factorial function in the iterative-controlled style is as follows. In this definition variable r is the accumulator.

```
fac ≡   λn  . fac′ n 1
fac′ ≡  λxr . if  (zero x)
                  r
                  (fac′ (− x 1) (∗ x r))
```

It seems that in the calculating process there is only one level for recursive calls, for example the value of fac 3

```
fac 3 →
fac′ 3 1 →
fac′ 2 3 →
fac′ 1 6 →
fac′ 0 6 →
6
```

There is no need to large stack memory for activation records, only two variables are required, one for the value n and another variable for the accumulator r. The calculating process is described and controlled by these variables.

It is important to observe that the called process does not return to the caller process to execute any operation, since there is no operation to be executed.

Iterative behaviour seems to be a very nice method, but it is applicable for cases where the size of the accumulator is constant during the calculation, and what is more, unfortunately there are procedures for which there is no possibility to convert them into iterative-controlled behaviour forms. But the continuation passing style solves this problem.

2.2 Tail position and tail call

In the previous example it was shown that the called process does not return to the caller process, and for this case we say that a tail call was executed.

More precisely, the procedure E is in *tail position* of the enclosing procedure F if F does not have any action to performed after E is returned. This means

that the return value of E is the result of F. *Tail call* means a call to expression in tail position, and the recursion is said to be *tail recursion* if the recursive calls are tail calls.

Thus in the case of tail calls there is no need for extra memory to the control information, the result of the procedure E is the result of F. Namely, after executing E, the control of execution is passed to the process which is the continuation of F.

There is a general method to convert procedures into this form, we have to write procedures in continuation passing style where the continuation represents what is left to do.

3 CPS—Continuation Passing Style

Continuation is a function and the result of the procedure is applied to it. A new variable is introduced, this variable represents the *continuation*. It usually has the name k.

There are many methods to convert an expression into continuation passing style [2, 8]. For example, a formal method published by Plotkin is as follows.

$$\begin{aligned}
[\mathsf{x}]\ \mathsf{k} &= \mathsf{k}\ \mathsf{x} \\
[\mathsf{n}]\ \mathsf{k} &= \mathsf{k}\ \mathsf{n} \\
[\lambda \mathsf{x}\ .\ \mathsf{E}]\ \mathsf{k} &= \mathsf{k}\ (\lambda \mathsf{x} \mathsf{v}\ .\ [\mathsf{E}]\ \mathsf{v}) \\
[\mathsf{E}\mathsf{F}]\ \mathsf{k} &= [\mathsf{E}]\ (\lambda \mathsf{v}\ .\ [\mathsf{F}]\ (\lambda \mathsf{w}\ .\ \mathsf{v}\ \mathsf{w}\ \mathsf{k}))
\end{aligned}$$

where the expression $[\,.\,]$ is due to convert, x is a variable and n is a constant. For example, if the continuation is $\mathsf{k} \equiv \texttt{print}$, then using the second rule to form $[6]$ `print`, it results `print 6` as it was expected.

For reductions it is not too hard to prove that $\mathsf{E} \to \mathsf{F} \iff [\mathsf{E}]\ \mathsf{k} \to [\mathsf{F}]\ \mathsf{k}$.

It is known that $(\lambda \mathsf{x}\,\mathsf{y}\ .\ \mathsf{y})\ \mathsf{1} \to \lambda \mathsf{y}\ .\ \mathsf{y} \equiv \mathsf{Id}$. The next example shows that $[(\lambda \mathsf{x}\,\mathsf{y}\ .\ \mathsf{y})\ \mathsf{1}]\ \mathsf{k} \to [\lambda \mathsf{y}\ .\ \mathsf{y}]\ \mathsf{k}$.

$$\begin{aligned}
&[(\lambda \mathsf{x}\,\mathsf{y}\ .\ \mathsf{y})\ \mathsf{1}]\ \mathsf{k} = \\
&[\lambda \mathsf{x}\,\mathsf{y}\ .\ \mathsf{y}]\ (\lambda \mathsf{v}\ .\ [\mathsf{1}]\ (\lambda \mathsf{w}\ .\ \mathsf{v}\ \mathsf{k}\ \mathsf{w})) = \\
&[\lambda \mathsf{x}\,\mathsf{y}\ .\ \mathsf{y}]\ (\lambda \mathsf{v}\ .\ (\lambda \mathsf{w}\ .\ \mathsf{v}\ \mathsf{k}\ \mathsf{w})\ \mathsf{1}) = \\
&(\lambda \mathsf{v}\ .\ (\lambda \mathsf{w}\ .\ \mathsf{v}\ \mathsf{k}\ \mathsf{w})\ \mathsf{1})\ (\lambda \mathsf{p}\,\mathsf{x}\ .\ [\lambda \mathsf{y}\ .\ \mathsf{y}]\ \mathsf{p}) \\
&(\lambda \mathsf{v}\ .\ (\lambda \mathsf{w}\ .\ \mathsf{v}\ \mathsf{k}\ \mathsf{w})\ \mathsf{1})\ (\lambda \mathsf{p}\,\mathsf{x}\ .\ \mathsf{p}\ (\lambda \mathsf{q}\,\mathsf{y}\ .\ \mathsf{q}\ \mathsf{y})) \to \\
&(\lambda \mathsf{w}\ .\ (\lambda \mathsf{p}\,\mathsf{x}\ .\ \mathsf{p}\ (\lambda \mathsf{q}\,\mathsf{y}\ .\ \mathsf{q}\ \mathsf{y}))\ \mathsf{k}\ \mathsf{w})\ \mathsf{1}) \to \\
&(\lambda \mathsf{p}\,\mathsf{x}\ .\ \mathsf{p}\ (\lambda \mathsf{q}\,\mathsf{y}\ .\ \mathsf{q}\ \mathsf{y}))\ \mathsf{k}\ \mathsf{1} \to \\
&\mathsf{k}\ (\lambda \mathsf{q}\,\mathsf{y}\ .\ \mathsf{q}\ \mathsf{y}) =
\end{aligned}$$

$k\ (\lambda q\, y\ .\ [y]\ q) =$
$[\lambda y\ .\ y]\ k$

There is a simple method to convert the expression to a tail call form, the method is presented by the calculation of $\mathtt{fac}\ 3$ [10]. We see an intermediate state:
$*\ 3(*\ 2\ (\mathtt{fac}\ 1))$.
Replace the call to $\mathtt{fac}\ 1$ with a new variable x,
$*\ 3\ (*\ 2\ x)$,
and create a λ-abstraction with this new variable:
$\lambda x.*\ 3\ (*\ 2\ x)$,
this is a continuation k of the expression $\mathtt{fac}\ 1$, that is
$k\ (\mathtt{fac}\ 1) =$
$(\lambda x.*\ 3(*\ 2\ x))\ (\mathtt{fac}\ 1) \rightarrow$
$*\ 3\ (*\ 2\ (\mathtt{fac}\ 1))$.
This means that the steps of calculation have form $k\ (\mathtt{fac}\ i)$. Now we create a new version of fac that takes an additional argument k for continuation and calls that function as the original body of $\mathtt{fac}$, that is
$\mathtt{fac\text{-}cps}\ n\ k \rightarrow k\ (\mathtt{fac}\ n)$.
The function $\mathtt{fac\text{-}cps}$ has the form as follows.

$$\begin{aligned}\mathtt{fac\text{-}cps} \equiv \lambda n k\ .\mathtt{if}\quad & (=\ n\ 0)\\ & (k\ 1)\\ & (\mathtt{fac\text{-}cps}\ (-\ n\ 1)\ (\lambda v\ .\ (k\ (*\ n\ v))))\end{aligned}$$

For example the value of $\mathtt{fac\text{-}cps}\ 3\ k$:

$\mathtt{fac\text{-}cps}\ 3\ k \quad =$
$\mathtt{fac\text{-}cps}\ 2\ (\lambda v\ .\ (k\ (*\ 3\ v))) \quad =$
$\mathtt{fac\text{-}cps}\ 1\ (\lambda v'\ .\ ((\lambda v\ .\ (k\ (*\ 3\ v)))\ (*\ 2\ v'))) \rightarrow$
$\mathtt{fac\text{-}cps}\ 1\ (\lambda v'\ .\ (k\ (*\ 3\ (*\ 2\ v')))) =$
$\mathtt{fac\text{-}cps}\ 0\ (\lambda v''\ .\ ((\lambda v'\ .\ (k\ (*\ 3\ (*\ 2\ v'))))(*\ 1\ v''))) \quad \rightarrow$
$\mathtt{fac\text{-}cps}\ 0\ (\lambda v''\ .\ (k\ (*\ 3\ (*\ 2\ (*\ 1\ v''))))) =$
$(\lambda v''\ .\ (k\ (*\ 3\ (*\ 2\ (*\ 1\ v''))))) \ 1 \rightarrow$
$(k\ (*\ 3\ (*\ 2\ (*\ 1\ 1)))) \rightarrow$
$k\ 6$

If $k \equiv \mathtt{print}$, then $\mathtt{fac\text{-}cps}\ 3\ \mathtt{print} = \mathtt{print}\ 6$, as it was expected.

We remark that continuations can be used to implement for example branching, loops, goto, return, and give possibility to write such types of control flow that coroutines, exceptions and threads.

4 Tail Call Optimisation in various languages

Tail Call Optimisation (TCO) is a common technique for transforming some execution units of the program to operate in constant stack space. The function with the recursive call is transformed into a loop, while preserving the semantics with the appropriate condition.

We chose Scala, a functional language running on JVM and the purely functional language Haskell. We did so because the problem is most relevant to functional languages where recursive calls are the only source of iterative behaviour. We also wanted to compare the different approaches to this problem.

4.1 TCO in Scala

The Scala compiler implements a limited form of the TCO.

The problem with the recursive approach is that calls to non-static functions in the JVM are dynamically dispatched. There is a direct and an indirect cost of this, mostly studied in C++ programs [3]. However extensive research had been done on the resolution of virtual calls in Java programs [11].

The system only handles self-recursion, so two functions mutually calling each other will not be transformed into a single cycle. The designers of the compiler introduced this constraint because they didn't want to cause duplications in the generated code, that could cause the program to slow down.

TCO can only be used on non-overrideable functions, because the dynamic method invocation of the JVM prevents further optimisations [9].

With the `@tailrec` annotation, the programmer can ensure that the TCO can be performed by the compiler, otherwise the compilation fails.

Example

```
def factorialAcc(acc: Int, n: Int): Int = {
  if (n<=1) acc
  else factorialAcc(n*acc, n-1)
}
```

The program is compiled to the same bytecode as:

```
def factorialAcc(acc: Int, n: Int): Int = {
  while(n <= 1) {
    acc = n*acc
    n = n-1
  }
  acc
}
```

So in this example we can get the elegant functional solution with no additional costs.

4.2 TCO in Haskell

Tail call elimination in Haskell is a little different than in languages with strict execution. It allows not only tail call functions to be executed in constant stack space, but a wider class of functions, that are called *productive* functions [7].

Every function is productive if only contains recursive calls in a data constructor.

For example take the three function definitions below:

```
infinite_list = 1 : infinite_list

infinite_number = go 0
  where go n = (let n' = n+1 in n' `seq` go n')

infinite_number' = 1 + infinite_number'
```

The `infinite_list` function is productive, because the recursive call is a parameter of the `:` data constructor, therefore the execution will not result in a stack overflow. And it will generate an infinite list of 1's.

The `infinite_number` function has a tail call, where the result is accumulated as an argument, and strictly evaluated by the `seq` function. If we would allow the lazy execution of the accumulator parameter, the tail call would be eliminated, but because the parameter is constantly growing, the "out of memory" error would be inevitable as seen in the case of `infinite_number'`. See also the strict folding functions `foldr'` and `foldl'` in [6].

Let's see the result of the execution of the three statements above:

```
> infinite_list
[1,1,1,1,1... -- This goes forever, but does not result in stack overflow.
```

```
> infinite_number
              -- Goes on forever, but also in constant stack size
> infinite_number'
<interactive>: out of memory
```

5 Structures for stackless programming

5.1 The Haskell Cont monad

The `Cont` monad can be found in the Monad transformer library [5], in the `Control.Monad.Cont` module. It is a monadic structure for writing functions with continuation passing style.

```
newtype Cont r a = Cont {
    runCont :: (a -> r) -> r  -- Returns the value after applying
                              -- the given (final) continuation to it.
}

instance Monad (Cont r) where
    return a = Cont (λf -> f a)
      -- When returning, simple supply the result to the final continuation.

    m >>= k  = Cont $ λc -> runCont m (λa -> runCont (k a) c)
      -- When binding, set the continuation of the first expression to
      -- the second expression (that gets the final continuation).$
```

After making a `Monad` instance for the `Cont` datatype, we can use it to create a factorial calculation in a simple way. The `fac` function simply executes `fac_cont` with an identity transformation as the final continuation.

The `fac_cont n` applies the final continuation with return, or executes `fac_cont (n-1)` with the multiplication as a continuation.

```
fac :: Int -> Int
fac n = runCont (fac_cont n) id
  where fac_cont :: Int -> Cont Int Int
        fac_cont 0 = return 1
        fac_cont n = do fprev <- fac_cont (n-1)
                        return (fprev * n)
```

For demonstrating the power of our continuation-using factorial function, we also present a naïve recursive implementation.

```
fac_naive :: Int -> Int
fac_naive n = n * fac_naive (n-1)
```

Then we execute both for a number larger than the maximum stack size, and inspect the results:

```
> fac 1000000
0    (Because of the overflow)
> fac_naive 1000000
<interactive>: out of memory
```

If we follow the execution of the `fac_cont` function we can observe that this is identical to the previous `fac-cps` example in Section 3.

```
> fac_cont 3
runCont (fac_cps 3) id
runCont (fac_cps 2) (λa1 → runCont (return (a1*3)) id)
runCont (fac_cps 1) ((λa2 → runCont (return (a2*2)))
                      (λa1 → runCont (return (a1*3)) id))
runCont (fac_cps 0) (λa3 → runCont (return (a3*1))
                      ((λa2 → runCont (return (a2*2)))
                        (λa1 → runCont (return (a1*3)) id)))
runCont (return (1)) ((λa2 → runCont (return (1*2)))
                       (λa1 → runCont (return (a1*3)) id))
runCont (return (1*2))) (λa1 → runCont (return (a1*3)) id)
runCont (return (1*2*3)) id
1*2*3
6
```

6 Stackless elements of the Miller programming language

In this section of the article we present the aspects of the Miller* programming language that are related to stackless programming.

* George Armitage Miller (February 3, 1920 – July 22, 2012) was one of the founders of the cognitive psychology field. He also contributed to the birth of psycholinguistics and cognitive science in general. Miller wrote several books and directed the development of WordNet, an online word-linkage database usable by computer programs [12]. We chose his name for our language because of his great contribution for the understanding on the usage of human memory, while our language focuses on the usage of electronic memory.

6.1 The Miller programming language

The Miller programming language is an industry-oriented programming language, focused on the development of performance-critical applications for special hardware.

The simplest execution unit of Miller is the *bubble*. A bubble is a separate compilation unit. Bubbles can contain simple sequential code, but no control structures such as branches and cycles. At the end of the bubble there is a section where conditions decide which bubble will be executed next. These are the *exit points* of the bubble.

Bubbles can also form a network, interconnected by their exit points. A *graph bubble* embeds a network of bubbles into itself. The nested bubbles can only be used by transferring the control to one of the entry points of the graph bubble. If an exit point of an inner bubble is connected to the entry point of another bubble inside it is called a *local exit point*. Otherwise if it's connected to an exit point of the containing graph bubble it is a *far exit point*.

The graph bubble creates an encapsulation for its inner bubbles, and defines an interface through which they can be accessed. Graph bubbles can also be nested into other graph bubbles. Nested bubbles can use any program element defined in their graph bubbles.

It is easy to see that graph bubbles, bubbles and exit points are equivalent in expressive power to the control structures of structured programming. Branches and loops can be simulated using a network of bubbles.

7 Defining control flow with bubbles

The general case of defining the transfer of control is to set exit points of the bubbles. This also allows the programmer to create control cycles. We experimented with this mode of control, but found that it is too cumbersome for actual programming.

Non-sequential code (for example branches and cycles) will be transformed into a network of bubbles. However, the language does not require the programmer to manually create these bubbles and their connections. An imperative programming interface is specified from which the compiler creates the final bubble graph. This interface contains *if-then* branching, *if-then-else* branching, special branching operations, and a *do-while* loop.

Nested bubbles can be instantiated in their ancestor bubbles any number of times.

Example

The following example shows a typical conversion from imperative frontend to a network of bubbles. It shows how the while cycle is transformed to bubbles in a naïve algorithm to compute the greatest common divisor of two positive integers.

Compile[... while ($b \neq 0$)
if($a > b$) { $a := a - b$; $b := b - a$; }
...]

The while cycle will be transformed by a stateful transformation. The condition and the loop body are separated.

Condition($b \neq 0$)
Core (Compile[if($a > b$) $a := a - b$; $b := b - a$;])
State(bubble previous_bubble{...}...)

Then another transformation will create bubbles from the condition and the loop core and combine them into the state.

State(
bubble _loop_condition ⇒ _loop_core, _exit {...}
bubble _loop_core ⇒ _loop_condition{
Compile[if($a > b$) { $a := a - b$; $b := b - a$; }]})

Using bubbles for the transfer of control does not need a stack. The programmer cannot return control to the caller from an execution module, just pass the control to the next execution unit. If call-and-return behaviour is expected, limited depth function calls provide help.

The result of the execution of the bubble body decides through which exit the control flow is passed.

Control flow of bubbles currently cannot follow continuation passing style, since it is not possible to transfer the exit point. A language feature is under planning which allows compile time passing of control. This is the parametrisation of the bubble exit points.

8 Parameter passing with interfaces

The bubble states that through an exit point which variables are passed to the next bubble. This is the exit specification. The bubble also declares the

variables that it uses, this is the entry specification. The interface checker verifies that the exit and entry specifications match.

It is very important to clarify that no copy operations happen, for this is not the traditional way of parameter passing. All variables declared in the exit and entry specification must appear in one of the ancestor bubbles.

The interface check theoretically prevents that the program has access to uninitialised variables. In practice this does not always happen. For example, it cannot be statically proven that a loop cycle running on an array gives all elements a starting value.

Nevertheless, the interface check gives us the same confidence that can be given by inspecting the initial assignment of variables, and does not require any data copy to be made for parameter passing. In addition, it also enables to create variables with constant values locally, in the scope of one bubble.

Example: factorial function in Miller

We present two approaches to calculate the factorial function. The first approach is a naïve recursive function, using stack. It is presented in a C-like pseudo-code. The second is a stackless approach, with bubbles accessing global variables, and using iterative control structure. It is presented with the pseudo-code version of the language Miller. We give the sequence of evaluation for each implementation.

Please note that, although the algorithm is the same as the example presented in the first part of the article, it is written in procedural and not in functional style.

```
int32 fac( int32 n ) {
    if( n ≤ 1 ) {
        return 1;
    } else {
        return n * fac(n-1);
    }
}
```

The next table shows the content of the stack after each step of execution. The ‘n’ columns show the value of the variable `n` in the given context. The `ret` columns show the points where the control is returned after the return call.

step #	ret	n	ret	n	ret	n
1	caller	3				
2	caller	3	fact	2		
3 (return 1)	caller	3	fact	2	fact	1
4 (return 2)	caller	3	fact	2		
5 (return 6)	caller	3				

The second part of the example shows the factorial function implemented with stackless bubbles of Miller. A few notes on the implementation:

- The language syntax may change, the purpose of the example is to give an idea about the implementation.

- The while cycle is needed because we have to connect the again exit point with the entry point of the cycle.

```
int32 n;
int32 val;

bubble fact(n)
  exits out(val) {
    val = 1;
    while(true)  cycle;
}
bubble fact::cycle(n,val)
   exits again(n,val)
   far exits out(val) {
     if( n ≤ 1 ) {
        exit out(val);
     } else {
        exit again(n-1, val*n);
     }
}
```

The next table shows the evaluation of the `fact`(3) expression in Miller with only two global variables. The two columns represent the values of the corresponding global variables.

step	n	val
1 (in fact)	3	
2 (in cycle)	3	1
3 (in cycle)	2	3
4 (in cycle)	1	6
5 (in out)		6

As can be seen, the expression is evaluated in constant stack size.

8.1 Limited depth function calls

There are situations where calling functions and returning the control is highly preferred to low level passing of control. The Miller language provides limited function calls in such situations.

Currently it's the programmers responsibility to return the control to the caller from the called method. This approach enables the function to be a complete system of bubbles, and any of them can return, if appropriate.

An important aspect of the functions is that the depth of the call chain is limited. We can calculate it in the following way:

- The bubble that calls no other bubbles have the call depth of zero.
- When a bubble calls other bubbles, its call depth is greater by one than the maximum of the call depths of the called bubbles.

Because the depth of all units must be bounded, the function calls cannot be recursive. This kind of control flow would require a stack to implement.

Thanks to the limitations on the function calls it becomes possible to store all function arguments and return addresses in registers, which is a common practice in performance critical systems. For example, if we limit our call chains to a depth of five, at most five registers would be enough to store the return addresses of the calls.

Of course, if values are passed to the called functions, more registers will be needed. To implement calls inside the bubble system, the compiler creates a calling bubble, which executes the call operation.

9 Evaluation of expressions using sandbox

The sandbox is a tool for generating instructions to evaluate complex expressions. It has a finite amount of registers and a larger amount of memory locations.

The compiler always optimises the usage of registers and memory locations to minimise the number of temporary variables. This means that the subexpressions to be evaluated first are the subexpressions that need more registers.

The sandbox works according to a simple strategy. As long as there is space the intermediate values are stored on registers, then it puts them on the specified memory areas. For now, this is enough, when it will be necessary, we design an algorithm that takes into account the different types of memory as well.

Example

In this example we present two methods to evaluate a simple expression. The first method uses the stack to evaluate expressions of any size, while the second uses a sandbox with a finite amount of registers to evaluate expressions.

The evaluation of the expression $\mathtt{a} \wedge \mathtt{b}$ to the lower part of the `ax` register with a very simple code generator is based on stack operations.

```
... instructions for the evaluation of a
    to the lower part of ax register ...
push ax
... instructions for the evaluation of b
    to the lower part of ax register ...
pop bx       (loading the previously stored value of a)
and al,bl    (executing the instruction on
              the lower part of ax and bx registers)
```

The evaluation of $\mathtt{a} \wedge \mathtt{b}$ to the lower part of ax register with our sandbox model:

```
... instructions for the evaluation of a
    to the lower part of ax register ...
... instructions to acquire a new register r
    from the sandbox ... (that may cause moving previously stored data
                          from a register to the memory.)
... instructions for the evaluation of b
    to the allocated r register ...
and ax,r
... instructions to release the allocated r register ...
```

If more registers are needed for calculating the subexpressions then more registers can be acquired. When the sandbox has temporary registers, the allocation of registers doesn't generate any instructions.

10 Summary

We investigated the techniques of generating programs that do not use a runtime stack for the calls and the evaluation of complex expressions.

We followed two different paths to address this problem. The first way was the formal method of continuation passing style, that solved the problem in a functional way. In functional languages this formal method can be implemented easily.

The second path was a practical method, implemented in the imperative Miller language. It replaced function calls with passing of control between bubbles, and function arguments with controlled global variables. This method can be used on the special hardware, which is not equipped with an efficient stack implementation. Our research was motivated by these problems.

Acknowledgements

We wish to thank the head of our research group Gergely Dévai and academic staff of the research group at ELTE Software Technology Lab for their useful and constructive work on this project and we are really thankful for making our publication possible.

We would like to thank the support of Ericsson Hungary and the grant EIT-KIC 12-1-2012-0001 that is supported by the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fund.

References

[1] I. Baxter, Answer # 1, http://stackoverflow.com/questions/1016218/how-does-a-stackless-language-work, 2013. ⇒167

[2] O. Danvy, K. Millikin, *On One-Pass CPS Transformations*, BRICS, Department of Computer Science, University of Aarhus, RS-07-6, 2007. ⇒170

[3] K. Driesen, U. Hölzle, The direct cost of virtual function calls in C++, *SIGPLAN Not.*, **31**, 10 (1996) 306–323. ⇒172

[4] D. P. Friedman, M. Wand, *Essentials of Programming Languages* (3rd edition), The MIT Press, Cambridge, MA, 2008. ⇒169

[5] A. Gill, *Hackage, The Monad Transformer Library package*, (16. 07. 2013.), http://hackage.haskell.org/package/mtl-2.0.1.0 ⇒174
[6] The Glasgow Haskell Team, *Haskell, The Data.List module*, (16. 07. 2013.), http://www.haskell.org/ghc/docs/latest/html/libraries/base/Data-List.html ⇒173
[7] Haskell Wiki, *Tail recursion*, (16. 07. 2013.), http://www.haskell.org/haskellwiki/Tail_recursion ⇒173
[8] S. Krishnamurthi, *Programming Languages: Application and Interpretation* (2nd edition), ebook, http://cs.brown.edu/~sk/Publications/Books/ProgLangs/, 2007. ⇒170
[9] T. Lindholm, F. Xellin, G. Bracha, A. Buckley, *The Java Virtual Machine Specification*, (28. 02. 2013.), http://docs.oracle.com/javase/specs/jvms/se7/html ⇒172
[10] A. Myers, *Continuation-passing style*, Lecture notes for CS 6110, Cornell University, 2013. ⇒171
[11] W. Sundaresan, L. J. Hendren, C. Razafimahefa, R. Vallée-Rai, P. Lam, E. Gagnon, C. Godin, Practical Virtual Method Call Resolution for Java, *SIGPLAN Not.*, **35**, 10 (2000) 264–280. ⇒172
[12] Wikipedia, *George Armitage Miller*, (16. 07. 2013.), http://en.wikipedia.org/wiki/George_Armitage_Miller ⇒175

Received: August 6, 2013 • Revised: October 27, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 184–211

Yang-Mills lattice on CUDA

Richárd FORSTER
Eötvös University
email: forceuse@inf.elte.hu

Ágnes FÜLÖP
Eötvös University
email: fulop@caesar.elte.hu

Abstract. The Yang-Mills fields have an important role in the non-Abelian gauge field theory which describes the properties of the quark-gluon plasma. The real time evolution of the classical fields is given by the equations of motion which are derived from the Hamiltonians to contain the term of the SU(2) gauge field tensor. The dynamics of the classical lattice Yang-Mills equations are studied on a 3 dimensional regular lattice. During the solution of this system we keep the total energy on constant values and it satisfies the Gauss law. The physical quantities are desired to be calculated in the thermodynamic limit. The broadly available computers can handle only a small amount of values, while the GPUs provide enough performance to reach out for higher volumes of lattice vertices which approximates the aforementioned limit.

1 Introduction

In particle physics there are many fundamental questions which demand the GPU, from both theoretical and experimental point of view. In the CERN NA61 collaboration one of the most important research field is the quark-gluon plasma's critical point examination. The topics of theoretical physics includes the lattice field theory which is a crossover phase transition in the quark gluon plasma and SU(N) gauge theory with topological lattice action in the QCD.

We present an algorithm which determines the real time dynamics of Yang-Mills fields which uses the parallel capabilities of the CUDA platform. We

Computing Classification System 1998: I.1.4
Mathematics Subject Classification 2010: 81T25
Key words and phrases: lattice gauge field, parallel computing, GPU, CUDA

compare this and the original sequential CPU based program in terms of efficiency and performance.

The real time evolution of these non-Abelian gauge fields is written by the Hamiltonian lattice SU(2) equation of motions [9, 1]. A lattice method was developed to solve these systems on the 3 dimensional space which satisfies Noether theory [2]. This algorithm keeps the Gauss law, the constraint of total energy and the unitary, orthogonality of the suitable link variable. It enables us to study the chaotic behavior as full complex Lyapunov spectrum of SU(2) Yang-Mills fields and the entropy-energy relation utilizing the Kolmogorov-Sinai entropy which was extrapolated to the large size limit by this numerical algorithm [6].

In the parallel algorithm all the links are computed concurrently by assigning a thread to each of them. By taking advantage of the GPUs highly parallel architecture this increases the precision of the calculation by allowing us to utilize more dense lattices. Just by adding a few more points to the lattice, the link count can increase drastically which makes this problem a very parallel friendly application which can utilize the high amount of computational resources available in modern day GPUs [8].

By extending the available CPU based implementation we were able to achieve a 28 times faster runtime compared to the original algorithm. This approach does not involve any special optimization strategies which will impose more performance boost in future releases.

In the original concept the calculations on the GPU were using only single precision floating point numbers to make the evaluations work on older generation cards too, but with the possibility to utilize double precision values, it is possible to achieve higher precision in energy calculation on the GPU as well.

There are more kind of open questions in the high energy physics which can be interesting for the GPU like studying the Yang-Mills-Higgs equations and the monopoles in the lattice QCD. The action function permits to use the thermalization of the quantum field theory in the non-equilibrium states. The solution of these problems requires high calculation power and efficiency.

This paper is constructed as follows. First we review basic definitions dealing with the Yang-Mills fields on lattice, then introducing the basic principles of the GPU programming in CUDA and finally we present numerical results providing comparisons between the CPU and GPU runtimes, extrapolate them for large N values, show the ratio between the sequential and parallel fraction of the algorithm, concluding with the thermodynamic limit of the total energy.

2 Homogeneous Yang-Mills fields

The non-Abelian gauge field theory was introduced as generalizing the gauge invariant of electrodynamics to non-Abelian Lie groups which leads to understand the strong interaction of elementary particles. The homogeneous Yang-Mills contains the quadratic part of the gauge field strength tensor [4, 10].

The $F^a_{\mu\nu}$ forms the component of an antisymmetric gauge field tensor in the Minkowski space, it is expressed by gauge fields A^a_μ:

$$F^a_{\mu\nu} = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + g f^{abc} A^b_\mu A^c_\nu, \tag{1}$$

where $\mu, \nu = 0, 1, 2, 3$ are space-time coordinates, the symmetry generators are labeled by $a, b, c = 1, 2, 3$, g is the bare gauge coupling constant and f^{abc} is the structure constant of the continuous Lie group. The generators of the Lie group fulfills the following relationship $[T^b, T^c] = i f^{bcd} T^d$.

The equation of motion can be expressed by covariant derivative in the adjoin representation:

$$\partial^\mu F^a_{\mu\nu} + g f^{abc} A^{b\mu} F^c_{\mu\nu} = 0. \tag{2}$$

We use Hamiltonian approach to investigate the real time dynamics of these systems SU(2). The lattice regularization of such theories were studied numerically.

3 Lattice Yang-Mills theory

The real time coupled differential equations of motion are solved by numerical method for basic variables which form matrix-valued link in the 3 dimensional lattice with lattice elementary size a (Figure 1). These are group elements which are related to the Yang-Mills potential A^c_i:

$$U_{x,i} = \exp(a A^c_i(x) T^c), \quad \text{where} \quad T^c \quad \text{is a group generator.}$$

For SU(2) symmetry group these links are given by the Pauli matrices τ, where $T^c = -(ig/2)\tau^c$. The indices x, i denote the link of the lattice which starts at the 3 dimensional position x and pointing into the nearest neighbor in direction i, $x + i$.

In this article we study the Hamiltonian lattice which can be written as a sum over single link contribution [1, 3]:

$$H = \sum_{x,i} \left[\frac{1}{2} \langle \dot{U}_{x,i}, \dot{U}_{x,i} \rangle + \left(1 - \frac{1}{4} \langle U_{x,i}, V_{x,i} \rangle \right) \right], \tag{3}$$

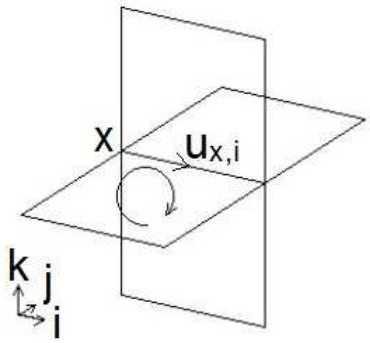

Figure 1: Wilson loop

where $\langle A, B\rangle = \mathrm{tr}(AB^\dagger)$ and $V_{x,i}$ are complement link variables, these are constructed by products of $U_{x,i}$-s along all link triples which close with given link (x, i) an elementary plaquette. The canonical variable is $P_{x,i} = \dot{U}_{x,i}$ and a dot means the time derivative.

The non-Abelian gauge field strength can be expressed by the oriented plaquette i.e. product of four links on an elementary box with corners $(x, x+i, x+i+j, x+j)$:

$$U_{x,ij} = U_{x,i}U_{x+i,j}U_{x+i+j,-i}U_{x+j,-j},$$

where $U_{x,-i} = U^\dagger_{x-i,i}$.

Then the local magnetic field strength $B^c_{x,k}$ is related to the plaquette:

$$U_{x,ij} = \exp(\epsilon_{ijk}a^2B^c_{x,k}T^c), \tag{4}$$

where ϵ_{ijk} is +1 or -1 if i, j, k is an even or odd permutation of 1,2,3 and vanishes otherwise. The electric field $\mathcal{E}^c_{x,i}$ is given in the continuum limit:

$$\mathcal{E}^c_{x,i} = \frac{2}{ag^2}\mathrm{tr}(T^c\dot{U}_{x,i}U^\dagger_{x,i}). \tag{5}$$

We use the SU(2) matrices which can be expressed by the quaternion representation (u_0, u_1, u_2, u_3) for one link, where the components u_i $i = 0, \ldots, 3$ are real numbers, it can be written:

$$\begin{aligned} U &= u_0 + i\tau^a u^a \\ U &= \begin{pmatrix} u_0 + iu_3, iu_1 + u_2 \\ iu_1 - u_2, u_0 - iu_3 \end{pmatrix}. \end{aligned} \tag{6}$$

The determinant of the quaternion is:

$$\det U = u_0^2 + u_1^2 + u_2^2 + u_3^2 = 1.$$

The length of the quaternion $\det U = \|U\|$ is conserved, because $\dot{u}_0 u_0 + \dot{u}_a u_a = 0$. The three electric fields $E^a_{x,i}$ which are updated on each link by the next form:

$$\dot{\mathcal{E}}^a_{x,i} = \frac{i}{ag^2} \sum_j \mathrm{tr}\left[\frac{1}{2}\tau^a (U_{x,ij} - U^\dagger_{x,ij})\right], \tag{7}$$

where the value of j runs over four plaquettes which are attached to the link (x, i).

The time evolution of the electric fields constraints the Gauss law:

$$D^{ab}_i \mathcal{E}^b_{x,i} = 0. \tag{8}$$

This means charge conservation.

The Hamiltonian equations of motion are derived from expression (3) by canonical method and these can be solved with dt discrete time steps. The algorithm satisfies the constraints of total energy and the Gauss law which is detailed in the next Section 3.1. Update of link variables is derived from the following implicit recursion forms of the lattice equation of motions:

$$\begin{aligned} U_{t+1} - U_{t-1} &= 2dt(P_t - \epsilon U_t), \\ P_{t+1} - P_{t-1} &= 2dt(V(U_t) - \mu U_t + \epsilon P_t), \end{aligned} \tag{9}$$

$$\epsilon = \frac{\langle U_t, P_t \rangle}{\langle U_t, U_t \rangle}, \qquad \mu = \frac{\langle V(U_t), U_t \rangle + \langle P_t P_t \rangle}{\langle U_t, U_t \rangle}, \tag{10}$$

where ε, μ are the Lagrange multipliers and the symmetry SU(N) fulfills the unitarity $\langle U_t, U_t \rangle = 1$ and the orthogonality $\langle U_t, P_t \rangle = 0$ conditions.

3.1 Constraint values

This algorithm fulfills the constraints of the system's total energy and the Gauss law.

The total energy E_{tot} is determined by the sum over each link of lattice for every time steps. The energy is defined for a single link at the time step t:

$$E_l = \frac{1}{2} < P, P > + 1 - \frac{1}{4} < U, V >, \tag{11}$$

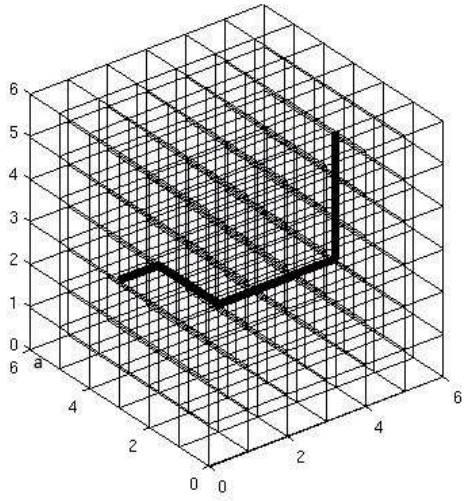

Figure 2: Flux line on the three dimensional lattice, where a means the size of the box

where $U = U_t$, $V = V_t$ and $P = P_t$. The value E_{tot} does not change during the time evolution. This is a constraint to use the Noether theorem [2]. The Gauss law is a constraint quantity:

$$\Gamma = \sum_{l^+} PU^\dagger - \sum_{l^-} U^\dagger P = 0, \tag{12}$$

where the sum is performed over links l^+ which is started at the box on the lattice and the l^- means the links to end at that side of the grid. This is conserved by the Hamiltonian equations of motion:

$$\dot{\Gamma} = \sum_{l^+} VU^\dagger - \sum_{l^-} U^\dagger V = 0. \tag{13}$$

Corresponding to the quantum electrodynamics' law the charge and flux line initialization (Figure 2) occur with the following recursion expressions on the lattice:

$$\begin{aligned} P_1 &= QU_1, \\ P_n &= U_{n-1}^\dagger P_{n-1} U_n \quad (1 \le n \le N), \end{aligned}$$

where the starting value of charge is Q and the end of this quantity equals to $-F^{\dagger}QF$. Flux line ordered product is written by the expression

$$F = \prod_{i=1}^{N-1} U_i. \tag{14}$$

The condition of neutrality is expressed by these equations:

$$\left.\begin{array}{rl} Q - F^{\dagger}QF & = 0 \\ trQ & = 0 \end{array}\right\} \Rightarrow Q = \frac{q}{2}(F^{\dagger} - F).$$

In the next section we discuss the connection between the Hamiltonian expression and Wilson action, because this plays an important role in the strong interaction.

3.2 Relation between Wilson action and Hamiltonian lattice

The Wilson action should be summed over all elementary squares of the lattice $S = \sum_{p_{x,i,j}} S_{p_{x,i,j}}$. The action function of the gauge theory on lattice is written over plaquette sum to use the nearest neighbour pairs. Because in the continuous time limit the lattice spacing a_t becomes different for the time direction, therefore the time-like plaquettes has other shape than the space-like ones. Therefore the coupling on space-like and time-like plaquettes are no longer equal in the action:

$$S = \frac{2}{g^2}\sum_{p_t}(N - tr(U_{p_t})) - \frac{2}{g^2}\sum_{p_s}(N - tr(U_{p_s})). \tag{15}$$

The time like plaquette is denoted by U_{p_t} and the space like is U_{p_s}.

Consider the path is a closed contour i.e. Wilson loop (Figure 1), where the trace of the group element corresponding to such a contour which is invariant under gauge changes and independent of the starting point. The product of such group elements along connected lines is a gauge covariant quantity, the trace over such products along a closed path is invariant. This lattice system is very suitable for describing gauge theories. Because the U_{p_t} can be series expansion by a_t:

$$U_{p_t} = U(t)U^{\dagger}(t + a_t) = UU^{\dagger} + a_t U\dot{U}^{\dagger} + \frac{a_t^2}{2}U\ddot{U}^{\dagger} + \ldots$$

$$N - tr(U_{p_t}) = -\frac{a_t^2}{2}tr(U\ddot{U}^{\dagger}) \qquad \text{up to } O(a_t^3) \text{ correction.}$$

We investigate the unitarity of the expression $UU^{\dagger} = 1$ at the beginning of Section 3, therefore this implies the following:

$$U\dot{U} + U\dot{U}^{\dagger} = 0 \quad \text{and} \quad \ddot{U}U^{\dagger} + 2\dot{U}\dot{U}^{\dagger} + U\ddot{U}^{\dagger} = 0.$$

The homogeneous non-Abelian gauge action can be written in the next form:

$$\Delta S_H = \frac{2}{g^2}\left(\frac{a_t^2}{2}\sum_i tr(\dot{U}_i\dot{U}_i^{\dagger}) - \sum_{ij}(N - tr(U_{ij}))\right). \tag{16}$$

The first sum is over all links and the second one goes over space-like plaquettes. The scaled Hamiltonian was derived in the next form. General discretized ansatz can be written as:

$$S = a_t\sum_t a_s^3\sum_s L, \tag{17}$$

than the scaled Hamiltonian becomes:

$$a_t H = \frac{2}{g^2}\left(\frac{a_t^2}{2}\sum_i tr(\dot{U}_i\dot{U}_i^{\dagger}) + \sum_{ij}(N - tr(U_{ij}))\right). \tag{18}$$

In the next section we derived the algorithm in the explicit form.

4 Lattice field algorithm

In this section we introduce the numerical solving of the coupling differential equations of motion by CPU [2]. The initial condition is uniformly random in the SU(2) group space to fulfil the constraints unitarity, orthogonality and Gauss law. The update algorithm satisfies the periodic boundary.

4.1 Implicit-explicit-endpoint algorithm

First we determine the forms μ and c corresponding to orthogonality and unitarity conditions which were introduced in Section 3. We denote:

$$P' = P_{t+1} \qquad P = P_t.$$

The equations of motion (9) are written:

$$P' = P + (V - \mu U + \varepsilon P'), \tag{19}$$

$$U' = U + (P' - \varepsilon U). \tag{20}$$

The Lagrange multiplicators μ, ε are given in the next form to satisfy the symmetry SU(2):

$$\begin{aligned}(1-\varepsilon)P' &= P + (V - \mu U),\\ U' &= (1-\varepsilon)U + P',\end{aligned}$$

where $c = 1 - \varepsilon$. First we obtain the value μ from orthogonality:

$$\begin{aligned}c\langle U', P'\rangle &= \langle cU, P\rangle + \langle P', P\rangle + c\langle U, V - \mu U\rangle + \langle P', V - \mu U\rangle,\\ 0 &= 0 + c(\langle U, V\rangle - \mu) + c\langle P', P'\rangle,\\ \mu &= \langle U, V\rangle + \langle P', P'\rangle.\end{aligned}$$

On the next step we obtain c from unitarity:

$$\begin{aligned}\langle U', U'\rangle &= c\langle U', U\rangle + \langle U', P'\rangle,\\ 1 = c\langle U', U\rangle &= c(\langle cU, U\rangle + \langle P', U\rangle),\\ 1 &= c^2 + c\langle P', U\rangle,\end{aligned}$$

$$c\langle U, P'\rangle = \langle U, P\rangle + \langle U, V - \mu U\rangle = \langle U, V\rangle - \mu = -\langle P', P'\rangle,$$

$$1 = c^2 - \langle P', P'\rangle \Rightarrow c = \sqrt{1 + \langle P', P'\rangle}.$$

$(c > 1, \quad \varepsilon < 0)$.

In the next section we express the explicit and implicit form of the algorithm.

4.1.1 Algorithm

The method is written in implicit form. The final expressions of these equations of motion are the following:

$$\begin{aligned}V^{\dagger} &= V - \langle U, V\rangle U,\\ \tilde{P} &= P + V^{\dagger},\\ cP' &= \tilde{P} - (c^2 - 1)U,\\ U' &= cU + \tilde{P}.\end{aligned}$$

The resolution of implicit recursion is:

$$\begin{aligned}c(P' + U') &= \tilde{P} + c^2 U + (1 - c^2)U + cP',\\ cU' &= \tilde{P} + U,\end{aligned}$$

but $U' = cU + P'$, so

$$\begin{aligned} P' &= U' - cU = \frac{1}{c}(\tilde{P} + U) - cU, \\ cP' &= \tilde{P} + (1 - c^2)U. \end{aligned}$$

The length of cP' becomes:

$$\begin{aligned} c^2\langle P', P'\rangle &= \langle \tilde{P}, \tilde{P}\rangle + 2(1-c^2)\langle \tilde{P}, U\rangle + (1-c^2)^2\langle U, U\rangle, \\ c^2(c^2-1) &= \langle \tilde{P}, \tilde{P}\rangle + (1-c^2)^2, \\ (c^4 - c^2) - (c^4 - 2c^2 + 1) &= \langle \tilde{P}, \tilde{P}\rangle, \\ c^2 - 1 &= \langle \tilde{P}, \tilde{P}\rangle, \end{aligned}$$

$\Rightarrow \quad c = \sqrt{1 + \langle \tilde{P}, \tilde{P}\rangle}$.

Finally the algorithm explicitly:

$$\begin{aligned} V^\dagger &= V - \langle U, V\rangle U, \\ \tilde{P} &= P + V^\dagger, \\ c &= \sqrt{1 + \langle \tilde{P}, \tilde{P}\rangle}, \\ P' &= \frac{1}{c}(\tilde{P} + U) - cU, \\ U' &= cU + P'. \end{aligned}$$

This algorithm was applied on GPU in Section 6. These processes are compared with the original sequential method on the CPU against the parallel version on the GPU.

5 Compute unified device architecture

In the last decade the performance increase of the central processing units have slowed down drastically compared to a decade earlier. At the same time the graphical processing units are showing a very intense evolution booth in performance and architecture thanks to their origin from the graphical computations and thanks to the never-ending need for more computational power (values on Figure 3 were taken from [12]).

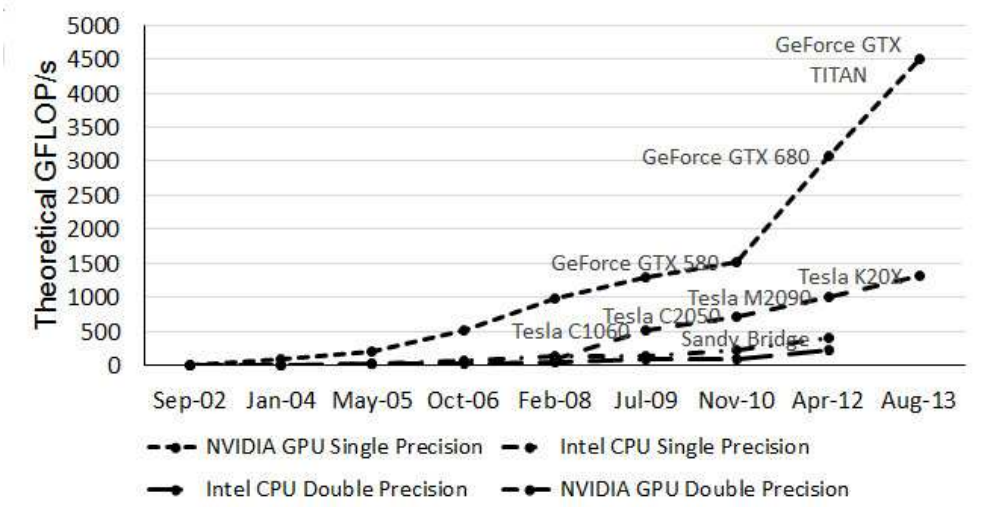

Figure 3: Performance increase of the CPU and the GPU

Our idea is to process the Yang-Mills model's high volume data on the GPU. This way we can use bigger lattices for calculation, achieving higher precision and faster runtime. With the many core architecture through the CUDA it is now possible to evaluate the actual status of the lattice by checking the individual link values in parallel.

5.1 The compute unified device architecture

Thanks to the modern GPUs now we can process efficiently very big datasets in parallel [7]. This is supported by the underlying hardware architecture that now allows us to create general purpose algorithms running on the graphical hardware. There is no need to introduce any middle abstractions to be able to use these processors as they have evolved into a more general processing unit (Figure 4 [14]). The compute unified device architecture (CUDA) divides the GPUs into smaller logical parts to have a deeper understanding of the platform. [12] With the current device architecture the GPUs are working like coprocessors in the system, the instructions are issued through the CPU. In the earlier generations we were forced to use the device side memory as the GPUs were not capable to accept direct memory pointers. If we wanted to

utilize the GPUs, then all the data were supposed to be copied over to the device prior the calculations.

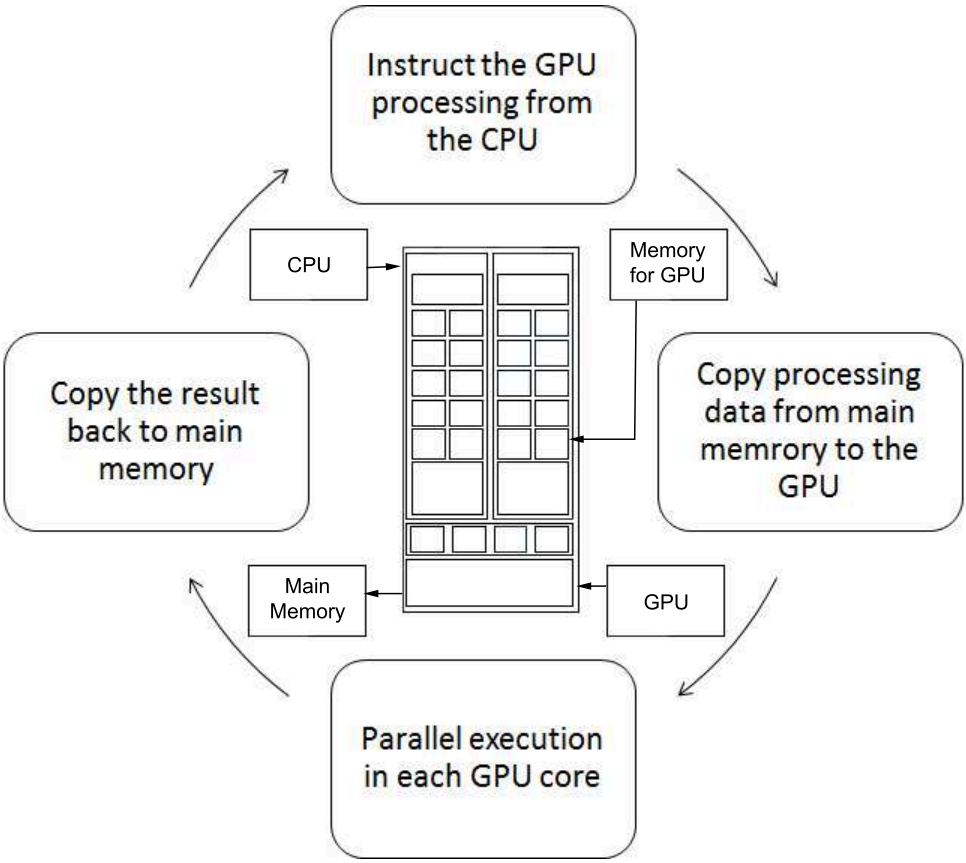

Figure 4: CUDA processing flow

While this is still the main idea behind our parallel calculations as the second generation of Compute Capability devices were released it has became possible to issue direct memory transactions thanks to the Unified Virtual Addressing [17]. This has made it possible to use pointers and memory allocations not only on the host side, but on the device as well. In earlier generations it the thread local, shared and global memories have used different address spaces, which made it impossible to use C/C++ like pointers on the device as the value of those pointers were unknown at compile time.

5.1.1 Thread hierarchy

CUDA is very similar to the C language, the biggest difference in its syntax is the <<< and >>> brackets which are used for kernel launches [8]. Kernels are special C like functions with void return value, that will create all the threads and that will run on the GPU. It also has the usual parameter list which contains all the variables we want to pass our input through and to receive the results of our computations. It is important, that for such inputs and outputs the memory has to be allocated on the GPU prior the kernel call. All of our threads are started in a grid which consists of many blocks which will contain the threads (Figure 5).

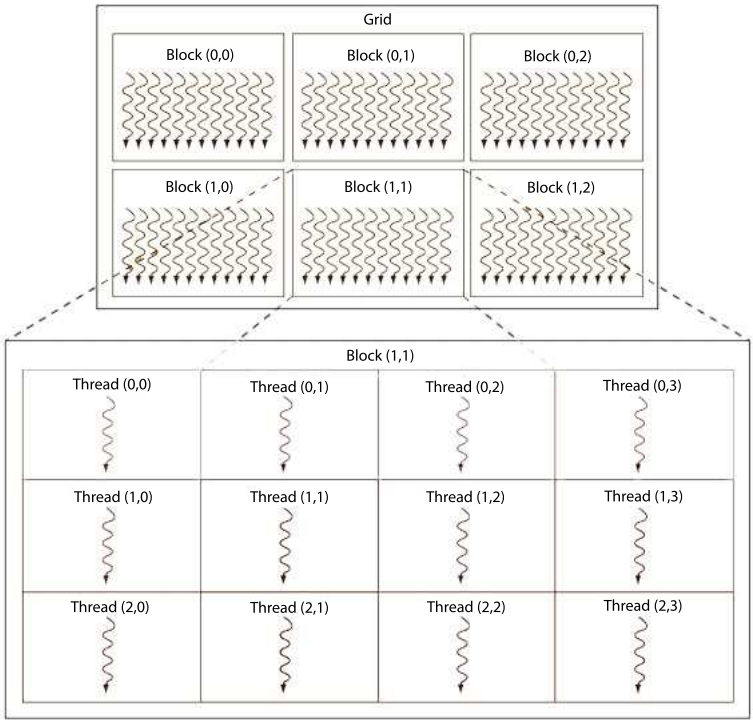

Figure 5: CUDA thread hierarchy

The maximum number of threads that we can start depends on the actual compute capability of the GPU, but it is important that this number does not equal to the actual threads being processed at the same time on the GPU. The

size of the grid and the size of the block depends on the compute capability of the hardware, but looking solely on the capability restraints we cannot show the maximum threads being processed.

A GPU can have different number of Streaming Multiprocessors (SM) and different amount of memory. The CUDA restrictions are containing occupancy restrictions as well. There are three kinds of these restrictions: resource limitations, block, and thread availability. The SMs are having the maximum limit on the number of maximum online blocks. Performance wise it is not a good practice to create algorithms that will be able to reach the resource constraints even with a very low amount of threads i.e. with high register utilization per thread [8].

We should divide our algorithm to be called by different kernels thus decreasing the resource dependency of our code. The biggest impact on the performance is the memory utilization. It is the most important aspect to keep all of our midterm results on the device and to keep the host-device memory transactions to the minimum [11]. The data must be coalesced in the memory to provide the maximum possible throughput. At the same time the registers and the shared memory are faster by at least a factor of 100 than the global memory. This is because the global memory is on the card, while the shared memory and the registers are on the chip.

5.1.2 Memory access strategies

For older generation of GPUs with Compute Capability 1.x it is important to use the right access patterns on the global memory. If we will try to use the same value from thousands of threads, then the access will have to be serialized on these GPUs, while the newer ones have caching functionality to help on such scenarios. The most optimal access is the map access, where all threads will manipulate their own values, more specifically thread n will access the nth position of the input or output array.

If the data stored in the memory can be accessed in a sequential order and it is aligned to a multitude of 128 byte address then the data fetching will be the most optimal on the devices with Compute Capability 1.x (Figure 1). The GPU can issue 32, 64 or 128 bytes transactions based on the utilization of the hardware.

If the data is in a non-sequential order (Figure 2), then additional memory transactions will be required to process everything. We mention here, by using non-sequential ordering it is possible the transactions will fetch more data, than we are really going to use at the time. This can be quite a wasteful

Compute capability:	1.0 and 1.1	1.2 and 1.3	2.x and 3.0
Memory transactions:	Uncached		Cached
	1 x 64B at 128 1 x 64B at 192	1 x 64B at 128 1 x 64B at 192	1 x 128B at 128

Table 1: Aligned and sequential memory access

approach.

Compute capability:	1.0 and 1.1	1.2 and 1.3	2.x and 3.0
Memory transactions:	Uncached		Cached
	8 x 32B at 128 8 x 32B at 160 8 x 32B at 192 8 x 32B at 224	1 x 64B at 128 1 x 64B at 192	1 x 128B at 128

Table 2: Aligned and non-sequential memory access

If the data is misaligned (Figure 3), then it will invoke more transactions as smaller ones will be required to fetch everything. This case can be problematic even on the cached devices. All tables are representing the data taken from [12].

Compute capability:	1.0 and 1.1	1.2 and 1.3	2.x and 3.0
Memory transactions:	Uncached		Cached
	7 x 32B at 128 8 x 32B at 160 8 x 32B at 192 8 x 32B at 224 1 x 32B at 256	1 x 128B at 128 1 x 64B at 192 1 x 32B 256	1 x 128B at 128 1 x 128B at 256

Table 3: Misaligned and sequential memory access

Overall it is important to design our data structures to be able to accommodate them to the aforementioned restrictions to be able to achieve the maximum possible memory throughput.

5.1.3 Other architectures and models

Our development and research was conducted on the aforementioned CUDA platform. Other architectures and programming models are available that provide high parallel performance. Currently the biggest competition for the NVIDIA GPUs are the AMD Radeon GPUs. But the biggest competition is in the discrete GPU market. In the HPC segment the NVIDIA GPUs are the mainstream solutions [18], when there is any GPU utilization in the supercomputers. For example the fastest GPU cluster based supercomputer, the TITAN incorporates NVIDIA Tesla K20X GPUs as coprocessors.

On the other hand Intel is developing it's own coprocessor for the HPC segment, the Intel Xeon Phi processor. Currently the fastest supercomputer in the TOP500 is the Tianhe-2 (MilkyWay-2) accelerated with these Xeon Phi processors, while the previously mentioned TITAN is at the second place [18]. The Linpack performance benchmark shows a $33,862.7$ TFlop/s capability for the Tianhe-2, while it shows a $17,590.0$ TFlop/s for TITAN. But if we take a look at the number of the processor cores, the Tianhe-2 uses $3,120,000$ cores, while the TITAN uses only $560,640$ cores. It is difficult to make a direct interpolation for the achieved performance in the case if we will double the cores in the TITAN, so we will take a look at the individual performance of each computers accelerator core.

The Intel Xeon Phi 3100 series of accelerators have 57 x86 cores and are capable of 1003 GFlop/s performance in double precision calculations while drawing 300 Watt of power [15]. On the other hand the NVIDIA Tesla K20X has 2688 CUDA cores, capable of 1310 GFlop/s performance also in double precision calculations while drawing only 235 Watt of power [16]. If we take the total core numbers of the supercomputers, then the Tianhe-2 has 48000 Xeon Phi processors [5], while the TITAN has 18688 Tesla K20X processors [13]. Theoretically if we double the number of K20X cards, we will have more performance than the Tianhe-2, while still utilizing less GPU, than how many MICs they are using. This shows that the Kepler GPU architecture based Tesla accelerators are more efficient than the Knights Corner MIC architecture based Xeon Phi accelerators.

Booth architectures support the OpenCL, OpenAAC programming languages which all are GPU based languages. On the NVIDIA GPUs the CUDA model is the most important as the GPUs are in connection with the programming model and as they develop the GPUs and provide new functionality, the same functionality becomes supported in the next CUDA version. This way they have the freedom to let developers utilize their GPUs how they see it the most efficient.

5.2 Single instruction multiple thread architecture

Our current CPUs are SIMD processors, where SIMD stands for Single Instruction Multiple Data. This implies that the multicore CPUs can evaluate a given instruction for multiple data which can achieve even higher amount of instructions per cycle with the Intel Hyper Threading Technology. In our case the test machine CPU has two physical cores, that can run up to four threads simultaneously thanks to the aforementioned technology. So in this case we will have four instructions evaluated concurrently. On the other hand the GPUs are SIMT architecture based processors. This stands for Single Instruction Multiple Thread which is very similar to the SIMD architecture. The biggest difference is in the maximum number of threads. Theoretically we are not bound by the maximum number of threads that we can launch, as the latest Kepler GPUs can initialize billions of threads at the same time. The key factor is that the performance per thread is quite low, but the GPU can handle thousands of those cores in a single clock cycle. Of course the actual number of running threads will be lower, but still bigger then what we can have on the CPUs. The basic idea behind the SIMT architecture is that we summon as many threads as many data we have for evaluation. This implies that for higher utilization we need to provide more data to work on. But even with maximum thread occupancy it doesn't directly mean we will achieve the maximum computational performance.

5.3 Computational architecture

In this subsection we will see what kind of technical details the GPUs have and how it affects the actual utilization of the given architecture. The actual number of threads running on the GPU comes through the term of warps. A warp is a set of 32 threads in a given Streaming Multiprocessor. Based on the actual architecture and compute capability the maximum number of warps per SM can differ (Table 4), but the size of a warp is constant 32. This means that in an optimal solution the maximum number threads running at the same time is:

$$\#SM * \#WARP * 32.$$

This implies that all n threads are running the same instruction at the same time. But in a not so optimal scenario it is possible that the threads are diverging from the size of the warp. This means that the execution flow differs among the different threads, so it will not be possible to evaluate all the 32 threads in the warp, because they are using the same program counter. In

this case the scheduler will have to take into account that there are slower warps in which the threads are serialized, and this will decrease the overall performance. This can happen if a thread has to evaluate if statement branches or long cycles. In the case of cycles the compiler can make some optimization as it will unroll the loops, but there is no way to predict the flow among the if statements.

	Compute Capability						
Technical Specifications	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Maximum dimensionality of grid of thread blocks	2				3		
Maximum x-dimension of a grid of thread blocks	65535					$2^{31}-1$	
Maximum y- or z-dimension of a grid of thread blocks	65535						
Maximum dimensionality of thread block	3						
Maximum x- or y-dimension of a block	512				1024		
Maximum z-dimension of a block	64						
Maximum number of threads per block	512				1024		
Warp size	32						
Maximum number of resident blocks per multiprocessor	8					16	
Maximum number of resident warps per multiprocessor	24		32		48	64	
Maximum number of resident threads per multiprocessor	768		1024		1536	2048	

Table 4: Compute capability technical specifications (for description see Section 5.1.1)

6 Parallel Yang-Mills algorithm

In our computational problem we are calculating newer and newer states of all the links in the system. This is a great example to use the map access pattern as all threads will have a link to work on. The algorithm was implemented on the CUDA platform.

6.1 Main idea

In every given timestep a kernel generates the new values for the links. The kernel call itself is in a cycle that will move until a given step count. The real difficulty arise from the sanity checks of the system. After a given timestep the algorithm checks if the system is still valid and such if any further developments are possible. This will require the actual results on the GPU to be checked if are still valid. To copy them back to the host side is just a waste of memory bandwidth and time. To check it on the device requires to have parallel algorithms for the subsequent operations. To check the systems validity we have to summarize the energy values of the links, thus we need to make a parallel summation. Thankfully the NVIDIA Thrust library already has this algorithm implemented, so we have used this approach. For this to work we have to give the values to the algorithm in the form of Thrust defined vectors. It is possible to cast raw memory pointers to vector containers, so there is no incompatibility between the Thrust defined vectors and the user defined global memory allocations. The result will be only one value which will be much easier to be transferred to the host side for evaluation and this will be done only once in every check. This isn't necessarily a good practice, because to achieve the highest memory throughput we should copy high amount of data instead of little fragments, but currently we only have to copy just these summation, so its really just one value per iteration, without implying any throughput problems.

6.2 Restrictions

The current implementation provides a direct port of the original Yang-Mills algorithm. As such, it does not provide any GPU specific optimizations which should further improve the already high amount of performance gain over the original CPU based version. The calculations are running on the three dimensional space with varying amount of precision, or varying dense of the lattice. The more dense it is, the more links it will generate, thus heavily increasing the inputs and the required calculations. Because this is a direct

port we do not separate the algorithm based on different functionalities, just applying the same algorithm for all the links in parallel. This implies that for the actual implementation the biggest restrain the available memory is. In this sense how dense an actual lattice can be for processing depends on how much free memory we have on the GPU, as all the links will have to be stored on the device.

6.3 Implementation

For implementation and testing we have used a GeForce GTX 580 with compute capability 2.0.

	GeForce GTX 580
Technical Specifications	Compute Capability 2.0
Transistors (Million)	3000
Memory (MB)	1536
SM Count	16
Memory Bandwidth (GB/s)	192.4
GFLOPs	1581.1
TDP (watts)	244

Table 5: GeForce GTX 580 technical specifications

In our case the maximum number of executed threads is 24576 (Table 5). This means that there will be this amount of instructions which evaluated at every given clock cycle in parallel. To be really efficient though it is important to do not introduce diverging threads. In our case the Yang-Mills algorithm doesn't provide any instructions that will result in diverging threads. The links of 3 dimensional lattice are aligned into an array which are distributed into a 1 dimensional grid. We compute a state of lattice through a grid of CUDA threads by giving a link to a thread for computation. As a new state is reached we use the Thrust reduction algorithm to do the required summation on the new values to check the actual properties while keeping the whole dataset on the GPU.

6.3.1 Compute unified device architecture based algorithm

Essentially there is no real difference between the original (see Section 4) and the CUDA based algorithm, the equations (19), (20) are the same after all.

But powerful distinction between the two is the indexing of the given links $U_{x,i}$ (Section 4.1.1) which are stored in a row ordered array.

The index of an actual thread can be calculated with the next statement, assuming that we are using a one dimensional grid (Section 5.1.1) with one dimensional blocks.

```
int idx = blockDim.x*blockIdx.x+threadIdx.x;
```

A simple optimization is the including of shared memory (Section 5.1.1).

We are accessing the links many times during an evaluation, so it is a good optimization strategy to load the links into the shared memory.

```
__shared__ float s_aux[1000];
__shared__ float s_U[1000];
__shared__ float s_V[1000];
```

The most compute heavy parts of the algorithm are the subroutines to upgrade the links and to calculate the complement of the lattice variable. As an implication of this the GPU based algorithm contains the accelerated versions of the aforementioned functions.

These functions are CUDA kernels so they are required to be __global__ functions. We are starting these kernels with 512 threads for each thread block with as many blocks as much we need to have the same amount of threads as much links we have.

Naturally this can give us more threads than the number of the available links, so a condition check is given to do not utilize the unnecessary threads.

This way we can evaluate our algorithm (Section 4.1.1) on all the maximum allowed threads at the same (Section 6.3), alas on the same amount of links.

6.4 Numerical results

We introduced a method to solve the Yang-Mills equations in time by expressions (19), (20) (see Section 4). The link variables were expressed by quaternion representation and due to the SU(2) symmetry three dimensional polar coordinate system was applied.

We numerically computed the differential equation of motion by real-time implicit-explicit algorithm (Section 4.1) to choose random initial configurations on any finite lattice SU(2). This process fulfils the constraints of total energy E_{tot} by Lagrange multiplicators, unitarity and orthogonality of the

SU(2) symmetry conditions and Gauss law. We applied periodic boundary conditions and the nearest neighbor intersection on finite lattice.

The physical quantities required the high precision calculation and the size of elementary lattice box expected the smallest value as possible i.e. to achieve the extrapolation of the thermodynamic limit. An efficient algorithm was achieved on the GPU by parallelism in Section 6. This process is much more effective on the three dimensional lattice.

For overall testing the following system was used (Table 6):

CPU	GPU	OS	Compiler	CUDA version
Intel Core i5 650	GeForce GTX 580	Windows 8 Pro	Visual C++ 2010	5.0

Table 6: The used system's specification

We compared the runtime of the CPU to the GPU (Figure 6), the GPU gives substantially better results considering the same lattice size which shows acceleration of a magnitude of 28 in single precision and 11 in double precision.

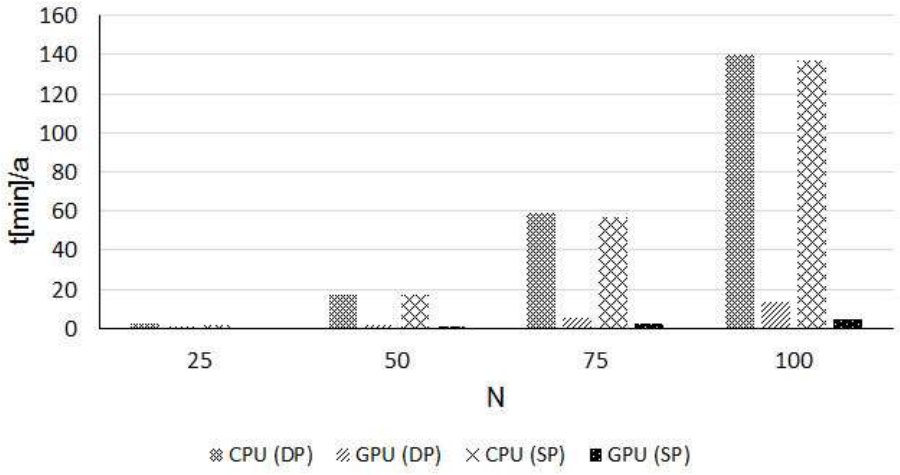

Figure 6: Runtime on the CPU and on the GPU with $N = 25, 50, 75, 100$

Even if we use single precision values for our calculations, the CPU cannot

provide any strong performance compared to the GPU because the latter has a lot more processing power.

Due to the limited resources of the CPU it is really difficult to provide a real comparison, thus we provide an extrapolation of the higher dense lattice computation runtime (Figure 7). By measuring how much time a lattice with N^3 vertices takes to be evaluated, we can see how much time a single link takes. Taking this into account we calculated the number of links on the three dimensional lattice and multiplied this number with that single link runtime as it follows:

$$\mathrm{const}_1 = t_1/(24N_1^3); \quad \text{extrapolated runtime} = \mathrm{const}_1 24N^3,$$

where t_1 is the runtime of a lattice with $N_1 = 25$ and $24N^3$ is the number of all links in a lattice with N^3 vertices. This value was calculated for booth single and double precision driven CPU and GPU based runtimes.

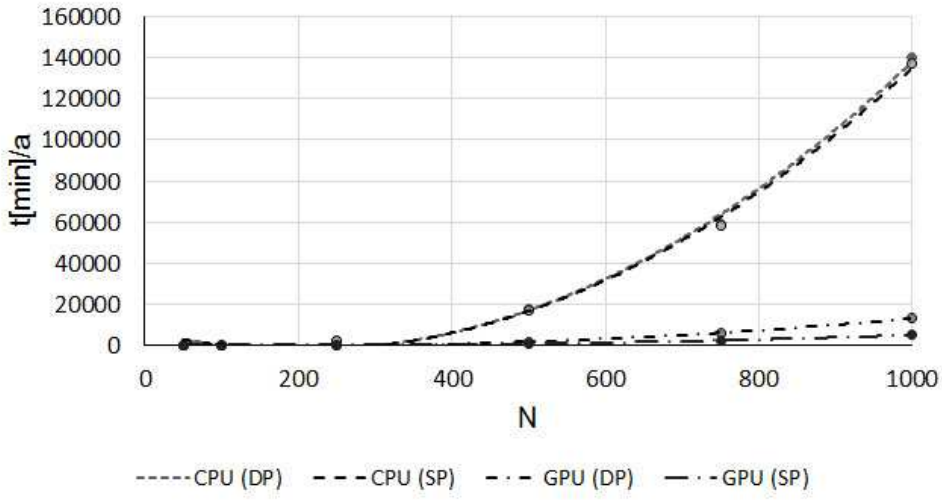

Figure 7: Extrapolated runtime of the algorithm for large N on the CPU and extrapolated runtime on the GPU. Measured range: 0 <= N <= 100; Extrapolated range: 100 < N <= 1000

From (Figure 7) it can be read that the GPU based calculations will remain faster compared to the CPU implementation even on much denser lattices. The actual measured range of the lattice size is 0 <= N <= 100 and the further

extrapolated range is $100 < N <= 1000$. (Figure 7) shows that the factor 27.68 by which the GPU is faster than the CPU on the actually calculated lattices is still valid on the extrapolated interval where this factor is 27.69. Naturally the single precision values should imply a faster runtime and the used hardware provides a significantly higher single precision peak performance, than what it gives for the double precision. Still the single precision based implementation isn't much faster than it's double precision counterpart. The reason for that is that the algorithm at hand cannot utilize the hardware efficiently.

We are mentioned it many times, that the problem at hand is very parallelization. This means that the algorithm that we are using has a very good sequential part to parallel part ratio which implies the parallelization nature of the problem. The values on (Figure 8) shows that as we increase the size of our lattice this ratio starts to grow, but very steadily. This is because the sequential part is very limited compared to the parallel part which already has a huge amount of acceleration over the original algorithm, where:

Let t_{seq}/a denote the runtime of the sequential portion of the code and t_{par}/a denote the runtime of the parallel portion booth values in seconds.

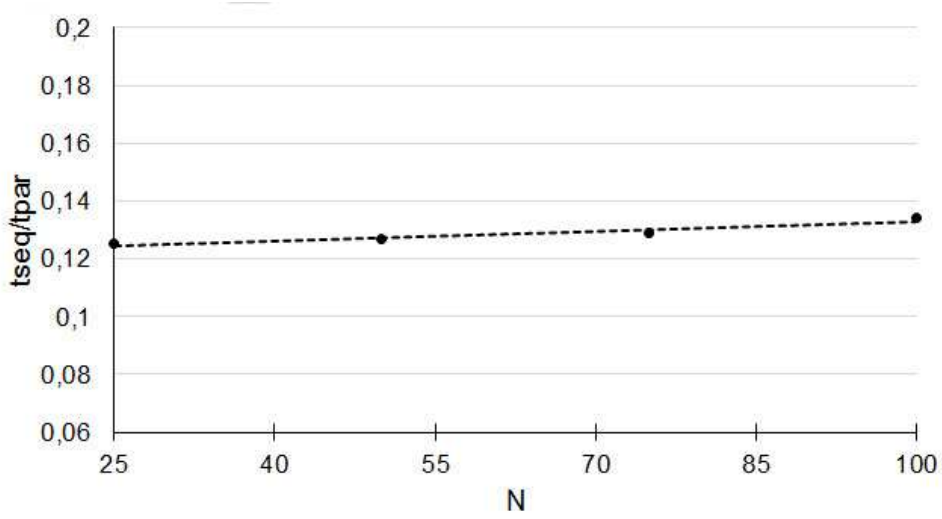

Figure 8: Ratio of the sequential and parallel runtime in the function of N, where tseq[sec]/a is the sequential fraction of the runtime and tpar[sec]/a is the parallel fraction

The fraction of the two t_{seq}/t_{par} is small, because the dominant factor is the parallel part as it will take more time to be evaluated, than the lesser sequential part. The parallel routines are handling the massive datasets, updating all the link variables in the lattice, while the sequential parts are calculating the Langrange multiplicators which results in the conservation of the total energy, the unitarity and orthogonality of the SU(2) symmetry condition in this dynamical system.

The single precision calculations are considerably faster than the double precision evaluations, so it is an important question if the single precision numbers are sufficient for our needs or not. The single precision values suffers a little loss thanks to the half precision, but the two values are still equal up to the fifth decimal value, above that the deviation of the energy only exists because of the higher precision of the double values (Figure 9).

Let E_d denote the energy based on the double precision values of energy and E_s denote the energy based on the single precision values, and $(E_d - E_s)g^2a$ is drawn in the function of t/a, where runtime measured in seconds.

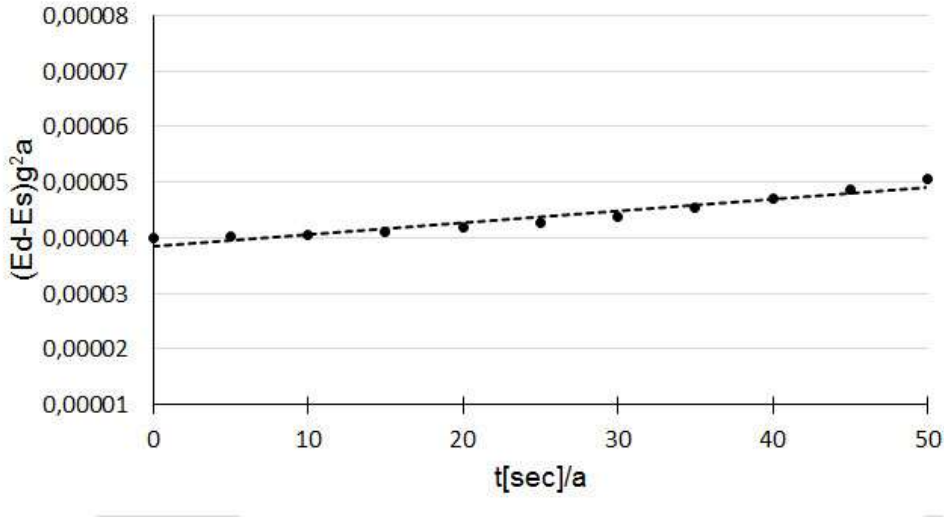

Figure 9: The energy difference in the function of the time i.e. $(E_d - E_s)g^2a$ vs $t[sec]/a$ for single precision (E_s) and double precision (E_s) values on the GPU

The fundamental value of certain physical quantities can be determined by finite-size scaling. We determined the extrapolation of the energy to the

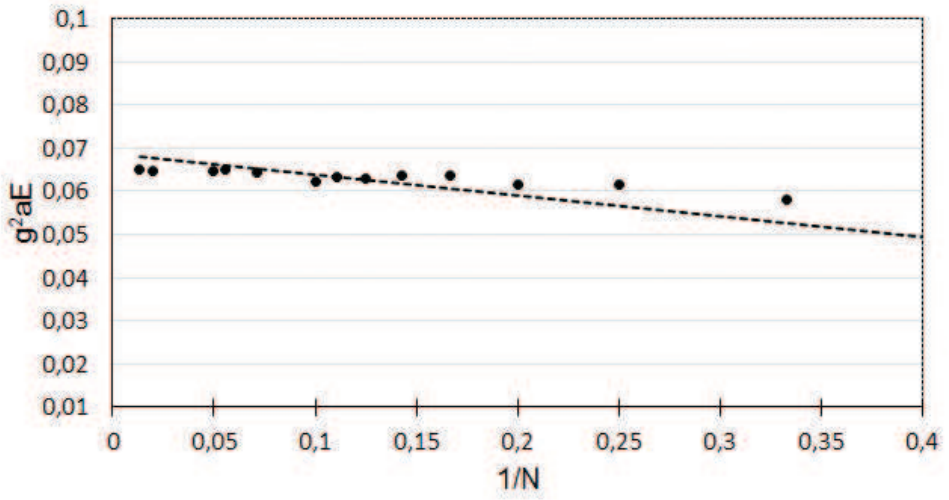

Figure 10: The thermodynamic limit of the energy ($N \to \infty$)

thermodynamic limit ($N \to \infty$) which is demonstrated on (Figure 10). The correspondence proved to be almost linear by assuming $g^2 aE \sim \frac{1}{N}$ scaling with finite-size.

In this chapter through numerical calculations we have proved that the GPU is a valuable computing platform even for the Yang-Mills algorithm, providing faster real-time performance than what the CPU has, allowing us to reach higher precision without sacrificing too much time.

7 Summary

We studied the time dependent behaviour of Yang-Mills fields which are expressed by coupled differential equations.

As the Fermi GPU architecture was build up from the ground to be compatible with the C++ programming standards it have became simple to port the existing applications to the GPU. In our case the direct port Yang-Mills model's algorithm was capable to achieve at least 11 times performance boost compared to the original.

Just changing the underlying hardware the algorithms still produced the

same result, thus keeping the physical principles valid. Physical constant quantities remains constraint while solving the equation of motion by parallel algorithm, such as the total energy.

The behaviour of the GPU makes it possible to solve the more complicated systems for example Yang-Mills-Higgs fields. This means more precision can be achieved on these systems. This is especially important where high precision computations in thermodynamic limit are mandatory.

References

[1] T. S. Biró, C. Gong, B. Müller, A. Trayanov, Hamiltonian dynamics of Yang-Mills fields on a lattice, *Int. J. of Modern Phys. C* **5** (1994) 113–149. ⇒185, 186

[2] T. S. Biró, Conserving algorithms for real-time non-Abelian lattice gauge theories, *Int. J. of Modern Phys. C* **6** (1995) 327–344. ⇒185, 189, 191

[3] T. S. Biró, A. Fülöp, C. Gong, S. Matinyan, B. Müller, A. Trajanov, Chaotic dynamics in classical lattice field theories, *165th WE-Heraeus Seminar on Theory of Spin Lattices and Lattice Gauge Models 14-19 Oct 1996. Bad Honnef, Germany Lect. Notes in Physics* **494** (1997) 164–176. ⇒186

[4] M. Creutz, *Quarks, Gluons and Lattices*, Cambridge University Press, Cambridge CB2 1RP, 1983. ⇒186

[5] J. Dongarra, *Visit to the National University for Defense Technology Changsha, China*, University of Tennessee ⇒199

[6] A. Fülöp, T. S. Biró, Towards the equation of state of a classical SU(2) lattice gauge theory, *Phys. Rev. C* **64** (2001) 064902(5). ⇒185

[7] A. Iványi (ed.), *Algorithms of Informatics, Volume 3,* AnTonCom Infokommunikációs Kft. 2011, ⇒194

[8] D. B. Kirk, W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publisher, Burlington, MA, 2012. ⇒185, 196, 197

[9] B. Müller, A. Trayanov, Deterministic chaos on non-Abelian lattice gauge theory, *Phys. Rev. Letters* **68,** 23 (1992) 3387–3390. ⇒185

[10] I. Montvay, G. Münster, *Quantum Fields on a Lattice*, Cambridge University Press, Cambridge CB2 1RP, 1994. ⇒186

[11] J. Sanders, E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 2010, NVIDIA Corporation, ⇒197

[12] *CUDA C Programming Guide*, NVIDIA Corp., 2013, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. ⇒193, 194, 198

[13] *TITAN Supercomputer*, http://www.olcf.ornl.gov/titan/ ⇒199

[14] *Tosaka, CUDA*, 2008 http://en.wikipedia.org/wiki/CUDA ⇒194

[15] *Intel Xeon Phi 3100 Series Specification*, `http://www.cpu-world.com/CPUs/Xeon_Phi/Intel-XeonPhi3120P.html` ⇒199

[16] *NVIDIA Tesla K20X Specification*, http://www.nvidia.com/object/tesla-servers.html ⇒199
[17] *Whitepaper NVIDIA's Next Generation Compute Architecture: Fermi*, NVIDIA Corp., 2009, `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf` ⇒195
[18] Official list of the top 500 supercomputers, http://top500.org ⇒199

Received: July 11, 2013 • Revised: November 25, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 212–226

On the k-Fibonacci words

José L. RAMÍREZ
Instituto de Matemáticas y sus Aplicaciones
Universidad Sergio Arboleda, Colombia
email: josel.ramirez@ima.usergioarboleda.edu.co

Gustavo N. RUBIANO
Departamento de Matemáticas
Universidad Nacional de Colombia, Colombia
email: gnrubianoo@unal.edu.co

Abstract. In this paper we define the k-Fibonacci words in analogy with the definition of the k-Fibonacci numbers. We study their properties and we associate to this family of words a family of curves with interesting patterns.

1 Introduction

Fibonacci numbers and their generalizations have many interesting properties and applications to almost every fields of science and arts (e.g. see [13]). The Fibonacci numbers F_n are the terms of the sequence $0, 1, 1, 2, 3, 5, \ldots$ wherein each term is the sum of the two previous terms, beginning with the values $F_0 = 0$, and $F_1 = 1$.

Besides the usual Fibonacci numbers many kinds of generalizations of these numbers have been presented in the literature. In particular, a generalization is the k-Fibonacci numbers [11].

For any positive real number k, the k-Fibonacci sequence, say $\{F_{k,n}\}_{n\in\mathbb{N}}$ is defined recurrently by

$$F_{k,0} = 0,\ F_{k,1} = 1 \quad \text{and} \quad F_{k,n+1} = kF_{k,n} + F_{k,n-1},\ n \geqslant 1. \tag{1}$$

Computing Classification System 1998: I.3.7, G.2.0
Mathematics Subject Classification 2010: 11B39, 05A05 , 68R15
Key words and phrases: Fibonacci word, k-Fibonacci numbers, k-Fibonacci words, k-Fibonacci curves

In [11], k-Fibonacci numbers were found by studying the recursive application of two geometrical transformations used in the four-triangle longest-edge (4TLE) partition. These numbers have been studied in several papers, see [5, 10, 11, 12, 18, 19, 23].

The characteristic equation associated to the recurrence relation (1) is $x^2 = kx + 1$. The roots of this equation are

$$r_{k,1} = \frac{k + \sqrt{k^2+4}}{2}, \quad \text{and} \quad r_{k,2} = \frac{k - \sqrt{k^2+4}}{2}.$$

Some of the properties that the k-Fibonacci numbers verify are (see [11, 12] for the proofs).

- Binet Formula: $F_{k,n} = \frac{r_{k,1}^n - r_{k,2}^n}{r_{k,1} - r_{k,2}}$.
- Combinatorial Formula: $F_{k,n} = \sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-1-i}{i} k^{n-1-2i}$.
- $\lim_{n\to\infty} \frac{F_{k,n}}{F_{k,n-1}} = r_{k,1}$.

On the other hand, there is a word-combinatorial interpretation of the Fibonacci sequence. Fibonacci words are words over $\{0,1\}$ defined recursively as follows:

$$f_0 = 1, \qquad f_1 = 0, \qquad f_n = f_{n-1}f_{n-2}, \quad n \geqslant 2.$$

The words f_n are referred to as the finite Fibonacci words and it is clear that $|f_n| = F_{n+1}$. The limit

$$\mathbf{f} = \lim_{n\to\infty} f_n = 0100101001001010010100100101\cdots$$

is called the Fibonacci word. This word is certainly one of the most studied words in the combinatorics on words, (see, e.g., [2, 6, 7, 9, 15, 22]). It is the archetype of a Sturmian word [14]. This word can be associated with a curve, which has fractal properties obtained from combinatorial properties of $\mathbf{f}$ [3, 16, 21].

In this paper we introduce a family of words $\mathbf{f}_k$ that generalize the Fibonacci word. Specifically, the k-Fibonacci words are words over $\{0,1\}$ defined inductively as follows

$$f_{k,0} = 0, \qquad f_{k,1} = 0^{k-1}1, \qquad f_{k,n} = f_{k,n-1}^k f_{k,n-2},$$

for all $n \geq 2$ and $k \geq 1$. Then it is clear that $|f_{k,n}| = F_{k,n+1}$. The infinite word

$$\mathbf{f}_k := \lim_{n\to\infty} f_{k,n}$$

is called the k-Fibonacci word. In connection with this definition, we investigate some new combinatorial properties and we associate a family of curves with interesting patterns.

2 Definitions and notation

The terminology and notations are mainly those of Lothaire [14] and Allouche and Shallit [1].

Let Σ be a finite alphabet, whose elements are called symbols. A word over Σ is a finite sequence of symbols from Σ. The set of all words over Σ, i.e., the free monoid generated by Σ, is denoted by Σ^*. The identity element ϵ of Σ^* is called the empty word and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. For any word $w \in \Sigma^*$, $|w|$ denotes its length, i.e., the number of symbols occurring in w. The length of ϵ is taken to be equal to 0. If $a \in \Sigma$ and $w \in \Sigma^*$, then $|w|_a$ denotes the number of occurrences of a in w.

For two words $u = a_1a_2\cdots a_k$ and $v = b_1b_2\cdots b_s$ in Σ^* we denote by uv the concatenation of the two words, that is, $uv = a_1a_2\cdots a_kb_1b_2\cdots b_s$. If $v = \epsilon$ then $u\epsilon = \epsilon u = u$, moreover, by u^n we denote the word $uu\cdots u$ (n times). A word v is a subword (or factor) of u if there exist $x, y \in \Sigma^*$ such that $u = xvy$. If $x = \epsilon$ ($y = \epsilon$), then v is called prefix (suffix) of u.

The reversal of a word $u = a_1a_2\cdots a_n$ is the word $u^R = a_n\cdots a_2a_1$ and $\epsilon^R = \epsilon$. A word u is a palindrome if $u^R = u$.

An infinite word over Σ is a map $\mathbf{u} : \mathbb{N} \to \Sigma$. It is written $\mathbf{u} = a_1a_2a_3\cdots$. The set of all infinite words over Σ is denoted by Σ^ω.

Example 1 *The word* $\mathbf{p} = (p_n)_{n\geq 1} = 0110101000101\cdots$, *where* $p_n = 1$ *if* n *is a prime number and* $p_n = 0$ *otherwise, is an example of an infinite word.* $\mathbf{p}$ *is called the characteristic sequence of the prime numbers.*

Definition 2 *Let* Σ *and* Δ *be alphabets. A morphism is a map* $h : \Sigma^* \to \Delta^*$ *such that* $h(xy) = h(x)h(y)$ *for all* $x, y \in \Sigma^*$. *It is clear that* $h(\epsilon) = \epsilon$.

There is a special class of words, with many remarkable properties, the so-called Sturmian words. These words admit several equivalent definitions (see, e.g. [1] or [14]).

Definition 3 *Let* $\mathbf{w} \in \Sigma^{\omega}$*. We define* $P(\mathbf{w}, n)$*, the complexity function of* $\mathbf{w}$*, to be the map that counts, for all integer* $n \geq 0$*, the number of subwords of length* n *in* $\mathbf{w}$*. An infinite word* $\mathbf{w}$ *is a Sturmian word if* $P(\mathbf{w}, n) = n + 1$ *for all integer* $n \geq 0$*.*

Since for any Sturmian word $P(\mathbf{w}, 1) = 2$, then Sturmian words are over two symbols. The word $\mathbf{p}$, in Example 1, is not a Sturmian word because $P(\mathbf{p}, 2) = 4$.

Given two real numbers $\alpha, \beta \in \mathbb{R}$ with α irrational and $0 < \alpha < 1, 0 \leq \beta < 1$, we define the infinite word $\mathbf{w} = w_1 w_2 w_3 \cdots$ as

$$w_n = \lfloor (n+1)\alpha + \beta \rfloor - \lfloor n\alpha + \beta \rfloor.$$

The numbers α and β are the slope and the intercept, respectively. This word is called mechanical. The mechanical words are equivalent to Sturmian words [14]. As special case, when $\beta = 0$, we obtain the characteristic words.

Definition 4 *Let* α *be an irrational number with* $0 < \alpha < 1$*. For* $n \geq 1$*, define*

$$w_\alpha(n) := \lfloor (n+1)\alpha \rfloor - \lfloor n\alpha \rfloor,$$

and

$$\mathbf{w}(\alpha) := w_\alpha(1) w_\alpha(2) w_\alpha(3) \cdots,$$

then $\mathbf{w}(\alpha)$ *is called a characteristic word with slope* α*.*

On the other hand, note that every irrational $\alpha \in (0, 1)$ has a unique continued fraction expansion

$$\alpha = [0, a_1, a_2, a_3, \ldots] = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}},$$

where each a_i is a positive integer. Let $\alpha = [0, 1 + d_1, d_2, \ldots]$ be an irrational number with $d_1 \geq 0$ and $d_n > 0$ for $n > 1$. To the directive sequence $(d_1, d_2, \ldots, d_n, \ldots)$, we associate a sequence $(s_n)_{n \geq -1}$ of words defined by

$$s_{-1} = 1, \quad s_0 = 0, \quad s_n = s_{n-1}^{d_n} s_{n-2}, \quad (n \geq 1).$$

Such a sequence of words is called a standard sequence. This sequence is related to characteristic words in the following way. Observe that, for any $n \geq 0$, s_n is a prefix of s_{n+1}, which gives meaning to $\lim_{n\to\infty} s_n$ as an infinite word. In fact, one can prove [14] that each s_n is a prefix of $\mathbf{w}(\alpha)$ for all $n \geq 0$ and

$$\mathbf{w}(\alpha) = \lim_{n\to\infty} s_n. \tag{2}$$

Example 5 *The infinite Fibonacci word* $\mathbf{f} = 0100101001001010\cdots$ *is a Sturmian word [14], exactly* $\mathbf{f} = \mathbf{w}\left(\frac{1}{\phi^2}\right)$ *where* $\phi = \frac{1+\sqrt{5}}{2}$ *is the golden ratio.*

Definition 6 *The Fibonacci morphism* $\sigma : \{0,1\} \to \{0,1\}$ *is defined by* $\sigma(0) = 01$ *and* $\sigma(1) = 0$.

The Fibonacci word $\mathbf{f}$ satisfies that $\lim_{n\to\infty} \sigma^n(1) = \mathbf{f}$ [1].

3 The k-Fibonacci words

Definition 7 *The* n*th* k*-Fibonacci words are words over* $\{0,1\}$ *defined inductively as follows*

$$f_{k,0} = 0, \qquad f_{k,1} = 0^{k-1}1, \qquad f_{k,n} = f_{k,n-1}^{k} f_{k,n-2},$$

for all $n \geq 2$ *and* $k \geq 1$. *The infinite word*

$$\mathbf{f}_k := \lim_{n\to\infty} f_{k,n}$$

is called the k*-Fibonacci word.*

It is clear that $|f_{k,n}| = F_{k,n+1}$. For $k = 1$ we have the word $\overline{\mathbf{f}} = 1011010110110\ldots$, where $\overline{a}$ is a morphism, with $a \in \{0,1\}$, defined by $\overline{0} = 1, \overline{1} = 0$.

Example 8 *The first* k*-Fibonacci words are*

$$\mathbf{f}_1 = 1011010110110\cdots = \overline{\mathbf{f}}, \quad \mathbf{f}_2 = 0101001010010\cdots, \quad \mathbf{f}_3 = 0010010010001\cdots,$$
$$\mathbf{f}_4 = 0001000100010\cdots, \quad \mathbf{f}_5 = 0000100001000\cdots, \quad \mathbf{f}_6 = 0000010000010\cdots.$$

Definition 9 *The k-Fibonacci morphism* $\sigma_k : \{0,1\} \to \{0,1\}$ *is defined by* $\sigma_k(0) = 0^{k-1}1$ *and* $\sigma_k(1) = 0^{k-1}10$.

Theorem 10 *For all* $n \geq 0$, $\sigma_k^n(0) = f_{k,n}$ *and* $\sigma_k^{n+1}(1) = f_{k,n+1}f_{k,n}$. *Hence, the* k*-Fibonacci word* $\mathbf{f}_k$ *satisfies that* $\lim_{n\to\infty} \sigma^n(0) = \mathbf{f}_k$.

Proof. We prove the two assertions about σ_k^n by induction on n. They are clearly true for $n = 0, 1$. Assume for all $j < n$; we prove them for n:

$$\sigma_k^{n+1}(0) = \sigma_k^n(0^{k-1}1) = (\sigma_k^n(0))^{k-1}\sigma_k^n(1) = f_{k,n}^{k-1}f_{k,n}f_{k,n-1} = f_{k,n}^k f_{k,n-1} = f_{k,n+1}.$$

$$\sigma_k^{n+2}(1) = \sigma_k^{n+1}(0^{k-1}10) = (\sigma_k^{n+1}(0))^{k-1}\sigma_k^{n+1}(1)\sigma_k^{n+1}(0) = f_{k,n+1}^{k-1}f_{k,n+1}f_{k,n}f_{k,n+1}$$

$$= f_{k,n+1}^k f_{k,n}f_{k,n+1} = f_{k,n+2}f_{k,n+1}. \qquad \square$$

Proposition 11

1. $|f_{k,n}|_1 = F_{k,n}$ *and* $|f_{k,n+1}|_0 = F_{k,n+1} + F_{k,n}$ *for all* $n \geqslant 0$.
2. $\lim\limits_{n\to\infty} \dfrac{|f_{k,n}|}{|f_{k,n}|_0} = \dfrac{r_{k,1}^2}{1 + r_{k,1}}$.
3. $\lim\limits_{n\to\infty} \dfrac{|f_{k,n}|}{|f_{k,n}|_1} = r_{k,1}$.
4. $\lim\limits_{n\to\infty} \dfrac{|f_{k,n}|_0}{|f_{k,n}|_1} = 1 + \dfrac{1}{r_{k,1}}$.

Proof.

1. It is clear by induction on n.

2. $$\lim_{n\to\infty} \frac{|f_{k,n}|}{|f_{k,n}|_0} = \lim_{n\to\infty} \frac{F_{k,n+1}}{F_{k,n} + F_{k,n-1}} = \lim_{n\to\infty} \frac{\frac{F_{k,n+1}}{F_{k,n}}}{1 + \frac{F_{k,n-1}}{F_{k,n}}} = \frac{r_{k,1}^2}{1 + r_{k,1}}.$$

3. $$\lim_{n\to\infty} \frac{|f_{k,n}|}{|f_{k,n}|_1} = \lim_{n\to\infty} \frac{F_{k,n+1}}{F_{k,n}} = r_{k,1}.$$

4. $$\lim_{n\to\infty} \frac{|f_{k,n}|_0}{|f_{k,n}|_1} = \lim_{n\to\infty} \frac{F_{k,n} + F_{k,n-1}}{F_{k,n}} = 1 + \frac{1}{r_{k,1}}.$$ $\square$

Proposition 12 *The* k*-Fibonacci word and the* n*th* k*-Fibonacci word satisfy that*

1. *The word* 11 *is not a subword of the* k*-Fibonacci word,* $k \geq 2$.
2. *Let* ab *be the last two symbols of* $f_{k,n}$. *For* $n \geq 1$, *we have* $ab = 10$ *if* n *is even and* $ab = 01$ *if* n *is odd,* $k \geq 2$.
3. *The concatenation of two successive* k*-Fibonacci words is "almost commutative", i.e.,* $f_{k,n-1}f_{k,n-2}$ *and* $f_{k,n-2}f_{k,n-1}$ *have a common prefix the length* $F_{k,n} + F_{k,n-1} - 2$ *for all* $n \geq 2$.

Proof.

1. It suffices to prove that 11 is not a subword of $f_{k,n}$, for all $n \geq 0$. By induction on n. For $n = 0, 1$ it is clear. Assume for all $j < n$; we prove it for n. We know that $f_{k,n} = f_{k,n-1}^k f_{k,n-2}$ so by the induction hypothesis we have that 11 is not a subword of $f_{k,n-1}$ and $f_{k,n-2}$. Therefore, the only possibility is that 1 is a suffix and prefix of $f_{k,n-1}$ or 1 is a suffix of $f_{k,n-1}$ and a prefix of $f_{k,n-2}$, both there are impossible.

2. By induction on n. For $n = 1, 2$ it is clear. Assume for all $j < n$; we prove it for n. We know that $f_{k,n+1} = f_{k,n}^k f_{k,n-1}$, if $n+1$ is even then by the induction hypothesis the last two symbols of $f_{k,n-1}$ are 10, therefore the last two symbols of $f_{k,n+1}$ are 10. Analogously, if $n+1$ is odd.

3. By induction on n. For $n = 1, 2$ it is clear. Assume for all $j < n$; we prove it for n. By definition of $f_{k,n}$, we have

$$\begin{aligned} f_{k,n-1}f_{k,n-2} &= f_{k,n-2}^k f_{k,n-3} \cdot f_{k,n-3}^k f_{k,n-4} \\ &= (f_{k,n-3}^k f_{k,n-4})^k \cdot f_{k,n-3}^k f_{k,n-3} f_{k,n-4}, \end{aligned}$$

and

$$\begin{aligned} f_{k,n-2}f_{k,n-1} &= f_{k,n-3}^k f_{k,n-4} \cdot f_{k,n-2}^k f_{k,n-3} \\ &= f_{k,n-3}^k f_{k,n-4} \cdot (f_{k,n-3}^k f_{k,n-4})^k \cdot f_{k,n-3} \\ &= (f_{k,n-3}^k f_{k,n-4})^k f_{k,n-3}^k f_{k,n-4} f_{k,n-3}. \end{aligned}$$

Hence the words have a common prefix of length $k(kF_{k,n-2} + F_{k,n-3}) + kF_{n-2} = k(F_{k,n-1} + F_{k,n-2})$. By the induction hypothesis $f_{k,n-3}f_{k,n-4}$ and $f_{k,n-4}f_{k,n-3}$ have a common prefix of length $F_{k,n-2} + F_{k,n-3} - 2$. Therefore the words have a common prefix of length

$$k(F_{k,n-1} + F_{k,n-2}) + F_{k,n-2} + F_{k,n-3} - 2 = F_{k,n} + F_{k,n-1} - 2. \qquad \square$$

Definition 13 *Let $\Phi : \{0,1\}^* \to \{0,1\}^*$ be a map such that Φ deletes the last two symbols, i.e., $\Phi(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_{n-2}$ $(n \geq 2)$.*

Corollary 14 *The nth k-Fibonacci word, satisfy for all $n \geqslant 2$ that*

1. $\Phi(f_{k,n-1}f_{k,n-2}) = \Phi(f_{k,n-2}f_{k,n-1})$.
2. $\Phi(f_{k,n-1}f_{k,n-2}) = f_{k,n-2}\Phi(f_{k,n-1}) = f_{k,n-1}\Phi(f_{k,n-2})$.
3. *If* $f_{k,n} = \Phi(f_{k,n})ab$, *then* $\Phi(f_{k,n-2})ab\Phi(f_{k,n-1}) = f_{k,n-1}\Phi(f_{k,n-2})$.

4. If $f_{k,n} = \Phi(f_{k,n})ab$*, then* $\Phi(f_{k,n-2})(ab\Phi(f_{k,n-1}))^k = \Phi(f_{k,n})$.

Proof. Parts (a) and (b) follow immediately from Proposition 12-(3) and because of $|f_{k,n}| \geqslant 2$ for all $n \geqslant 2$. (c) In fact, if $f_{k,n} = \Phi(f_{k,n})ab$ then from Proposition 12-(2) we have $f_{k,n-2} = \Phi(f_{k,n-2})ab$. Hence $\Phi(f_{k,n-2})ab\Phi(f_{k,n-1}) = f_{k,n-2}\Phi(f_{k,n-1}) = f_{k,n-1}\Phi(f_{k,n-2})$. (d) It is clear from (c) and definition of $f_{k,n}$. □

Theorem 15 $\Phi(f_{k,n})$ *is a palindrome for all* $n \geq 1$ *and* $k \geq 1$.

Proof. By induction on n. If $n = 2$ then $\Phi(f_{k,2}) = (0^{k-1}1)^{k-1}0^{k-1}$ is a palindrome. Now suppose that the result is true for all $j < n$; we prove it for n.

$$\begin{aligned}(\Phi(f_{k,n}))^R &= (\Phi(f_{k,n-1}^k f_{k,n-2}))^R = (f_{k,n-1}^k \Phi(f_{k,n-2}))^R = \Phi(f_{k,n-2})^R (f_{k,n-1}^k)^R \\ &= \Phi(f_{k,n-2})(f_{k,n-1}^R)^k.\end{aligned}$$

If n is even then $f_{k,n} = \Phi(f_{k,n})10$ and from Corollary 14-(4), we have that

$$\begin{aligned}(\Phi(f_{k,n}))^R &= \Phi(f_{k,n-2})((\Phi(f_{k,n-1})01)^R)^k = \Phi(f_{k,n-2})(10(\Phi(f_{k,n-1}))^R)^k \\ &= \Phi(f_{k,n-2})(10\Phi(f_{k,n-1}))^k = \Phi(f_{k,n}).\end{aligned}$$

If n is odd, the proof is analogous. □

Corollary 16 *1. If* $f_{k,n} = \Phi(f_{k,n})ab$ *then* $ba\Phi(f_{k,n})ab$ *is a palindrome.*

2. If u *is a subword of the* k*-Fibonacci word, then so is its reversal,* u^R .

Theorem 17 *Let* $\alpha = [0, \overline{k}]$ *be an irrational number, with* k *a positive integer, then*

$$\mathbf{w}(\alpha) = \mathbf{f}_k.$$

Proof. Let $\alpha = [0, \overline{k}]$ an irrational number, then its associated standard sequence is

$$s_{-1} = 1, \quad s_0 = 0, \quad s_1 = s_0^{k-1}s_{-1} = 0^{k-1}1 \text{ and } s_n = s_{n-1}^k s_{n-2}, \ n \geq 2.$$

Hence $\{s_n\}_{n\geq 0} = \{f_{k,n}\}_{n\geq 0}$ and from equation (2), we have

$$\mathbf{w}(\alpha) = \lim_{n\to\infty} s_n = \mathbf{f}_k.$$

□

Remark. Note that

$$[0,\overline{k}] = \cfrac{1}{k+\cfrac{1}{k+\cfrac{1}{k+\cfrac{1}{\ddots}}}} = \frac{-k+\sqrt{k^2+4}}{2} = -r_{k,2}$$

From the above theorem, we conclude that k-Fibonacci words are Sturmian words.

A fractional power is a word of the form $z = x^n y$, where $n \in \mathbb{Z}^+$ and $x \in \Sigma^+$, and y is power power prefix of x. If $|z| = p$ and $|x| = q$, we say that z is a p/q-power, or $z = x^{p/q}$. In the expression $x^{p/q}$, the number p/q is the power's exponent. For example, 01201201 is a 8/3-power, $01201201 = (012)^{8/3}$. The index of an infinite word $\mathbf{w}\in \Sigma^\omega$ is defined by

$$\mathrm{Ind}(w) := \sup\{r \in \mathbb{Q}_{\geqslant 1} : w \text{ contains an } r\text{-power}\}$$

For example $\mathrm{Ind}(\mathbf{f})> 3$ because the cube $(010)^3$ occurs in $\mathbf{f}$ at position 6. In [15] the authors proof that $\mathrm{Ind}(\mathbf{f})=2+\phi \approx 3.618$. A general formula for the index of a Sturmian word was given in [8].

Theorem 18 *If* $\mathbf{u}$ *is a Sturmian word of slope* $\alpha = [0, a_1, a_2, a_3, \ldots]$*, then*

$$\mathrm{Ind}(w) = \sup_{n\geqslant 0}\left\{2 + a_{n+1} + \frac{q_{n-1}-2}{q_n}\right\},$$

where q_n *is the denominator of* $\alpha = [0, a_1, a_2, a_3, \ldots, a_n]$ *and satisfies* $q_{-1} = 0, q_0 = 1, q_{n+1} = a_{n+1}q_n + q_{n-1}$.

Corollary 19 *The index of* k*-Fibonacci words is* $Ind(\mathbf{f}_k) = 2 + k + \frac{1}{r_{k,1}}$.

Proof. $\mathbf{f}_k$ is a Sturmian word of slope $\alpha = [0,\overline{k}]$, then it is clear that $q_n = F_{k,n+1}$, and from above theorem

$$\mathrm{Ind}(\mathbf{f}_k) = \sup_{n\geqslant 0}\left\{2 + k + \frac{F_{k,n}-2}{F_{k,n+1}}\right\} = 2 + k + \frac{1}{r_{k,1}}.$$ □

4 The k-Fibonacci Word Curve

The Fibonacci word can be associated to a curve from a drawing rule. We must travel the word in a particular way, depending on the symbol read a particular action is produced, this idea is the same as that used in the L-Systems [17]. In this case, the drawing rule is called "odd-even drawing rule" [16], this is defined as shown in the following table.

Symbol	**Action**
1	Draw a line forward.
0	Draw a line forward and if the symbol 0 is in a position even then turn θ degree and if 0 is in a position odd then turn $-\theta$ degrees.

Definition 20 *The* n*th-curve of Fibonacci, denoted by* $\mathcal{F}_n$*, is obtained by applying the odd-even drawing rule to the word* f_n*. The Fibonacci word fractal* $\mathcal{F}$*, is defined as*

$$\mathcal{F} := \lim_{n\to\infty} \mathcal{F}_n.$$

Example 21 *In Figure 1 we show the curve* $\mathcal{F}_{10}$ *and* $\mathcal{F}_{17}$*. The graphics in this paper were generated using the software* `Mathematica 9.0`*,* [20].

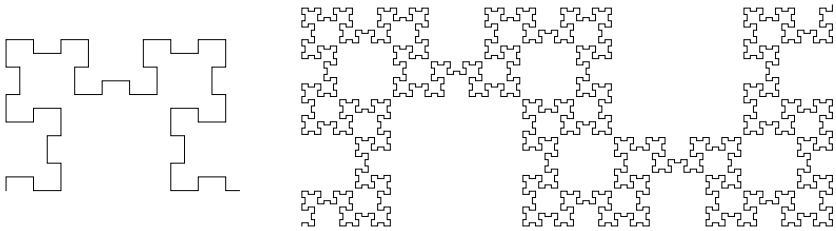

Figure 1: Fibonacci curves $\mathcal{F}_{10}$ and $\mathcal{F}_{17}$ corresponding to the words f_{10} and f_{17}

Properties of Fibonacci Word Fractal can be found in [3, 4, 16].

Definition 22 *The* n*th* k*-curve of Fibonacci, denoted by* $\mathcal{F}_{k,n}$*, is obtained by applying the odd-even drawing rule to the word* $f_{k,n}$*. The* k*-Fibonacci word curve* $\mathcal{F}_k$ *is defined as*

$$\mathcal{F}_k := \lim_{n\to\infty} \mathcal{F}_{k,n}.$$

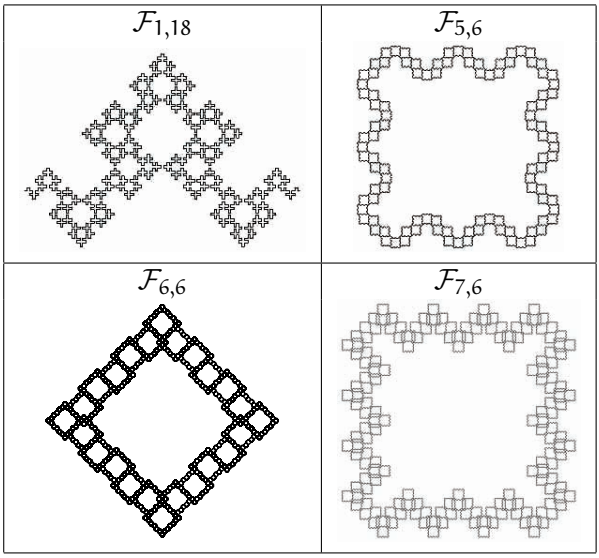

Table 1: Some curves $\mathcal{F}_{k,n}$ with $\theta = 90°$

In Table 1, we show some curves $\mathcal{F}_{k,n}$ with an angle $\theta = 90°$.

In Table 2, we show some curves $\mathcal{F}_{k,n}$ with an angle $\theta = 60°$. In general these curves have a lot of patterns because the index is large, see Corollary 19.

Proposition 23 *The* k*-Fibonacci word curve and the curve* $\mathcal{F}_{k,n}$ *have the following properties:*

1. *The* k*-Fibonacci curve* $\mathcal{F}_k$ *is composed only of segments of length 1 or 2.*
2. *The* $\mathcal{F}_{k,n}$ *is symmetric.*
3. *The number of turns in the curve* $\mathcal{F}_{k,n}$ *is* $F_{k,n} + F_{k,n-1}$.
4. *If* n *is even then the* $\mathcal{F}_{k,n}$ *curve is similar to the curve* $\mathcal{F}_{k,n-2}$ *and if* n *is odd then the* $\mathcal{F}_{k,n}$ *curve is similar to the curve* $\mathcal{F}_{k,n-3}$.

Proof.

1. It is clear from Proposition 12-1, because 110 and 111 are not subwords of $\mathbf{f}_k$.
2. It is clear from Theorem 15, because $f_{k,n} = \Phi(f_{k,n})ab$, where $\Phi(f_{k,n})$ is a palindrome.
3. It is clear from definition of odd-even drawn rule and because $|f_{k,n+1}|_0 = F_{k,n+1} + F_{k,n}$.

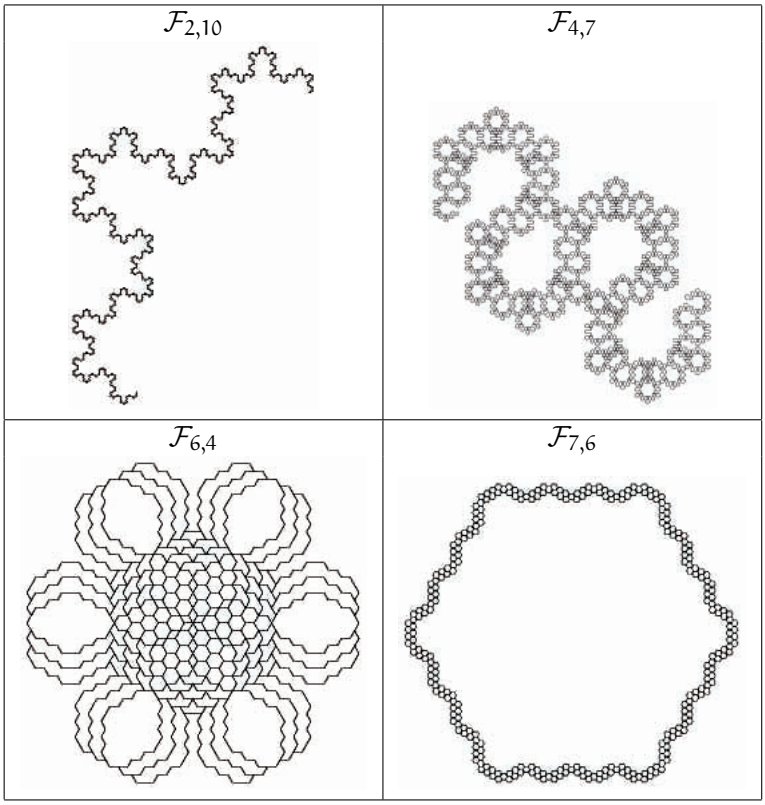

Table 2: Some curves $\mathcal{F}_{k,n}$ with $\theta = 60°$

4. If n is even. It is clear that $\sigma_k^2(f_{k,n-2}) = f_{k,n}$. We are going to proof that σ_k^2 guaranties the odd-even alternation required by the odd-even drawing rule. In fact, $\sigma_k^2(0) = \sigma_k(0^{k-1}1) = (0^{k-1}1)^k 0$ and $\sigma_k^2(1) = (0^{k-1}1)^k 0^k 1$. As k is even, then $|\sigma_k^2(0)|$ and $|\sigma_k^2(1)|$ are odd. Hence if $|w|$ is even (odd) then $|\sigma_k^2(w)|$ is even (odd). Since σ_k^2 preserves the parity of length then any subword in the k-Fibonacci word preserves the parity of position.

   Finally, we have to proof that the resulting angle of a pattern must be preserved or inverted by σ_k^2. Let $a(w)$ be the function that gives the resulting angle of a word w through the odd-even drawing rule of angle

θ. Note that $a(00) = 0^\circ, a(01) = -\theta^\circ$ and $a(10) = \theta^\circ$. Therefore

$$\begin{aligned}
a(\sigma_k^2(00)) &= a((0^{k-1}1)^k 0(0^{k-1}1)^k 0) \\
&= a((0^{k-1}1)^k)a(00^{k-1})a(1(0^{k-1}1)^{k-1}0) \\
&= -k\theta^\circ + 0^\circ + k\theta^\circ = 0^\circ \\
a(\sigma_k^2(01)) &= a((0^{k-1}1)^k 0(0^{k-1}1)^k 0^k 1) \\
&= a((0^{k-1}1)^k)a(00^{k-1})a(1(0^{k-1}1)^{k-1}0)a(0^{k-1}1) \\
&= -k\theta^\circ + 0^\circ + k\theta^\circ - \theta^\circ = -\theta^\circ \\
a(\sigma_k^2(10)) &= a((0^{k-1}1)^k 0^k 1(0^{k-1}1)^k 0) \\
&= a((0^{k-1}1)^k)a(0^k)a(1(0^{k-1}1)^k 0) \\
&= -k\theta^\circ + 0^\circ + (k+1)\theta^\circ = \theta^\circ
\end{aligned}$$

Then σ_k^2 inverts the resulting angle, i.e., $a(w) = -a(\sigma_k^2(w))$ for any word w. Therefore the image of a pattern by σ_k^2 is the rotation of this pattern by a rotation of $-\theta^\circ$. Since $\sigma_k^2(f_{k,n-2}) = f_{k,n}$, then the curve $\mathcal{F}_{k,n}$ is similar to the curve $\mathcal{F}_{k,n-2}$.

If n is odd the proof is similar, but using σ_k^3. □

Example 24 *In Figure 2 $\mathcal{F}_{4,4}$ looks similar to $\mathcal{F}_{4,6}, \mathcal{F}_{4,8}$ and so on.*

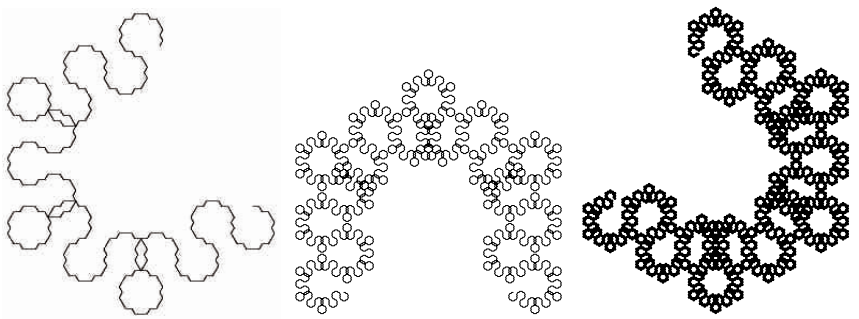

Figure 2: Curves $\mathcal{F}_{4,4}, \mathcal{F}_{4,6}, \mathcal{F}_{4,8}$ with $\theta = 60^\circ$

In Figure 3 $\mathcal{F}_{5,3}$ looks similar to $\mathcal{F}_{5,6}$.

Acknowledgements

The first author was partially supported by Universidad Sergio Arboleda under grant number USA-II-2012-14.

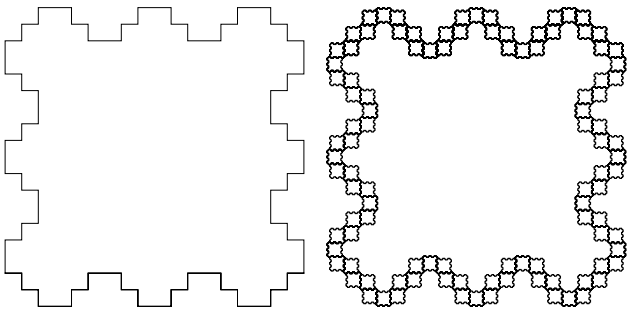

Figure 3: Curves $\mathcal{F}_{5,3}, \mathcal{F}_{5,6}$ with $\theta = 60°$

References

[1] J. Allouche, J. Shallit, *Automatic Sequences*, Cambridge University Press, 2003. ⇒214, 216

[2] J. Berstel, Fibonacci words—a survey, in: G. Rosenberg, A. Salomaa (Eds.), *The Book of L*, Springer, Berlin, 1986, pp. 11–26. ⇒213

[3] A. Blondin-Mass, S. Brlek, A. Garon, S. Labb, Two infinite families of polyominoes that tile the plane by translation in two distinct ways, *Theoret. Comput. Sci.* **412** (2011) 4778–4786. ⇒213, 221

[4] A. Blondin-Mass, S. Brlek, S. Labb, M. Mends France, Complexity of the Fibonacci snowflake, *Fractals* **20** (2012) 257–260. ⇒221

[5] C. Bolat, H. Kse, On the properties of k-Fibonacci numbers, *Int. J. Contemp. Math. Sciences*, **22,** 5 (2010) 1097–1105. ⇒213

[6] J. Cassaigne, On extremal properties of the Fibonacci word, *RAIRO - Theor. Inf. Appl.* **42,** (4) (2008) 701–715. ⇒213

[7] W. Chuan, Fibonacci words, *Fibonacci Quart.*, **30,** 1 (1992) 68–76. ⇒213

[8] D. Damanik, D. Lenz, The index of Sturmian sequences, *European J. Combin.*, **23** (2002) 23–29. ⇒220

[9] A. de Luca, Sturmian words: structure, combinatorics, and their arithmetics, *Theor. Comput. Sci.* **183,** 1 (1997) 45–82. ⇒213

[10] S. Falcon, A. Plaza, On k-Fibonacci sequences and polynomials and their derivatives, *Chaos Solitons Fractals* **39,** 3 (2009) 1005–1019. ⇒213

[11] S. Falcon, A. Plaza, On the Fibonacci k-numbers, *Chaos Solitons Fractals* **32,** 5 (2007) 1615–1624. ⇒212, 213

[12] S. Falcon, A. Plaza, The k-Fibonacci sequence and the Pascal 2-triangle, *Chaos Solitons Fractals* **33,** 1 (2007) 38–49. ⇒213

[13] T. Koshy, *Fibonacci and Lucas numbers with Applications,* Wiley-Interscience, 2001. ⇒212

[14] M. Lothaire, *Algebraic Combinatorics on Words,* Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2002. ⇒213, 214, 215, 216

[15] F. Mignosi, G. Pirillo, Repetitions in the Fibonacci infinite word, *RAIRO – Theor. Inf. Appl.* **26** (1992) 199–204. ⇒213, 220
[16] A. Monnerot, The Fibonacci word fractal, preprint, 2009. ⇒213, 221
[17] P. Prusinkiewicz, A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, Nueva York, 2004. ⇒221
[18] J. Ramírez, Incomplete k-Fibonacci and k-Lucas numbers, *Chinese Journal of Mathematics* (2013). ⇒213
[19] J. Ramírez. Some properties of convolved k-Fibonacci numbers. *ISRN Combinatorics* (2013) ID759641, 5pp. ⇒213
[20] J. Ramírez, G. Rubiano, Generating fractals curves from homomorphisms between languages [with Mathematica®] (Spanish), *Rev. Integr. Temas Mat.* **30,** 2 (2012) 129–150. ⇒221
[21] J. Ramírez, G. Rubiano, R. de Castro, A generalization of the Fibonacci word fractal and the Fibonacci snowflake, preprint arXiv:1212.1368, 2013. ⇒213
[22] W. Rytter, The structure of subword graphs and suffix trees of Fibonacci words, *Theoret. Comput. Sci.* **363,** 2 (2006) 211–223. ⇒213
[23] A. Salas. About k-Fibonacci numbers and their associated numbers, *Int. Math. Forum.* **50,** 6 (2011) 2473–2479. ⇒213

Received: June 12, 2013 • Revised: October 25, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 227–244

Some more on the basis finite automaton

Boris MELNIKOV
Russia, Samara State University
email: bormel@rambler.ru

Aleksandra MELNIKOVA
Russia, National Research Nuclear University "MEPhI"
email: super-avahi@yandex.ru

Abstract. We consider in this paper the basis finite automaton and its some properties. We shall also consider some properties of special binary relation defined on the sets of states of canonical automata for the given language and for its mirror image. We shall also consider an algorithm of constructing the basis automaton defining the language which has a priory given variant of this relation.

1 Introduction

The basis automaton for the given regular language was firstly defined in [11]. And in [6], we considered an extension of the basis automaton, which can describe *all the possible* labels of inputs, outputs and loops for *any* state of *any* nondeterministic automaton defining the given language.

The basis automaton can be considered as a complete invariant of regular language, like automaton of canonical form and Conway's universal automaton ([2, 5]). But using the basis automaton, we can formulate some properties of regular language; using other formalisms, these properties could be formulated in a more complicated way. Some of such properties were already considered in [8, 9, 11].

In this paper, we shall consider some other such properties and some examples for them. Among other things, we shall consider some properties of special binary relation defined on the sets of states of two canonical automata: for the

Computing Classification System 1998: F.4.3
Mathematics Subject Classification 2010: 68Q45
Key words and phrases: nondeterministic finite automata, basis automaton, state-marking functions

given language and for its mirror image. We shall also consider an algorithm of constructing the basis automaton defining the language, for which there holds the *a priory given* variant of this relation.

2 Preliminaries

We shall use the notation and preliminaries of [6, 7]. Let us repeat the main ones for the ease of reading.

We shall consider nondeterministic finite automaton

$$K = (Q, \Sigma, \delta, S, F) \tag{1}$$

without ε-edges, i.e., we consider transition function δ of automaton (1) as

$$\delta : Q \times \Sigma \to \mathcal{P}(Q) .$$

Its language will be denoted by $\mathcal{L}(K)$; unless other fact is formulated, we shall suppose that $\mathcal{L}(K) = L$.

The input language of the state $q \in Q$, i.e., the language of automaton $(Q, \Sigma, \delta, S, \{q\})$, will be denoted by $\mathcal{L}^{in}_K(q)$. Similarly, the output language of the state $q \in Q$, i.e., the language of automaton $(Q, \Sigma, \delta, \{q\}, F)$, will be denoted by $\mathcal{L}^{out}_K(q)$.

$\widetilde{L}$ is the canonical automaton defining L, without the useless ("dead") state. Let automata $\widetilde{L}$ and $\widetilde{L^R}$ for the given language L be as follows:

$$\widetilde{L} = (Q_\pi, \Sigma, \delta_\pi, \{s_\pi\}, F_\pi) \quad \text{and} \quad \widetilde{L^R} = (Q_\rho, \Sigma, \delta_\rho, \{s_\rho\}, F_\rho)$$

(where π and ρ are indexes which indicate languages of two canonical automata, i.e., languages L and L^R respectively).

Binary relation $\# \subseteq Q_\pi \times Q_\rho$ is defined in the following way. For some states $A \in Q_\pi$ and $X \in Q_\rho$, condition $A\#X$ holds if and only if there exist some words $u \in \mathcal{L}^{in}_{\widetilde{L}}(A)$ and $v \in \mathcal{L}^{out}_{\widetilde{L^R}}(X)$, such that $uv^R \in \mathcal{L}(K)$. In [7], we considered a simple algorithm for constructing this relation.

Also in [6, 8, 7], we considered state-marking functions φ^{in} and φ^{out} for automaton (1); those are the function of the type

$$\varphi^{in}_K : Q \to \mathcal{P}(Q_\pi) \quad \text{and} \quad \varphi^{out}_K : Q \to \mathcal{P}(Q_\rho)$$

defined in the following way. We set $\varphi^{in}_K(q) \ni A$ (where $q \in Q$ and $A \in Q_\pi$) if and only if

$$(\exists u \in \Sigma^*)\,(u \in \mathcal{L}^{in}_K(q)\,\&\,u \in \mathcal{L}^{in}_{\widetilde{L}}(A))\,, \quad \text{i.e.,} \quad \mathcal{L}^{in}_K(q) \cap \mathcal{L}^{in}_{\widetilde{L}}(A) \neq \emptyset\,.$$

Similarly, we set $\varphi_K^{out}(q) \ni X$ (where $q \in Q$ and $X \in Q_\rho$) if and only if

$$\left(\mathcal{L}_K^{out}(q)\right)^R \cap \mathcal{L}_{\widetilde{L^R}}^{in}(X) \neq \emptyset\,.$$

A simple algorithm for constructing these functions was also given in [7].

For language L defined by automaton (1), we define the equivalent basis automaton $\mathcal{BA}(L)$; in this paper, we use the version of its definition of [6]. Thus, for the given regular language L, this automaton will be denoted by

$$\mathcal{BA}(L) = (\hat{Q}, \Sigma, \hat{\delta}, \hat{S}, \hat{F}), \tag{2}$$

where [1]:

- $\hat{Q}$ is the set of pairs of the type $\begin{smallmatrix} A \\ X \end{smallmatrix}$, such that $A \in Q_\pi$, $X \in Q_\rho$ and $A\#X$;
- transition function $\hat{\delta}$ is defined in the following way: for each $\begin{smallmatrix} A \\ X \end{smallmatrix}, \begin{smallmatrix} B \\ Y \end{smallmatrix} \in \hat{Q}$ and $a \in \Sigma$, we have $\begin{smallmatrix} A \\ X \end{smallmatrix} \xrightarrow[\hat{\delta}]{a} \begin{smallmatrix} B \\ Y \end{smallmatrix}$ if and only if $A \xrightarrow[\delta_\pi]{a} B$ and $Y \xrightarrow[\delta_\rho]{a} X$;
- $\hat{S} = \left\{ \begin{smallmatrix} s_\pi \\ X \end{smallmatrix} \,\middle|\, s_\pi \# X \right\}$;
- similarly, $\hat{F} = \left\{ \begin{smallmatrix} A \\ s_\rho \end{smallmatrix} \,\middle|\, A \# s_\rho \right\}$.

Thus, we can think that considering the given regular *language* L, we also have notation for *its*:

- two automata of canonical form (i.e., $\widetilde{L}$ and $\widetilde{L^R}$), and also their states, transition functionc etc;
- binary relation $\#$;
- state-marking functions φ^{in} and φ^{out};
- basis automaton $\mathcal{BA}(L)$.

We also shall sometimes consider automaton $(\widetilde{L^R})^R$ which also defines language L.

[1] See [7] for some more details, e.g., for binary relation $\#$.

3 The correctness of the definition $\mathcal{BA}(L)$: the complete proof

As we said before, the definition of the basis automaton was firstly given in [11], we use the equivalent definition of [6]. Also in [11], there was given the proof of the correctness of that definition. But that proof was incomplete: in fact, we have proved only that each word of the given language can be accepted by automaton $\mathcal{BA}(L)$. In this section, we consider the complete version of this proof. This complete version will be also used in Section 7.

Proposition 1 $\quad \mathcal{L}\Big(\mathcal{BA}(L)\Big) = L.$

Proof. We shall prove the equivalence of automata $\widetilde{L}$ and $\mathcal{BA}(L)$.

1. Firstly, let us consider some word $u \in L$, i.e., $u \in \mathcal{L}(\widetilde{L})$. Then $u^R \in L^R$, i.e., $u^R \in \mathcal{L}(\widetilde{L^R})$. Let $|u| = n$.

Let the accepting of the word u by automaton $\widetilde{L}$ is the following sequence of transitions beginning (the only) initial state s_π:

$$p_0{=}s_\pi,\ p_1,\ p_2,\ \ldots,\ p_{n-2},\ p_{n-1},\ f_\pi{=}p_n\,, \tag{3}$$

where $p_i \in Q_\pi$ for each $i \in \{1, \ldots, n{-}1\}$ (i.e., each p_i is some state of automaton $\widetilde{L}$), and $f_\pi \in F_\pi$ (i.e., f_π is *some* final state of automaton $\widetilde{L}$). Because $\widetilde{L}$ is deterministic automaton, sequence (3) is the (unique) accepting run of $\widetilde{L}$ on u.

Similarly, automaton $\widetilde{L^R}$ reading letters of the word u^R has the following sequence of transitions:

$$r_0{=}s_\rho,\ r_1,\ r_2,\ \ldots,\ r_{n-2},\ r_{n-1},\ f_\rho = r_n \tag{4}$$

(where $r_i \in Q_\rho$ for each $i \in \{1, \ldots, n{-}1\}$, and $f_\rho \in F_\rho$); as before, s_ρ is the only initial state. Sequence (4) is also defined by the given word u (or u^R) in the only way. The numbers of elements of the sequences (3) and (4) are the same; they are equal to $n+1$. The sequence of transitions for automata $\widetilde{L}$, $\widetilde{L^R}$ and $(\widetilde{L^R})^R$ reading words u and u^R is shown on the following diagram:

$$\begin{array}{rccccccccccc}
\widetilde{L}: & s_\pi & \longrightarrow & p_1 & \longrightarrow & p_2 & \ldots\ p_{n-2} & \longrightarrow & p_{n-1} & \longrightarrow & f_\pi & \\
 & & a_1 & & a_2 & & \ldots & & a_{n-1} & & a_n & \\
 & f_\rho & \longleftarrow & r_{n-1} & \longleftarrow & r_{n-2} & \ldots\ r_2 & \longleftarrow & r_1 & \longleftarrow & s_\rho & : \widetilde{L^R} \\
(\widetilde{L^R})^R: & f_\rho & \longrightarrow & r_{n-1} & \longrightarrow & r_{n-2} & \ldots\ r_2 & \longrightarrow & r_1 & \longrightarrow & s_\rho &
\end{array}$$

Let us rewrite the sequence (4) in the reverse order:

$$f_\rho,\, r_{n-1},\, r_{n-2},\, \ldots,\, r_2,\, r_1,\, s_\rho\,, \tag{5}$$

then let us combine elements of (3) and (5) in the following sequence of the pairs:

$$\frac{s_\pi}{f_\rho}, \quad \frac{p_1}{r_{n-1}}, \quad \frac{p_2}{r_{n-2}}, \quad \ldots, \quad \frac{p_{n-2}}{r_2}, \quad \frac{p_{n-1}}{r_1}, \quad \frac{f_\pi}{s_\rho}.$$

By the definition of relation #, each pair of this sequence can be considered as a state of automaton $\mathcal{BA}(L)$, because for each such pair (let it be $\frac{q_i}{r_{n-i}}$), we can write a word of L (i.e., the given word u) by $u = vw$, where

$$v = a_1 \ldots a_i \in \mathcal{L}^{in}_{\widetilde{L}}(p_i) \quad \text{and} \quad w = a_{i+1} \ldots a_n \in \left(\mathcal{L}^{in}_{\widetilde{L^R}}(r_{n-i})\right)^R.$$

Besides, by definition of $\mathcal{BA}(L)$, state $\frac{s_\pi}{f_\rho}$ belongs to $\hat{S}$ (the set of initial states of automaton $\mathcal{BA}(L)$), and state $\frac{f_\pi}{s_\rho}$ belongs to $\hat{F}$ (the set of final states). And for each $i \in \{0, \ldots, n-1\}$, we have

$$\frac{p_i}{r_{n-i}} \xrightarrow[\mathcal{BA}(L)]{a_i} \frac{p_{i+1}}{r_{n-(i+1)}}$$

by definition of $\mathcal{BA}(L)$. Therefore $L \subseteq \mathcal{L}\Big(\mathcal{BA}(L)\Big)$.

2. Secondly, let us consider some word $u \in \mathcal{L}\Big(\mathcal{BA}(L)\Big)$. Let also $|u| = n$, and $u = a_1 a_2 \ldots a_n$. Then for $\mathcal{BA}(L)$ and u, we can write the following sequence of transitions:

$$\frac{p_0}{r_n} \xrightarrow[\mathcal{BA}(L)]{a_1} \frac{p_1}{r_{n-1}} \xrightarrow[\mathcal{BA}(L)]{a_2} \frac{p_2}{r_{n-2}} \quad \ldots \quad \frac{p_{n-2}}{r_2} \xrightarrow[\mathcal{BA}(L)]{a_{n-1}} \frac{p_{n-1}}{r_1} \xrightarrow[\mathcal{BA}(L)]{a_n} \frac{p_n}{r_0}\,. \tag{6}$$

By definition of $\mathcal{BA}(L)$ we obtain, that $p_0 = s_\pi$, and also

$$p_i \xrightarrow[\widetilde{L}]{a_{i+1}} p_{i+1} \quad \text{for each} \quad i \in \{0, \ldots, n-1\}\,.$$

We have to prove, that $p_n \in F_\pi$.

Let $p_n \notin F_\pi$. By definition of relation # (see [7]) and pair $\frac{p_n}{r_0}$ for it, *there exists* a word $u' \in L$, such that $u' = vw$, where

$$v \in \mathcal{L}^{in}_{\widetilde{L}}(p_n) \quad \text{and} \quad w^R \in \mathcal{L}^{in}_{\widetilde{L^R}}(r_0)\,.$$

By (6), we can think that $v = u$; then $uw \in L$, and therefore $u^R \in \mathcal{L}^{out}_{\widetilde{L^R}}(r_0)$. Then (because $\varepsilon \in \mathcal{L}^{in}_{\widetilde{L^R}}(r_0)$ and automaton $\widetilde{L^R}$ is deterministic) $u^R \in L^R$, therefore $u \in L$. $\square$

4 Some properties of the state-marking functions

In this section we consider some properties of the state-marking functions. They were formulated in [10], and afterwards were used in some other our papers. In this section we shall prove them without other facts, i.e., using the definitions only. All these properties combine in the common expressions the values of input and output languages of the states (i.e., of $\mathcal{L}^{in}$ and $\mathcal{L}^{out}$, see [7]):

- of *any* nondeterministic finite automaton K defining considered regular language;
- of canonical automata for languages $\mathcal{L}(\mathsf{K})$ and $\mathcal{L}(\mathsf{K}^{\mathsf{R}})$.

It is important to remark, that by following Propositions 6 and 7, corresponding languages are also input and output languages of the states of the equivalent *basis* automaton.

The first proposition of this section formulates the *sufficient* condition of the given word: whether or not it belongs to the corresponding output language.

Proposition 2

$$\mathcal{L}^{out}_{\mathsf{K}}(\mathfrak{q}) \subseteq \bigcup_{\widetilde{\mathfrak{q}} \in \varphi^{in}_{\mathsf{K}}(\mathfrak{q})} \mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\widetilde{\mathfrak{q}})\,.$$

Proof. Let for some word ν and state of canonical automaton $\widetilde{\mathfrak{q}} \in \varphi^{in}_{\mathsf{K}}(\mathfrak{q})$ condition $\nu \notin \mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\widetilde{\mathfrak{q}})$ holds. Then we prove, that $\nu \notin \mathcal{L}^{out}_{\mathsf{K}}(\mathfrak{q})$.

Consider some word

$$\mathfrak{u} \in \mathcal{L}^{in}_{\mathsf{K}}(\mathfrak{q}) \cap \mathcal{L}^{in}_{\widetilde{\mathsf{L}}}(\widetilde{\mathfrak{q}})\,;$$

such a word $\mathfrak{u}$ exists by definition of the function φ^{in}. Automaton $\widetilde{\mathsf{L}}$ is deterministic, then $\mathfrak{u}\nu \notin \mathcal{L}(\mathsf{K})$. Therefore, condition $\nu \in \mathcal{L}^{in}_{\mathsf{K}}(\mathfrak{q})$, which is equivalent to $\mathfrak{u}\nu \in \mathcal{L}(\mathsf{K})$, contradicts the equality $\mathcal{L}(\widetilde{\mathsf{L}}) = \mathcal{L}(\mathsf{K})$. □

The "mirror" fact is the following

Proposition 3

$$\mathcal{L}^{in}_{\mathsf{K}}(\mathfrak{q}) \subseteq \Big(\bigcap_{\widetilde{\mathfrak{q}} \in \varphi^{out}_{\mathsf{K}}(\mathfrak{q})} \mathcal{L}^{out}_{\widetilde{\mathsf{L}^{\mathsf{R}}}}(\widetilde{\mathfrak{q}}) \Big)^{\mathsf{R}}. \quad \square$$

In the two following propositions we consider subsets of some language using output languages of the states.

Proposition 4

$$\mathcal{L}^{in}_{K}(q)\cdot\Big(\bigcap_{\widetilde{q}\in\varphi^{in}_{K}(q)}\mathcal{L}^{out}_{\widetilde{L}}(\widetilde{q})\Big)\subseteq\mathcal{L}(K)\,.$$

Proof. Consider any word $u\in\mathcal{L}^{in}_{K}(q)$. By definition of function φ^{in}, for some state $\widetilde{q}\in\varphi^{in}_{K}(q)$, the word u belongs to the language $\mathcal{L}^{in}_{\widetilde{L}}(\widetilde{q})$. For each word

$$v\in\bigcap_{\widetilde{q}\in\varphi^{in}_{K}(q)}\mathcal{L}^{out}_{\widetilde{L}}(\widetilde{q})\,,$$

condition $v\in\mathcal{L}^{out}_{\widetilde{L}}(\widetilde{q})$ holds. Therefore,

$$uv\in\mathcal{L}^{in}_{\widetilde{L}}(\widetilde{q})\cdot\mathcal{L}^{out}_{\widetilde{L}}(\widetilde{q})\subseteq\mathcal{L}(\widetilde{L})=\mathcal{L}(K)\,,$$

and the last condition proves the proposition. □

The "mirror" fact is the following

Proposition 5

$$\Big(\bigcap_{\widetilde{q}\in\varphi^{out}_{K}(q)}\mathcal{L}^{out}_{\widetilde{L^R}}(\widetilde{q})\Big)^{R}\cdot\mathcal{L}^{out}_{K}(q)\subseteq\mathcal{L}(K)\,.\quad\square$$

5 The first example

In this section, we shall consider an example for Proposition 4. For this thing, we shall use the automaton considered in detail in [7, Sect. 3].

For the ease of reading, let us give this figure once again (Fig. 1). And the equivalent canonical automaton (i.e., $\widetilde{L}$) is given on Fig. 2.

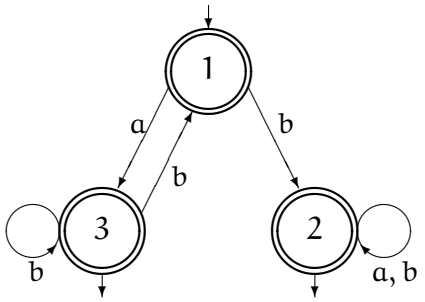

Figure 1

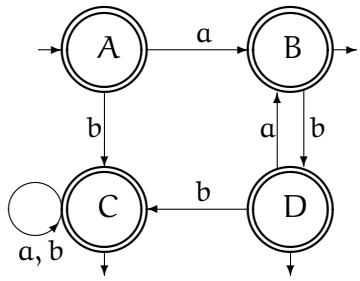

Figure 2

For automaton on Fig. 1, we shall consider state 3. By simple definitions of [7], input language of this state can be defined by the following automaton of Fig. 3:

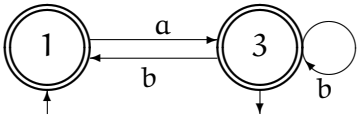

Figure 3

Then it can also be defined by regular expression

$$\mathtt{a}(\mathtt{b}+\mathtt{b}\mathtt{a})^*. \tag{7}$$

For state 3, this language is the first factor of condition of Proposition 4.

By [7, Sect. 3], $\varphi^{in}_{\mathsf{K}}(3) = \{\mathsf{B}, \mathsf{C}, \mathsf{D}\}$. There is evident, that $\mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\mathsf{C}) = \Sigma^*$. Then we have to define the intersection of languages $\mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\mathsf{B})$ and $\mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\mathsf{D})$.

The automaton defining language of their intersection is shown on the following Fig. 4. It can be simply constructed using (deterministic) automaton of Fig. 2. E.g., the (initial) state marked $\mathsf{B}\cap\mathsf{D}$ symbolized the intersection of languages $\mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\mathsf{B})$ and $\mathcal{L}^{out}_{\widetilde{\mathsf{L}}}(\mathsf{D})$. We have

$$\mathsf{B} \xrightarrow[\widetilde{\mathsf{L}}]{\mathtt{b}} \mathsf{D} \quad \text{and} \quad \mathsf{D} \xrightarrow[\widetilde{\mathsf{L}}]{\mathtt{b}} \mathsf{C}\,,$$

then in constructed automaton, we have $\mathsf{B}\cap\mathsf{D} \xrightarrow{\mathtt{b}} \mathsf{C}\cap\mathsf{D}$, etc.

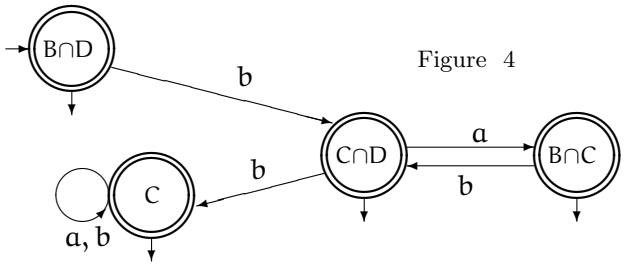

Figure 4

The language of this automaton can also be defined be regular expression

$$\varepsilon + \mathtt{b}(\mathtt{a}\mathtt{b})^*(\mathtt{a} + \varepsilon + \mathtt{b}(\mathtt{a}+\mathtt{b})^*)\,.$$

Then considering the last expression and (7), we obtain, that each word defined by the following expression

$$\mathtt{a}(\mathtt{b}+\mathtt{b}\mathtt{a})^* \cdot (\varepsilon + \mathtt{b}(\mathtt{a}\mathtt{b})^*(\mathtt{a} + \varepsilon + \mathtt{b}(\mathtt{a}+\mathtt{b})^*)))$$

belongs to L.

6 Input and output languages of the states of the basis automaton

In this section we consider properties of input and output languages of the states of the basis automaton; these properties also can be called by the properties of the table of binary relation #. Those are properties, which combine:

- input and output languages of the basis automaton;
- and also input and output languages of two canonical automata (i.e., of automata $\widetilde{L}$ and $\widetilde{L^R}$).

The canonical automaton $\widetilde{L^R}$ contains no more than 2^m-1 states (where m is the number of states of automaton $\widetilde{L}$). [2] Then we can limit by this value the number of possible columns of the table of binary relation #, which has m rows. Besides, this table cannot have duplicate rows and duplicate columns.

The next propositions 6–9 are proved by the definition of the basis automaton.

Proposition 6 *Let there are given some regular language* L *and some state* $A \in Q_\pi$ *of automaton* $\widetilde{L}$*. Then for each state* X *of automaton* $\widetilde{L^R}$*, such that* $A\#X$*, the following condition holds:*

$$\mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X} = \mathcal{L}^{in}_{\widetilde{L}}(A).$$

Proof. Condition

$$\mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X} \subseteq \mathcal{L}^{in}_{\widetilde{L}}(A)$$

is the direct consequence of the definition of automaton $\mathcal{BA}(L)$. Let us prove the reverse inclusion

$$\mathcal{L}^{in}_{\widetilde{L}}(A) \subseteq \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X},$$

i.e., that for every word $w \in \Sigma^*$, the following fact holds:

$$w \in \mathcal{L}^{in}_{\widetilde{L}}(A) \quad \text{implies} \quad w \in \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X}.$$

We shall prove this fact by induction by $|w|$.

The basis of induction (i.e., $w = \varepsilon$) is evident, because if $\varepsilon \in \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X}$ then $\varepsilon \in \mathcal{L}^{in}_{\widetilde{L}}(A)$. Then let us prove *the step of induction*.

[2] Remark once again, that we consider canonical automaton without possible "dead" state.

Let $w = w'a$, where $w' \in \Sigma^*$ and $a \in \Sigma$; let also $w \in \mathcal{L}^{in}_{\widetilde{L}}(A)$. Because $A \# X$, there exist words $u, v \in \Sigma^*$, such that

$$uv \in L\,, \quad u \in \mathcal{L}^{in}_{\widetilde{L}}(A) \quad \text{and} \quad v^R \in \mathcal{L}^{in}_{\widetilde{L^R}}(X).$$

Because w also belongs to the language $\mathcal{L}^{in}_{\widetilde{L}}(A)$, we obtain, that $wv \in L$, i.e., $w'av \in L$. Let also:

- B be some state of automaton $\widetilde{L}$, such that $w' \in \mathcal{L}^{in}_{\widetilde{L}}(B)$;
- Y be some state of automaton $\widetilde{L^R}$, such that $v^R a \in \mathcal{L}^{in}_{\widetilde{L^R}}(Y)$;

both the states (B and Y) do exist, because $w'av \in L$. Then $B \# Y$, and, by the induction hypothesis, $w' \in \mathcal{L}^{in}_{\mathcal{BA}(L)}((B, Y))$. And using the fact $\delta_{\mathcal{T}}((B, Y), a) \ni \binom{A}{X}$, we obtain that $w \in \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X}$. □

The "mirror" fact is the following

Proposition 7 *Let there are given some regular language L and some state $X \in Q_\rho$ of automaton $\widetilde{L^R}$. Then for each state A of automaton $\widetilde{L}$, the following condition holds:*

$$\mathcal{L}^{out}_{\mathcal{BA}(L)}\binom{A}{X} = \left(\mathcal{L}^{in}_{\widetilde{L^R}}(X)\right)^R = \mathcal{L}^{out}_{(\widetilde{L^R})^R}(X). \quad \square$$

Proposition 8 *Let there is given some regular language L. Then for each state $A \in Q_\pi$ of automaton $\widetilde{L}$, the following condition holds:*

$$\mathcal{L}^{out}_{\widetilde{L}}(A) = \bigcup_{X \in Q_\pi} \mathcal{L}^{out}_{\mathcal{BA}(L)}\binom{A}{X}.$$

Proof. Consider some word $uv \in L$, such that

$$u \in \mathcal{L}^{in}_{\widetilde{L}}(A) \quad \text{and} \quad v \in \mathcal{L}^{out}_{\widetilde{L}}(A).$$

By [11] (and also by consequence of the proof of Proposition 1), automaton $\mathcal{BA}(L)$ has the only accepting path for the word uv, and for some $x \in Q_\rho$ the following conditions hold:

$$u \in \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X} \quad \text{and} \quad v \in \mathcal{L}^{out}_{\mathcal{BA}(L)}\binom{A}{X}.$$

Then combining all the possible ν, we obtain, that

$$\mathcal{L}^{out}_{\widetilde{L}}(A) \subseteq \bigcup_{X \in Q_\pi} \mathcal{L}^{out}_{\mathcal{BA}(L)}\binom{A}{X}.$$

The reverse inclusion, i.e., that

$$\bigcup_{X \in Q_\pi} \mathcal{L}^{out}_{\mathcal{BA}(L)}\binom{A}{X} \subseteq \mathcal{L}^{out}_{\widetilde{L}}(A),$$

is evident. □

The "mirror" fact is the following

Proposition 9 *Let there is given some regular language* L. *Then for each state* $X \in Q_\rho$ *of automaton* $\widetilde{L^R}$, *the following condition holds:*

$$\left(\mathcal{L}^{out}_{\widetilde{L^R}}(X)\right)^R = \mathcal{L}^{in}_{(\widetilde{L^R})^R}(X) = \bigcup_{A \in Q_\rho} \mathcal{L}^{in}_{\mathcal{BA}(L)}\binom{A}{X}. \quad \square$$

Proposition 10 *Let canonical automaton* $\widetilde{L}$ *for the given regular language* L *has at least* 2 *states, and* $A, B \in Q_\pi$ *is a pair of such states. Then there exists a state* $X \in Q_\rho$ *of automaton* $\widetilde{L^R}$, *such that automaton* $\mathcal{BA}(L)$ *contains exactly one state of the following* 2 *ones:* $\binom{A}{X}$ *and* $\binom{B}{X}$.

Proof. This proposition can be considered as a consequence of the classical algorithm of constructing of canonical automaton (which includes imperative combining equivalent states) and also [7, Th. 4.1]. □

The "mirror" fact is the following

Proposition 11 *Let canonical automaton* $\widetilde{L^R}$ *for the mirror image of the given regular language* L *has at least* 2 *states, and* $X, Y \in Q_\rho$ *is a pair of such states. Then there exists a state* $A \in Q_\pi$ *of automaton* $\widetilde{L}$, *such that automaton* $\mathcal{BA}(L)$ *contains exactly one state of the following* 2 *ones:* $\binom{A}{X}$ *and* $\binom{A}{Y}$. □

Two the following propositions (we briefly talked about them at the beginning of this section) are the direct consequences of two previous propositions. They formulate facts about the possible size of the table of binary relation # for the given regular language.

Proposition 12 *Let regular language* L *be given, and automaton* $\widetilde{L}$ *contains exactly* n *states (i.e.,* $|Q_\pi| = m$*). Then automaton* $\widetilde{L^R}$ *contains no more than* 2^m-1 *states (i.e.,* $|Q_\rho| \le 2^m-1$*).* □

Proposition 13 *Let regular language* L *be given, and automaton* $\widetilde{L^R}$ *contains exactly* n *states (i.e.,* $|Q_\rho| = n$*). Then automaton* $\widetilde{L}$ *contains no more than* 2^n-1 *states (i.e.,* $|Q_\pi| \le 2^n-1$*).* □

In the next section we shall obtain, that such maximums (i.e., the values 2^m-1 and 2^n-1) can be achieved.

7 On the possible set of the states of the basis automaton

In this section we shall formulate some properties for *all the possible* states of a basis automaton (or, in other words, all the possible variants of binary relation #). We shall obtain, that if there hold all the limitations formulated in previous section for the table of the binary relation #, then such table can really describe such relation for some regular language. Besides, such proof is *constructive*, i.e., we obtain *an algorithm* of constructing the basis automaton for regular language having such table of *a priori given* binary relation #.

Proposition 14 *Let binary relation # be given, and for it, all the limitations formulated before hold.*[3] *Then there exists a regular language, for which corresponding binary relation # coincides with the given one.*

Proof. Thus, we think, that the sets of states Q_π (where $|Q_\pi| = m$) and Q_ρ (where $|Q_\rho| = n$) are already given. Binary relation $\# \subseteq Q_\pi \times Q_\rho$ is also given. These objects satisfy all the limitations formulated before.

[3] Let us formulate them once again, for the table of #. Let $|Q_\pi| = m$ and $|Q_\rho| = n$. Then:

- $1 \le m \le 2^n-1$;
- $1 \le n \le 2^m-1$;
- there are no duplicate rows;
- there are no duplicate columns;
- there are no empty rows (i.e., there are no row A, such that $A\#X$ holds for none column X);
- there are no empty columns.

Consider the following alphabet:

$$\Sigma_{\#} = \left\{ a_{X}^{A} \;\middle|\; A \in Q_\pi,\, X \in Q_\rho \right\}.$$

Over this alphabet, consider the arbitrary states $s_\pi \in Q_\pi$ and $s_\rho \in Q_\rho$. For them, consider following automaton

$$K^{\#s_\pi s_\rho} = (\, Q_\pi, \Sigma_{\#}, \delta_\pi, \{s_\pi\}, F_\pi \,)$$

(or, briefly, $K^{\#}$, when s_π and s_ρ are meant), where:

- $F_\pi = \{\, f_\pi \in Q_\pi \,|\, f_\pi \# s_\rho \,\}$;[4]
- transition function δ_π is defined in the following way:

$$\text{for each } A, B \in Q_\pi \text{ and } X \in Q_\rho\,, \quad \delta_\pi(A, a_{X}^{B}) = \begin{cases} \{B\}, & \text{if} \quad A \# X\,; \\ \emptyset, & \text{otherwise} \end{cases}$$

(we allow for the possibility $A = B$).

Let us prove that the language $\mathcal{L}(K^{\#})$ is the desired one.

By the construction, automaton $\mathcal{L}(K^{\#})$ is deterministic. The conditions of Proposition 10 hold, because we made automaton using relation corresponding #; therefore automaton $K^{\#}$ has no pairs of equivalent states. Also by the construction, the transition graph of is automaton is strongly connected ([3]); then it contains no useless states. Therefore, automaton $K^{\#}$ is canonical (for its language), i.e.,

$$\text{for the language} \quad L^{\#} = \mathcal{L}(K^{\#})\,, \quad \text{we have} \quad K^{\#} = \widetilde{L^{\#}}\,.$$

Over the same alphabet $\Sigma_{\#}$, consider also automaton

$$K_{\#s_\pi s_\rho} = (\, Q_\rho, \Sigma_{\#}, \delta_\rho, \{s_\rho\}, F_\rho \,)$$

(or, briefly, $K_{\#}$, when s_π and s_ρ are meant), where:

- $F_\rho = \{\, f_\rho \in Q_\pi \,|\, s_\pi \# f_\rho \,\}$;

[4] Such a choice is possible, because of limitations formulated before. Remark also, that choosing various $s_\pi \in Q_\pi$ and $s_\rho \in Q_\rho$, we obtain a *set* of languages having the given binary relation #.

- transition function δ_ρ is defined in the following way:

$$\text{for each } B \in Q_\pi \text{ and } X, Y \in Q_\rho\,,\ \ \delta_\rho(Y, a_{\substack{B\\X}}) = \begin{cases} \{X\}, & \text{if } B\#Y\,; \\ \emptyset, & \text{otherwise} \end{cases}$$

  (we allow for the possibility $X = Y$).

Like automaton $K^\#$ we can prove, that automaton $K_\#$ is canonical (for its language), i.e., for the language $L_\# = \mathcal{L}(K_\#)$, we have $K_\# = \widetilde{L_\#}$.

Let us prove, that $L_\# = (L^\#)^R$. For this thing, consider any word $u \in L^\#$. Let

$$u = a_{\substack{A_1\\X_1}} a_{\substack{A_2\\X_2}} \dots a_{\substack{A_k\\X_k}}\,.$$

Then we can write the following sequence of transitions of canonical automaton $K^\# = \widetilde{L^\#}$ while readind the word u:

$$s_\pi \xrightarrow[K^\#]{a_{\substack{A_1\\X_1}}} A_1 \xrightarrow[K^\#]{a_{\substack{A_2\\X_2}}} A_2 \ \dots\ A_{k-1} \xrightarrow[K^\#]{a_{\substack{A_k\\X_k}}} A_k\,, \quad \text{where} \quad A_k \in F_\pi\,.$$

Since $A_k \in F_\pi$, we have $A_k \# s_\rho$. Then for this sequence, we can construct the following sequence of transitions of automaton $K_\# = \widetilde{L_\#}$:

$$s_\rho \xrightarrow[K_\#]{a_{\substack{A_k\\X_k}}} X_{k-1} \ \dots\ X_2 \xrightarrow[K_\#]{a_{\substack{A_2\\X_2}}} X_1 \xrightarrow[K_\#]{a_{\substack{A_1\\X_1}}} X_0$$

for the sequence of states $X_{k-1}, \dots, X_2, X_1$ selected before and some new state X_0. Like the proof of Proposition 1, we obtain that $X_0 \in F_\rho$.

We proved that $L^\# \subseteq (L_\#)^R$. The reverse inclusion, i.e., $(L_\#)^R \subseteq L^\#$, can be proved similarly.

Thus, automata $K^\#$ and $K_\#$ are canonical automata for the languages $L^\#$ and $(L^\#)^R = L_\#$. Then for them, we can construct the following *basis* automaton

$$\mathcal{BA}(L^\#) = (\,\mathcal{T}, \Sigma_\#, \delta_T, S_T, F_T\,),$$

(over the alphabet $\Sigma_\#$ defined before), where:

- $\mathcal{T} = \left\{ \begin{smallmatrix} A\\X \end{smallmatrix} \,\middle|\, A \in Q_\pi,\, X \in Q_\rho,\, A\#X \right\}$;
- for each $A, B \in Q_\pi$ and $X \in Q_\rho$, such that $A\#X$ (we admit the possibility of $A = B$), we set

$$\delta_T\left(\begin{smallmatrix} A\\X \end{smallmatrix}, a_{\substack{B\\X}}\right) = \left\{ \begin{smallmatrix} B\\Y \end{smallmatrix} \,\middle|\, (\exists Y \in Q_\rho)\,(B\#Y) \right\};$$

- for each other cases $a \in \Sigma_{\#}$,[5] $A, B \in Q_\pi$ and $X, Y \in Q_\rho$, we set

$$\delta_T(\begin{smallmatrix} A \\ X \end{smallmatrix}, a) = \emptyset;$$

- $S_T = \left\{ \begin{smallmatrix} s_\pi \\ X \end{smallmatrix} \;\middle|\; s_\pi \# X \right\}$;
- $S_T = \left\{ \begin{smallmatrix} A \\ s_\rho \end{smallmatrix} \;\middle|\; A \# s_\rho \right\}$;

(where the states s_π and s_ρ were also previously selected). This automaton is $\mathcal{BA}(L^{\#})$ by the process of its constructing. And also by its constructing, its set of states $\mathcal{T}$ forms the given binary relation #. □

As a consequence of the Proposition 14 we obtain, that these maximums of the number of states of two canonical automata (i.e., the values 2^m-1 and 2^n-1) can be achieved. But there is important to remark the following thing. In some books ([1] etc.), there are examples, when the given automaton contains n states, and the equivalent canonical automaton contains 2^n-1 states.[6] This fact (i.e., the possible fulfilment the upper bound 2^n-1) is *not* a consequence of these results: it proved there for *arbitrary* nondeterministic finite automata (having no limitations), and we consider it for automata which are mirror automata for canonical ones. E.g. we can say, that these automata are *not* deterministic, but they are unambiguous.

8 The second example

Let us consider a simple example for automata defined in previous section. However, we describe the whole process of constructing detailed.

Thus, let us consider the following binary relation #:

#	X	Y
A	#	–
B	#	#

Table 1

Certainly, this relation satisfies all the limitations formulated before.

[5] Remark once again, that we consider automata without ε-edges.

[6] Or 2^n states, when we assume the possible "dead" state, considering the canonical automaton as a total automaton.

By the previous section, corresponding alphabet $\Sigma_{\#}$ is the following:

$$\Sigma_{\#} = \left\{ a_{X}^{A}, a_{X}^{B}, a_{Y}^{A}, a_{Y}^{B} \right\}.$$

Let $s_\pi = A$, $s_\rho = Y$ (also by the previous section, we can choose such states). Then $F_\pi = \{B\}$, $F_\rho = \{X\}$, and we obtain the following canonical automaton for the language $L^{\#}$:

	$L^{\#}$	a_{X}^{A}	a_{X}^{B}	a_{Y}^{A}	a_{Y}^{B}
$\rightarrow$	A	A	B	–	–
$\leftarrow$	B	A	B	A	B

Table 2

For the convenience, let us rename the letters in the following way:

$$a_{X}^{A} = a, \quad a_{X}^{B} = b, \quad a_{Y}^{A} = c, \quad a_{Y}^{B} = d.$$

Then we can rewrite the considered automaton by the following Table 3 or Fig. 5:

	$L^{\#}$	a	b	c	d
$\rightarrow$	A	A	B	–	–
$\leftarrow$	B	A	B	A	B

Table 3

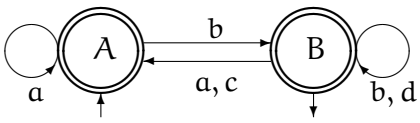

Figure 5

The mirror automaton $(L^{\#})^{R}$ is the following (Table 4 or Fig. 6; for the table of nondeterministic automaton, we use the agreements of [7]):

	$(L^{\#})^{R}$	a	b	c	d
$\leftarrow$	A	A, B	–	B	–
$\rightarrow$	B	–	A, B	–	B

Table 4

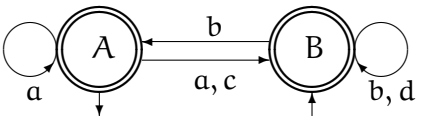

Figure 6

The process of determinization is given by the following table:

		a	b	c	d
→	B	–	A, B	–	B
←	A, B	A, B	A, B	B	B

Table 5

Renaming {B} for Y and {A, B} for X, we obtain the following automaton $\widetilde{L'} = L_{\#}$ (where $L' = (L^{\#})^{R}$; see Table 6 or Fig. 7):

	$L_{\#}$	a	b	c	d
→	Y	–	X	–	Y
←	X	X	X	Y	Y

Table 6

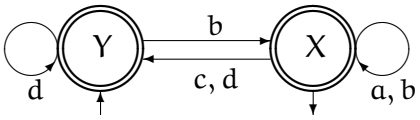

Figure 7

And using the last automaton, we obtain automata $(L_{\#})^{R}$ and then $\mathcal{BA}(L^{\#})$ (Fig. 8 and 9).

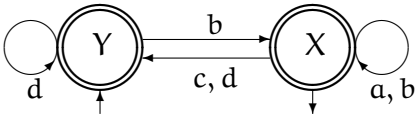

Figure 8

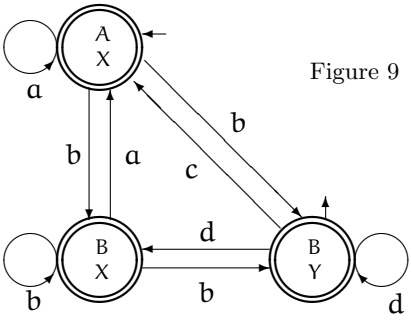

Figure 9

Certainly, they both also define the language $L^{\#}$. (Compare the given Table 1 and the obtained Fig. 9.)

9 Conclusion

Let us describe some possible problems for the future solution. Thus, we are going to:

- to show the relationship between the basis automaton and Conway's universal automaton;

- to use some propositions proved in this paper to describe the minimization algorithms for nondeterministic finite automata;
- vice versa, to use automaton $\mathsf{K}^{\#}$ to describe algorithms of automatic constructing some counter-examples for the algorithms of state minimization (the most famous example of such a counter-example is so called automaton Waterloo, [4]).

References

[1] W. Brauer, *Automatentheorie. Eine Einführung in die Theorie endlicher Automaten.* Teubner, 1984. ⇒241

[2] J. Conway, *Regular Algebra and Finite Machines.* Chapman and Hall, 1971. ⇒227

[3] F. Harary, *Graph Theory.* Addison-Wesley, 1969. ⇒239

[4] T. Kameda, P. Weiner, On the state minimization of nondeterministic finite automata, *IEEE Trans. on Comp.,* **C-19,** 7 (1970) 617–627. ⇒244

[5] S. Lombardy, J. Sakarovitch, The Universal Automaton, *Logic and Automata, Texts in Logic and Games, Amsterdam Univ. Press,* **2** (2008) 457–504. ⇒227

[6] B. Melnikov, On an expansion of nondeterministic finite automata, *J. of Applied Math. and Computing (The Korean J. of Comp. and Appl. Math.),* **24,** 1–2 (2007) 155–165. ⇒227, 228, 229, 230

[7] B. Melnikov, Once more on the edge-minimization of nondeterministic finite automata and the connected problems, *Fundamenta Informaticae,* **104,** 3 (2010) 267–283. ⇒228, 229, 231, 232, 233, 234, 237, 242

[8] B. Melnikov, A. Melnikova, Some properties of the basis finite automaton, *J. of Applied Math. and Computing (The Korean J. of Comp. and Appl. Math.),* **9,** 1 (2002) 131–150. ⇒227, 228

[9] B. Melnikov, A. Melnikova, A new algorithm of constructing the basis finite automaton, *Informatica (Lithuanian Acad. Sci. Ed.),* **13,** 3 (2002) 299–310. ⇒227

[10] B. B. Melnikov, N. Sciarini-Guryanova, Possible edges of a finite automaton defining a given regular language, *J. of Applied Math. and Computing (The Korean J. of Comp. and Appl. Math.),* **9,** 2 (2002) 475–485. ⇒232

[11] A. Vakhitova, The basis automaton for the given regular language, *J. of Applied Math. and Computing (The Korean J. of Comp. and Appl. Math.),* **6,** 3 (1999) 617–624. ⇒227, 230, 236

Received: August 2, 2013 • Revised: December 27, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 245–270

Parallel enumeration of degree sequences of simple graphs II

Antal IVÁNYI
Eötvös Loránd University,
Faculty of Informatics
Budapest
email: tony@onf.elte.hu

Loránd LUCZ
Eötvös Loránd University,
Faculty of Informatics
Budapest
email: lorand.lucz@gmail.com

Gergő GOMBOS
Eötvös Loránd University,
Faculty of Informatics
Budapest
email: ggombos@inf.elte.hu

Tamás MATUSZKA
Eötvös Loránd University,
Faculty of Informatics
Budapest
email: matuszka1987@gmail.com

Abstract. In the paper we report on the parallel enumeration of the degree sequences (their number is denoted by $G(n)$) and zerofree degree sequences (their number is denoted by $(G_z(n))$ of simple graphs on $n = 30$ and $n = 31$ vertices. Among others we obtained that the number of zerofree degree sequences of graphs on $n = 30$ vertices is $G_z(30) = 5\,876\,236\,938\,019\,300$ and on $n = 31$ vertices is $G_z(31) = 22\,974\,847\,474\,172\,374$. Due to Corollary 21 in [52] these results give the number of degree sequences of simple graphs on 30 and 31 vertices.

1 Introduction

In the practice an often occuring problem is the ranking of different objects (examples can be found e.g. in [52]), assigning points to the objects and then ranking of the objects on the base of the sum of the assigned to them points.

Computing Classification System 1998: G.2.2.
Mathematics Subject Classification 2010: 05C85, 68R10
Key words and phrases: simple directed graphs, linear Erdős-Gallai and Havel-Hakimi algorithm, enumeration of graphical sequences

Especially extensive bibliography has the case when the results are represented by a simple graph and the problem is the test, reconstruction and enumeration of the degree sequences. Havel in 1955 [42], Erdős and Gallai in 1960 [16, 32, 77], Hakimi in 1962 [39], Knuth in 2008 [61], Tripathi et al. in 2010 [89] proposed a method to decide, whether a sequence of nonnegative integers can be the degree sequence of a simple graph. Sierksma and Hoogeven in 1991 [83] compared seven known methods. The running time of their algorithms in worst case is $\Omega(n^2)$. In 2007 Takahashi [86], in 2009 Hell and Kirkpatrick [43], in 2011 Iványi et al. [52] and in April of 2012 Király [58] proposed an algorithm, whose worst running time is $\Theta(n)$.

There are several new proofs for the classical Havel-Hakimi and Erdős-Gallai theorems [26, 32, 63, 70, 75, 87, 88, 89].

Extensions of the algorithms for $(0, b)$-graphs [8, 9, 24, 23, 25, 27, 69, 75, 90, 92] and (a, b)-graphs [44, 45, 46, 53] are also known.

As an application of our linear time algorithm we describe Erdős-Gallai-Enumerative algorithm (EGE) and its parallel version used to enumerate the different degree sequences of simple graphs for 30 vertices. We also present the linear test version of Havel-Hakimi algorithm (HHL).

Let $n \geq 1$. We call a sequence $\mathbf{s} = (s_1, \ldots, s_n)$ (l, u, n)-*bounded*, if $0 \leq s_i \leq n$ for $i = 1, \ldots, n$, n-*bounded*, if it is $(0, n-1, n)$-bounded, *n-regular*, if the conditions $n - 1 \geq s_1 \geq \cdots \geq s_n \geq 0$ hold, and n-*even*, if the sum of the elements of $\mathbf{s}$ is even. If there exists a graph with n vertices which has the degree sequence $\mathbf{s}$, then we say that $\mathbf{s}$ is n-*graphical*. If such graph does not exist, then we say that $\mathbf{s}$ is *nongraphical*. A sequence is *zerofree*, if it does not contain zero. If n is not necessary, then we omit it in the terms n-bounded, n-regular, n-even and n-graphical. The first i elements of an n-regular $\mathbf{s}$ are called *the head*, and the last $n-i$ elements are called *the tail*, belonging to the element i of $\mathbf{s}$.

2 Earlier results

A classical problem of the graph theory is the enumeration of the sorted degree sequences of different graphs—among others simple graphs. For example *The On-Line Encyclopedia of Integer Sequences* contains for $n = 1, \ldots, 29$ vertices the number of degree sequences of simple graphs (the values for $n = 20, \ldots, 23$ were set in July of 2011 by Nathann Cohen [28], and for $24, \ldots, 29$ in 15 November, 2011 by us [48, 52]) and the number of zerofree degree sequences of simple graphs (the values for $n = 1, \ldots, 9$ were set in 12 June, 2004 by

N. J. Sloane, for $n = 10, \ldots, 20$ in 12 August, 2006 by Gordon Royle, for $n = 21, 22$, and 23 in August 31, 2011, and in December 10, 2012 by Frank Ruskey [80], and the values for $n = 24, \ldots, 29$ by us [50, 51].

In this section we review the theoretical and practical results connected with the enumeration of simple graphs.

2.1 Exact enumeration results

It is known [52, equation (23)] that if $n \geq 1$, then the number $R(n)$ of the regular sequences is

$$R(n) = \binom{2n-1}{n} \tag{1}$$

and the number $R_z(n)$ of the zerofree regular sequences is [52, equation (24)]

$$R_z(n) = \binom{2n-2}{n} \tag{2}$$

implying [52]

$$\lim_{n\to\infty} \frac{R(n+1)}{R(n)} = \lim_{n\to\infty} \frac{R_z(n+1)}{R_z(n)} = 4 \tag{3}$$

and

$$\lim_{n\to\infty} \frac{R_z(n)}{R(n)} = \frac{1}{2}, \tag{4}$$

and

$$R(n) = \frac{4^n}{2\sqrt{\pi n}} + O\left(\frac{4^n}{n^{3/2}}\right) \tag{5}$$

and

$$R_z(n) = \frac{4^n}{4\sqrt{\pi n}} + O\left(\frac{4^n}{n^{3/2}}\right). \tag{6}$$

Table 1 in [52] shows the values values of $R(n)$ for $n = 1, \ldots, 38$, Table 4 in [51] for $n = 39, \ldots, 60$, and in [47, 51, 68] the values are presented for $n = 1, \ldots, 1200$. Table 1 in Subsection 3.3 presents the values $R(n)/R(n+1)$ for $n = 1, \ldots, 32$ and [68] for $n = 1, \ldots, 1200$.

Figure 1 in Subsection 3.3 shows the values of $R_z(n)/R_z(n+1))$ for $n = 1, \ldots, 32$.

In 1987 Ascher derived the following explicit formula for the number $E(n)$ of even sequences.

Lemma 1 (Ascher [2], Sloane and Pfoffe [85]) *If* $n \geq 1$, *then the number of even sequences* $E(n)$ *is*

$$E(n) = \frac{1}{2}\left(\binom{2n-1}{n} + \binom{n-1}{\lfloor n/2 \rfloor}\right). \tag{7}$$

Proof. See [2]. □

Table 1 in [52] contains the values of $E(n)$ and $E(n+1)/E(n)$ for $n = 1, \ldots, 31$.

(7) implies (see [52])

$$\lim_{n\to\infty} \frac{E(n+1)}{E(n)} = 4 \tag{8}$$

and

$$E(n) = \frac{4^n}{8\sqrt{\pi n}} + O\left(\frac{4^n}{n^{3/2}}\right). \tag{9}$$

further (1) and (7) imply

$$\lim_{n\to\infty} \frac{E(n)}{R(n)} = \frac{1}{2}, \tag{10}$$

(2) and (7) imply

$$\frac{R_z(n)}{E(n)} = \frac{2n-2}{2n-1} = 1 - \frac{1}{2n-1} \quad \text{and} \quad \lim_{n\to\infty} \frac{R_z(n)}{E(n)} = 1. \tag{11}$$

Table 1 in [52] shows the values of $E(n)$ for $n = 1, \ldots, 38$, Table 4 in [51] for $n = 39, \ldots, 60$, the *list* of [64] for $n = 1, \ldots, 1000$, and [68] for $n = 31, \ldots, 1200$.

Figure 3 in [52] shows the values of $E_z(n)$ for $n = 1, \ldots, 20$, and [68] $n = 1, \ldots, 1200$. Table 5 in [51] shows the values of $E_z(n/R(n)$ for $n = 1, \ldots, 20$.

Using (1) and (7) we computed $E(n)$ and $E(n+1)/E(n)$ for $i = 1, \ldots, 750$ (see [52, 68]). Recently Librandi [64] published the values of $E(n)$ up to $n = 1000$ and we continued the computations up to 1200 [51, 68].

The following theorem gives a very useful connection between the values of $G(n)$ and $G_z(n)$: it helped to decrease the computing time of $G(29)$ with about 50 %.

Lemma 2 (Iványi, Lucz, Móri, Sótér [52]) *If* $n \geq 2$, *then the number of* n*-graphical sequences* $G(n)$ *can be computed from the number of* $(n-1)$*-graphical sequences* $G(n-1)$ *and the number of* n*-graphical zero-free sequences* G_z*:*

$$G(n) = G(n-1) + G_z(n),$$

and if $n \geq 1$ *then*

$$G(n) = 1 + \sum_{i=2}^{n} G_z(i).$$

Proof. If an even sequence $s = (s_1, \ldots, s_n)$ contains at least one zero, then $s_n = 0$ and $s' = (s_1, \ldots, s_{n-1})$ is graphical or not. If $\mathbf{a} = (a_1, \ldots, a_{n-1})$ is $(n-1)$-graphical, then $\mathbf{a}' = (a_1, \ldots, a_{n-1}, 0)$ is n-graphical.

The set of the n-graphical sequences $\mathcal{S}(n)$ consists of two subsets: set of zerofree sequences $\mathcal{S}_z(n)$ and the set of the remaining sequences $\mathcal{S}_0(n)$. There is a bijection between the set of the $(n-1)$-graphical sequences and such n-graphical sequences, which contain at least one zero. Therefore $|\mathcal{S}| = |\mathcal{S}_z| + |\mathcal{S}_0| = G_z(n) + G(n-1)$. □

Using the parallel version EGP (see the next section) of EGE we computed $G(n)$ up to $n = 29$. These numbers can be found in Table 2 of [52].

Theorem 3 (Burns [22]) *There exist positive constants* c *and* C *such that the following bounds of the function* $G(n)$ *are true for* $n \geq 1$*:*

$$\frac{4^n}{cn} < G(n) < \frac{4^n}{(\log n)^C \sqrt{n}}. \tag{12}$$

Proof. See [22]. □

This result implies that the asymptotic density of the graphical sequences is zero among the even sequences.

Corollary 4 *If* $n \geq 1$*, then there exists a positive constant* C *such that*

$$\frac{G(n)}{E(n)} < \frac{1}{(\log_2 n)^C} \tag{13}$$

and

$$\lim_{n\to\infty} \frac{G(n)}{E(n)} = 0. \tag{14}$$

Proof. (13) is a direct consequence of (7) and (12). □

Table 1 in [52] contains the values of $G(n)$ and $G(n+1)/G(n)$ for $n = 1, \ldots, 29$. Table 5 in [51] contains values of $G_z(n)$, $G_z(n)/R(n)$, and $G(n)/R(n)$ for $n = 1, \ldots, 29$.

We remark that a zerofree degree sequence belongs to a graph not containing isolated vertex, therefore the number of zerofree graphical degree sequences

$G_z(n)$ is at the same time also the number of degree sequences of simple graphs, not containing isolated vertex.

There are several classic asymptotic results, e.g. due to Bender and Canfield [7], Bollobás [17, 18, 19], Harary and Palmer [41], Kleitman and Winston [56, 60], Reid [78], Winston and Kleitman [91]. A modern direction is to get approximate results by sampling of random graphs (see e.g. the papers of Erdős, Király and Miklós [34], further of Miklós, Erdős and Soukup [**?**].

An interesting connected problem is the characterization of pairs of different directed graphs having a pair of prescribed indegree and outdegree sequences [8, 9, 10, 11, 12, 14, 15, 20, 40, 72, 76, 81].

Another interesting related questions are the unicity of the realizations of the degree sequences [29, 55, 62, 82] and the parallel realization of degree sequences [1].

Several recent papers consider the problem of approximate enumeration of the number of all realizations of simple graphs (see e.g. [13, 34, 35, 36, 37, 38, 59, 71]). In 1978 Bender and Canfield [7] characterized the asymptotic number of realizations of given graphical degree sequences, while in 2012 Zoltán Király [58] proposed an algorithm which with polynomial delay lists all realizations of a given graphical sequence.

2.2 Earlier algorithmic results

In this subsection the linear Havel-Hakimi algorithm (HHL) based on Havel-Hakimi theorem [39, 42] and the enumerating Erdős-Gallai algorithm (EGE) based on Erdős-Gallai theorem [32] are shortly described.

2.2.1 Linear Havel-Hakimi algorithm (HHL)

In a previous paper [52] we described the classical Havel-Hakimi [39, 42] and Erdős-Gallai [32] algorithms and their some improvements as linear Erdős-Gallai (EGL) and jumping Erdős-Gallai (EGLJ) algorithms.

it is worth to remark that our linear Erdős-Gallai algorithm is applied in the solution of different problems connected with degree sequences [5, 6, 21, 31].

Here we present the linear version of Havel-Hakimi algorithm (HHL) [46] and compare it with the previous linear algorithms EGL and EGLJ [52]. It is important to remark that this linear version of HH only tests the investigated sequences without their reconstruction.

In the worst case the original Havel-Hakimi algorithm requires quadratic time to test the $(0, 1, n)$-regular sequences. Using the new concepts weight

point and reserve we reduced the worst running time to $O(n)$.

Let $s = (s_1, \ldots, s_n)$ be a potential graphical sequence. The definition of the *weight point* w_i belonging to s_i was introduced in [52] in connection with ERDŐS-GALLAI-LINEAR: if $s_1 \geq i$, then w_i is the largest k ($1 \leq k \leq n$) having the property $s_k \geq i$. But if $s_1 < i$, then $w_i = 0$. EGL exploits the property w_i ensuring that if $i \leq w_i$, then the key expression $\min j, s_k$ in the Erdős-Gallai theorem equals i, otherwise equals s_k.

In HHL the weight point w_i determines the increment of the tail capacity when we switch to the investigation of the next element of s.

The reserve r_i belonging to s_i is defined as the unused part of the actual tail capacity and can be computed by the formulas

$$r_1 = w_1 - 1 - s_1 \tag{15}$$

and

$$r_i = w_i + r_{i-1} - s_i \quad \text{for} \quad 2 \leq i \leq n-1. \tag{16}$$

Theorem 5 *The running time of* HAVEL-HAKIMI-LINEAR *is in best case* $\Theta(1)$*, and in worst case it is* $\Theta(n)$.

Proof. If the condition in line 1 or 3 holds, then the running time is $\Theta(1)$. If not, then we decrease the actual w at most n times and the remaining operations require $O(1)$ operations for all reductions. □

2.2.2 Enumerating Erdős-Gallai algorithm (EGE)

A classical problem of the graph theory is the enumeration of the degree sequences of different graphs—among others simple graphs. For example *The On-Line Encyclopedia of Integer Sequences* [84] contains for $n = 1, \ldots, 30$ vertices the number of degree sequences of simple graphs (the values for $n = 20, \ldots, 23$ were set in July of 2011 by Nathann Cohen, in November 15, 2011 for $24, \ldots, 29$ and in 29 July of 2013 for $n = 30$ by us [48]).

We applied the new quick EGL to get these numbers for larger values of n.

Our starting point was to test all regular sequences and so to enumerate the graphical ones. Equation 1 gives the number of regular sequences.

According to Erdős-Gallai theorem [32] the sum of the elements of a graphical sequence is always even. Therefore it is sufficient to test only the even sequences. In 1987 Ascher [2] derived Lemma 1, containing an explicit formula for the number of even sequences $E(n)$.

According to Lemma 2 it is enough to test only the zerofree even sequences.

This lemma was the base of Erdős-Gallai Enumerative algorithm (EGE) used to enumerate the graphical sequences for $n = 23, \ldots, 29$ [51].

We enumerated the graphical sequences of simple graphs on $n = 30$ and 31 vertices using algorithm EGE2. The running time of EGE was substantially (with about 30 %) decreased due to Lemma 9.

We prepare the enumeration of degree sequences of simple graphs on 32 vertices. The running time of EGE2 would be about 320 years for a computer with one processor having 2,2 GHz speed. We wish to decrease the running time of EGE2 using Lemmas 10 and 11.

2.3 Earlier simulation results

The papers [44, 45, 46, 51, 52, 66] and OEIS [64, 73, 74] contain many simulation results. We describe them together with the new results in Subsection 3.3.

It is worth to mention other methods of enumeration of graph sequences as generation of random graphs (e.g. [65] and generation of graphical partitions (see e.g. [3, 4, 30, 33].

3 New results

In this section we describe the new mathematical and simulation results.

3.1 New enumerative results

At first we give a new formula for the number of zerofree even sequences. This formula is more sophisticated than Ascher's formula, and its application requires more time, but it has the adventage that we can extend it to a formula for $E_z(n)$. Let $\mathbf{s}$ be an n-even sequence and let $\mathbf{s}' = (s'_1, \ldots, s'_n)$ be defined by $s'_i = s_i + n - i$ for $i = 1, \ldots, n$. Then the number of different possible sequences $\mathbf{s}$ is $E(n)$ and the number of different sequences $\mathbf{s}'$ is $R(n)$.

If $j = 0,\ 1,\ 2$, or 3 and $n = 4k + j$, then let $E(n)$ be denoted by $E(k, j)$.

Lemma 6 *If* $n \geq 1$ *and* $n = 4k + j$, *then*

$$E(k, 0) = \sum_{i=0}^{2k-1} \binom{4k-1}{2i} \binom{4k}{4k-2i}, \tag{17}$$

$$E(k, 1) = \sum_{i=0}^{2k} \binom{4k}{2i} \binom{4k+1}{4k-2i+1}, \tag{18}$$

$$E(k,2) = \sum_{i=0}^{2k} \binom{4k+1}{2i+1} \binom{4k+2}{4k-2i+1}, \tag{19}$$

$$E(k,3) = \sum_{i=0}^{k} \binom{4k+2}{2i+1} \binom{4k+3}{4k-2i+2}. \tag{20}$$

Proof. Let

$$\sum_{i=1}^{n} s_i = S(\mathbf{s}) \quad \text{and} \quad \sum_{i=1}^{n} s'_i = S'(\mathbf{s}). \tag{21}$$

According to the value of j we consider four cases. Since $\mathbf{s}$ is an even sequence, therefore $S(\mathbf{s})$ is even in all cases.

1. If $j = 0$, then

$$S'(\mathbf{s}) = S(\mathbf{s}) + \sum_{i=0}^{4k-1} i = S(\mathbf{s}) + 2k(4k-1), \tag{22}$$

and so $S(\mathbf{s}')$ is also even, therefore it contains an even number of odd elements. The interval $[0, 8k-2]$ contains $8k-1$ elements and among them $4k$ even and $4k-1$ odd elements, so for $\mathbf{s}'$ we can choose $2i$ odd elements from $4k-1$ candidates and $4k-2i$ ($i = 0,\ 1,\ \ldots,\ 2k-1$) even elements from $4k+1$ candidates, so

$$E(k,0) = \sum_{i=0}^{2k-1} \binom{4k-1}{2i} \binom{4k}{4k-2i}. \tag{23}$$

2. If $j = 1,\ 2$ or $j = 3$, then the proof is similar to the proof in the first case.

□

For example let $n = 4$, then $k = 1$, $j = 0$ and

$$E(4) = E(1,0) = \sum_{i=0}^{1} \binom{3}{2i} \binom{4}{4-2i} = 1 \cdot 1 + 3 \cdot 6 = 19. \tag{24}$$

As another example let $n = 6$, then $k = 1$, $j = 2$ and

$$E(6) = \sum_{i=0}^{2} \binom{5}{1} \binom{6}{3} + \binom{5}{5} \binom{6}{1} = 530 + 200 + 6 = 236. \tag{25}$$

Let the number of zerofree even sequences denoted by $E_z(n)$. Let $\mathbf{q} = (q_1, \ldots, q_n)$ be a zerofree n-even sequence and let $\mathbf{q}' = (q'_1, \ldots, q'_n)$ be defined by $q'_i = q_i + n - i$ for $i = 1, \ldots, n$. Then the number of different possible sequences $\mathbf{q}$ is $E_z(n)$ and the number of different sequences $\mathbf{q}'$ is $R_z(n)$.

Theorem 7 *Let* $n = 4k + j$ *for* $k = 0, 1, \ldots$ *and* $j = 0, 1, 2, 3$, *further let* $E_z(n)$ *be denoted by* $E_z(k, j)$. *Then*

$$E_z(k, 0) = \sum_{i=0}^{2k-1} \binom{4k-1}{2i} \binom{4k-1}{4k-2i}, \tag{26}$$

$$E_z(k, 1) = \sum_{i=0}^{2k} \binom{4k}{2i} \binom{4k}{4k-2i+1}, \tag{27}$$

$$E_z(k, 2) = \sum_{i=0}^{2k} \binom{4k+1}{2i+1} \binom{4k+1}{4k-2i+1}, \tag{28}$$

$$E_z(k, 3) = \sum_{i=0}^{2k+1} \binom{4k+2}{2i+1} \binom{4k+2}{4k-2i+2}. \tag{29}$$

Proof. Let

$$\sum_{i=1}^{n} q_i = Q(\mathbf{q}) \quad \text{and} \quad \sum_{i=1}^{n} q'_i = Q'(\mathbf{q}). \tag{30}$$

According to the value of j we consider four cases. Since $\mathbf{q}$ is an even sequence, therefore $Q(\mathbf{q})$ is alwys even.

1. If $j = 0$, then

$$Q'(\mathbf{q}) = Q(\mathbf{q}) + \sum_{i=0}^{4k-1} i = Q(\mathbf{q}) + 2k(4k-1) \tag{31}$$

is even, therefore the number of odd elements of $\mathbf{q}'$ is also even. The interval $[1, 8k-2]$ contains $8k-2$ elements and among them $4k-1$ even and $4k-1$ odd elements, so for $\mathbf{q}'$ we can choose $2i$ odd elements from $4k-1$ candidates and $4k-2i$ ($i = 0, \ldots, 2k-1$) even elements from $4k-1$ candidates, so we get (26).

2. If $j = 1$, then

$$Q'(\mathbf{q}) = Q(\mathbf{q}) + \sum_{i=0}^{4k} i = Q(\mathbf{q}) + 2k(4k+1), \tag{32}$$

is even, therefore the number of odd elements of $\mathbf{q}'$ is also even. The interval $[1, 8k]$ contains $8k$ elements and among them $4k$ odd and $4k$ even elements, so for $\mathbf{q}'$ we can choose $2i$ odd elements from $4k$ candidates and $4k-2i+1$ ($i = 0, \ldots, 2k$) even elements from $4k-1$ candidates, so we get (27).

3. If $j = 2$, then

$$Q'(\mathbf{q}) = Q(\mathbf{q}) + \sum_{i=0}^{4k+1} i = Q(\mathbf{q}) + (2k+1)(4k+1) \tag{33}$$

is odd, therefore the number of odd elements of $\mathbf{q}'$ is also odd. The interval $[1, 8k+2]$ contains $8k+2$ elements and among them $4k+1$ even and $4k+1$ odd elements, so for $\mathbf{q}'$ we can choose $2i+1$ odd elements from $4k+2$ candidates and $4k-2i-1$ ($i = 0, \ldots, 2k-1$) even elements from $4k+1$ candidates, so we get (28).

4. If $j = 3$, then

$$Q'(\mathbf{q}) = Q(\mathbf{q}) + \sum_{i=0}^{4k+2} i = Q(\mathbf{q}) + (2k+1)(4k+3), \tag{34}$$

and so $Q(\mathbf{q}')$ is also odd, therefore $\mathbf{q}'$ contains an odd number of odd elements. The interval $[1, 8k+4]$ contains $8k+4$ elements and among them $4k+2$ even and $4k+2$ odd elements, so for $\mathbf{q}'$ we can choose $2i+1$ odd elements from $4k+2$ candidates and $4k-2i-1$ ($i = 0, \ldots, 2k-1$) even elements from $4k+2$ candidates, so

$$E_z(k, 3) = \sum_{i=0}^{2k+1} \binom{4k+2}{2i+1} \binom{4k+2}{4k-2i+2}. \tag{35}$$

□

Table 1 shows the values of $R(n)/R(n+1)$, $R_z(n)/R_z(n+1)$, $E(n)/R(n)$, $E(n)/E(n+1)$, $E_z(n)/E_z(n+1)$, and $E_z(n)/R_z(n)$ for $n = 1, \ldots, 32$.

n	$\frac{R(n)}{R(n+1)}$	$\frac{R_z(n)}{R_z(n+1)}$	$\frac{E(n)}{R(n)}$	$\frac{E(n)}{E(n+1)}$	$\frac{E_z(n)}{E_z(n+1)}$	$\frac{E_z(n)}{R_z(n)}$
1	0.333333	0.000000	1.00000000	0.000000	0.000000	− − −
2	0.300000	0.250000	0.66666667	0.500000	0.500000	1.000000
3	0.287714	0.266667	0.60000000	0.222220	0.222222	0.500000
4	0.277778	0.257857	0.487179	0.321427	0.321429	0.600000
5	0.270562	0.266667	0.523810	0.254545	0.254545	0.500000
6	0.269231	0.265151	0.510823	0.277778	0.277778	0.523810
7	0.266667	0.263736	0.505828	0.260698	0.260698	0.500000
8	0.264706	0.262500	0.502720	0.265559	0.265559	0.505828
9	0.263158	0.261437	0.501440	0.260687	0.260687	0.500000
10	0.261905	0.260526	0.500682	0.261276	0.261276	0.501440
11	0.260870	0.259740	0.500357	0.259555	0.259555	0.500000
12	0.260000	0.259058	0.500171	0.259243	0.259243	0.500357
13	0.259259	0.258461	0.500089	0.258415	0.258416	0.500000
14	0.258621	0.257937	0.500043	0.257982	0.257982	0.500089
15	0.258065	0.257471	0.500022	0.257460	0.257460	0.500000
16	0.257578	0.257056	0.500011	0.257068	0.257068	0.500022
17	0.257143	0.256684	0.500005	0.256682	0.256682	0.500000
18	0.256757	0.256349	0.500003	0.256352	0.256352	0.500006
19	0.256410	0.256046	0.500001	0.256045	0.256045	0.500000
20	0.256098	0.255769	0.500001	0.255770	0.255770	0.500001
21	0.255814	0.255517	0.50000034	0.255517	0.255517	0.500000
22	0.255556	0.255285	0.50000016	0.255286	0.255286	0.50000034
23	0.255319	0.255072	0.50000009	0.255072	0.255072	0.50000000
24	0.255102	0.254876	0.50000004	0.254876	0.254876	0.50000000
25	0.254902	0.254694	0.50000002	0.254694	0.254694	0.50000009
26	0.254717	0.254525	0.50000001	0.254525	0.254525	0.50000000
27	0.254545	0.254368	0.50000001	0.254368	0.254368	0.50000000
28	0.254386	0.254221	0.50000000	0.254221	0.254221	0.50000000
29	0.254237	0.254083	0.50000000	0.254083	0.254083	0.50000000
30	0.254098	0.253854	0.50000000	0.253955	0.253955	0.50000000
31	0.253968	0.253834	0.50000000	0.253834	0.253834	0.50000000
32	0.253846	0.253720	0.50000000	0.253720	0.253720	0.50000000

Table 1: The values of $R(n)/R(n+1)$, $R_z(n)/R_z(n+1)$, $E(n)/R(n)$, $E(n)/E(n+1)$, $E_z(n)/E_z(n+1)$, and $E_z(n)/R_z(n)$ for $n = 1, \ldots, 32$

It is remarkable that in $R(101)/R(102)$ and $R_z(101)/R_z(102)$ the first nine decimal digits are equal.

For example let $n = 4$, then $k = 1$, $j = 0$ and

$$E_z(4) = E_z(1,0) = \binom{3}{0}\binom{3}{4} + \binom{3}{2}\binom{3}{2} = 1 \cdot 0 + 3 \cdot 3 = 9. \tag{36}$$

If $n = 5$, then $k = 1$, $j = 1$ and

$$E_z(5) = \binom{4}{0}\binom{4}{5} + \binom{4}{2}\binom{4}{3} + \binom{4}{4}\binom{4}{1} = 0 + 24 + 4 = 28. \tag{37}$$

If $n = 6$, then $k = 1$, $j = 2$ and

$$E_z(6) = \binom{5}{1}\binom{5}{5} + \binom{5}{3}\binom{5}{3} + \binom{5}{5}\binom{5}{1} = 5 + 100 + 5 = 110. \tag{38}$$

If $n = 7$, then $k = 1$, $j = 3$ and

$$E_z(7) = \binom{6}{1}\binom{6}{6} + \binom{6}{3}\binom{6}{4} + \binom{6}{5}\binom{6}{2} = 6 + 300 + 90 = 396. \tag{39}$$

If $n = 8$, then $k = 2$, $j = 0$ and

$$E_z(8) = \binom{7}{0}\binom{7}{8} + \binom{7}{2}\binom{7}{6} + \binom{7}{4}\binom{7}{4} + \binom{7}{6}\binom{7}{2} = 1519. \tag{40}$$

Simulaton results in Table 1 show, that if $1 \le n \le 32$ and n is odd, then $E_z(n)/R_z(n) = 0.5$. This property is true for larger odd n's too.

Lemma 8 *If* $1 \le k \le 600$*, then*

$$\frac{E_z(2k-1)}{R_z(2k-1)} = 0.5. \tag{41}$$

Proof. See the computed values of $R_z(n)$ and $E_z(n)$ in [68]. □

Table 2 contains the ratios $E_z(n)/G_z(n)$ for $n = 23, \ldots, 29$ and the ratios $T(n)/G_z(n)$ for $n = 30$ and $n = 31$.

The data in Table 2 show that the function G_z/E_z) is decreasing. We suppose that this function tends monotonically decreasing to zero when n tends to infinity (in a similar way as the function $G(n)/E(n)$ tends to zero according to Corollary 23 [52, page 260].

Table 3 contains the values of $G_z(n)$, $T(n)$, and $G_z(n)/T(n)$ for $n = 30$ and $n = 31$: the ratio of the graphical and tested sequences is much higher and these ratios are increasing. These changes are dut to the fact that EGE2 jumps many nongraphical zerofree ebven sequences withous testing them.

3.2 New algorithmic results

Using the following Lemma 9 later we will further fasten EGE.

If $b = (b_1, \ldots, b_n)$ is a regular sequence, then $c = (c_1, \ldots, c_n)$ is called *lexicographically* i*-smaller, than* b if there exist indices i and j such that

$$1 \le i < j <= n, \tag{42}$$

n	$G_z(n)$	$E_z(n)$	$G_z(n)/E_z(n)$
17	130 038 230	282 861 360	0.459724
18	499 753 855	1 101 992 870	0.453500
19	1 924 912 894	4 298 748 300	0.447784
20	7 429 160 296	16 789 046 494	0.442500
21	28 723 877 732	65 641 204 200	0.437589
22	111 236 423 288	256 895 980 068	0.433002
23	431 403 470 222	1 006 308 200 040	0.428699
24	1 675 316 535 350	3 945 186 233 014	0.424648
25	6 513 837 679 610	15 478 849 767 888	0.420821
26	25 354 842 100 894	60 774 332 618 300	0.417197
27	98 794 053 269 694	238 775 589 937 976	0.413752
28	385 312 558 571 890	938 702 947 395 204	0.410473
29	1 504 105 116 253 904	3 692 471 324 505 040	0.407344
30	5 876 236 938 019 300	14 532 512 180 224 216	0.404351
31	22 974 847 474 172 100	57 224 797 531 384 560	0.400148

Table 2: The values of $G_z(n)$, $E_z(n)$, and $G_z(n)/E_z(n)$ for $n = 17, \ldots, 31$

n	$G_z(n)$	$T(n)$	$G_z(n)/T(n)$
31	5 876 236 938 019 300	6 790 865 476 867 340	86, 531487
32	22 974 847 471 172 100	26 507 499 250 791 700	86, 673010

Table 3: The values of $G_z(n)$, $T(n)$, and $G_z(n)/T(n)$ for $n = 30$ and $n = 31$

further

$$c_k = b_k \quad \text{for} \quad k = 1, \ldots, i, \tag{43}$$

and

$$\sum_{k=i+1}^{n} c_k \leq \sum_{k=i+1}^{n} b_k. \tag{44}$$

Lemma 9 *If* $b = (b_1, \ldots, b_n)$ *and is a nongraphical sequence and* $c = (c_1, \ldots, c_n)$ *is lexicographically* i*-smaller than* b *for some* i $(1 \leq i < n$*, then* c *is also graphical.*

Proof. See [54]. □

Using this lemma the running time of EGLJ decreased substantially. It was very useful when we enumerated the edge sequences of the simple graphs on

30 vertices between June 21 and 18 July of 2013 and on 31 vertices between 18 July and 24 August (the results see in Table 4).

Using the results of Tripathi and Vijay [52, Lemma 6 and Theorem 7] we can substantially decrease the average testing time of the zerofree even sequences. It is known that the expected number of checking points proposed by Tripathi and Vijay is about $n/2$ [52].

Algorithm EGE2 [51, pages 274–277] used in the enumerations for $n = 30$ and $n = 31$ vertices is based on Lemma 9.

We develop algorithm EGE3 for the enumeration of the degree sequences in the case of 32 vertices. EGE3 will be based on Lemmas 10 and 11.

Lemma 10 *If the investigated by EG3 sequence is graphical and has the form* $b = b_{i_1}^{c_1} b_{i_2}^{c_2}$ *and the upper bound* $c_1(c_1-1)$ *of the inner capacity of the* c_1*-length head of* b *covers* H_n*, that is if*

$$c_1(c_1 - 1) \geq H_n \tag{45}$$

then all sequences starting with $b_1^{c_1}$ *are graphical.*

Proof. See [54]. □

The next lemma allows to enumerate many graphical sequences without their time consuming testing.

Lemma 11 *If the investigated by EG3 graphical sequence has the form* $b = b_1^{c_1} b_2^{c_2} \dots b_p^{c_p} 1^{c_{p+1}}$*, where* $p \geq 1$*,* $b_1 > b_2 > \dots > b_p \geq 3$*,* $c_1, \dots, c_{p+1} \geq 1$*, then all zerofree even sequences starting with the prefix*

$$b_1^{c_1} b_2^{c_2} \dots b_{p-1}^{c_{p-1}} b_p^{c_p - 1} (b_p - 1)$$

are also graphical.

Proof. See [54]. □

3.3 New simulation results

Table 4 contains the values of $G_z(n)$ and $G(n)$ for $n = 1, \dots, 31$. The values for $n = 1, \dots, 9$ were computed by E. Weisstein, for $n = 10, \dots, 20$ by G. Royle in 2006, for $n = 21, 22$ and $n = 23$ by F. Ruskey in 2006, for $n = 24, \dots, 29$ by T. Matuszka in January of 2013, for $n = 30$ by L. Lucz in July of 2013 and for $n = 31$ also by L. Lucz in September of 2013 [48, 50, 51, 52, 79].

Column 4 of Table 4 supports the following conjecture formulated by Gordon Royle in 2012.

Conjecture 12 (Royle, 2012). *If* n *tends to infinity, then* $G_z(n+1)/G_z(n)$ *tends to 4.*

We think, that the following conjecture is also true.

Conjecture 13 *If* n *tends to infinity, then* $G(n+1)/G(n)$ *tends to 4.*

We observed that when we enumerated these sequences, that in the case $n = 30$ vertices 85.40 percent, while in the case $n = 31$ vertices 86.67 percent of the investigated potential degree sequences was graphical. Therefore it is useful if we know without a linear time testing that a given tested sequence is graphical.

Figure 1 shows the number of the tested and the graphical sequences as the function of the index of the slices when $n = 30$.

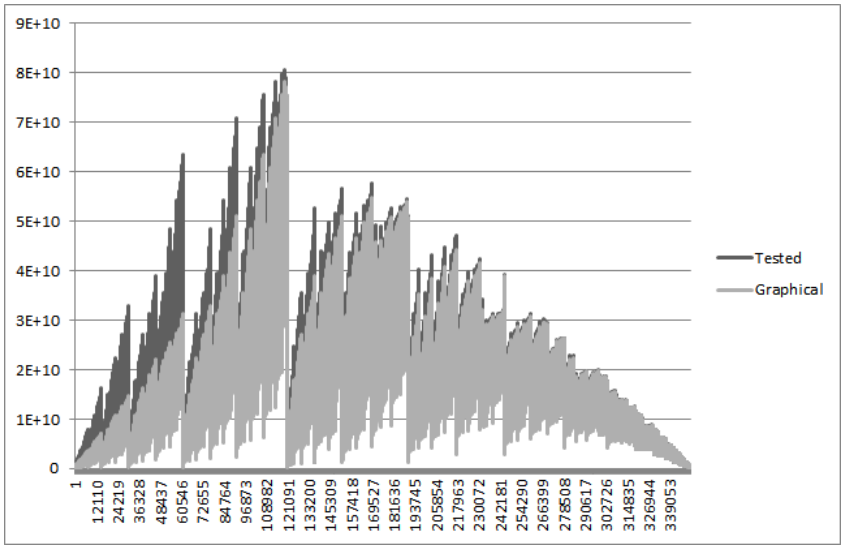

Figure 1: The number of tested (trimmed even) sequences and the number of graphical sequences as the function of the index of slices when $n = 30$

Figure 2 shows the similar data for $n = 31$.

We remark that on the site of the journal the Figures 1 and 2 are color (the graphical sequences are represented by red, while the tested sequences by blue color).

Table 5 contains the data of PC's used for the enumeration of $G_z(31)$, where Comp. alg. = Computer Algebra, Prog. lang. = Program languages, Core =

n	$G_z(n)$	$G(n)$	$\frac{G_z(n+1)}{G_z(n)}$	$\frac{G(n+1)}{G(n)}$
1	0	1	0.000000	0.500000
2	1	2	0.500000	0.500000
3	2	4	3.500000	3.750000
4	7	11	2.857143	2.818182
5	20	31	3.550000	3.290323
6	71	102	3.380282	3.352941
7	240	342	3.629167	3.546784
8	871	1 213	3.614237	3.595218
9	3 148	4 361	3.702351	3.672552
10	11 655	16 016	3.717889	3.705544
11	43 332	59 348	3.756323	3.742620
12	162 769	222 117	3.773434	3.786674
13	614 198	836 315	3.794439	3.802710
14	2 330 537	3 166 852	3.808465	3.817067
15	8 875 768	12 042 620	3.822189	3.828918
16	33 924 859	45 967 479	3.833125	3.839418
17	130 038 230	176 005 709	3.843130	3.848517
18	499 753 855	675 759 564	3.851172	3.856630
19	1 924 912 894	2 600 672 458	3.859479	3.863844
20	7 429 160 296	10 029 832 754	3.866369	3.870343
21	28 723 877 732	38 753 710 486	3.872612	3.876212
22	111 236 423 288	149 990 133 774	3.878257	3.881553
23	431 403 470 222	581 393 603 996	3.883410	3.886431
24	1 675 316 535 350	2 256 710 139 346	3.888124	3.890907
25	6 513 837, 679 610	8 770 547 818 956	3.894458	3.895031
26	25 354 842 100 894	34 125 389 919 850	3.895503	3.897978
27	98 794 053 269 694	132 919 443 189 544	3.900159	3.898843
28	385 312 558 571 890	518 232 001 761 434	3.903597	3.902238
29	1 504 105 116 253 904	2 022 337 118 015 338	3.906814	3.905666
30	5 876 236 938 019 300	7 898 574 056 034 638	3.909789	3.908734
31	22 974 847 474 172 100	30 873 429 530 206 738	− − −	− − −

Table 4: The number $G_z(n)$ of zerofree graphical sequences and the number $G(n)$ of graphical sequences for $n = 1, \ldots, 31$, further the ratios $G_z(n)/G_z(n+1)$ and $G(n)/G(n+1)$ for $n = 1, \ldots, 30$

Core(TM)Kása 1 = Z. Kása (Cluj), Kása 2 = Z. Kása (Tg.-Mureś), Kása 3 = Z. Kása // (Tg.-Mureś), Sp1 = Speed of a machine in GHz, Sp2 = Speed of the laboratory in GFLOPS, Intel (R) = Intel (R) Xeon (R).

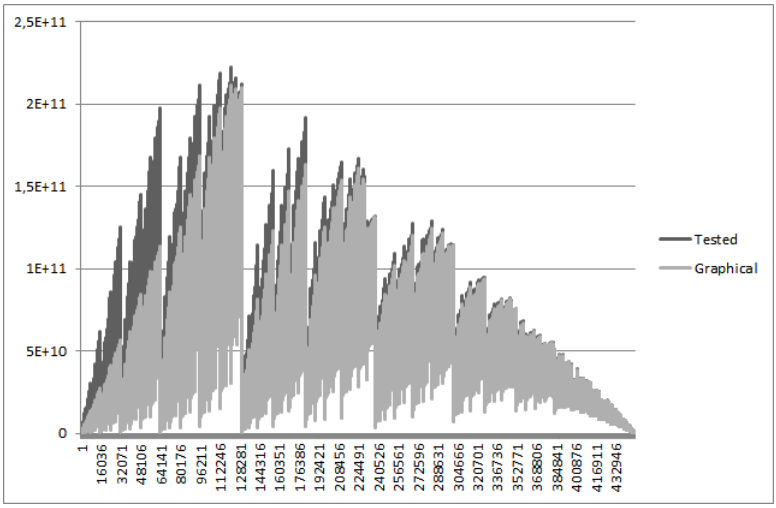

Figure 2: The number of tested (trimmed even) sequences and the number of graphical sequences as the function of the index of slices when $n = 31$

The total number of machines was 350.

Table 6 contains the algorithms, running times and number of jobs in the case $n = 25, \ldots, 31$.

3.4 The growth of the functions $R(n)$, $E(n)$, $R_z(n)$, $E_z(n)$, $G(n)$ and $G_z(n)$

In this subsection we present concrete values of the functions characterizing the sizes of the investigated sets of sequences.

The number $R(n)$ of the regular sequences is presented in Figure 1 of [52] for $n = 1, \ldots, 38$ and up to $n = 1200$ in [47].

The values of the zerofree regular $R_z(n)$ can be quickly computed using formula (22) in [47]. The values for $n = 1, \ldots, 1200$ can be found in [68].

The number $E(n)$ of even sequences is presented in Figure 1 of [52] for $n = 1, \ldots, 38$ and up to 1000 in [49] and up to $n = 1200$ in [68].

The number $E_z(n)$ of the zerofree even sequences is contained in Figure 3 of [52] for $n = 1, \ldots, 20$ (these data are the results of brute force simulation) and up to $n = 1200$ in [68].

The order of growth of these functions is $\Theta(4^n/n)$.

According to theorem of Burns [22, 52] the order of growth of $G(n)$ is smaller (see 12).

Laboratory	Number	Type	Sp1	Sp2
Central	87	Core 2 Duo	2.93	2041
Comp. algebra	13	Core 2 Duo	2.13	403
Data base	34	Core 2 Duo	3.25	1631
Graphical	16	Core 2 Quad	2.33	597
Prog. lang.	54	Core 4 Duo	3.25	6621
PC1	20	Core i5-2320	3.00	1920
PC3	28	Core i3-2100	3.10	1389
PC4	19	Core 2 Duo	2.93	446
PC5	19	Core 4 Duo	2.93	446
PC6	18	Core 2 Quad	2,33	672
PC7	18	Core 2 Quad	2.40	691
PC9	19	Core 2 Quad	2.66	810
Server	1	Core i5 650	3.20	26
Kása 1	1	AMD K7	0.75	8
Kása 2	1	Intel (R)	3.00	50
Kása 3	1	Intel (R)	2.13	23
P. Ősze	1	Core 4 Duo	2.20	37
Total	350			17811

Table 5: Names of laboratories, number of machines, type of machines, speed of machines in GHz, speed of laboratories in GLOPS, used in the case $\mathfrak{n} = 31$

The known values of $\mathsf{G}(\mathfrak{n})$ and $\mathsf{G}_z(\mathfrak{n})$ are summarized in Table 4.

3.5 Further plans

Our new program (EGE3) is able to jump the test of some part of zerofree graphical sequences [54]. Due to this property of the new program EGE3 the number of tested sequences is smaller than the number of zerofree graphical ones (see Table 7).

4 Summary

The log files and source codes of our programs can be found at

http://people.inf.elte.hu/lulsaai/Holzhacker .

n	Algorithm	Running time (in days)	Running time (in years)	Number of jobs
25	EGE	26	0.0712	435
26	EGE	70	0.1918	435
27	EGE	316	0.8657	435
28	EGE	1 130	3.0959	2 001
29	EGE	6 733	18.4466	15 119
30	EGE2	7 221	19.7835	351 155
31	EGE2	32 702	89.5954	448 957

Table 6: Number of vertices, used algorithm, total running time (in days and in years) and number of jobs

n	$T(n)$	$G_z(n)$
3	3	2
4	8	7
5	24	20
6	77	71
7	245	240
8	852	871
9	2991	3148
10	10807	11655
11	39407	43332
12	145673	162769
13	542531	614198
14	2036196	2330537
15	7684164	8875768
16	29143362	33924859
17	110973050	130038230
18	424055902	499753855
19	1625265958	1924912894
20	6245498873	7429160296

Table 7: Number of vertices (n), number of tested sequences ($T(n)$) and number of zerofree graphical sequences ($G_z(n)$)

Acknowledgements. The authors are indebted to Faculty of Informatics of Eötvös Loránd University for the possibility to run the server and client programs in its laboratories, further to Antal Sándor and Ferenc Saáry, Jr. (Eötvös Loránd University, Faculty of Informatics) for their technical help, for Bálint Csergő (Ulstream Hungary Kft.) for the conversion of the figures, for Zoltán Kása (Sapientia Hungarian University of Transylvania, Campus in Cluj and Tg.-Mureś) and for Péter Ősze (Ustream Hungary Kft.) for running the clients and Kristóf Szabados (Ericsson Hungary) for improving the jumping algorithm.

References

[1] S. R. Arikati, A. Maheshwari, Realizing degree sequences in parallel, *SIAM J. Discrete Math.* **9,** 2 (1996) 317–338. ⇒250

[2] M. Ascher, Mu torere: an analysis of a Maori game, *Math. Mag.* **60,** 2 (1987) 90–100. ⇒248, 251

[3] T. M. Barnes, C. D. Savage, A recurrence for counting graphical partitions, *Electron. J. Combin.* **2** (1995), Research Paper 11, 10 pages (electronic). ⇒252

[4] T. M. Barnes, C. D. Savage, Efficient generation of graphical partitions, *Discrete Appl. Math.* **78,** 1–3 (1997) 17–26. ⇒252

[5] M. D. Barrus, Hereditary unigraphs and Erdős-Gallai inequalities. arXiv:1302.2703v1 [mathCO] 12 February 2013, 23 pages. ⇒250

[6] M. D. Barrus, S. G. Hartke, K. F. Jao, D. B. West, Length threshold s for graphic lists given fixed largest and smallest entries and bounded gaps, *Discrete Math.* **312,** 9 (2012) 1494–1501. ⇒250

[7] E. A. Bender, E. N. Canfield, The asymptotic number of labeled graphs with given degree sequences, *J. Comb. Math.* **24** (1978) 296–307. ⇒250

[8] C. Berge, *Graphs and Hypergraphs*, North Holland, 1973. ⇒246, 250

[9] C. Berge, *Graphs* (third edition), North Holland, 2001 (first edition: 1989). ⇒246, 250

[10] A. Berger, *Directed degree sequences*, PhD Dissertation, Martin-Luther-Universität Halle-Wittenberg, 2011. `http://wcms.uzi.uni-halle.de/download.php?down=22851&elem=2544689`. ⇒250

[11] A. Berger, A note on the characterization of digraph sequences, *arXiv*, arXiv:1112.1215v1 [math.CO] (6 December 2011). ⇒250

[12] A. Berger, A note on the characterization of digraphic sequences, *Discrete Math.* **314** (2014) 38–41. ⇒250

[13] A. Berger, M. Müller-Hannemann, Uniform sampling of digraphs with a fixed degree sequence, in (ed. D. M. Thilikos) *36th Int. Workshop on Graph Theoretic Concepts in Computer Science* (June 28 - 30, 2010, Zarós, Crete, Greece), LNCS **6410** (2010) 220–231. ⇒250

[14] A. Berger, M. Müller-Hannemann, Dag characterizations of directed degree sequences, Technical Report 2011/6 of University Halle-Wittenberg, Institute of Computer Science. ⇒250

[15] A. Berger, M. Müller-Hannemann, How to attack the NP-complete dag realization problems in practice. *arXiv*, arXiv:1203.36v1, 2012. http://arxiv.org/abs/1203.3636 ⇒250

[16] N. Bödei, *Degree sequences of graphs* (Hungarian), Mathematical master thesis (supervisor A. Frank), Dept. of Operation Research of Eötvös Loránd University, Budapest, 2010, 43 pages. ⇒246

[17] B. Bollobás, The distribution of the maximum degree of a random graph, *DiscreteMath.* **32,** 2 (1980) 201–203. ⇒250

[18] B. Bollobás, A probabilistic proof of an asymptotic formula for the number of labelled regular graphs, *European J. Comb.* **1,** 4 (1980) 311–316. ⇒250

[19] B. Bollobás, Degree sequences of random graphs, *Discrete Mathematics* **33,** 1 (1981) 1–19. ⇒250

[20] A. Brualdi, Matrices of zeros and ones with fixed row and column sum vectors, *Linear Alg. Appl.* **33** (1980) 159–231. ⇒250

[21] J. C. Brunson, The S-metric, the Beichl-Croteaux approximation and preferential attachment, *arXiv*:1308.4067v1 [mathCO] 19 August 2013. ⇒250

[22] J. M. Burns, The number of degree sequences of graphs, PhD Dissertation, MIT, 2007. ⇒249, 262

[23] G. Cairns, S. Mendan, An improvement of a result of Zverovich-Zverovich, *arXiv* arXiv:1303.2144v1 [math.CO], 5 pages. ⇒246

[24] G. Cairns, S. Mendan, Degree sequences for graphs with loops, *arXiv* arXiv:1303.2145v1 [math.CO], 9 pages. ⇒246

[25] G. Cairns, S. Mendan, Y. Nikolayevsky, A sharp improvement of a result of Zverovich-Zverovich, *arXiv* arXiv:1310.3992v1 [math.CO], 7 pages. ⇒246

[26] S. A. Choudum, A simple proof of the Erdős-Gallai theorem on graph sequences, *Bull. Austral. Math. Soc.* **33** (1986) 67–70. ⇒246

[27] V. Chungphaisan, Conditions for sequences to be r-graphic, *Discrete Math.* **7** (1974) 31–39. ⇒246

[28] N. Cohen, Number of distinct degree sequences among all n-vertex graphs with no isolated vertices: new values for $n = 20, 21, 22$, and 23, in: ed. by N. J. A. Sloane, *The On-Line Encyclopedia of Integer Sequences,* 2012, http://oeis.org/A095268. ⇒246

[29] P. Das, Characterization of unigraphic and unidigraphic integer-pair sequences, Characterization of unigraphic and unidigraphic integer-pair sequences *Discrete Math.* **37,** 1 (1981) 51–66. ⇒250

[30] C. I. Del Genio, H. Kim, Z. Toroczkai, K. E. Bassler, Efficient and exact sampling of simple graphs with given arbitrary degree sequence, *PLoS ONE* **5,** 4 e10012 (2010). ⇒252

[31] D. Dimitrov, Efficient computation of trees with minimal atom-bound connectivity index. *arXiv*:1305.1155v2 [csDM] 4 October 2013. ⇒250

[32] P. Erdős, T. Gallai, Graphs with vertices having prescribed degrees (Hungarian), *Mat. Lapok* **11** (1960) 264–274. ⇒246, 250, 251

[33] P. Erdős, L. B. Richmond, On graphical partitions, *Combinatorica* **13,** 1 (1993) 57–63. ⇒252

[34] P. L. Erdős, S. Z. Kiss, I. Miklós, On the swap-distances of different realizations of a graphical degree sequence, *Comb. Probab. Comp.* **22,** 3 (2013) 366–383. ⇒ 250

[35] P. L. Erdős, Z. Király, I. Miklós, L. Soukup, Constructive sampling and counting graphical realizations of restricted degree sequences, *arXiv* arXiv:13017523v3 [math.CO], 24 pages. ⇒250

[36] P. L. Erdős, I. Miklós, Z. Toroczkai, A simple Havel-Hakimi type algorithm to realize graphical degree sequences of directed graphs, *Electron. J. Combin.* **17,** 1 (2010) R66, 10 pages. ⇒250

[37] P. L. Erdős, I. Miklós, Z. Toroczkai, A decomposition based proof for fast mixing of a Markov chain over balanced realizations of a joint degree matrix, *arXiv*, arXiv:1307.5295v1 [math.CO], 2013, 18 pages. ⇒250

[38] C. Greenhill, A polynomial bound on the mixing time of a Markov chain for sampling regular directed graphs, *Electron. J. Combin.* **18** &P234, 2011, 49 pages. ⇒250

[39] S. L. Hakimi, On the realizability of a set of integers as degrees of the vertices of a simple graph, *J. SIAM Appl. Math.* **10** (1962) 496–506. ⇒246, 250

[40] S. L. Hakimi, On the degrees of the vertices of a graph, *F. Franklin Institute* **279,** 4 (1965) 290–308. ⇒250

[41] F. Harary, E. M. Palmer, *Graphical Enumeration*, Academic Press, New York and London, 1973. ⇒250

[42] V. Havel, A remark on the existence of finite graphs (Czech), *Časopis Pěst. Mat.* **80** (1955) 477–480. ⇒246, 250

[43] P. Hell, D. Kirkpatrick, Linear-time certifying algorithms for near-graphical sequences. *Discrete Math.* **309,** 18 (2009) 5703–5713. ⇒246

[44] A. Iványi, Reconstruction of complete interval tournaments, *Acta Univ. Sapientiae, Inform.* **1,** 1 (2009) 71–88. ⇒246, 252

[45] A. Iványi, Reconstruction of complete interval tournaments. II, *Acta Univ. Sapientiae, Math.* **2,** 1 (2010) 47–71. ⇒246, 252

[46] A. Iványi, Degree sequences of multigraphs. *Annales Univ. Sci. Budapest., Sect. Comp.* **37** (2012) 195–214. ⇒246, 250, 252

[47] A. Iványi, L. Lucz, G. Gombos, T. Matuszka $C(2n+1, n+1)$: number of ways to put $n+1$ indistinguishable balls into $n+1$ distinguishable boxes = number of $(n+1)$-st degree monomials in $n+1$ variables = number of monotone maps from $1\,..\,n+1$ to $1\,..\,n+1$, in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2013. http://oeis.org/A001700. ⇒247, 262

[48] A. Iványi, L. Lucz, G. Gombos, T. Matuszka The number of degree-vectors for simple graph. in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2013. http://oeis.org/A004251. ⇒246, 251, 259

[49] A. Iványi, L. Lucz, G. Gombos, T. Matuszka, Number of bracelets (turn over necklaces) with n red, 1 pink and $n-1$ blue beads; also reversible strings with n red and $n-1$ blue beads, in (ed. by N. J. A. Sloane), *The On-Line Encyclopedia of Integer Sequences,* 2013. http://oeis.org/A005654. ⇒262

[50] A. Iványi, L. Lucz, G. Gombos, T. Matuszka, Number of distinct degree sequences among all n-vertex graphs with no isolated vertices: new values for $n = 24,\ 25,\ 26,\ 27,\ 28,$ and 29, in (ed. by N. J. A. Sloane): *The On-Line Encyclopedia of Integer Sequences,* 2013, http://oeis.org/A095268. ⇒247, 259

[51] A. Iványi, L. Lucz, T. Matuszka, S. Pirzada, Parallel enumeration of degree sequences of simple graphs. *Acta Univ. Sapientiae*, *Inform.* **4,** 2 (2012) 260–288. ⇒247, 248, 249, 252, 259

[52] A. Iványi, L. Lucz, T. F. Móri, P. Sótér, On the Erdős-Gallai and Havel-Hakimi algorithms. *Acta Univ. Sapientiae, Inform.* **3,** 2 (2011) 230–268. ⇒245, 246, 247, 248, 249, 250, 251, 252, 257, 259, 262

[53] A. Iványi, S. Pirzada, Comparison based ranking, in: *Algorithms of Informatics, Vol. 3* (ed. A. Iványi), AnTonCom, Budapest 2011, 1209–1258. ⇒246

[54] A. Iványi, K. Szabados, Parallel enumeration of degree sequences (Hungarian). *Alk. Mat. Lapok*, 40 pages, submitted. ⇒258, 259, 263

[55] R. H. Johnson, properties of unique realizations—a survey, *Discrete Math.* **3,** 1 (1980) 185–192. ⇒250

[56] J. H. Kim, B. Pixttel, Confirming the Kleitman-Winston conjecture on the largest coefficient in a q-Catalan number, *J. Comb. Theory A* **99,** 2 (2000) 19–206. ⇒250

[57] H. Kim, Z. Toroczkai, I. Miklós, P. L. Erdős, L. A. Székely: Degree-based graph construction, *J. Physics: Math. Theor. A* **42** (2009), 392001, 10 pages. ⇒

[58] Z. Király, Recognizing graphic degree sequences and generating all realizations. Egres Technical Reports, TR-2011-11 April 23, 2012, 12 pages. ⇒246, 250

[59] D. J. Kleitman, D. Wang, Algorithms for constructing graphs and digraphs with given valences and factors, *Discrete Math.* **6** (1973) 79–88. ⇒250

[60] D. J. Kleitman, K. J. Winston, Forests and score vectors. *Combinatorica* **1** (1981) 49–51. ⇒250

[61] D. E. Knuth, *The Art of Computer Programming, Volume 4A*, Addison Wesley, 2011. ⇒246

[62] M. Koren, Sequences with a unique realization by simple graphs, *J. Comb. Theory, Ser. B* **21,** 3 (1976) 235–244. ⇒250

[63] M. D. LaMar, Algorithms for realizing degree sequences of directed graphs. arXiv, 2010. http://arxiv.org/abs/0906.0343. ⇒246

[64] V. Librandi, Number of bracelets (turn over necklaces) with n red, 1 pink and $n-1$ blue beads for $n = 1,\ \ldots,\ 1000,$ in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences.* 2012. `http://oeis.org/A005654/b005654.txt` ⇒248, 252

[65] X. Lu, S. Bressan, Generating random graph sequences, in (ed. J. X. Yu, M. H. Kim, R. Unland): *DASFAA2011, Part I*, LNCS **6587**, Springer-Verlag, 2011, 570–579. ⇒252

[66] L. Lucz, *Analysis of degree sequences of graphs* (Hungarian), Student thesis awarded by first prize in the Hungarian Scientific Student Conference, Budapest, 2013. Eötvös Loránd University, Faculty of Informatics, Budapest, 2011. `http://people.inf.elte.hu/lulsaai/diploma`. ⇒252

[67] B. D. McKay, X. Wang, Asymptotic enumeration of tournaments with a given score sequence. *J. Comb. Theory A*, **73**, (1) (1996) 77–90. ⇒

[68] T. Matuszka, Programs and Results Connected with Degree Sequences. ELTE IK, Budapest, 2013. ⇒247, 248, 257, 262

[69] D. Meierling, L. Volkmann, A remark on degree sequences of multigraphs, *Math. Methods Operation Research*, **69,** 2 (2009) 369–374. ⇒246

[70] J. W. Miller, Reduced criteria for degree sequences, *Discrete Math.* **313,** 4 (2013) 550–562. ⇒246

[71] R. Milo, N. Kashtan, S. Itzkovitz, M.E.J. Newman, U. Alon, On the uniform generation of random graphs with prescribed degree sequences, *arXiv,* arXiv:cond-mat/0312028v2 [cond-mat.stat-mech], 2004, 4 pages. ⇒250

[72] Miklós, J. Podani, Randomization of presence-absence matrices: comments and new algorithms, *Ecology*, **85,** 1 (2004) 86–92. ⇒250

[73] T. D. Noe, Table of $\mathfrak{a}(\mathfrak{n})$ for $\mathfrak{n} = 1, \ldots, 100$, in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2010. `http://oeis.org/A001700`. ⇒252

[74] T. D. Noe, Table of binomial coefficients $C(2\mathfrak{n}-1, \mathfrak{n})$, in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2012, `http://oeis.org/A001791`. ⇒252

[75] S. Özkan, Generalization of the Erdős-Gallai inequality. *Ars Combin.*, **98** (2011) 295–302. ⇒246

[76] A. N. Patrinos, S. L. Hakimi, Relations between graphs and integer-pair sequences. *Discrete Math.*, **15,** 4 (1976) 347–358 ⇒250

[77] S. Pirzada, *Introduction to Graph Theory*, Universities Press (India) Private Limited, 2012. ⇒246

[78] R. C. Read, The enumeration of locally restricted graphs (I). *J. London Math. Soc.*, **34** (1959) 417–436. ⇒250

[79] G. Royle, Is it true that $\mathfrak{a}(\mathfrak{n}+1)/\mathfrak{a}(\mathfrak{n})$ tends to 4? In (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2012. `http://oeis.org/A095268` ⇒259

[80] F. Ruskey, Number of distinct degree sequences among all $\mathfrak{n}$-vertex graphs with no isolated vertices for $\mathfrak{n} = 21,\ 22$ and 23, in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2006. http://oeis.org/A095268. ⇒247

[81] H. J. Ryser, Matrices of zeros and ones. *Bull. of Amer. Math. Soc.* **66,** 6 (1960) 442–464. ⇒250

[82] R. L. Shuo-Yen, Graphic sequences with unique realization, *J. Comb. Theory, Ser. B*, **19** (1975) 42–68. ⇒250

[83] G. Sierksma, H. Hoogeveen, Seven criteria for integer sequences being graphic, *J. Graph Theory* **15**, (2) (1991) 223–231. ⇒246

[84] N. J. A. Sloane, The number of degree-vectors for simple graphs, in (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences*. 2011. `http://oeis.org/A004251`. ⇒251

[85] N. J. A. Sloane, S. Plouffe, *The Encyclopedia of Integer Sequences,* Academic Press, 1995. ⇒248

[86] M. Takahashi, *Optimization Methods for Graphical Degree Sequence Problems and their Extensions*, PhD thesis, Graduate School of Information, Production and systems, Waseda University, Tokyo, 2007. http://hdl.handle.net/2065/28387. ⇒246

[87] A. Tripathi, H. Tyagy, A simple criterion on degree sequences of graphs. *Discrete Appl. Math.*, **156,** 18 (2008) 3513–3517. ⇒246

[88] A. Tripathi, S. Vijay, A note on a theorem of Erdős & Gallai. *Discrete Math.*, **265,** 1–3 (2003) 417–420. ⇒246

[89] A. Tripathi, S. Venugopalan, D. B. West, A short constructive proof of the Erdős-Gallai characterization of graphic lists. *Discrete Math.*, **310,** 4 (2010) 833–834. ⇒246

[90] R. I. Tyshkevich, O. I. Melnikov, V. M. Kotov, On graphs and degree sequences: a canonical decomposition (Russian), *Kibernetika* **6** (1981) 5–8. ⇒246

[91] K. J. Winston, D. J. Kleitman, On the asymptotic number of tournament score sequences. *J. Comb. Theory A* **35,** 2 (1983) 208–230. ⇒250

[92] I. E. Zverovich, V. E. Zverovich, Contributions to the theory of graphic sequences, *Discrete Math.*, **105,** 1–3 (1992) 293–303. ⇒246

Received: August 10, 2013 • Revised: December 30, 2013

Acta Univ. Sapientiae, Informatica, **5,** 2 (2013) 271–287

Linear programming over exponent pairs

Andrew V. LELECHENKO
I. I. Mechnikov Odessa National University, Russia
email: 1@dxdy.ru

Abstract. We consider the problem of the computation of $\inf_P \theta p$ over the set of exponent pairs $P \ni p$ under linear constrains for a certain class of objective functions θ. An effective algorithm is presented. The output of the algorithm leads to the improvement and establishing new estimates in the various divisor problems in the analytical number theory.

1 Introduction

Exponent pairs are an extremely important concept in the analytical number theory. They are defined implicitly.

Definition 1 ([8, Ch. 2]) *A pair* (k, l) *of real numbers is called an exponent pair if* $0 \leqslant k \leqslant 1/2 \leqslant l \leqslant 1$*, and if for each* $s > 0$ *there exist integer* $r > 4$ *and real* $c \in (0, 1/2)$ *depending only on* s *such that the inequality*

$$\sum_{a < n \leqslant b} e^{2\pi i f(n)} \ll z^k a^l$$

holds with respect to s *and* u *when the following conditions are satisfied:*

$$u > 0, \qquad 1 \leqslant a < b < au, \qquad y > 0, \qquad z = ya^{-s} > 1;$$

$f(t)$ *being any real function with differential coefficients of the first* r *orders in* $[a, b]$ *and*

$$\left| f^{(\nu+1)}(t) - y\frac{d^\nu}{dt^\nu} t^{-s} \right| < (-1)^\nu cy \frac{d^\nu}{dt^\nu} t^{-s}$$

for $a \leqslant t \leqslant b$ *and* $0 \leqslant \nu \leqslant r - 1$.

Computing Classification System 1998: G.1.6, F.2.1.
Mathematics Subject Classification 2010: 90C05, 11Y16, 11L07, 11N37
Key words and phrases: linear programming, exponent pairs, divisor problem

But for the computational purposes more explicit construction is needed.

Proposition 2 *The set of the exponent pairs includes a convex hull* $\operatorname{conv} P$ *of the set* P *such that*

1. P *includes a subset of initial elements* P_0*, namely*
   (a) $(0,1)$ [8],
   (b) $(2/13+\varepsilon, 35/52+\varepsilon)$, $(13/80+\varepsilon, 1/2+13/80+\varepsilon)$, $(11/68+\varepsilon, 1/2+11/68+\varepsilon)$ [3],
   (c) $(9/56+\varepsilon, 1/2+9/56+\varepsilon)$ [5],
   (d) $(89/560+\varepsilon, 1/2+89/560+\varepsilon)$ [11],
   (e) $H_{05} := (32/205+\varepsilon, 1/2+32/205+\varepsilon)$ [4].
2. $A(k,l) \in P$ *and* $BA(k,l) \in P$ *for every* $(k,l) \in P$*, where operators* A *and* B *are defined as follows:*

$$A(k,l) = \left(\frac{k}{2(k+1)}, \frac{k+l+1}{2(k+1)}\right), \qquad B(k,l) = (l-1/2, k+1/2)\,.$$

Possibly the set of the exponent pairs includes elements $(k,l) \notin \operatorname{conv} P$, but at least $\operatorname{conv} P$ incorporates all *currently known* exponent pairs. Everywhere below writing "a set of exponent pairs" we mean P in fact.

Denote by Pp a set of exponent pairs, generated from the pair p with the use of operators A and BA. One can check that *currently*

$$\operatorname{conv} P = \operatorname{conv}\bigl(PH_{05} \cup \{(0,1),(1/2,1/2)\}\bigr).$$

Many asymptotic questions of the number theory (especially in the area of divisor problems) come to the optimization task

$$\inf_{(k,l)\in\operatorname{conv} P} \bigl\{\theta(k,l) \bigm| R_i(\alpha_i k + \beta_i l + \gamma_i),\quad i = 1,\dots,j\bigr\}, \tag{1}$$

where $\alpha_i, \beta_i, \gamma_i \in \mathbb{R}$, $R_i \in R_>, R_\geq$, predicate $R_>$ checks whether its argument is a positive value and $R_\geq$ checks whether its argument is non-negative, $i = 1,\dots,j$.

Graham [2] gave an effective method, which in many cases is able to determine

$$\inf_{(k,l)\in\operatorname{conv} P(0,1)} \theta(k,l)$$

with a given precision (for certain θ—even exactly), where

$$\theta \in \Theta := \left\{ (k,l) \mapsto \frac{ak+bl+c}{dk+el+f} \;\middle|\; \begin{array}{c} a,b,c,d,e,f \in \mathbb{R}, \\ dk+el+f > 0 \text{ for } (k,l) \in \operatorname{conv} P \end{array} \right\}.$$

We shall refer to this result as to *Graham algorithm.* Unfortunately, for some objective functions $\theta \in \Theta$ the algorithm fails and, as Graham writes, we should "resort to manual calculations and ad hoc arguments". We discuss possible improvements in Section 4.

The primary aim of the current paper is to provide an algorithm to determine

$$\inf_{(k,l)\in P} \theta(k,l) \tag{2}$$

under a nonempty set of linear constrains (thus $j \neq 0$) and

$$\theta = \max\{\theta_1(k,l), \dots, \theta_m(k,l)\}, \qquad \theta_1, \dots, \theta_m \in \Theta. \tag{3}$$

In Section 2 a useful computational concept of projective exponent pairs is explained. Section 3 is devoted to the exploration of the geometry of P and its results are of separate interest. In Section 4 Graham algorithm is discussed. Section 5 contains the description of the proposed algorithm to solve (2) under linear constrains and (3). In Section 6 new estimates and theoretical results on various divisor problems are given, derived from the observation of particular cases of the output of our algorithm.

2 Projective exponent pairs

Let us map exponent pairs into the real projective space (the concept of such mapping traces back to Graham [2]):

$$\mu\colon \mathbb{R}^2 \to \mathbb{R}^3/(\mathbb{R}\setminus\{0\}), \quad (k,l) \mapsto (k:l:1).$$

For the set of the exponent pairs the inverse mapping

$$\mu^{-1}\colon (k:l:m) \mapsto (k/m, l/m)$$

is also well-defined.

Operators A and BA are mapped by μ into linear operators over projective space:

$$A(k,l) \mapsto \mathcal{A}(k:l:1), \quad \mathcal{A} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}, \quad \mathcal{A}(k:l:m) = \begin{pmatrix} k \\ k+l+m \\ 2k+2m \end{pmatrix}$$

and

$$BA(k,l) \mapsto \mathcal{BA}(k:l:1),\ \ \mathcal{BA} = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \\ 2 & 0 & 2 \end{pmatrix}, \quad \mathcal{BA}(k:l:m) = \begin{pmatrix} l \\ 2k+m \\ 2k+2m \end{pmatrix}.$$

Thus $A = \mu^{-1}\mathcal{A}\mu$ and $BA = \mu^{-1}\mathcal{BA}\mu$

Such projective mappings are very useful to achieve better computational performance.

Firstly, we replace fractional calculations with integer ones.

Secondly, let M be a fixed composition of A and BA. We can evaluate Mp for a set of points p effectively: once precompute the matrix of the projective operator $\mathcal{M}$ and then just calculate $\mu^{-1}\mathcal{M}\mu p$ for each point p.

3 Exploring exponent pairs

Let us split Pp into *generations* $P_n p$ such that

$$P_0 p = \{p\}, \qquad P_n p = AP_{n-1}p \cup BAP_{n-1}p, \qquad n > 0.$$

Let us investigate properties of $P(0,1)$. As soon as

$$A(0,1) = (0,1), \qquad BA(0,1) = (1/2,1/2),$$

$$A(1/2,1/2) = BA(1/2,1/2) = (1/6,2/3),$$

we obtain

$$P(0,1) = \big\{(0,1),(1/2,1/2)\big\} \cup P(1/6,2/3).$$

So it is enough to study $P(1/6,2/3)$.

All initial exponent pairs satisfy inequalities

$$k + l \leqslant 1, \qquad k \leqslant 1/2, \qquad l \geqslant 1/2.$$

One can check that if (k,l) satisfies such inequalities, then $A(k,l)$ and $BA(k,l)$ also do. Thus all exponent pairs fits into the triangle

$$T := \triangle\big((1/2,1/2),(0,1),(0,1/2)\big). \tag{4}$$

Lemma 3 *Denote*

$$P' = (0,1/6) \times (2/3,1), \qquad P'' = (1/6,1/2) \times (1/2,2/3).$$

Let $p := (k,l)$ *be the exponent pair such that* $Ap \in P'$. *Then*

$$APp \subset P', \qquad BAPp \subset P'', \qquad Pp \subset \{p\} \cup P' \cup P''. \tag{5}$$

Proof. Suppose that (5) is true for all generations P_m, $m < n$. Let us prove that it is also true for generation P_n.

We have $BP' = P''$, so it is enough to prove that $AP_{n-1}p \subset P'$. Let (k, l) be an arbitrary element of $P_{n-1}p$ and let $(\kappa, \lambda) = A(k, l)$. There are three possibilities:

1. $(k, l) = p$. Then $(\kappa, \lambda) \in P'$ by conditions of the lemma.

2. $(k, l) \in P'$. Then
$$\kappa = \frac{1}{2} - \frac{1}{2(k+1)} < \frac{1}{2} - \frac{3}{7} = \frac{1}{14},$$
$$\lambda = \frac{1}{2} + \frac{l}{2(k+1)} > \frac{1}{2} + \frac{2/3}{7/3} = \frac{11}{14}.$$

3. $(k, l) \in P''$. Then
$$\kappa = \frac{1}{2} - \frac{1}{2(k+1)} < \frac{1}{2} - \frac{1}{3} = \frac{1}{6},$$
$$\lambda = \frac{1}{2} + \frac{l}{2(k+1)} > \frac{1}{2} + \frac{1/2}{3} = \frac{2}{3}.$$

□

An exponent pair $(1/6, 2/3)$ satisfies conditions of Lemma 3, because

$$A(1/6, 2/3) = (1/14, 11/14).$$

We note that the statement of Lemma 3 can be refined step-by-step, obtaining 4, 8, 16 and so on rectangles, covering Pp more and more precisely.

Remark 4 *There exists another approach to cover* Pp *or the whole* P. *For a set of pairs* (α_i, β_i) *determine with the use of Graham algorithm*

$$\theta_i = \inf_{(k,l) \in \operatorname{conv} Pp} (\alpha_i k + \beta_i l).$$

Then Pp *is embedded into a polygonal area, constrained with the set of inequalities*

$$\alpha_i k + \beta_i l \geqslant \theta_i$$

from the bottom and left (together they form a hyperbola-like line) and by the segment from $(0, 1)$ *to* $(1/2, 1/2)$.

Let us introduce an order $\prec$ on $P(1/6, 2/3)$, defined as

$$(k, l) \prec (\kappa, \lambda) \iff k < \kappa,\ l > \lambda.$$

Theorem 5 *Let* p *be the exponent pair from the statement of Lemma 3. Then the order* $\prec$ *is a strict total order on* $P_n p$ *and this order coincides with the order of the binary Gray codes [7, Ch. 7.2.1.1] over an alphabet* $\{A, BA\}$.

Proof. One can directly check that operator A saves the order:

$$p_1 \prec p_2 \Rightarrow Ap_1 \prec Ap_2$$

and operator B reverses it, so BA reverses it too:

$$p_1 \prec p_2 \Rightarrow BAp_1 \succ BAp_2.$$

Lemma 3 implies that for every $p_1, p_2 \in P_{n-1} p$

$$Ap_1 \prec BAp_2. \tag{6}$$

Combining these facts we obtain the statement of the theorem. □

In the case of $P(1/6, 2/3)$ inequality (6) can be refined up to

$$Ap_1 \prec (1/6, 2/3) \prec BAp_2. \tag{7}$$

Thus $\prec$ is a strict total order over the whole $P(1/6, 2/3)$.

Fig. 1 illustrates our results. Point $(1/6, 2/3)$ divides the set into rectangles P' and P''. These rectangles consists of pairs, where the last applied operator was A, and pairs, where the last applied operator was BA, respectively. All plotted points are total-ordered by $\prec$. Writing out points of the same generation from the left top corner to the right bottom corner we obtain a list of Gray codes. E. g., for the generation 3 we obtain a sequence of 8 codes:

A A A,
A ABA,
ABABA,
ABA A,
BABA A,
BABABA,
BA ABA,
BA A A.

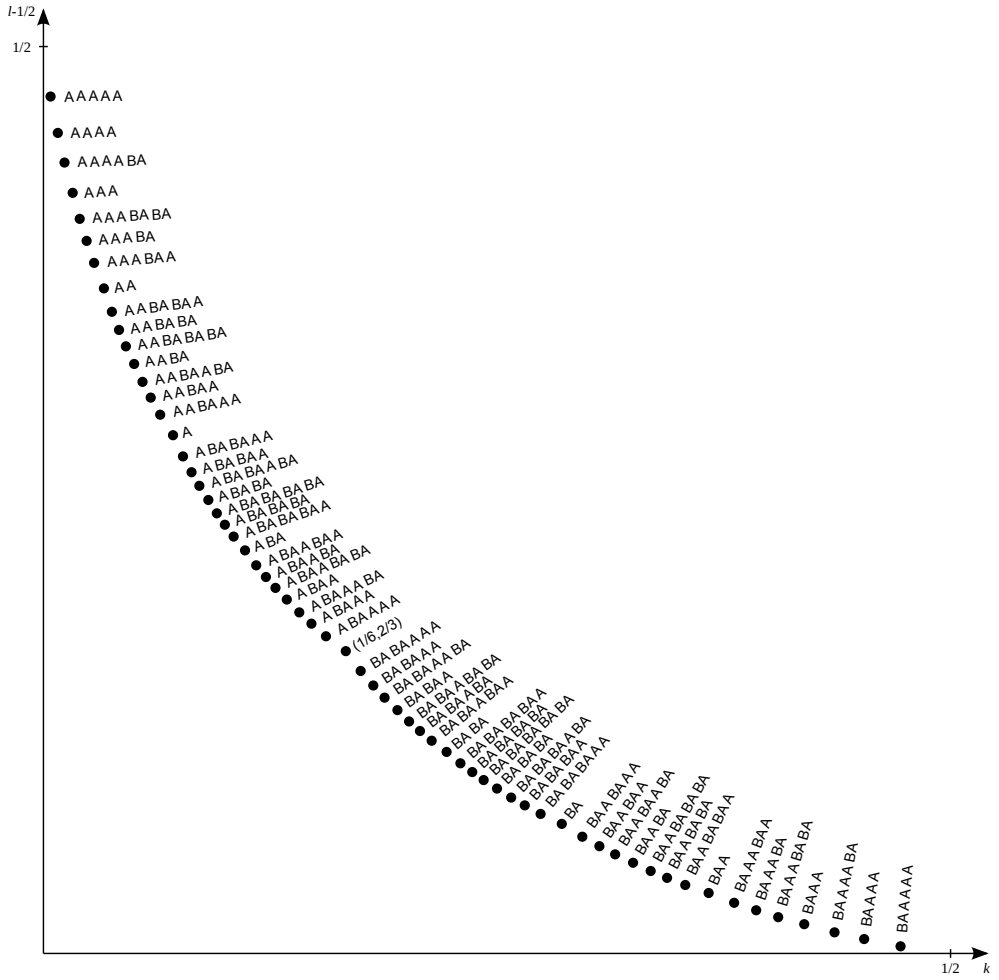

Figure 1: First six generations of $\mathsf{P}(1/6, 2/3)$ plotted in shifted coordinates $(k, l - 1/2)$.

As soon as

$$A^n(1/6, 2/3) \to (0+, 1-0) \qquad \text{as } n \to \infty$$

we obtain

$$\begin{aligned} p_n := A \cdot BA \cdot A^n(1/6, 2/3) &\to (1/6-0, 2/3+0), \\ BA \cdot BA \cdot A^n(1/6, 2/3) &\to (1/6+0, 2/3-0) \qquad \text{as } n \to \infty. \end{aligned}$$

So no point from $P(1/6, 2/3)$ is isolated: for every $p \in P(1/6, 2/3)$ and every $\varepsilon > 0$ there exist $p_1, p_2 \in P(1/6, 2/3)$ such that $p_1 \prec p \prec p_2$, $|p - p_1| < \varepsilon$ and $|p - p_2| < \varepsilon$.

We are even able to compute the slopes of left-hand and right-hand "tangents" at $(1/6, 2/3)$. Namely, using Section 2 and denoting $d_n = p_n - (1/6, 2/3)$ we get

$$d_n/|d_n| \to (-2/\sqrt{5}, 1/\sqrt{5}) \qquad \text{as } n \to \infty,$$

so the left-hand "tangent" at $(1/6, 2/3)$ has a slope $\arctan(-1/2)$. The right-hand "tangent" has a slope $\arctan(-2)$.

What about sets generated from other known initial exponent pairs, listed in Proposition 2? Lemma 3 and Theorem 5 remains valid. But inequality (7) does not hold and so $\prec$ is not a strict total order. E. g., for

$$p = A \cdot BA \cdot A^4 H_{05} = \left(\frac{8083}{50342}, \frac{1}{2} + \frac{4304}{25171} \right)$$

neither $p \prec H_{05}$, nor $p \succ H_{05}$.

As opposed to $P(1/6, 2/3)$, each point of the set Pp, $p \neq (1/6, 2/3)$, is isolated, because the initial point is. But for every such p each point of $P(1/6, 2/3)$ has an arbitrary close to it point from Pp.

Lemma 6 *Operators* A *and* BA *are contractions over the triangle* T*, which was defined in* (4)*.*

Proof. It is enough to prove that A is a contraction. Let us check that there exists $\alpha < 1$ such that for each $p_1, p_2 \in T$ we have

$$|Ap_1 - Ap_2| \leqslant \alpha |p_1 - p_2|.$$

Let $(k_1, l_1) := p_1$ and $(k_2, l_2) := p_2$. Then

$$|Ap_1 - Ap_2|^2 = \frac{1}{4}\left(\left(\frac{1}{k_1+1} - \frac{1}{k_2+1}\right)^2 + \left(\frac{l_1}{k_1+1} - \frac{l_2}{k_2+1}\right)^2\right) =$$
$$= \frac{1}{4}\left(\left(\frac{k_2-k_1}{(k_1+1)(k_2+1)}\right)^2 + \left(\frac{(l_1-l_2)(k_2+1)+l_2(k_2-k_1)}{(k_1+1)(k_2+1)}\right)^2\right).$$

But $k_1, k_2 \geqslant 0$, so

$$|Ap_1 - Ap_2|^2 \leqslant \frac{1}{4}\left((k_1-k_2)^2 + (|l_1-l_2| + |k_1-k_2|)^2\right).$$

Applying inequality $(x+y)^2 \leqslant 2(x^2+y^2)$ we finally obtain

$$|Ap_1 - Ap_2|^2 \leqslant \frac{3}{4}\left((k_1-k_2)^2 + (l_1-l_2)^2\right) = \frac{3}{4}|p_1-p_2|^2.$$

□

4 Notes on Graham algorithm

Below GX means a reference to [2, Step X at p. 209].

1. Graham algorithm is designed to search $\inf_{p\in P(0,1)} \theta p$ and relies on the fact that

$$P(0,1) = AP(0,1) \cup BAP(0,1).$$

This kind of decomposition does not hold for the whole P. Instead we have

$$P = AP \cup BAP \cup \big(P_0 \setminus \{(0,1)\}\big).$$

Thus in order to run Graham algorithm over P, not just over $P(0,1)$, it should be changed in following way. Establish a variable r to keep a current minimal value, setting it initially to $+\infty$. Add an additional step before G5: apply current θ on elements of P_0 and set $r \leftarrow \min(r, \min \theta P_0)$. At the end of the algorithm output r instead of simply $\min \theta P_0$.

2. Unfortunately, Graham algorithm over P is infinite: no analog of halting conditions at G3 provided by [2, Th. 3] can be easily derived. So we should stop depending on whether the desired accuracy is achieved. Cf. Step 2 in the Section 5 below.

3. Bad news: if [2, Th. 1, 2] does not specify the branch to choose at G4 then the original Graham algorithm halts. Good news: [2, Th. 2] can be generalized to cover a wider range of cases. In notations of the mentioned theorem for a given finite sequence $M \in \{A, BA\}^n$ if $\inf \theta BA = \inf \theta BAM$ and if

$$\min(rw + v - u, \alpha w + v - u) \geqslant 0$$

then $\inf \theta = \inf \theta A$, where

$$\alpha := \max\left\{k + l \;\middle|\; (k, l) \in AM\{(0, 1), (1/2, 1/2), (0, 1/2)\}\right\}.$$

4. For the case of linearly constrained optimization one can build a "greedy" modification of Graham algorithm: if at G5 one of the branches is entirely out of constrains then choose another one; otherwise choose a branch in a normal way. Such algorithm executes pretty fast, but misses optimal pairs sometimes.

5 Linear programming algorithm

Now let us return to the optimization problem (2). We will attack it with the use of backtracking.

Operators A and BA perform projective mappings of the plane $\mathbb{R}^2$, so both of them map straight lines into lines and polygons into polygons.

Let θ be as in (3) and a set of linear constrains LC be as in (1).

Denote

$$\theta_+(V) = \max\left\{\sup_{p \in V} \theta_i p\right\}_{i=1}^{m}, \qquad \theta_-(V) = \max\left\{\inf_{p \in V} \theta_i p\right\}_{i=1}^{m}.$$

Then

$$\theta_-(V) \leqslant \inf_{p \in V} \theta p \leqslant \theta_+(V)$$

and these bounds embrace $\inf_{p \in V} \theta p$ tighter and tighter as V becomes smaller. Both θ_+ and θ_- can be computed effectively by simplex method.

Let V be a polygon (or a set of polygons, lines and points) such that $P \subset V$. See Lemma 3 and paragraphs above and below it for possible constructions of V. For a set of linear constrains LC let $R(V, LC)$ be a predicate, which is true if and only if there exists a point $p \in V$, which satisfies all constrains from LC. This predicate can be computed effectively using algorithms for line segment intersections [1, p. 19–44].

The proposed algorithm consists of a routine $\mathsf{L}(\theta, \mathsf{LC}, r, \mathcal{M})$, which calls itself recursively. Here r keeps a current minimal value of $\theta(k, l)$ and $\mathcal{M}$ is a current projective transformation matrix. Initially $r \leftarrow +\infty$ and $\mathcal{M} \leftarrow \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$.

On each call routine L performs following steps:

1. Compute $t \leftarrow \min\{\theta\mu^{-1}\mathcal{M}\mu p\}$, where p runs over all known initial pairs p. If we get $t < r$ then update current minimal value: $r \leftarrow t$.

2. Check whether the desired accuracy is achieved, comparing r with the values of $\theta_+(\mu^{-1}\mathcal{M}\mu V)$ and $\theta_-(\mu^{-1}\mathcal{M}\mu V)$. If yes then return r and abort computations.

3. Set $\mathsf{LC}' \leftarrow \mathsf{LC} \cup \{\theta_i(k, l) < r\}_{i=1}^{m}$. Due to the nature of $\theta_i \in \Theta$ a constrain of form $\theta_i(k, l) < r$ is in fact a linear constrain.

4. If $\mathsf{R}(\mu^{-1}\mathcal{M}\mathcal{A}\mu V, \mathsf{LC}')$ (that means that there is at least a chance to meet exponent pair $p \in \mu^{-1}\mathcal{M}\mathcal{A}\mu V$, which satisfies LC and on which objective function is less than yet achieved value) then compute $t \leftarrow \mathsf{L}(\theta, \mathsf{LC}, r, \mathcal{M}\mathcal{A})$. If $t < r$ set $r \leftarrow t$ and recompute LC' as in Step 3 using the new value of r.

5. If $\mathsf{R}(\mu^{-1}\mathcal{M}\mathcal{B}\mathcal{A}\mu V, \mathsf{LC}')$ then compute $t \leftarrow \mathsf{L}(\theta, \mathsf{LC}, r, \mathcal{M}\mathcal{B}\mathcal{A})$. If $t < r$ set $r \leftarrow t$.

6. Return r.

The algorithm executes in finite time, because due to Lemma 6 both A and BA are contractions and sooner or later (depending on required accuracy) recursively called routines will abort at Step 2.

Step 3 plays a crucial role in chopping off non-optimal branches of the exhaustive search and preventing exponential running time. We are not able to provide any theoretical estimates, but in all our experiments (see Section 6 below) the number of calls of $\mathsf{L}(\cdot, \cdot, \cdot, \cdot)$ behaved like a linear function of the recursion's depth.

We have implemented our algorithm as a program, written in PARI/GP [10]. It appears that it runs pretty fast, in a fraction of a second on the modern hardware.

During computations elements of $\mathcal{M}$ can grow enormously. As soon as matrix $\mathcal{M}$ is applied on projective vectors we can divide $\mathcal{M}$ on the greatest common divisor of its elements to decrease their magnitude.

Now, under which circumstances an equality

$$\inf_{p\in P}\theta p = \inf_{p\in \operatorname{conv} P}\theta p$$

holds? Certainly it is true for $\theta \in \Theta$ and $j = 0$, because sets $\{\theta p = \text{const}\}$ are straight lines; this is the case of Graham algorithm.

Consider the case $\theta \in \Theta$ and $j \neq 0$. Constraining lines are specified by equations

$$l_i = \{\alpha_i k + \beta_i l + \gamma_i = 0\}, \qquad i = 1, \ldots, j.$$

Then

$$\inf_{p\in \operatorname{conv} P}\theta p = \min\left\{\inf_{p\in P}\theta p, \inf_{p\in l_1\cap \operatorname{conv} P}\theta p, \ldots, \inf_{p\in l_j\cap \operatorname{conv} P}\theta p\right\}.$$

But $\operatorname{conv} P$ is approximated by a polygon as in Remark 4, so $l_i \cap \operatorname{conv} P$ can be approximated too and consists of a single segment. Thus $\inf_{p\in l_i\cap \operatorname{conv} P}\theta p$ is computable.

The case when θ is as in (3) with $m > 1$ is different. Even without any constrains the value of $\inf_{p\in \operatorname{conv} P}\theta p$ may be not equal to $\inf_{p\in P}\theta p$. For example, take

$$\theta(k, l) = \max\{11k/10, l - 1/2\}.$$

Then

$$\inf_{p\in P}\theta p = \frac{176}{1025} \qquad \text{at } p = H_{05}.$$

But

$$\inf_{p\in \operatorname{conv} P}\theta p = \frac{176}{1057} \qquad \text{at } p = (160/1057, 1409/2114) := q,$$

and q is owned by a segment from $(0, 1)$ to H_{05}. However, in not-so-synthetic cases the proposed algorithm produces results, which are closer to optimal.

6 Applications

One can run algorithm from the previous section to obtain numerical results in partial cases for different objective functions and constrains. It gives us a way to catch site of some patterns and to suppose general statements on them. Nevertheless these patterns should be proved, not only observed. This is the main theme of the current section.

Consider the asymmetrical divisor problem. Denote

$$\tau(a_1,\dots,a_k;n)=\sum_{d_1^{a_1}\cdots d_k^{a_k}=n}1,$$

which is called *an asymmetrical divisor function.* Let $\Delta(a_1,\dots,a_m;x)$ be an error term in the asymptotic estimate of the sum $\sum_{n\leqslant x}\tau(a_1,\dots,a_m;n)$. (See [8] for the form of the main term.) What upper estimates of Δ can be given? The following result is one of the possible answers.

Theorem 7 ([8, Th. 5.11]) *Let* $a<b$ *and let* $(k,l)=A(\kappa,\lambda)$ *be an exponent pair. Then the estimate*

$$\Delta(a,b;x)\ll x^{\alpha}\log x,\qquad \alpha=\frac{2(k+l-1/2)}{(a+b)}$$

holds under the condition $(2l-1)a\geqslant 2kb$. *Here* $f(x)\ll g(x)$ *denotes* $f(x)=O\bigl(g(x)\bigr)$. *If otherwise* $(2l-1)a<2kb$, *then*

$$\Delta(a,b;x)\ll x^{\alpha}\log x,\qquad \alpha=\frac{k}{(1-l)a+kb}.$$

Taking into account Lemma 3 the condition $(k,l)=A(\kappa,\lambda)$ can be rewritten as $k<1/6$ and $l>2/3$. Thus

$$\theta_1=\frac{2(k+l-1/2)}{a+b},\qquad LC_1=\bigl\{(2l-1)a\geqslant 2kb,k<1/6,l>2/3\bigr\},$$
$$\theta_2=\frac{k}{(1-l)a+kb},\qquad LC_2=\bigl\{(2l-1)a<2kb,k<1/6,l>2/3\bigr\}.$$

Using proposed algorithm we can compute $\inf\theta_1$ under constrains LC_1 (which refers to the first case of Theorem 7), compute $\inf\theta_2$ under constrains LC_2 (which refers to the second case) and take lesser of the obtained values. Observed results shows that for $a=1$, $b=2^r$, $r\geqslant 10$, the second case provides better results and exponent pair has form

$$A^{r-1}BAA^{r-4}BABA\dots.$$

This leads us to the following statement.

Theorem 8 *For a fixed integer* $r\geqslant 5$ *we have* $\Delta(1,2^r;x)\ll x^{\alpha}\log x$, *where*

$$\alpha=\frac{2^r-2r}{2^{2r}-r\cdot 2^r-2r^2+2r-4}<\frac{1}{2^r+r}.$$

Proof. Consider an exponent pair

$$(k_r, l_r) := A^{r-1}BAA^{r-4}(1/6, 2/3).$$

We have

$$\mathcal{A} = \mathcal{S}\begin{pmatrix}1 & 1 & 0\\ 0 & 1 & 0\\ 0 & 0 & 2\end{pmatrix}\mathcal{S}^{-1}, \qquad \mathcal{S} = \begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 1\\ 0 & 2 & 1\end{pmatrix}.$$

Thus $\mathcal{A}^n = \mathcal{S}\left(\begin{smallmatrix}1 & n & 0\\ 0 & 1 & 0\\ 0 & 0 & 2^n\end{smallmatrix}\right)\mathcal{S}^{-1}$. Note that $\mu(1/6, 2/3) = (1 : 4 : 6)$ and

$$\mathcal{A}^{r-1}\mathcal{B}\mathcal{A}\mathcal{A}^{r-4}(1:4:6) = \begin{pmatrix}2^r - 2r\\ 2^{2r+1} - (3r+4)\cdot 2^r + 2r^2 + 2r + 4\\ 2^{2r+1} - (2r+4)\cdot 2^r + 4r\end{pmatrix}.$$

Applying μ^{-1} we get

$$k_r = \frac{2^r - 2r}{2^{2r+1} - (2r+4)\cdot 2^r + 4r}, \quad l_r = 1 - \frac{r\cdot 2^r - 2r^2 + 2r - 4}{2^{2r+1} - (2r+4)\cdot 2^r + 4r}.$$

Now for $r \geqslant 5$

$$2l_r - 2\cdot 2^r k_r - 1 = \frac{2r^2 + 4 - 2^{r+1}}{2^{2r} - (r+2)\cdot 2^r + 2r} < 0.$$

This proves that (k_r, l_r) satisfies the second case of Theorem 7 and finally

$$\alpha = \frac{k_r}{2^r k_r - l_r + 1}.$$

□

In the same manner one can estimate $\Delta(a, 2^r; x)$ for odd a. Here is one more example.

Theorem 9 *For a fixed integer* $r \geqslant 1$ *we have* $\Delta(3, 2^r; x) \ll x^{\alpha+\varepsilon}$*, where*

$$\alpha = \frac{1}{2^r + 3r - 88/17}.$$

Proof. Consider an exponent pair $(k, l) := A^{r-3}BAA(9/56 + \varepsilon, 37/56 + \varepsilon)$. □

In the case of $\Delta(a, b, c)$ one can derive objective function and constrains from [8, Th. 6.2, 6.3] and observe the output of the algorithm.

a	b	(k,l)	$\Xi(a,b)$
1	2	BAH_{05}	$269/1217$
1	3	$(BA)^2ABAH_{05}$	$1486/8647$
1	4	H_{05}	$111/790$
1	5	$ABAA^2BAA(BA)^2A^2M^\infty(0,1)$	$(15921-2c)/30437$
1	6	$(ABA)^3(BA)^3A^3BA(0,1)$	$669/6305$
1	7	$A(BA)^2BAA(BA)^2A^2M^\infty(0,1)$	$(9370-c)/34469$
1	8	$A(BA)^4(A^2BAA)^\infty(0,1)$	$(5+\sqrt{809})/392$
1	9	$A(BA)^2AM^\infty(0,1)$	$(10551-c)/56976$
1	10	$A(BA)^2(A^2(BA)^2)^2ABAH_{05}$	$150509/2096993$
2	3	$BAA(BA)^2A^2M^\infty(0,1)$	$(c-4047)/15688$
2	4	BAH_{05}	$269/2434$
2	5	$M^\infty(0,1)$	$(c-4311)/18672$
3	4	$BAAH_{05}$	$1819/19369$
3	5	$BAA(BA)^3A^2(BA)^3A(BA)^5A^2BA(0,1)$	$63916/774807$
4	5	$BAAH_{05}$	$1819/24903$

Table 1: Estimates of Ξ. Here $M=(BA)^6(ABA)^2BAA^2$ and $c=\sqrt{37368753}$.

Theorem 10 *For a fixed integer* $r\geqslant 10$ *we have*

$$\theta(1,2^r,2^r)=\frac{26\cdot 2^{2r}-(29r+41)2^r+16r^2+12r+32}{26\cdot 2^{3r}-(16r+41)2^{2r}+(24r-3)2^r+16r+12}<\frac{1}{2^r+1}.$$

Proof. Follows from [8, Th. 6.2] with $(k,l)=A^{r-1}BA^{r-2}BABA^2\cdot B(0,1)$. □

Finally, consider the asymmetric divisor problem with congruence conditions on divisors. Namely, let $\tau(a,m_a,r_a;b,m_b,r_b;n)$ be the number of (d_a,d_b) such that

$$d_a^a d_b^b=n,\qquad d_a\equiv r_a\pmod{m_a},\qquad d_b\equiv r_b\pmod{m_b}.$$

Menzer and Nowak showed in [9] that if $a<b$ then the error term in the asymptotic estimate of

$$\sum_{n\leqslant x}\tau(a,m_a,r_a;b,m_b,r_b;n)$$

has form $\left(x/m_a^a m_b^b\right)^{\Xi(a,b)+\varepsilon}$, where

$$\Xi(a,b):=\inf_{(k,l)\in\operatorname{conv}P}\max\left\{\frac{k+l}{(k+1)(a+b)},\frac{k}{kb+a(1+k-l)}\right\},$$

σ	$\mu(\sigma)$	Depth 100	Depth 1000
3/5	1409/12170	10	10
2/3	0.0879154	154	1609
3/4	0.0581840	154	1610
4/5	0.0422535	103	1003

Table 2: Estimates for $\mu(\sigma)$ and the number of calls to L.

where $\varepsilon > 0$ is arbitrary small. They also listed estimates of $\Xi(a, b)$ for $1 \leqslant a < b \leqslant 5$. As soon as $\Xi(a, b)$ is of form (3) we can refine all their results. See Table 1.

Various estimates of the Riemann zeta function depends on optimization tasks (1). The following theorem seems to be the simplest example.

Theorem 11 ([6, (7.57)]) *Let ζ denote the Riemann zeta function and $\sigma \geqslant 1/2$. Further, let $\mu(\sigma)$ be an infimum of all $\varkappa$ such that $\zeta(\sigma + it) \ll t^{\varkappa}$. Then*

$$\mu(\sigma) \leqslant \frac{k + l - \sigma}{2}.$$

for every exponent pair (k, l) such that $l - k \geqslant \sigma$.

Better results on μ leads to better estimates for power moments of ζ, and the last are helpful to improve estimates in multidimensional divisor problem. See [6, Th. 8.4, 13.2, 13.4].

Table 2 contains several results on $\mu(\sigma)$ obtained with the use of the proposed algorithm. Results are accompanied with the number of calls to $L(\cdot, \cdot, \cdot, \cdot)$ up to the given depth of search.

References

[1] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications* (3rd edition), Springer, Berlin, Heidelberg, 2008. ⇒280

[2] S. W. Graham, An algorithm for computing optimal exponent pair, *J. Lond. Math. Soc.* **33,** 2 (1986) 203–218. ⇒272, 273, 279, 280

[3] M. N. Huxley, Exponential sums and the Riemann zeta function, *Proc. International Number Theory Conference,* Laval, Canada, 1987, pp. 417–423. ⇒272

[4] M. N. Huxley, Exponential sums and the Riemann zeta function V, *Proc. Lond. Math. Soc.* **90,** 1 (2005) 1–41. ⇒272

[5] M. N. Huxley, N. Watt, Exponential sums and the Riemann zeta function, *Proc. Lond. Math. Soc.* **57,** 1 (1988) 1–24. ⇒272
[6] A. Ivić *The Riemann Zeta-function: Theory and Applications,* Dover Publications, Mineola, NY, 2003. ⇒286
[7] D. E. Knuth, *The Art of Computer Programming. Vol. 4A. Combinatorial Algorithms, part 1,* Addison–Wesley, Upper Saddle River, NJ, 2011. ⇒276
[8] E. Krätzel. *Lattice Points,* Kluwer, Dordrecht, Boston, London, 1988. ⇒271, 272, 283, 284, 285
[9] H. Menzer, W. G. Nowak, On an asymmetric divisor problem with congruence conditions, *Manuscr. Math.* **64,** 1 (1989) 107–119. ⇒285
[10] The PARI Group, Bordeaux, *PARI/GP, Version 2.6.1,* 2013. ⇒281
[11] N. Watt, Exponential sums and the Riemann zeta function II, *J. Lond. Math. Soc.* **39,** 3 (1989) 385–404. ⇒272

Received: August 28, 2013 • Revised: December 30, 2013

ALGORITHMS OF INFORMATICS Vol. 3 SELECTED TOPICS

Editor: A. Iványi
Publisher: Mondat Kft. (http://www.mondat.hu/), Vác, 2013.
ISBN: 978-963-87596-7-2, 528 pages

First and second volumes of *Algorithms of Informatics* appeared in 2007. These volumes contained 12 + 11 chapters. Volume 3 contains further eight chapters. The base of this volume is the third volume of the electronic edition, which appeared in 2011 and contained 7 chapters.

Volume 3 consists of three parts. Part 7 (DISCRETE METHODS) contains 4 chapters. Chapter 24 (*Comparison-based Ranking*) is written by Antal Iványi (Eötvös Loránd University) and Shariefuddin Pirzada (University of Kashmir), Chapter 25 (*Complexity of Words*) by Zoltán Kása (Sapientia Hungarian University of Transylvania) and Mira-Cristiana Anisiu (Tiberiu Popoviciu Institute of Numerical Analysis of Romanian Academy), Chapter 26 (*Perfect Arrays*) by Antal Iványi (Eötvös Loránd University) and Chapter 27 (*Score Sets and Kings*) by Shariefuddin Pirzada (Kashmir University), Antal Iványi (Eötvös Loránd University) and Muhammad Ali Khan (King Fahd University of Petroleum and Minerals).

Part 8 (PARALLEL METHODS) consists of two chapters. Chapter 28 (*GPGPU: Computing on Graphics Processors*) is due to László Szirmay-Kalos and László Szécsi (both Budapest University of Technology and Economics) and Chapter 29 (*Quantum-based Computing*) is due to László Bacsárdi, and Sándor Imre (all Budapest University of Technology and Economics).

In Part 9 (PRACTICAL METHODS) are presented two chapters. Chapter 30 (*Branch and Bound*) is written by Béla Vizvári, while Chapter 31 (*Conflict Situations*) by Ferenc Szidarovszky (The University of Arizona) and László Domoszlai (Eötvös Loránd University). The book is closed by bibliography, subject index, name index and color version of 16 figures.

The chapters were validated by teachers and researchers of Archidata, Gödöllő University of Agricultural Sciences, Eötvös Loránd University, Nimfea Nature Conservation Association, Sapientia Hungarian University of Transylvania, and University of Szeged.

The volume contains pseudocodes of the presented algorithms, further tables, figures, verbal description and resource requirements, which help to understand the algorithms.

Bibliography contains more than 300 references, mirroring—beside the alien books and papers—and near all of the publications written in Hungarian. Electronic version of the bibliography contains near 1000 links to home pages of cited authors, journals and publishers: if you ask, then the editor of the volume sends you the digital bibliography.

INFORMATIKAI ALGORITMUSOK 3. kötet

Editor: A. Iványi
Publisher: Mondat Kft. (http://www.mondat.hu/), Vác, 2013.
ISBN: 978-963-87596-8-9, 388 pages

First and second volumes of *Informatikai algoritmusok* (in Hungarian) were published in 2004 and 2005, resp. These volumes contained 17 + 14 chapters. Now we present Volume 3 containing further 8 chapters.

Volume 3 consists of three parts. Part 7 titled DISCRETE METHODS contains 3 chapters: Chapter 32 (*Heuristic Graph Searching*) is written by Tibor Gregorics (Eötvös Loránd University), Chapter 33 *(Comparison-based Ranking*) by Antal Iványi (Eötvös Loránd University) and Shariefuddin Pirzada (University of Kashmir), and Chapter 34 (*Complexity of Words*) by Zoltán Kása (Sapientia Hungarian University of Transylvania) and Mira-Cristiana Anisiu (Tiberiu Popoviciu Institute of Numerical Analysis of Romanian Academy).

Part 8 titled PARALLEL METHODS consists of two chapters. Chapter 35 (*GPGPU: Computing on Graphics Processors*) is due to László Szirmay-Kalos and László Szécsi (both Budapest University of Technology and Economics) and Chapter 36 (*Quantum-based Computing*) is due to László Bacsárdi, Máté Galambos and Sándor Imre (all Budapest University of Technology and Economics).

In Part 9 titled PRACTICAL METHODS we present three chapters. Chapter 37 (*Conflict Situations*) is written by Ferenc Szidarovszky (The University of Arizona) and László Domoszlai (Eötvös Loránd University), Chapter 38 (*Query Languages of Semantic Web*) by Balázs Pinczel, Balázs Kósa, Tamás Matuszka and Attila Kiss (all Eötvös Loránd University), and Chapter 39 (*Applications of Semantic Web*) by Gábor Rácz, Gergő Gombos and Attila Kiss (all Eötvös Loránd University).

The book is closed by bibliography, subject index, name index, and color version of 16 figures.

The chapters were validated by teachers and researchers of Archidata (György Antal), Gödöllő University of Agricultural Sciences (Sándor Molnár), Institute for Computer Science and Control (István Csaba Sidló), Nimfea Nature Conservation Association (Anna Iványi), Sapientia Hungarian University of Transylvania (Zoltán Kása), and University of Szeged (Zoltán Blázsik).

The volume contains pseudocodes of the presented algorithms, further tables, figures, verbal description and resource requirements, which help to understand the algorithms.

Bibliography contains near 300 references, mirroring—beside the publications of the Hungarian authors—also books and papers from all over the World.

Contents
Volume 5, 2013

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English.
Information about each series can be found at
http://www.acta.sapientia.ro.

Acta Universitatis Sapientiae, Informatica

Each volume contains two issues.

Sapientia University

Scientia Publishing House

ISSN 1844-6086
http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: `http://www.acta.sapientia.ro`.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (`acta-inf@acta.sapientia.ro`) only, using the LaTeX style and sample file at the address `http://www.acta.sapientia.ro`. Beside the LaTeX source a pdf format of the paper is necessary too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (`http://www.acm.org/about/class/1998`) and AMS Mathematics Subject Classification codes (`http://www.ams.org/msc/`).

References should be listed alphabetically based on the Intructions for Authors given at the address `http://www.acta.sapientia.ro`.

Illustrations should be given in Encapsulated Postscript (eps) format.

One issue is offered each author free of charge. No reprints will be available.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by Gloria Printing House
Director: Péter Nagy

ISSN 1844-6086
http://www.acta.sapientia.ro