About FOI-2015

This CSJM issue and a part of the next one are devoted to the first edition of International Scientific Workshop on “Foundations of Informatics” (FOI). They contain selected, revised and extended papers presented at this scientific event.

The event took place in Moldova on August 24 – 29, 2015. The organizers of Workshop FOI 2015 are Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova, Taras Shevchenko National University of Kyiv, Ukraine, East Computers, SRL in co-operation with the Information Society Development Institute and Tiraspol State University.

The first workshop on FOI was aimed to bring together researchers from East and West Europe and worldwide by adding synergy to their endeavors to lay down and foster the foundations of computer science also known as informatics.

About 40 scientists from 8 countries (Moldova, Netherlands, Russia, Romania, Turkey, Ukraine, UK, USA) attended the workshop: Every FOI working day started with the invited speakers' speeches:

- Acad. Solomon Marcus, Honorary member of "Simion Stoilow" Institute of Mathematics of the Romanian Academy;
- Dr Alexandru Baltag, Associate Researcher, Institute for Logic, Language and Computation, Amsterdam, Netherlands;
- Acad. Gheorghe Paun, Senior researcher of "Simion Stoilow" Institute of Mathematics of the Romanian Academy;
- Dr Sonja Smets, Associate Researcher, Institute for Logic, Language and Computation, Amsterdam, Netherlands;
- Dr. Razvan Diaconescu, Research Professor of "Simion Stoilow" Institute of Mathematics of the Romanian Academy;
- Dr. Vadim Ermolayev, Associate Professor, Dept. Information Technologies, Zaporizhzhya National University;
- Dr. Gabriel Ciobanu, Institute of Computer Science, Romanian Academy, Iasi Branch, Romania.

FOI-2015 has *brought more* focus on such foundational aspects of informatics treated by mathematical methods as Theory of computing, Theoretical aspects of software system development, Natural computing, Theoretical issues in automated reasoning, Logics in informatics, Formal languages and automata, Semantic technologies, Natural language processing, Cryptography and security.

Round tables organized at the end of each workshop day provided open debates on the state of the art in the declared problem and new directions of the respective field. Discussions in round tables were devoted to logics in informatics, reviewing research directions and results in institutions represented by the participants.

At the last day round table discussion the FOI-2015 participants have noted the high level of the event organization and specially mentioned the significance of presented results. Also the action items required for the renaissance of the East Europe research in domain were discussed. The necessity of periodical organization of conferences on FOI next years was formulated. At the same time many connections for personal participants' collaboration and their affiliation were established.

CSJM Editorial board

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Semantic Properties of T-consequence Relation in Logics of Quasiary Predicates[*]

Mykola Nikitchenko, Stepan Shkilniak

Abstract

In the paper we investigate semantic properties of program-oriented algebras and logics defined for classes of quasiary predicates. Informally speaking, such predicates are partial predicates defined over partial states (partial assignments) of variables. Conventional n-ary predicates can be considered as a special case of quasiary predicates. We define first-order logics of quasiary non-deterministic predicates and investigate semantic properties of T-consequence relation for such logics. Specific properties of T-consequence relation for the class of deterministic predicates are also considered. Obtained results can be used to prove logic validity and completeness.

Keywords: First-order logic, quasiary predicate, partial predicate, non-deterministic predicate.

1 Introduction

Mathematical logic is one of the basic disciplines for computer science. To use effectively mathematical logic it is important to construct logical systems that are adequate for problems considered in computer science. Classical logic, despite its numerous advantages, has some restrictions for its use in this area. For example, classical logic is based on the class of total n-ary predicates, while in computer science partial and non-deterministic predicates often appear. Therefore many logical

[*] This work was partially supported by the research project No. 11BF015-02 "Formal specifications and methods of development of reliable software systems", Taras Shevchenko National University of Kyiv, Ukraine.

systems which better reflect properties of such kind were constructed [1, 2]. One of specific features for computer science is quasiarity of predicates. Such predicates are partial predicates defined over partial states (partial assignments) of variables. Conventional n-ary predicates can be considered as a special case of quasiary predicates. In our previous works [3, 4, 5] we investigated the class of partial deterministic (single-valued) predicates and constructed corresponding logics. For such logic a natural extension of conventional logical consequence relation, called irrefutability relation, was used.

This paper aims to develop a semantic basis for construction of logics of non-deterministic (many-valued) quasiary predicates. To realize this idea we first construct predicate algebras using *composition-nominative approach* [6]. Terms of such algebras specify the language of logic. Then we define interpretation mappings. At last, we construct calculi of sequent type for defined logics. It is important to admit that constructed logics better reflect specifics of computer science problems, but the opposite side of this feature is that the methods of logic investigation turn out to be more complicated. In particular, the irrefutability relation collapses for the class of non-deterministic predicates. Therefore in this paper which is an extended version of [7] we concentrate on semantic properties of a special T-consequence relation. We also formulate properties of this relation for logics of deterministic predicates.

The rest of the paper is structured as follows. In Section 2 we define first-order algebras of quasiary predicates. In Sections 3 we define logics of quasiary predicates. Section 4 is devoted to semantic properties of such logics. In Section 5 T-consequence relation is specified and its main properties are studied. Section 6 is devoted to properties of T-consequence relation for the class of deterministic predicates. In Section 7 conclusions are formulated.

Arrows $\xrightarrow{t}$, $\xrightarrow{p}$, and $\xrightarrow{r}$ specify total, partial, and relational mappings respectively. Notations not defined in this paper are understood in a sense of [4].

2 First-order algebras of quasiary predicates

Let V be a nonempty *set of names.* According to tradition, names from V are also called *variables.* Let A be a set of basic values ($A \neq \emptyset$). Given V and A, the class ${}^{V}\!A$ of *nominative sets* is defined as the class of all partial mappings from V to A, thus, ${}^{V}\!A = V \xrightarrow{p} A$. Informally speaking, nominative sets represent states of variables.

Though nominative sets are defined as mappings, we follow mathematical tradition and also use set-like notation for these objects. In particular, the notation $d = [v_i \mapsto a_i \mid i \in I]$ describes a nominative set d; the notation $v_i \mapsto a_i \in_n d$ means that $d(v_i)$ is defined and its value is a_i ($d(v_i) \downarrow= a_i$). The main operation for nominative sets is a total unary parametric *renomination* $r_{x_1,\ldots,x_n}^{v_1,\ldots,v_n} : {}^{V}\!A \xrightarrow{t} {}^{V}\!A$, where $v_1, ..., v_n, x_1, ..., x_n \in V$, $v_1, ..., v_n$ are distinct names, $n \geq 0$, which is defined by the following formula:

$$r_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}(d) =$$
$$= [v \mapsto a \in_n d \mid v \notin \{v_1, ..., v_n\}] \cup [v_i \mapsto d(x_i) \mid d(x_i) \downarrow, i \in \{1, ..., n\}].$$

Intuitively, given d this operation yields a new nominative set changing the values of $v_1, ..., v_n$ to the values of $x_1, ..., x_n$ respectively. We also use simpler notation for this formula: $r_{\bar{x}}^{\bar{v}}(d) = d\nabla\bar{v} \mapsto d(\bar{x})$. Also note that we treat a parameter ${}_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}$ as a total mapping from $\{v_1, ..., v_n\}$ into $\{x_1, ..., x_n\}$ thus parameters obtained by pairs permutations are identical.

Operation of deleting a component with a name v from a nominative set d is denoted $d|_{-v}$. Notation $d =_{-v} d'$ means that $d|_{-v} = d'|_{-v}$. The set of *assigned names* (*variables*) in d is defined by the formula

$$asn(d) = \{v \in V \mid v \mapsto a \in_n d \text{ for some } a \in A\}.$$

Let $Bool = \{F, T\}$ be a set of Boolean values.

Let $PrR_A^V = {}^{V}\!A \xrightarrow{r} Bool$ be the set of all non-deterministic (relational) predicates over ${}^{V}\!A$. Such predicates are called *non-deterministic (relational) quasiary predicates.* The term 'relational' means that graphs of such predicates are binary relations from ${}^{V}\!A \times Bool$. Note that non-determinism in logic was intensively studied, see, for example, [8].

We will also use set-theoretic notations for quasiary predicates.

Full image of $d \in {}^V\!A$ under $p \in PrR_A^V$ is defined by the formula $p[d] = \{b \in Bool \mid (d, b) \in p\}$.

For $p \in PrR_A^V$ the truth and falsity domains of p are respectively

$$T(p) = \{d \in {}^V\!A \mid (d, T) \in p\} \text{ and } F(p) = \{d \in {}^V\!A \mid (d, F) \in p\}.$$

Considering predicates from PrR_A^V in set-theoretic style we can speak about such operations as union $\cup$ and intersection $\cap$. The following statement is obvious.

Lemma 1. *The set* $< PrR_A^V; \cup, \cap >$ *is a complete distributive lattice.*

The greatest and the least elements of this lattice are denoted $\top_A^V$ and $\perp_A^V$ respectively. For these elements $T(\top_A^V) = {}^V\!A$, $F(\top_A^V) = {}^V\!A$, $T(\perp_A^V) = \emptyset$, $F(\perp_A^V) = \emptyset$.

Operations over PrR_A^V are called *compositions*. The set $C(V)$ of first-order compositions is $\{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x\}$. Compositions have the following types:

$\vee : PrR_A^V \times PrR_A^V \xrightarrow{t} PrR_A^V$; $\neg, R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}, \exists x : PrR_A^V \xrightarrow{t} PrR_A^V$ and are defined by the following formulas ($p, q \in PrR_A^V$):

- $T(p \vee q) = T(p) \cup T(q)$; $F(p \vee q) = F(p) \cap F(q)$;
- $T(\neg p) = F(p)$; $F(\neg p) = T(p)$;
- $T(R_{\bar{x}}^{\bar{v}}(p)) = \{d \in {}^V\!A \mid r_{\bar{x}}^{\bar{v}}(d) \in T(p)\}$;
  $F(R_{\bar{x}}^{\bar{v}}(p)) = \{d \in {}^V\!A \mid r_{\bar{x}}^{\bar{v}}(d) \in F(p)\}$;
- $T(\exists x p) = \{d \in {}^V\!A \mid d\nabla x \mapsto a \in T(p) \text{ for some } a \in A\}$;
  $F(\exists x p) = \{d \in {}^V\!A \mid d\nabla x \mapsto a \in F(p) \text{ for all } a \in A\}$.

Here $d\nabla x \mapsto a = [v \mapsto c \in_n d \mid v \neq x] \cup [x \mapsto a]$. Conventional notation is $d[v \mapsto a]$.

Please note that definitions of compositions are similar to strong Kleene's connectives and quantifiers.

Also note that parametric compositions of existential quantification and renomination can also represent classes of compositions. Thus, notation $\exists x$ can represent one composition, when x is fixed, or a class $\{\exists x \mid x \in V\}$ of such compositions for various names.

A pair $AQR(V, A) =< PrR_A^V; C(V) >$ is called *a first-order algebra of non-deterministic quasiary predicates.*

It is not difficult to prove the following statement.

Lemma 2. *Singleton sets $\{\top_A^V\}$ and $\{\bot_A^V\}$ are sub-algebras of algebra $AQR(V, A)$.*

Algebras $AQR(V, A)$ (for various A) form a semantic base for the constructed first-order pure quasiary predicate logic L^{QR} (called also quasiary logic). Let us now proceed with formal definitions.

3 First-order pure quasiary logic

To define a logic we should first specify its semantic component, syntactic component, and interpretational component [3, 4, 5]. Then a consequence relation should be defined. Semantics of the logic under consideration is specified by algebras of the type $AQR(V, A)$ (for various A), so, we proceed with syntactic component of the logic.

3.1 Syntactic component

A syntactic component specifies the language of L^{QR}. Let $Cs(V)$ be *a set of composition symbols* that represent compositions in algebras defined above – $Cs(V) = \{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x\}$. For simplicity, we use the same notation for symbols of compositions and compositions themselves.

Let Ps be a set of *predicate symbols.* A triple $\Sigma^Q = (V, Cs(V), Ps)$ is a language *signature.* Given Σ^Q , we inductively define the language of L^{QR} – *the set of formulas* $Fr(\Sigma^Q)$:

1) if $P \in Ps$, then $P \in Fr(\Sigma^Q)$; such formulas are called atomic;

2) if Φ, $\Psi \in Fr(\Sigma^Q)$, then $(\Phi \vee \Psi) \in Fr(\Sigma^Q)$;

3) if $\Phi \in Fr(\Sigma^Q)$, then $(\neg\Phi) \in Fr(\Sigma^Q)$;

4) if $\Phi \in Fr(\Sigma^Q)$, $v_1, ..., v_n$, $x_1, ..., x_n \in V$, $v_1, ..., v_n$ are distinct names, $n \geq 0$, then $(R^{v_1,...,v_n}_{x_1,...,x_n}(\Phi)) \in Fr(\Sigma^Q)$;
   for such formulas notation $R^{v_1,...,v_n}_{x_1,...,x_n}\Phi$ or $R^{\bar{v}}_{\bar{x}}\Phi$ can be also used;

5) if $\Phi \in Fr(\Sigma^Q)$, $x \in V$, then $(\exists x\Phi) \in Fr(\Sigma^Q)$.

Extra brackets can be omitted using conventional rules of operation priorities. Derived operations like conjunction $\wedge$, implication $\rightarrow$ etc. are defined in a usual way.

3.2 Interpretational component

Given Σ^Q and nonempty set A we can consider an algebra of quasiary predicates $AQR(V, A) =< PrR^V_A; C(V) >$. Composition symbols have fixed interpretation, but we additionally need interpretation $I^{Ps} : Ps \xrightarrow{t} PrR^V_A$ of predicate symbols; obtained predicates are called *basic predicates.* A tuple $J = (\Sigma^Q, A, I^{Ps})$ is called *an interpretation.*

Formulas and interpretations in L^{QR} are called L^{QR}*-formulas* and L^{QR}*-interpretations* respectively. Usually the prefix L^{QR} is omitted. Given a formula Φ and an interpretation J we can speak of an *interpretation of* Φ in J. It is denoted by Φ_J.

3.3 Extensions of L^{QR}

The logic L^{QR} being a rather powerful logic still is not expressive enough to represent transformations required for proving its completeness. Therefore we introduce its two extensions: L^{UR} — *a logic with unessential variables*, and L^{UR}_ε — a logic with unessential variables and a parametric total deterministic *variable unassignment predicate* εz which checks if a variable z is unassigned in a given nominative set.

To define L^{UR} we should specify its semantic, syntactic, and interpretational components.

Let U be an infinite set of variables such that $V \cap U = \emptyset$. Variables from U are called *unessential variables* (analogs of fresh variables in classical logic) that should not affect the formula meanings.

Algebras

$$AQR(V \cup U, A) = < Pr_A^{V \cup U}; C(V \cup U) >$$

(for different A) form a semantic base for L^{UR}.

A syntactic component is specified by the set of formulas $Fr(\Sigma^U)$, where $\Sigma^U = (V \cup U, Cs(V \cup U), Ps)$ is the *signature* of L^{UR}.

An interpretational component restricts the class of L^{UR}-interpretations in such a way that interpretations of predicate symbols are neither sensitive to the values of the component with an unessential variable u in nominative sets, nor to presence of such components. Formally, a variable $u \in U$ is *unessential in an interpretation of predicate symbols* I^{Ps} if $I^{Ps}(P)[d] = I^{Ps}(P)[d']$ for all $P \in Ps$, $d, d' \in {}^{V \cup U}A$ such that $d =_{-u} d'$.

The following statement is obvious.

Lemma 3. *L^{UR} is a model-theoretic conservative extension of L^{QR}.*

Note that given $p \in PrR_A^{V \cup U}$ and $v \in V \cup U$ we say that v is unessential for p if $p[d] = p[d']$ for any $d, d' \in {}^{V \cup U}A$ such that $d =_{-v} d'$.

The next logic L_ε^{UR} is an extension of L^{UR} by a null-ary parametric composition (predicate) εz ($z \in V \cup U$) defined in interpretation J by the following formulas:

$$T(\varepsilon z_J) = \{d \in {}^{V \cup U}A \mid z \notin asn(d)\},$$
$$F(\varepsilon z_J) = \{d \in {}^{V \cup U}A \mid z \in asn(d)\}.$$

Thus, for this logic the set of compositions is equal to $\{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, \varepsilon z\}$.

Note that in free logic [9] $E!z$ corresponds to negation of εz.

Algebras of the form

$$ARE(V \cup U, A) = < Pr_A^{V \cup U}; \vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, \varepsilon z >$$

(for different A) constitute a semantic base for L_ε^{UR}.

A syntactic component is specified by the set of formulas $Fr(\Sigma_\varepsilon^U)$, where $\Sigma_\varepsilon^U = (V \cup U, \{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, \varepsilon z\}, Ps)$ is the signature of L_ε^{UR}.

An interpretational component of L_ε^{UR} is defined in the same way as for L^{UR}.

By construction of L_ε^{UR} we get the following statement.

Lemma 4. *L_ε^{UR} is a model-theoretic conservative extension of L^{UR}.*

Predicates εz specify cases when z is assigned or unassigned. This property can be used for construction of sequent rules for quantifiers.

For a formula Φ and a set of formulas Γ let $nm(\Phi)$ denote all names (variables) that occur in Φ, $nm(\Gamma)$ denote all names that occur in formulas of Γ. Names from $U\backslash nm(\Phi)$ are called fresh unessential variables for Φ and their set is denoted $fu(\Phi)$, in the same way $fu(\Gamma) = U\backslash nm(\Gamma)$ is the set of fresh unessential variables for Γ. We also use natural extensions of this notation for a case of several formulas and sets of formulas like $nm(\Gamma, \Delta, R_{\bar{v}}^{\bar{u}}(\exists x\Phi))$ and $fu(\Gamma, \Delta, R_{\bar{v}}^{\bar{u}}(\exists x\Phi))$. Such notation is also used when we consider properties of predicate algebras. We write $x \in \bar{v}$ to denote that x is a variable from $\bar{v}$. We write $\{\bar{v}, \bar{x}\}$ to denote the set of variables that occur in the sequences $\bar{v}$ and $\bar{x}$.

In the sequel we adopt the following convention: a, b denote elements from A; x, y, z, v, w (maybe with indexes) denote variables (names) from $V \cup U$; d, d', d_1, d_2 denote nominative sets from ${}^{V\cup U}\!A$; p, q denote predicates from $ARE(V \cup U, A)$; Φ, Ψ, Ξ denote L_{ε}^{UR}-formulas, Γ, Δ denote sets of L_{ε}^{UR}-formulas, J denotes L_{ε}^{UR}-interpretation.

4 Semantic properties of quasiary logics

The set of compositions $\{\vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, \varepsilon z\}$ of quasiary logics specifies four types of properties related to propositional compositions $\vee$ and $\neg$, to renomination composition $R_{\bar{x}}^{\bar{v}}$, to unassignment composition (predicate) εz, and to existential quantifier $\exists x$.

4.1 Properties related to propositional compositions

Properties of propositional compositions are traditional. In particular, disjunction composition is associative, commutative, and idempotent; negation composition is involutive

$$\neg\neg\text{: } \neg\neg p = p.$$

4.2 Properties related to renomination composition

Renomination composition is a new composition specific for logics of quasiary predicates. Its properties are not well-known therefore we describe them in more detail. The main attention will be paid to distributivity properties.

Lemma 5. *For every algebra ARE(V∪U, A) the following properties related to renomination composition hold:*

R∨: $R_{\bar{x}}^{\bar{v}}(p \vee q) = R_{\bar{x}}^{\bar{v}}(p) \vee R_{\bar{x}}^{\bar{v}}(q)$;

R¬: $R_{\bar{x}}^{\bar{v}}(\neg p) = \neg R_{\bar{x}}^{\bar{v}}(p)$;

RI: $R_{z,\bar{x}}^{z,\bar{v}}(p) = R_{\bar{x}}^{\bar{v}}(p)$;

RU: $R_{z,\bar{x}}^{y,\bar{v}}(p) = R_{\bar{x}}^{\bar{v}}(p)$, *y is unessential for* $R_{\bar{x}}^{\bar{v}}(p)$;

RR: $R_{\bar{x}}^{\bar{v}}(R_{\bar{y}}^{\bar{w}}(p)) = R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}} (p)$;

R: $R(p) = p$;

R∃s: $R_{\bar{x}}^{\bar{v}}(\exists yp) = \exists y(R_{\bar{x}}^{\bar{v}}(p))$, $y \notin \{\bar{v}, \bar{x}\}$;

R∃r: $\exists yp = \exists z R_z^y(p)$, *z is unessential for p;*

R∃: $R_{\bar{x}}^{\bar{v}}(\exists yp) = \exists z R_{\bar{x}}^{\bar{v}}(R_z^y(p))$, *z is unessential for* $R_{\bar{x}}^{\bar{v}}(\exists yp)$;

R∃R: $R_{z,\bar{x}}^{y,\bar{v}}(\exists yp) = R_{\bar{x}}^{\bar{v}}(\exists yp)$.

Here $R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}}$ represents two successive renominations $R_{\bar{y}}^{\bar{w}}$ and $R_{\bar{x}}^{\bar{v}}$.

Proof. We prove the lemma by showing that truth and falsity domains of predicates in the left- and right-hand sides of equalities coincide. Let us consider properties R∃s, R∃r, and R∃ only.

For R∃s we have:

$d \in T(R_{\bar{x}}^{\bar{v}}(\exists yp)) \Leftrightarrow r_{\bar{x}}^{\bar{v}}(d) \in T(\exists yp) \Leftrightarrow r_{\bar{x}}^{\bar{v}}(d)\nabla y \mapsto a \in T(p)$ for some $a \in A \Leftrightarrow$ (since $y \notin \{\bar{v}, \bar{x}\}$) $r_{\bar{x}}^{\bar{v}}(d\nabla y \mapsto a) \in T(p)$ for some $a \in A$ $\Leftrightarrow d\nabla y \mapsto a \in T(R_{\bar{x}}^{\bar{v}}(p))$ for some $a \in A \Leftrightarrow d \in T(\exists y(R_{\bar{x}}^{\bar{v}}(p)))$;

$d \in F(R_{\bar{x}}^{\bar{v}}(\exists yp)) \Leftrightarrow r_{\bar{x}}^{\bar{v}}(d) \in F(\exists yp) \Leftrightarrow r_{\bar{x}}^{\bar{v}}(d)\nabla y \mapsto a \in F(p)$ for all $a \in A \Leftrightarrow$ (since $y \notin \{\bar{v}, \bar{x}\}$) $r_{\bar{x}}^{\bar{v}}(d\nabla y \mapsto a) \in F(p)$ for all $a \in A$ $\Leftrightarrow d\nabla y \mapsto a \in F(R_{\bar{x}}^{\bar{v}}(p))$ for all $a \in A \Leftrightarrow d \in F(\exists y(R_{\bar{x}}^{\bar{v}}(p)))$.

For R∃r we have:

$d \in T(\exists z(R_z^y(p))) \Leftrightarrow d\nabla z \mapsto a \in T(R_z^y(p))$ for some $a \in A \Leftrightarrow$ $r_z^y(d\nabla z \mapsto a) \in T(p)$ for some $a \in A \Leftrightarrow (d\nabla z \mapsto a)\nabla y \mapsto a \in T(p)$ for some $a \in A \Leftrightarrow$ (since z is unessential for p) $d\nabla y \mapsto a \in T(p)$ for some $a \in A \Leftrightarrow d \in T(\exists yp)$.

In the same way we demonstrate coincidence of the falsity domains for R$\exists r$.

By R$\exists s$ and R$\exists r$ we obtain R$\exists$. $\square$

4.3 Properties related to unassignment composition

Here we formulate only that null-ary unassignment composition (predicate) is total deterministic predicate, i.e.

$$T(\varepsilon y) \cup F(\varepsilon y) = {}^{V}\!A \text{ and } T(\varepsilon y) \cap F(\varepsilon y) = \emptyset.$$

4.4 Properties related to quantifier composition

The following lemmas describe properties of quantifiers.

Lemma 6. *For every algebra $ARE(V \cup U, A)$ and every $p \in PrR_A^{V \cup U}$ the following properties hold ($x \neq y$):*

$$T\exists v : T(R_y^x(p)) \cap F(\varepsilon y) \subseteq T(\exists xp);$$
$$F\exists v : F(\exists xp) \cap F(\varepsilon y) \subseteq F(R_y^x(p));$$
$$T\exists u : T(R_y^x(p)) \subseteq T(\varepsilon y) \cup T(\exists xp);$$
$$F\exists u : F(\exists xp) \subseteq T(\varepsilon y) \cup F(R_y^x(p)).$$

Proof. To prove $T\exists v$ consider arbitrary $d \in T(R_y^x(p)) \cap F(\varepsilon y)$. This means that y is assigned in d with some value a and $d\nabla x \mapsto a \in T(p)$, therefore $d \in T(\exists xp)$.

Property $F\exists v$ is proved in the same manner.

Properties $T\exists u$ and $F\exists u$ are obtained from $T\exists v$ and $F\exists v$ using the following property of Boolean algebra of sets:

$$\text{SI: } S_1 \cap S_2 \subseteq S_3 \Leftrightarrow S_1 \subseteq \overline{S_2} \cup S_3,$$

where $\overline{S_2}$ denotes supplement of S_2 and the properties that $\overline{T(\varepsilon y)} = F(\varepsilon y)$ and $\overline{F(\varepsilon y)} = T(\varepsilon y)$. $\square$

Lemma 7. *For every algebra $ARE(V \cup U, A)$ the following property holds ($x \neq y$):*

$\exists e_L$*:* $T(\exists xp) =_{-y} (T(R_y^x(p)) \cap F(\varepsilon y))$ if *y is unessential for p.*

Proof. Let $d \in T(\exists xp)|_{-y}$. It means that there exists $d' \in {}^{V \cup U}\!A$ and $a \in A$ such that $d'\nabla x \mapsto a \in T(p)$ and $d' =_{-y} d$. Since y is not essential for p, then $(d'\nabla x \mapsto a)\nabla y \mapsto a \in T(p)$. By definition, we get that $d'\nabla y \mapsto a \in T(R_y^x(p)) \cap F(\varepsilon y)$. But $d = d'|_{-y}$. Thus,

$$T(\exists xp)|_{-y} \subseteq (T(R_y^x(p)) \cap F(\varepsilon y))|_{-y}.$$

The inverse follows from $T\exists v$. $\square$

5 T-consequence relation for sets of formulas

Traditionally, for logics of quasiary predicates a conventional logical consequence is considered [3, 4].

Let $\Gamma \subseteq Fr(\Sigma_\varepsilon^U)$ and $\Delta \subseteq Fr(\Sigma_\varepsilon^U)$ be sets of formulas. Δ is a *consequence* of Γ in an interpretation J (denoted $\Gamma_J \models \Delta$), if $\bigcap\limits_{\Phi\in\Gamma} T(\Phi_J) \cap \bigcap\limits_{\Psi\in\Delta} F(\Psi_J) = \emptyset$.

Δ is a logical consequence of Γ (denoted $\Gamma \models \Delta$), if $\Gamma_J \models \Delta$ in every interpretation J. The introduced relation of logical consequence specifies irrefutability.

For the class of non-deterministic predicates the logical consequence relation collapses, i.e. it is empty. Indeed, for any Γ and Δ we have that $\Gamma_J \not\models \Delta$ if we in J interpret predicate symbols as non-deterministic predicate $\top_A^V$ (Lemma 2).

Therefore we introduce another consequence relation which arises naturally in Computer Science [10].

Δ is a *T-consequence* of Γ in an interpretation J (denoted by $\Gamma_J \models_T \Delta$), if $\bigcap\limits_{\Phi\in\Gamma} T(\Phi_J) \subseteq \bigcup\limits_{\Psi\in\Delta} T(\Psi_J)$. Δ is a T-consequence of Γ (denoted by $\Gamma \models_T \Delta$), if $\Gamma_J \models_T \Delta$ in every interpretation J.

We will also use the following notation: $T^\wedge(\Gamma_J) = \bigcap\limits_{\Phi\in\Gamma} T(\Phi_J)$ and $T^\vee(\Gamma_J) = \bigcup\limits_{\Phi\in\Gamma} T(\Phi_J)$.

Now we describe the main properties of T-consequence relation.

First, let us give the following definitions for arbitrary consequence relation $\models_*$ [11]:

– $\models_*$ is called *paraconsistent* if there exist Γ, Δ, and Φ, such that $\Gamma, \Phi \wedge \neg\Phi \not\models_* \Delta$;

- $\models_*$ is called *paracomplete* if there exist Γ, Δ, and Ψ such that $\Gamma \not\models_* \Psi \vee \neg\Psi, \Delta$;
- $\models_*$ is called *paranormal* if there exist Γ, Δ, Φ, and Ψ such that $\Gamma, \Phi \wedge \neg\Phi \not\models_* \Psi \vee \neg\Psi, \Delta$.

We say that $\models_*$ is consistent, complete, and normal if it is not paraconsistent, not paracomplete, and not paranormal respectively.

It is easy to see that paranormality implies paraconsistency and paracompleteness; consistency or completeness implies normality.

Theorem 1. *T-consequence relation is paraconsistent, paracomplete, and paranormal.*

Proof. To prove the theorem it is sufficient do demonstrate only paranormality of T-consequence relation. Indeed, let $\Gamma = \emptyset$, $\Delta = \emptyset$, Φ be $P_1 \in Ps$, Ψ be $P_2 \in Ps$ such that $P_1 \neq P_2$. Then it is easy to check that interpreting P_1 as predicate $\top_A^V$ and P_2 as predicate $\bot_A^V$ we get that $P_1 \wedge \neg P_1 \not\models_T P_2 \vee \neg P_2$. □

In a similar way we can prove the following statement.

Lemma 8. *For T-consequence relation the following properties hold:*

- $\Gamma, \neg\Phi \models_T \Delta$ *and* $\Gamma \not\models_T \Phi, \Delta$ *for some* Γ, Δ, *and* Φ*;*
- $\Gamma, \Phi \models_T \Delta$ *and* $\Gamma \not\models_T \neg\Phi, \Delta$ *for some* Γ, Δ, *and* Φ*;*
- $\Gamma \models_T \neg\Phi, \Delta$ *and* $\Gamma, \Phi \not\models_T \Delta$ *for some* Γ, Δ, *and* Φ*;*
- $\Gamma \models_T \Phi, \Delta$ *and* $\Gamma, \neg\Phi \not\models_T \Delta$ *for some* Γ, Δ, *and* Φ*.*

This lemma states that rules of sequent calculi permitting moving (negated) formulas from one side of a sequent to its another side are not valid for T-consequence relations. Consequently, sequent calculi for $\models_T$ will be more complicated.

Still, such transformations are possible for a formula interpreted as total deterministic predicate (see Theorem 2(4)).

Lemma 9. *Let Φ be a formula, $\Gamma, \Gamma', \Delta, \Delta'$ be sets of formulas. Then*

(M) *if* $\Gamma \subseteq \Gamma'$ *and* $\Delta \subseteq \Delta'$, *then* $\Gamma \models_T \Delta \Rightarrow \Gamma' \models_T \Delta'$*;*

(C) $\Phi, \Gamma \models_T \Delta, \Phi$.

Proof of the lemma follows immediately from definitions.

Now we continue with those properties of T-consequence relation which induce sequent rules for the logic under consideration. Such properties are constructed upon semantic properties of compositions. To do this the following lemma is often used.

Theorem 2. *Let* Φ*,* Ψ*, and* Ξ *be formulas,* Γ *and* Δ *be sets of formulas,* J *be* L_ε^{UR}*-interpretation. Then*

(1) *if* $T(\Phi_J) = T(\Psi_J)$*, then*
$$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Psi, \Gamma\ {}_J{\models_T}\ \Delta \text{ and } \Gamma\ {}_J{\models_T}\ \Phi, \Delta \Leftrightarrow \Gamma\ {}_J{\models_T}\ \Psi, \Delta;$$

(2) *if* $T(\Phi_J) = T(\Psi_J) \cap T(\Xi_J)$*, then*
$$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Psi, \Xi, \Gamma\ {}_J{\models_T}\ \Delta,$$
$$\Gamma\ {}_J{\models_T}\ \Phi, \Delta \Leftrightarrow (\Gamma\ {}_J{\models_T}\ \Psi, \Delta \text{ and } \Gamma\ {}_J{\models_T}\ \Xi, \Delta);$$

(3) *if* $T(\Phi_J) = T(\Psi_J) \cup T(\Xi_J)$*, then*
$$\Gamma\ {}_J{\models_T}\ \Phi, \Delta \Leftrightarrow \Gamma\ {}_J{\models_T}\ \Psi, \Xi, \Delta,$$
$$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow (\Psi, \Gamma\ {}_J{\models_T}\ \Delta \text{ and } \Xi, \Gamma\ {}_J{\models_T}\ \Delta);$$

(4) *if* $T(\Phi_J) \cup F(\Phi_J) = {}^V\!A$ *and* $T(\Phi_J) \cap F(\Phi_J) = \emptyset$*, then*
$$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Gamma\ {}_J{\models_T}\ \neg\Phi, \Delta \text{ and} \neg\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Gamma\ {}_J{\models_T}\ \Phi, \Delta;$$
$$\Gamma \models_T \Delta \Leftrightarrow (\Phi, \Gamma \models_T \Delta \text{ and } \Gamma \models_T \Delta, \Phi);$$

(5) *if* $y \in fu(\Gamma, \Delta)$*, then* $\Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Gamma\ {}_J{\models_T}\ \Delta, \varepsilon y$*;*

(6) *if* $T(\Phi_J) =_{-y} T(\Psi_J)$ *for* $y \in fu(\Psi, \Gamma, \Delta)$*, then*
$$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow \Psi, \Gamma\ {}_J{\models_T}\ \Delta.$$

Proof. Property (1) is obvious. For (2) we have
$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow T(\Phi_J) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J) \Leftrightarrow$
$\Leftrightarrow T(\Psi_J) \cap T(\Xi_J) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J) \Leftrightarrow \Psi, \Xi, \Gamma\ {}_J{\models_T}\ \Delta$.

In the same way the second part of (2) and property (3) are proved. Let us consider (4). We have
$\Phi, \Gamma\ {}_J{\models_T}\ \Delta \Leftrightarrow T(\Phi_J) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J) \Leftrightarrow$

$\Leftrightarrow T^{\wedge}(\Gamma_J) \subseteq \overline{T(\Phi_J)} \cup T^{\vee}(\Delta_J) \Leftrightarrow T^{\wedge}(\Gamma_J) \subseteq T(\neg\Phi_J) \cup T^{\vee}(\Delta_J) \Leftrightarrow$
$\Leftrightarrow \Gamma\, _J{\models_T}\, \neg\Phi, \Delta$.

In the same way other properties of (4) are proved.

Let us consider (5). By Lemma 9(M) we have that $\Gamma\, _J{\models_T}\ \Delta \Rightarrow \Gamma\, _J{\models_T}\ \Delta, \varepsilon y$. We need to prove that $\Gamma\, _J{\models_T}\ \Delta, \varepsilon y \Rightarrow \Gamma\, _J{\models_T}\ \Delta$. It is equivalent to
$T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J) \cup T(\varepsilon y) \Leftrightarrow T^{\wedge}(\Gamma_J) \cap F(\varepsilon y) \subseteq T^{\vee}(\Delta_J)$.

Let $d \in T^{\wedge}(\Gamma_J) \cap F(\varepsilon y)$. Since y is unessential for Γ, it means that $d|_{-y} \in T^{\wedge}(\Gamma_J)$. From this follows that $d|_{-y} \in T^{\vee}(\Delta_J)$. Since y is unessential for Δ, it means that $d \in T^{\vee}(\Delta_J)$. Thus, $T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J)$ that proves the property under consideration.

Let us consider (6). We should prove that

$$T(\Phi_J) \cap T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J) \Leftrightarrow T(\Psi_J) \cap T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J).$$

Let $d \in T(\Psi_J) \cap T^{\wedge}(\Gamma_J)$. Since $T(\Phi_J) =_{-y} T(\Psi_J)$, there exists $d' \in T(\Phi_J)$ such that $d' =_{-y} d$. Since y is unessential for Γ, we have that $d' \in T^{\wedge}(\Gamma_J)$. Hence $d' \in T^{\vee}(\Delta_J)$. Again, y is also unessential for Δ therefore $d \in T^{\vee}(\Delta_J)$. This proves the direct implication.

Let us prove the inverse implication. First, we prove that $T(\Phi_J) \subseteq T(\Psi_J)$. Indeed, let $d \in T(\Phi_J)$. Since $T(\Phi_J) =_{-y} T(\Psi_J)$, there exists $d' \in T(\Psi_J)$ such that $d' =_{-y} d$. Since y is unessential for Ψ, $d \in T(\Psi_J)$. Thus, $T(\Phi) \subseteq T(\Psi)$.

From this follows that $\Psi, \Gamma\, _J{\models_T}\ \Delta \Rightarrow \Phi, \Gamma\, _J{\models_T}\ \Delta$.

This completes the proof of (6). □

Theorem 3. *The following properties hold for T-consequence relation.*

– *Properties related to propositional compositions:*

$\neg\neg_{\mathrm{L}}$) $\neg\neg\Phi, \Gamma \models_T \Delta \Leftrightarrow \Phi, \Gamma \models_T \Delta$.

$\neg\neg_{\mathrm{R}}$) $\Gamma \models_T \Delta, \neg\neg\Phi \Leftrightarrow \Gamma \models_T \Delta, \Phi$.

$\vee_{\mathrm{L}}$) $\Phi \vee \Psi, \Gamma \models_T \Delta \Leftrightarrow (\Phi, \Gamma \models_T \Delta$ *and* $\Psi, \Gamma \models_T \Delta)$.

$\neg\vee_{\mathrm{L}}$) $\neg(\Phi \vee \Psi), \Gamma \models_T \Delta \Leftrightarrow \neg\Phi, \neg\Psi, \Gamma \models_T \Delta$.

$\vee_{\mathrm{R}}$) $\Gamma \models_T \Delta, \Phi \vee \Psi \Leftrightarrow \Gamma \models_T \Delta, \Phi, \Psi$.

$\neg\vee_{\mathrm{R}}$) $\Gamma \models_T \Delta, \neg(\Phi \vee \Psi) \Leftrightarrow (\Gamma \models_T \Delta, \neg\Phi$ *and* $\Gamma \models_T \Delta, \neg\Psi)$.

– *Properties related to renomination compositions:*

$R\vee_{\mathrm{L}}$) $R_{\bar{x}}^{\bar{v}}(\Phi\vee\Psi),\Gamma\models_T\Delta\Leftrightarrow R_{\bar{x}}^{\bar{v}}(\Phi)\vee R_{\bar{x}}^{\bar{v}}(\Psi),\Gamma\models_T\Delta.$

$\neg R\vee_{\mathrm{L}}$) $\neg R_{\bar{x}}^{\bar{v}}(\Phi\vee\Psi),\Gamma\models_T\Delta\Leftrightarrow\neg(R_{\bar{x}}^{\bar{v}}(\Phi)\vee R_{\bar{x}}^{\bar{v}}(\Psi)),\Gamma\models_T\Delta.$

$R\vee_{\mathrm{R}}$) $\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(\Phi\vee\Psi)\Leftrightarrow\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(\Phi)\vee R_{\bar{x}}^{\bar{v}}(\Psi).$

$\neg R\vee_{\mathrm{R}}$) $\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(\Phi\vee\Psi)\Leftrightarrow\Gamma\models_T\Delta,\neg(R_{\bar{x}}^{\bar{v}}(\Phi)\vee R_{\bar{x}}^{\bar{v}}(\Psi)).$

$\mathrm{R_L}$) $R(\Phi),\Gamma\models_T\Delta\Leftrightarrow\Phi,\Gamma\models_T\Delta.$

$\mathrm{R_R}$) $\Gamma\models_T\Delta,R(\Phi)\Leftrightarrow\Phi,\Gamma\models_T\Delta,\Phi.$

$\mathrm{RI_L}$) $R_{z,\bar{x}}^{z,\bar{v}}(\Phi),\Gamma\models_T\Delta\Leftrightarrow R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta.$

$\neg\mathrm{RI_L}$) $\neg R_{z,\bar{x}}^{z,\bar{v}}(\Phi),\Gamma\models_T\Delta\Leftrightarrow\neg R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta.$

$\mathrm{RI_R}$) $\Gamma\models_T\Delta,R_{z,\bar{x}}^{z,\bar{v}}(\Phi)\Leftrightarrow\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(\Phi).$

$\neg\mathrm{RI_R}$) $\Gamma\models_T\Delta,\neg R_{z,\bar{x}}^{z,\bar{v}}(\Phi)\Leftrightarrow\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(\Phi).$

$\mathrm{RU_L}$) $R_{z,\bar{x}}^{y,\bar{v}}(\Phi),\Gamma\models_T\Delta\Leftrightarrow R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta,$ *if* $y\in fu(\Phi).$

$\neg\mathrm{RU_L}$) $\neg R_{z,\bar{x}}^{y,\bar{v}}(\Phi),\Gamma\models_T\Delta\Leftrightarrow\neg R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta,$ *if* $y\in fu(\Phi).$

$\mathrm{RU_R}$) $\Gamma\models_T\Delta,R_{z,\bar{x}}^{y,\bar{v}}(\Phi)\Leftrightarrow\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(\Phi),$ *if* $y\in fu(\Phi).$

$\neg\mathrm{RU_R}$) $\Gamma\models_T\Delta,\neg R_{z,\bar{x}}^{y,\bar{v}}(\Phi),\Leftrightarrow\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(\Phi),$ *if* $y\in fu(\Phi).$

$\mathrm{RR_L}$) $R_{\bar{x}}^{\bar{v}}(R_{\bar{y}}^{\bar{w}}(\Phi)),\Gamma\models_T\Delta\Leftrightarrow R_{\bar{x}}^{\bar{v}}\circ_{\bar{y}}^{\bar{w}}(\Phi),\Gamma\models_T\Delta.$

$\neg\mathrm{RR_L}$) $\neg R_{\bar{x}}^{\bar{v}}(R_{\bar{y}}^{\bar{w}}(\Phi)),\Gamma\models_T\Delta\Leftrightarrow\neg R_{\bar{x}}^{\bar{v}}\circ_{\bar{y}}^{\bar{w}}(\Phi),\Gamma\models_T\Delta.$

$\mathrm{RR_R}$) $\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(R_{\bar{y}}^{\bar{w}}(\Phi))\Leftrightarrow\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}\circ_{\bar{y}}^{\bar{w}}(\Phi).$

$\neg\mathrm{RR_R}$) $\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(R_{\bar{y}}^{\bar{w}}(\Phi))\Leftrightarrow\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}\circ_{\bar{y}}^{\bar{w}}(\Phi).$

$\mathrm{R\neg_L}$) $R_{\bar{x}}^{\bar{v}}(\neg\Phi)),\Gamma\models_T\Delta\Leftrightarrow\neg R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta.$

$\neg\mathrm{R\neg_L}$) $\neg R_{\bar{x}}^{\bar{v}}(\neg\Phi)),\Gamma\models_T\Delta\Leftrightarrow\neg\neg R_{\bar{x}}^{\bar{v}}(\Phi),\Gamma\models_T\Delta.$

$\mathrm{R\neg_R}$) $\Gamma\models_T\Delta,R_{\bar{x}}^{\bar{v}}(\neg\Phi))\Leftrightarrow\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(\Phi).$

$\neg\mathrm{R\neg_R}$) $\Gamma\models_T\Delta,\neg R_{\bar{x}}^{\bar{v}}(\neg\Phi))\Leftrightarrow\Gamma\models_T\Delta,\neg\neg R_{\bar{x}}^{\bar{v}}(\Phi).$

$\mathrm{R\exists R_L}$) $R_{\bar{v},y}^{\bar{u},x}(\exists x\Phi),\Gamma\models_T\Delta\Leftrightarrow R_{\bar{v}}^{\bar{u}}(\exists x\Phi),\Gamma\models_T\Delta.$

$\neg\mathrm{R\exists R_L}$) $\neg R_{\bar{v},y}^{\bar{u},x}(\exists x\Phi),\Gamma\models_T\Delta\Leftrightarrow\neg R_{\bar{v}}^{\bar{u}}(\exists x\Phi),\Gamma\models_T\Delta.$

$\mathrm{R\exists R_R}$) $\Gamma\models_T\Delta,R_{\bar{v},y}^{\bar{u},x}(\exists x\Phi)\Leftrightarrow\Gamma\models_T\Delta,R_{\bar{v}}^{\bar{u}}(\exists x\Phi).$

$\neg$R$\exists$R$_\mathrm{R}$) $\Gamma \models_T \Delta, \neg R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi) \Leftrightarrow \Gamma \models_T \Delta, \neg R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$.

R$\exists$s$_\mathrm{L}$) $R^{\bar{v}}_{\bar{x}}(\exists y\Phi), \Gamma \models_T \Delta \Leftrightarrow \exists y R^{\bar{v}}_{\bar{x}}(\Phi), \Gamma \models_T \Delta$, *if* $y \notin \{\bar{v}, \bar{x}\}$.

$\neg$R$\exists$s$_\mathrm{L}$) $\neg R^{\bar{v}}_{\bar{x}}(\exists y\Phi), \Gamma \models_T \Delta \Leftrightarrow \exists y \neg R^{\bar{v}}_{\bar{x}}(\Phi), \Gamma \models_T \Delta$, *if* $y \notin \{\bar{v}, \bar{x}\}$.

R$\exists$s$_\mathrm{R}$) $\Gamma \models_T \Delta, R^{\bar{v}}_{\bar{x}}(\exists y\Phi) \Leftrightarrow \Gamma \models_T \Delta, \exists y R^{\bar{v}}_{\bar{x}}(\Phi)$, *if* $y \notin \{\bar{v}, \bar{x}\}$.

$\neg$R$\exists$s$_\mathrm{R}$) $\Gamma \models_T \Delta, \neg R^{\bar{v}}_{\bar{x}}(\exists y\Phi) \Leftrightarrow \Gamma \models_T \Delta, \exists y \neg R^{\bar{v}}_{\bar{x}}(\Phi)$, *if* $y \notin \{\bar{v}, \bar{x}\}$.

R$\exists_\mathrm{L}$) $R^{\bar{v}}_{\bar{x}}(\exists y\Phi), \Gamma \models_T \Delta \Leftrightarrow \exists z R^{\bar{v}}_{\bar{x}} \circ^y_z (\Phi), \Gamma \models_T \Delta$,
if $z \in fu(R^{\bar{v}}_{\bar{x}}(\exists y\Phi))$.

$\neg$R$\exists_\mathrm{L}$) $\neg R^{\bar{v}}_{\bar{x}}(\exists y\Phi), \Gamma \models_T \Delta \Leftrightarrow \neg\exists z R^{\bar{v}}_{\bar{x}} \circ^y_z (\Phi), \Gamma \models_T \Delta$,
if $z \in fu(R^{\bar{v}}_{\bar{x}}(\exists y\Phi))$.

R$\exists_\mathrm{R}$) $\Gamma \models_T \Delta, R^{\bar{v}}_{\bar{x}}(\exists y\Phi) \Leftrightarrow \Gamma \models_T \Delta, \exists z R^{\bar{v}}_{\bar{x}} \circ^y_z (\Phi)$,
if $z \in fu(R^{\bar{v}}_{\bar{x}}(\exists y\Phi))$.

$\neg$R$\exists_\mathrm{R}$) $\Gamma \models_T \Delta, \neg R^{\bar{v}}_{\bar{x}}(\exists y\Phi) \Leftrightarrow \Gamma \models_T \Delta, \neg\exists z R^{\bar{v}}_{\bar{x}} \circ^y_z (\Phi)$,
if $z \in fu(R^{\bar{v}}_{\bar{x}}(\exists y\Phi))$.

– *Properties related to unassignment predicates:*

ε_LR) $\Gamma \models_T \Delta \Leftrightarrow \varepsilon y, \Gamma \models_T \Delta$ *and* $\Gamma \models_T \Delta, \varepsilon y$.

ε_R) $\Gamma \models_T \Delta \Leftrightarrow \Gamma \models_T \Delta, \varepsilon z$, *if* $z \in fu(\Gamma, \Delta)$.

– *Properties related to quantifiers:*

$\exists$e$_\mathrm{L}$) $\exists x\Phi, \Gamma \models_T \Delta \Leftrightarrow R^x_z(\Phi), \Gamma \models_T \Delta, \varepsilon z$, *if* $z \in fu(\Gamma, \Delta, \exists x\Phi)$.

$\neg\exists$e$_\mathrm{L}$) $\neg\exists x\Phi, \Gamma \models_T \Delta, \varepsilon y \Leftrightarrow \neg\exists x\Phi, \neg R^x_y(\Phi), \Gamma \models_T \Delta, \varepsilon y$.

$\exists$e$_\mathrm{R}$) $\Gamma \models_T \Delta, \exists x\Phi, \varepsilon y \Leftrightarrow \Gamma \models_T \Delta, \exists x\Phi, R^x_y(\Phi), \varepsilon y$.

$\neg\exists$e$_\mathrm{R}$) $\Gamma \models_T \Delta, \neg\exists x\Phi \Leftrightarrow \Gamma \models_T \Delta, \neg Rx_z(\Phi), \varepsilon z$,
if $z \in fu(\Gamma, \Delta, \exists x\Phi)$.

Proof. Proof of the formulated properties is based on semantic properties of compositions and properties of T-consequence relation. Consider, for instance, properties $\vee_\mathrm{L}$ and $\neg\vee_\mathrm{L}$. For $\vee_\mathrm{L}$ we have that $\Phi \vee \Psi, \Gamma\, {}_J{\models_T} \Delta \Leftrightarrow (T(\Phi_J) \cup T(\Psi_J)) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J) \Leftrightarrow (T(\Phi_J) \cap T^\wedge(\Gamma_J)) \cup (T(\Psi_J) \cap T^\wedge(\Gamma_J)) \subseteq T^\vee(\Delta_J) \Leftrightarrow (T(\Phi_J) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J)$ and $T(\Psi_J) \cap T^\wedge(\Gamma_J) \subseteq T^\vee(\Delta_J)) \Leftrightarrow (\Phi, \Gamma\, {}_J{\models_T} \Delta$ and $\Psi, \Gamma\, {}_J{\models_T} \Delta)$

for any interpretation J. Thus, $\Phi \vee \Psi, \Gamma \models_T \Delta \Leftrightarrow (\Phi, \Gamma \models_T \Delta$ and $\Psi, \Gamma \models_T \Delta)$.

For $\neg\vee_{\mathrm{L}}$ we have that $\neg(\Phi \vee \Psi), \Gamma\,_J\!\models_T \Delta \Leftrightarrow T(\neg(\Phi \vee \Psi)_J) \cap T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J) \Leftrightarrow F(\Phi_J) \cap F(\Psi_J) \cap T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J) \Leftrightarrow T(\neg\Phi_J) \cap T(\neg\Psi_J) \cap T^{\wedge}(\Gamma_J) \subseteq T^{\vee}(\Delta_J) \Leftrightarrow \neg\Phi, \neg\Psi, \Gamma\,_J\!\models_T \Delta$ for any interpretation J. Thus, $\neg(\Phi \vee \Psi), \Gamma \models_T \Delta \Leftrightarrow \neg\Phi, \neg\Psi, \Gamma \models_T \Delta$.

Properties related to renomination composition hold by Lemma 5.

Property $\varepsilon_{\mathrm{LR}}$ follows from Theorem 2(4); property ε_{R} follows from Theorem 2(5).

Properties related to quantifiers are consequences of Lemma 6, Lemma 7, and Theorem 2(6). Detailed proof is omitted here. $\square$

Properties presented in Theorem 3 induce sequent rules for calculus formalizing $\models_T$. For example, property $\exists\mathrm{e}_{\mathrm{L}}$ induces a rule

$$\frac{R_z^x(\Phi), \Gamma \to \Delta, \varepsilon z}{\exists x\Phi, \Gamma \to \Delta}, \ z \in fu(\Gamma, \Delta, \exists x\Phi).$$

Detailed construction of such a calculus will be presented in forthcoming papers.

6 T-consequence relation for the class of deterministic predicates

A class of deterministic quasiary predicates is an important subclass of the class of non-deterministic predicates.

Predicate $p \in PrR_A^V$ is called *deterministic (partial single-valued)* if $T(p) \cap F(p) = \emptyset$. The class of such predicates is denoted Pr_A^V.

Lemma 10. *Class Pr_A^V is a sub-algebra of algebra $AQR(V, A)$.*

The lemma is proved by direct checking that all compositions preserve the class of deterministic predicates. Defined algebra is denoted $AQ(V, A)$.

Such algebras form a semantic base for a logic of quasiary deterministic predicates L^Q. The extended logics L^U and L_ε^U are defined in the same way as for non-deterministic predicates. In this section a sign $\models_T$ denotes a T-consequence relation for the class of deterministic quasiary predicates.

Here we present only those properties of logics of deterministic predicates that differ from corresponding properties for non-deterministic predicates. In particular, a formula from the right side of T-consequence relation can be placed as a negated formula into the left side of the relation. Such property does not hold for the class of non-deterministic predicates.

Lemma 11. *The following properties hold for T-consequence relation for the logic of deterministic quasiary predicates:*

- $\Gamma \models_T \Delta, \Phi \Rightarrow \neg\Phi, \Gamma \models_T \Delta$;
- $\Gamma \models_T \Delta, \neg\Phi \Rightarrow \Phi, \Gamma \models_T \Delta$.

Proof. For any Γ, Δ, Φ, and interpretation J we have that
$\Gamma\ {}_J{\models_T}\ \Delta, \Phi \Leftrightarrow T^{\wedge}(\Gamma_J) \subseteq T(\Phi) \vee T^{\vee}(\Delta_J) \Leftrightarrow T^{\wedge}(\Gamma_J) \cap \overline{T(\Phi)} \subseteq T^{\vee}(\Delta_J)$.
Since $T(\neg\Phi) \subseteq \overline{T(\Phi)}$ for deterministic predicates, we have that

$$T^{\wedge}(\Gamma_J) \cap \overline{T(\Phi)} \subseteq T^{\vee}(\Delta_J) \Rightarrow T^{\wedge}(\Gamma_J) \cap T(\neg\Phi) \subseteq T^{\vee}(\Delta_J).$$

Thus, $\Gamma\ {}_J{\models_T}\ \Delta, \Phi \Rightarrow \neg\Phi, \Gamma\ {}_J{\models_T}\ \Delta$. Therefore

$$\Gamma \models_T \Delta, \Phi \Rightarrow \neg\Phi, \Gamma \models_T \Delta.$$

The second property is proved in the same manner. □

Properties concerning paraconsistency, paracompleteness, and paranormality also differ for the class of deterministic predicates.

Theorem 4. *T-consequence relation for the logic of deterministic quasiary predicates is consistent, paracomplete, and normal.*

Proof. For the class of deterministic predicates we have that $\Phi \wedge \neg\Phi, \Gamma\ {}_J{\models_T}\ \Psi, \Delta \Leftrightarrow (T(\Phi_J) \cap T(\neg\Phi_J)) \cap T^{\wedge}(\Gamma_J) \subseteq T(\Psi_J) \vee T^{\vee}(\Delta_J)$ for any Γ, Δ, Φ, Ψ, and interpretation J. This inclusion holds because for deterministic predicates $T(\Phi_J) \cap T(\neg\Phi_J) = \emptyset$. Thus, $\models_T$ is consistent; consequently, $\models_T$ is normal. To demonstrate paracompleteness of $\models_T$ we take an interpretation J such that $T(\Phi_J) \neq \emptyset$ and $\Psi_J = \perp_A^V$. Then $\Phi\ {}_J{\not\models_T}\ \Psi \vee \neg\Psi$. □

In general, comparing properties of T-consequence relation for classes of deterministic and non-deterministic predicates, we can say that the latter is poorer and the former is richer. In particular, if we consider F-consequence relation, which is dual to T-consequence relation, and a combined TF-consequence relation, then all three relations coincide for the class of non-deterministic predicates, but they are different for the class of deterministic predicates. Also, the irrefutability consequence relation is quite natural for the class of deterministic predicates while it collapses for the class of non-deterministic predicates.

7 Conclusion

In the paper we have investigated a special kind of program-oriented algebras and logics defined for classes of non-deterministic and deterministic quasiary predicates. We have considered the main semantic properties of T-consequence relations for such logics. We have presented properties related to propositional compositions, renomination, variable unassignment predicates, and quantifier compositions. These properties form a basis for construction of sequent calculi for logics of non-deterministic and deterministic quasiary predicates. We plan to present such calculi and prove their validity and completeness in forthcoming papers.

References

[1] *Handbook of Logic in Computer Science*, S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (eds.), in 5 volumes, Oxford Univ. Press, Oxford, 1993–2001.

[2] *Handbook of Philosophical Logic*, D.M. Gabbay, F. Guenthner (eds.), 2nd Edition, in 17 volumes, Springer, 2001–2011.

[3] M. Nikitchenko, S. Shkilniak. *Mathematical logic and theory of algorithms*, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2008, 528 p. (In Ukrainian)

[4] M. Nikitchenko, S. Shkilniak. *Applied Logic*, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2013, 278 p. (in Ukrainian).

[5] M. Nikitchenko, V. Tymofieiev. *Satisfiability in composition-nominative logics*, Central European Journal of Computer Science, vol. 2, no. 3, 2012, pp. 194–213.

[6] N. Nikitchenko. *A Composition Nominative Approach to Program Semantics*, Technical Report IT-TR 1998-020, Technical University of Denmark, 1998.

[7] M. Nikitchenko, S. Shkilniak. *Semantic properties of logics of quasiary predicates*, Workshop on Foundation of Informatics: Proceedings FOI-2015, August 24-29, 2015, Chisinau/Inst. of Mathematics and Computer Science, Acad. of Sciences of Moldova; ed.: S. Cojocaru, C. Gaindric, 2015, pp. 180–197.

[8] A. Avron, A. Zamansky. *Non-deterministic semantics for logical systems*, in *Handbook of Philosophical Logic*, D.M. Gabbay, F. Guenthner (eds.), 2nd ed., vol. 16, Springer Netherlands, 2011, pp. 227–304.

[9] E. Bencivenga. *Free Logics*, in *Handbook of Philosophical Logic*, D. Gabbay and F. Guenthner (eds.), vol. III: Alternatives to Classical Logic, Dordrecht: D. Reidel, 1986, pp. 373–426.

[10] A. Kryvolap, M. Nikitchenko, W. Schreiner. *Extending Floyd-Hoare logic for partial pre- and postconditions*, CCIS, 412, 2013, Springer, Heidelberg, pp. 355–378.

[11] J.-Y. Béziau. *What is paraconsistent logic?*, in D. Batens, C. Mortensen, G. Priest, J.P. Van Bendegem (Eds.), Frontiers of Paraconsistent Logic, Proceedings of the 1st World Congress on Paraconsistency, held in Ghent, Belgium, July 29–August 3, 1997, Research Studies Press, Baldock, UK, 2000, pp. 95–111.

Mykola Nikitchenko, Stepan Shkilniak,

Received September 20, 2015

Mykola Nikitchenko
Taras Shevchenko National University of Kyiv
01601, Kyiv, Volodymyrska st, 60
Phone: +38044 2590519
E–mail: `nikitchenko@unicyb.kiev.ua`

Stepan Shkilniak
Taras Shevchenko National University of Kyiv
01601, Kyiv, Volodymyrska st, 60
Phone: +38044 2590519
E–mail: `sssh@unicyb.kiev.ua`

BioMaxP : A Formal Approach for Cellular Ion Pumps

Bogdan Aman Gabriel Ciobanu

Abstract

We look at the living cells as complex systems of ion pumps working in parallel to ensure proper physiologic functionalities. To model such a system of pumps, we define a simple and elegant approach that allows working with multisets of ions, explicit interpretation of the transportation (from inside to outside, and from outside to inside) based on the number of existing ions, and a maximal parallel execution of the involved pumps.

1 Introduction

All living cells can be seen as complex systems of interacting components, having different concentrations of ions (e.g., Na^+, K^+ and Ca^{++}) across the cell membrane. Under resting conditions, Na^+ and Ca^{++} ions enter the cells and K^+ ions exit the cell because the concentration of K^+ is high inside the cell and low outside, while the opposite situation is found for Na^+ and Ca^{++}. A fundamental mechanism in most of the living cells is the Na^+/K^+-$ATPase$ that is essential for the maintenance of Na^+ and K^+ concentrations across the membrane by transporting Na^+ out of the cell and K^+ back into the cell. This pump is the first discovered ion transporter; for this discovery, the Danish chemist Jens Skou received the Nobel Prize in 1997.

In this paper we model the movement of ions and the conformational transformations of ion transporters (NaK ion pumps, Na and K ion channels) by using BioMaxP , a very simple but powerful approach. We use an operational semantics able to capture quantitative aspects (e.g., number of ions) and abstract conditions associated with evolution (e.g., the number of ions is between certain thresholds). The modelling aims to facilitate a better understanding of the living cell viewed as a complex system of parallel ion transporters.

The novelty of our approach is that we model systems composed of more than one pump as usually done (e.g., see [5]). Since the pumps non-deterministically choose which ions to transport, the complexity of such systems increases with the number of pumps. For further notions about NaK pump, the interested reader can consult [1].

2 Syntax and Semantics of BioMaxP

The prototyping language BioMaxP provides sufficient expressiveness to model in an elegant way the interaction in complex systems of parallel ion pumps. The cell is a complex system of parallel pumps trying to keep the equilibrium of ions inside the cell. In order to model these pumps, we enforce that their functioning takes place only if the number of various types of ions is between some accepted limits given by *min* and *max* values. Therefore the syntax and semantics emphasize the process of counting them, and the way the quantities of ions vary during evolution. The semantics of BioMaxP is provided by multiset labelled transitions in which multisets of actions are executed in parallel.

Syntax of BioMaxP The syntax of BioMaxP is given in Table 1, where the following are assumed:

- a set $Chan$ of ion transportation channels a, and a set Id of process identifiers (each $id \in Id$ has its arity m_{id});
- for each $id \in Id$ there is a unique process $id(u_1, \dots, u_{m_{id}} : T_1, \dots, T_{m_{id}}) \overset{def}{=} P_{id}$, where the distinct variables u_i are parameters, and the T_i are ions types;
- v is a tuple of expressions built from values, variables and allowed operations;
- T represent ions types.

Table 1. BioMaxP Syntax

Processes			
P, Q	$::=$	$a^{min}!(v : T)$ then P |	(sending)
		$a^{max}?(f(u : T))$ then P |	(receiving)
		$id(v)$ |	(recursion)
		$P \mid Q$	(parallel)

A constraint ^{min} associated with a sending action $a^{min}!(z : T)$ then P makes the channel a available for sending z units/ions of type T only if the total available quantity of ions of type t is greater than min. A constraint ^{max} associated to a receiving action $a^{max}?(x : T)$ then P along a channel a is activated only if the number of ions of the type T available is less than max. The function f of the receiving action can be either id (we often omit it), meaning that the received ions are to be transported, or add, meaning that the ions are received from some other process. The only variable binding constructor is $a^{max}?(u : T)$ then P; it binds the variable u within P. The free variables of a process P are denoted by $fv(P)$; for a process definition, is assumed that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, where u_i are the process parameters. Processes are defined up-to an alpha-conversion, and $\{v/u\}P$ denotes P in which all free occurrences of the variable u are replaced by v, eventually after alpha-converting P in order to avoid clashes. Processes are further constructed from the parallel composition $P \mid Q$. A system of parallel pumps is represented as a process with some initial values for the numbers of ions.

Remark 1 *In order to focus on the local interaction aspects of* BioMaxP *, we abstract from arithmetical operations, considering by default that the simple ones (comparing, addition, subtraction) are included in the language.*

Operational Semantics of BioMaxP The operational semantics rules of BioMaxP is presented in Table 2. The multiset labelled transitions of form $P \xrightarrow{\Lambda} P'$ use a multiset Λ to indicate the actions executed in parallel in one step. When the multiset Λ contains only one action λ, in order to simplify the notation, $P \xrightarrow{\{\lambda\}} P'$ is simply written as $P \xrightarrow{\lambda} P'$. We assume that in order to interact the processes can commute, namely $P \mid Q$ is the same process as $Q \mid P$.

Table 2. BioMaxP Operational Semantics

$$(\textsc{Com}) \quad \frac{v : T \quad and \quad min \leq |T| \leq max}{a^{min}!\langle v\rangle \ \mathsf{then}\ P \mid a^{max}?(f(u:T)) \ \mathsf{then}\ P' \xrightarrow{\{v/u\}} P \mid \{v/u\}P' \\ \text{and } |T| = |T| - v \text{ if } f = id \text{ or } |T| = |T| + v \text{ if } f = add}$$

$$(\textsc{Call}) \quad \frac{\{v/u\}P_{id} \xrightarrow{id} P'_{id}}{id(v) \xrightarrow{id} P'_{id}} \text{ where } id(v:T) \overset{def}{=} P_{id}$$

$$(\textsc{Par1}) \quad \frac{P_1 \xrightarrow{\Lambda_1} P'_1 \qquad P \not\rightarrow}{P_1 \mid P \xrightarrow{\Lambda_1} P'_1 \mid P} \qquad (\textsc{Par2}) \quad \frac{P_1 \xrightarrow{\Lambda_1} P'_1 \quad P_2 \xrightarrow{\Lambda_2} P'_2}{P_1 \mid P_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} P'_1 \mid P'_2}$$

In rule (Com), an output process $a^{min}!\langle v\rangle$ then P succeeds in sending a tuple of values v over channel a to process $a^{max}?(u:T)$ then P if v has the same type T as u and if the number of ions of type T is between min and max, namely $v : T$ and $min \leq |T| \leq max$. Both processes continue to execute, the first one as P and the second one as $\{v/u\}P'$. Once the ions are send away, $f = id$, the number of ions of type T becomes $T - |v|$, while if they are received, $f = add$, then the number of ions of type T becomes $T + |v|$. Rule (Call) describes the evolution of a recursion process. Rules (Par1) and (Par2) are used to compose larger processes from smaller ones by putting them in parallel, and considering the union of multisets of actions. In rule (Par2), $P \not\rightarrow$ denotes a process P that cannot evolve. It can be noticed that in rule (Par2) we use negative premises: an activity is performed based on the absence of actions. This is due to the fact that sequencing the evolution can only be defined using negative premises, as done for sequencing processes [6, 10].

Example 1 *The use of* BioMaxP *for specifying complex systems of pumps is illustrated by describing in an explicit way the molecular interactions and conformational transformations of a large system of ion transporters, namely Na^+K^+ ATPases and Na and K ion channels, that are concerned with the movement of sodium-potassium ions in and out of a cell whenever certain thresholds are verified. The system we consider is formed from n_1 NaK pumps, n_2 Na channels and n_3 K channels. Each pump i is modelled by three processes: one that models the interaction of the pump with the environment, one modelling the interaction with the cell and another one that models the transport of ions through the membrane. The molecular components are processes modelled as the ends of a channel (one end for input, and another for output), while the molecular interaction coincides with communication on channels.*

The initial system of pumps is described in BioMaxP *by:*

$$\begin{gathered}
Cell(NaEnv, KEnv, NaCell, KCell, AtP, ADP, P) = \\
\mid NaKPumpEnv(0) \mid NaKPumpCell(0) \mid NaKPump(0) \\
\cdots \mid NaKPumpEnv(n_1\text{-}1) \mid NaKPumpCell(n_1\text{-}1) \mid NaKPump(n_1\text{-}1) \\
\mid NaPumpEnv(n_1) \mid NaPumpCell(n_1) \mid NaPump(n_1) \\
\cdots \mid NaPumpEnv(n_1+n_2-1) \mid NaPumpCell(n_1+n_2-1) \mid NaPump(n_1+n_2-1) \\
\mid KPumpEnv(n_1+n_2) \mid KPumpCell(n_1+n_2) \mid KPump(n_1+n_2) \\
\cdots \mid KPumpEnv(n_1+n_2+n_3\text{-}1) \mid KPumpCell(n_1+n_2+n_3\text{-}1) \mid \\
KPump(n_1+n_2+n_3\text{-}1) \\
CreateATP \mid ConsumeADP
\end{gathered}$$

We present in detail some of the above processes. The others are written in a similar manner.

- $Cell(NaCell, KCell, AtP, ADP, NaEnv, KEnv, P)$ *is the system in which several quantities of ions are initialized.*
- *Each NaK-ATPase is described by three processes:*
  - $*$ $NaKPumpEnv(id) = site2[id]^{160}?(add(y_{na} : NaEnv))$
    then $site2[id]^{2}!\langle 2K\rangle$
    then $p[id]^{6}?(add(y_p : P))$
    then $NaKPumpEnv(id)$

The environment site of the pump contains the channel $site2[id]$ used for receiving three ions of Na^+ and also for sending two ions of K^+, and also the channel $p[id]$ for receiving the produced P molecules. The sending and receiving operations modify also the number of ions present in the system of pumps: e.g., when sending two K^+ ions, an operation of the form $KEnv = Kenv - 2$ is performed, while receiving the $y_{na} : NaEnv$ ions an operation of the form $NaEnv = NaEnv + 3$ is performed due to the add function that is used to add the amount of received ions to the corresponding multiset.

$$\begin{aligned} * \; NaKPumpCell(id) = {} & site1[id]^{(12,1)}!\langle(3Na, ATP)\rangle \\ & \text{then } adp[id]^{6}?(add(x_{adp} : ADP)) \\ & \text{then } site1[id]^{150}?(add(x_k : KCell)) \\ & \text{then } NaKPumpCell(id) \end{aligned}$$

The cell site of the pump contains the channel $site1[id]$ used for sending three ions of Na^+ and one ATP and also for receiving two ions of K^+, and also the channel $adp[id]$ for receiving the produced ADP molecules.

$$\begin{aligned} * \; NaKPump(id) = {} & site1[id]^{(28,9)}?((x_{na} : NaCell, x_{atp} : ATP)) \\ & \text{then } adp[id]^{0}!\langle ADP\rangle \\ & \text{then } site2[id]^{100}!\langle 2Na\rangle \\ & \text{then } site2[id]^{6}?(y_k : KEnv) \\ & \text{then } p[id]^{0}!\langle P\rangle \text{ then } site1[id]^{110}!\langle 2K\rangle \\ & \text{then } NaKPump(id) \end{aligned}$$

This process describes the evolution of the pump, namely the transport of Na and K ions between the environment and the cell.

3 Timed Automata

Due to their simplicity, timed automata, extended with integer variables, structured data types, user defined functions, and channel synchronization, have been used by several tools (e.g., Uppaal) for the simulation and verification of timed automata [2]. In what follows we consider a particular case of timed automata, namely we ignore the time aspects as they are not relevant to our approach and will refer to timed automata as automata.

Syntax Assume a finite set of integer variables $\mathcal{C}$ ranged over by x, y, ... standing for data, and a finite alphabet Σ ranged over by a, b, ... standing for actions. A constraint is a conjunctive formula of constraints of the form $x \sim m$ for $x \in \mathcal{C}$, $\sim \in \{\leq, <, ==, >, \geq\}$, and $m \in \mathbb{N}$. The set of constraints, ranged over by g, is denoted by $\mathcal{B}(\mathcal{C})$.

Definition 1 *An* **automaton** $\mathcal{A}$ *is a tuple* $\langle N, n_0, E\rangle$*, where*

- N *is a finite set of nodes;*
- n_0 *is the initial node;*
- $E \subseteq N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times \mathbb{N}^{\mathcal{C}} \times N$ *is the set of edges.*

$n \xrightarrow{g,a,r} n'$ *is a shorthand notation for* $\langle n, g, a, r, n'\rangle \in E$*.* r *denotes fresh assignments to variables after the transition is performed.*

Networks of Automata A network of automata is the parallel composition $\mathcal{A}_1 \mid \ldots \mid \mathcal{A}_n$ of a set of automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ combined into a single system. Synchronous communication inside the network is by handshake synchronization of input and output actions. In this case, the action alphabet Σ consists of $a?$ symbols (for input actions), $a!$ symbols (for output actions), and τ symbols (for internal actions). A detailed example is found in [9]. A network can perform action transitions (following an enabled edge). An action transition is enabled if all guards on the corresponding edges are satisfied.

Let u, v, ... denote assignments mapping $\mathcal{C}$ to naturals $\mathbb{N}$. $g \models u$ means that the values u satisfy the guard g. Let n_i stand for the ith element of a node vector n, and $n[n'_i/n_i]$ for the vector n with n_i being substituted with n'_i. A network state is a pair $\langle n, u\rangle$, where n denotes a vector of current nodes of the network (one for each automaton), and u is an assignment storing the current values of all network integer variables.

Definition 2 *The operational semantics of an automaton is a transition system where states are pairs $\langle n, u\rangle$ and transitions are defined by the rules:*

- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i], u'\rangle$ *if* $n_i \xrightarrow{g,\tau,r} n'_i$, $g \models u$ *and* $u' = r[u]$;
- $\langle n, u\rangle \xrightarrow{\tau} \langle n[n'_i/n_i][n'_j/n_j], u'\rangle$ *if there exist* $i \neq j$ *such that*
  1. $n_i \xrightarrow{g_i, a?, r_i} n'_i$, $n_j \xrightarrow{g_j, a!, r_j} n'_j$, $g_i \wedge g_j \models u$,
  2. $u' = r_i[r_j[u]]$.

4 Relating BioMaxP to Automata

In order to use existing tools such as Uppaal for the verification of complex systems of parallel pumps, we establish a relationship between BioMaxP and automata.

Building an automaton for each process: Given a process P without the parallel operator at the top level, we associate to it an automaton $\mathcal{A} = \langle N, n_0, E\rangle$, where $n_0 = l_0$, $N = \{l_0\}$, $E = \emptyset$. The initial values of the BioMaxP system composed of P are set as the initial values of the automaton $\mathcal{A}$. The nodes of the associated automata are labelled using a fresh label l, and an index such that the nodes are uniquely labelled in this automaton (we start with the index 0, and increment it when necessary). The components N and E are updated depending on the structure of process P:

- for $P = a^{min}!\langle v\rangle$ then P_1 we have
  - $N = N \cup \{l_{i+1}\}$ where $i = max\{j \mid l_j \in N\}$;
    - The added node l_{i+1} indicates the execution of the process P, leading to P_1.
  - $E = E \cup \{n, min \leq |T|, a!, , l_{i+1}\}$;
    - If $i > 0$ it means that the automaton already contains some edges, and the process P was launched from the then branch of a process P'. Since the translation is made depending on the structure of the processes, it means that the action leading to P is already modelled

in the automaton. If $P' = b^{min'}!\langle w\rangle$ then P or $P' = b^{max'}?(u : T')$ then P, then the action of P' is modelled by an edge with the last component l_k, and thus $n = l_k$.
    * Otherwise, $n = l_0$.

  The edge encodes the then branch leading to process P_1. Channel a is an urgent channel (communication takes place as soon as possible).

- for $P = a^{max}?(f(u : T))$ then P_1 we have
  - $N = N \cup \{l_{i+1}\}$ where $i = max\{j \mid l_j \in N\}$;
  - $E = \begin{cases} E \cup \{l_i, |T| \leq max, a!, |T| = |T| - |u|, l_{i+1}\}, \text{if } f = id; \\ E \cup \{l_i, |T| \leq max, a!, |T| = |T| + |u|, l_{i+1}\}, \text{if } f = add. \end{cases}$

  A similar reasoning as for the previous case. Depending on function f, ions are removed or added from the number representing the existing ions of type T.
- for $P = P_1 \mid \ldots \mid P_k$, $k > 1$, and P_j does not contain operator $\mid$ at top level, then
  - $N = N \cup \{l_{i+1}\}$ where $i = max\{j \mid l_j \in N\}$;
    * If P contains some indexed nodes l (namely $l_0, \ldots, l_i$), then add l_{i+1} to N.
  - $E = E \cup \{n, , a!, \{x = 0\}, l_{i+1}\}$;
    * If $i > 0$, using a similar argument as for the communication actions, it holds that $n = l_k$. We use a new channel labelled a as a broadcast channel, in order to start at the same time all the parallel processes from P.
    * Otherwise, $n = l_0$.

  The new edge leads to process P_1. For each of the other processes P_j, $j > 1$, a new automaton $\mathcal{A}_j = \langle N_j, n_{j0}, E_j, I_j\rangle$ is build, where:
    * $n_{j0} = l_0$; $N_j = \{l_0, l_1\}$; $E_j = \{l_0, , a?, \{x = 0\}, l_1\}$; $I_j(l_0) = \emptyset$.

  The automaton is constructed recursively using the definition of P_j.

Building an automaton for each process leads to the next result about the equivalence between a BioMaxP process P and its corresponding automaton $\mathcal{A}_P$ in state $\langle n_P, u_P\rangle$ (i.e., $(\mathcal{A}_P, \langle n_P, u_P\rangle)$). Their transition systems differ not only in transitions, but also in states; thus, we adapt the notion of bisimilarity:

Definition 3 *A symmetric relation $\sim$ between* BioMaxP *processes and their corresponding automata is a bisimulation if whenever $(N, (\mathcal{A}_N, \langle n_N, u_N\rangle)) \in\sim$ if $P \xrightarrow{\lambda} P'$, then $\langle n_P, u_P\rangle \xrightarrow{\tau} \langle n_{P'}, u_{P'}\rangle$ and $(P', (\mathcal{A}_{P'}, \langle n_{P'}, u_{P'}\rangle)) \in\sim$ for some P'.*

After defining bisimulation, we can state the following result.

Theorem 1 *Given a* BioMaxP *process P, there exists an automata $\mathcal{A}_P$ with a bisimilar behaviour. Formally, $P \sim \mathcal{A}_P$.*

Proof. [Sketch] The construction of the automaton simulating a given BioMaxP process is presented above. A bisimilar behaviour is given by the fact that a communication rule is matched by a synchronization between the edges obtained by translations. □

Thus, the size of an automata $\mathcal{A}_P$ is polynomial with respect to the size of a BioMaxP process P, and the state spaces have the same number of states.

Reachability Analysis. Qualitative properties abstract away from any quantitative information like time aspects or energy costs of targeted biological systems. One of the most useful question to ask about an automaton is the reachability of a given set of final states. Such final states may be used to characterize safety properties of a system.

Definition 4 *For an automata with initial state $\langle n_0, u_0\rangle$, $\langle n, u\rangle$ is reachable if and only if $\langle n_0, u_0\rangle \overset{\tau}{\rightarrow}^* \langle n, u\rangle$. More generally, given a constraint $\phi \in \mathcal{B}(\mathcal{C})$ if $\langle n, u\rangle$ is reachable for some u satisfying ϕ, then a state $\langle n, \phi\rangle$ is reachable.*

The reachability problem is decidable [4]. The reachability problem can be also defined for BioMaxP networks.

Definition 5 *Starting from a* BioMaxP *process P_0, a process P_1 is reachable if and only if $P_0 \overset{\lambda}{\rightarrow}^* P_1$.*

The following result is a consequence of Theorem 1.

Corollary 1 *For a* BioMaxP *process, the reachability problem is decidable.*

5 Conclusion

Previously, we provided a formal description of the sodium-potassium ion transport across cell membranes in terms of the π-calculus [8]. In [7], the transfer mechanisms were described step by step, and a software tool called Mobility Workbench [11] was used to verify some properties of the described system formed of only one pump. Inspired by the functioning of this pump, we introduced and studied a ratio-based type system using thresholds in a bio-inspired framework [3]. The aim was to avoid errors in the definition of the formal models used to mimic the evolution of some biologic processes.

In this paper we try to unify and extend our previous attempts to model the movement of ions in the sodium-potassium-pump by using BioMaxP , a simple and elegant approach able to capture the quantitative aspects (e.g., number of ions) and abstract conditions associated with evolution (e.g., the number of ions is between certain thresholds). This approach facilitates a better understanding of the processes happening in a cell viewed as a complex system of ion pumps working in parallel. The novelty is that we are able to model systems consisting of more than just a NaK pump by adding different amounts of other types of ion pumps.

Acknowledgements. The work was supported by a grant of the Romanian National Authority for Scientific Research, project number PN-II-ID-PCE-2011-3-0919.

References

[1] B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, P. Walter. *Molecular Biology of the Cell*, 6th edition, Garland Science, New York (2014).

[2] R. Alur, D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science* **126**, 183–235 (1994).

[3] B. Aman, G. Ciobanu. Behavioural Types Inspired by Cellular Thresholds. *Lecture Notes in Computer Science* **8368**, 1–15 (2014).

[4] J. Bengtsson, W. Yi. Timed Automata: Semantics, Algorithms and Tools. *Lecture Notes in Computer Science* **3098**, 87–124 (2004).

[5] D. Besozzi, G. Ciobanu. A P System Description of the Sodium-Potassium Pump. *Lecture Notes in Computer Science* **3365**, 210–223 (2005).

[6] B. Bloom, S. Istrail, A.R. Meyer. Bisimulation Can't Be Traced: Preliminary Report. *In 15th ACM Symposium on Principles of Programming Languages*, 229–239, 1988.

[7] G. Ciobanu. Software Verification of the Biomolecular Systems. in *Modelling in Molecular Biology*, Natural Computing Series, Springer, 40–59 (2004).

[8] G. Ciobanu, V. Ciubotariu, B. Tanasă. A π-calculus Model of the Na Pump, *Genome Informatics* **13**, 469–472 (2002).

[9] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-time Systems. *Information and Computation* **111**, 192–224 (1994).

[10] F. Moller. *Axioms for Concurrency.* PhD Thesis, Department of Computer Science, University of Edinburgh, 1989.

[11] B. Victor, F. Moller. The Mobility Workbench - A Tool for the π-Calculus. *Lecture Notes in Computer Science* **818**, 428–440 (1994).

Bogdan Aman, Gabriel Ciobanu

Received July 21, 2015

Romanian Academy, Institute of Computer Science
Blvd. Carol I no.11, 700506 Iaşi, Romania
E–mail: `baman@iit.tuiasi.ro, gabriel@info.uaic.ro`

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Structuring of Specification Modules (extended)[*]

Răzvan Diaconescu

Abstract

This paper has two goals. One goal is to provide a brief introduction to the concept of modularisation in the context of formal specifications. The other goal is to survey some recent developments in this area, including parameter instantiation with sharing and module systems for behavioural specifications.

1 Composition of Specification Modules

1.1 Formal methods in software engineering

The field of formal methods for software and hardware systems is without alternative in safety-critical or security areas where one cannot take the risk of failure. Moreover formal methods are crucial in ensuring a smooth development of large software systems as well as their evolution and maintainability. In general one may distinguish between two classes of formal methods:

1. model checking; and
2. formal specification.

While the former has gained certain prominence, being especially succesfull in discovering errors in the system development process, only the latter may fully guarantee correctness.

[*] This work has been supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439.

1.2 Formal specification

In simple terms, formal specification activity can be described at two levels:

– At *specification* level, the system is specified axiomatically in a formal logical system (not necessarily a conventional one).
– At *verification* level, the properties of the system are derived as theorems, getting a fully formal proof.

Perhaps the most prominent classes of formal specifications are those that are based upon algebraic principles (in a wide sense of the term). While traditionally *algebraic specification* [3] is based upon a general form of algebra (such as the equational logic of many-sorted algebra), modern algebraic specification employs a wide variety of logical systems of higher levels of sophistication. Modern algebraic specification systems include CASL [2], CafeOBJ [20], Maude [10], Specware [29], etc. Heterogeneous environments [20, 32] constitute a recent integrating trend in logic-based formal specification that provides a very flexible approach when choosing the appropriate language and formalism. The theoretical infrastructure developed over many decades of research in formal specification has been exported also to other areas of computing science such as ontologies (e.g. [30]) or declarative programming.

In some cases there is a rather strong integration between the specification and the verification levels (like in OBJ, CafeOBJ, Maude, etc.). These are systems that in some sense are most faithful to the original algebraic specification culture that has equational logic at its core. The algebraic features of those systems smoothen up significantly the verification process by integrating techniques with good computational properties such as rewriting. In a way such specification languages, also called *executable* languages, may function quite well as very high level (declarative) programming languages. However equational logic poses some limitations in the specification power. In other cases (such as CASL, Specware, etc.) there is not a direct integration between the specification and the verification levels, allowing for more sophisticated logics meaning significant gains in specification power. In these cases the formal verification methodologies may involve external automatic provers (SPASS [41], etc.) and/or proof assistants (Coq, Isabelle, etc.). The difference between these two lies at the level of human involvement, while in the former situations this is minimal, in the case of the proof assistants the formal proofs are human guided.

An important distinctive feature of formal/algebraic specifications is its rock solid mathematical foundations based upon the principle of specification languages being rigorously based upon underlying formal logical systems such that each language constructs reflect rigorously as a mathematical concept in the underlying logic. Modern specification theory is based upon Goguen and Burstall's theory of *institutions* [24, 14] which is a general *category theoretic* [31] approach to logic. This makes the general theory independent of the actual choice of a logical system, which means great uniformity and applicability to a wide variety of specification languages.

1.3 Modular specifications

Modularisation is the only way to cope with the complexity of formal specifications of large software systems. It greatly enhances readability and reusability of specification code. Moreover structuring techniques also reduce the complexity of the formal verification process.

The work on specification modules has a long history that starts with the specification language Clear developed by Goguen and Burstall [9]. That laid general founding principles for module systems of formal specifications that have been realised by a multitude of subsequent languages and systems, notably including programming languages such as ADA or ML.

The main structuring constructs include

– Module imports (various kinds);
– Module sum/aggregation;
– Renamings;
– Information hiding;
– Parameterised (generic) modules and parameters instantiation by *views*;
– Complex module expressions;

The current modularisation theory and methodologies include many developments that have been fuelled by the actual practice in formal specification and by the design of new modern languages and systems supporting this activity. Moreover the modularisation technology has been exported from the area of formal specifications to other computer science areas, e.g. ontologies (the OMG standard *OntoIOp*).

Two quite major ideas have shaped the current approaches to specification modules.

1.3.1 Institution theory

The first idea is to abstract away from the details of the logic underlying the actual specification language. The module composition techniques are largely independent of the logical formalism employed by the respective specification language; in fact this important observation constituted the main idea behind the language Clear and has triggered the conception of institution theory [24] as an abstract framework for modularisation. This very general mathematical study of formal logical systems, with emphasis on semantics, is based upon a mathematical definition for the informal notion of logical system, called *institution*. This definition accommodates not only well established logical systems but also very unconventional ones and moreover it has served and it may serve as a template for defining new ones. Institution theory approaches logic from a relativistic, non-substantialist perspective, quite different from the common reading of logic. This is not opposed to the established logic tradition, since it rather includes it from a higher abstraction level. However the real difference is made at the level of methodology, top-down (in the case of institution theory) versus bottom-up (in the case of conventional logic tradition). Institution theory has had a strong impact in computer science and logic for over more than three decades (see [16]), and it continues to attract an ever growing interest in institution theory by computer scientists and (more recently) logicians.

Let us very briefly recall here the main concept of institution theory.

Definition 1 (Institutions) *An* institution *is a tuple* $(\mathit{Sign}, \mathit{Sen}, \mathit{Mod}, (\models_\Sigma)_{\Sigma\in|\mathit{Sign}|})$ *that consists of*

– *a category Sign whose objects are called* signatures*,*

– *a functor* $\mathit{Sen}\colon \mathit{Sign} \to \mathbb{S}\mathrm{et}$ *(to the category of sets) giving for each signature a set whose elements are called* sentences *over that signature,*

– *a (contravariant) functor* $\mathit{Mod}\colon (\mathit{Sign})^{op} \to \mathbb{CAT}$ *(to the 'category' of categories), giving for each signature* Σ *a category whose objects are called* Σ-models*, and whose arrows are called* Σ-(model) homomorphisms*, and*

– *a relation* $\models_\Sigma \subseteq |\mathit{Mod}(\Sigma)| \times \mathit{Sen}(\Sigma)$ *for each* $\Sigma \in |\mathit{Sign}|$*, called the* satisfaction relation*,*

such that for each morphism $\varphi\colon \Sigma \to \Sigma' \in \mathit{Sign}$*, the* Satisfaction Condition

$$M' \models_{\Sigma'} \mathit{Sen}(\varphi)(\rho) \text{ if and only if } \mathit{Mod}(\varphi)(M') \models_\Sigma \rho \tag{1}$$

holds for each $M' \in |\mathit{Mod}(\Sigma')|$ *and* $\rho \in \mathit{Sen}(\Sigma)$*.*

The literature (e.g. [14, 39]) shows myriads of logical systems from computing or from mathematical logic captured as institutions. In fact, an informal thesis underlying institution theory is that any 'logic' may be captured by the above definition. While this should be taken with a grain of salt, it certainly applies to any logical system based on satisfaction between sentences and models of any kind.

1.3.2 Core specification building operators

A second important idea envisaged in many works (e.g. [38, 4, 39], etc.) is to have a core set of specification building operators, with a clearly defined semantics, that can be used to define composition operators in actual module systems. The most prominent such set of specification building operators has been introduced in [38]; we recall them below in the more modern form of [8].

Given any institution $(\mathit{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ with designated classes of signature morphisms $\mathcal{T}$ and $\mathcal{D}$ the class of the $(\mathcal{T}, \mathcal{D})$-*structured specifications* [8] is the least class such that

– it contains all finite *presentations*, i.e. pairs (Σ, E) with Σ signature and E finite set of Σ-sentences; we also define
  – $\Phi(\Sigma, E) = \Sigma$ and
  – $\mathit{Mod}(\Sigma, E) = \{M \in |\mathit{Mod}(\Sigma)| \mid M \models E\}$,

– if SP_1 and SP_2 are structured specifications such that $\Phi(\mathrm{SP}_1) = \Phi(\mathrm{SP}_2)$, then $\mathrm{SP}_1 \cup \mathrm{SP}_2$ is also a structured specification and we define
  – $\Phi(\mathrm{SP}_1 \cup \mathrm{SP}_2) = \Phi(\mathrm{SP}_i)$ and
  – $|\mathit{Mod}(\mathrm{SP}_1 \cup \mathrm{SP}_2)| = |\mathit{Mod}(\mathrm{SP}_1) \cap \mathit{Mod}(\mathrm{SP}_2)|$,

– if SP is a structured specification and $(\varphi\colon \Phi(\mathrm{SP}) \to \Sigma') \in \mathcal{T}$, then $\mathrm{SP} \star \varphi$ is structured specification and
  – $\Phi(\mathrm{SP} \star \varphi) = \Sigma'$ and
  – $|\mathit{Mod}(\mathrm{SP} \star \varphi)| = \{M' \mid \mathit{Mod}(\varphi)(M') \in \mathit{Mod}(\mathrm{SP})\}$, and

– if SP' is a structured specification and $(\varphi\colon \Sigma \to \Phi(\mathrm{SP}')) \in \mathcal{D}$, then $\varphi \Box \mathrm{SP}'$ is structured specification and
  – $\Phi(\varphi \Box \mathrm{SP}') = \Sigma$ and
  – $|\mathit{Mod}(\varphi \Box \mathrm{SP}')| = \{M'\restriction_\varphi \mid M' \in \mathit{Mod}(\mathrm{SP})\}$.

Given a structured specification SP as above $\Phi(\mathrm{SP})$ is its *signature* $\Phi(\mathrm{SP})$ and $\mathit{Mod}(\mathrm{SP})$ is its *class of models.* The signature and the class of models of a specification SP represents its semantics.

When the actual specification language provides tight semantics capabilities, then an initial or final semantics operator is also typically included. However occasionally (see [18]) these specification building operators do not suffice, therefore other operators have also to be considered.

1.3.3 Inclusion systems

At the end of this section we would like to briefly present a category theoretic device that plays an important technical role in the theory of specification modules.

Inclusion systems were introduced in [23] with the aim of supporting an abstract general study of structuring specifications and programming modules that is independent of any underlying logic. While computer science is usually a heavy consumer of mathematics, inclusion systems can be seen as opposite, i.e. a rare contribution of computer science to pure mathematics. Inclusion systems have been used in a series of general module algebra studies such as [23, 27, 14], and also in institutional model theory studies [14, 36, 13, 1, 14]. Inclusion systems capture categorically the concept of set-theoretic inclusion in a way reminiscent of the manner in which the rather notorious concept of factorisation system [7] captures categorically the set-theoretic injections; however, in many applications the former are more convenient than the latter. The definition below can be found in the recent literature on inclusion systems (see, e.g. [14]), and differs slightly from the original one of [23].

Definition 2 (Inclusion system) *An* inclusion system *for a category $\mathbb{C}$ is a structure $\langle \mathcal{I}, \mathcal{E} \rangle$ such that $\mathcal{I}$ and $\mathcal{E}$ are broad subcategories of $\mathbb{C}$ satisfying the following two properties:*

- *$\mathcal{I}$ is a partial order (with the ordering relation denoted by $\subseteq$), and*
- *every arrow f in $\mathbb{C}$ can be factored uniquely as $f = e_f ; i_f$, with $e_f \in \mathcal{E}$ and $i_f \in \mathcal{I}$.*

In [12] it was shown that the category $\mathcal{I}$ of abstract inclusions determines the category $\mathcal{E}$ of abstract surjections. In this sense, [12] gives an explicit equivalent definition of inclusion systems that relies only on the category $\mathcal{I}$ of abstract inclusions.

The standard example of inclusion system is that from $\mathbb{S}\mathrm{et}$, with set theoretic inclusions in the role of abstract inclusions and surjective functions in the role of abstract surjections. The literature contains many other examples of inclusion systems for the categories of signatures and for the categories of models of various institutions from logic or specification theory.

2 Recent Developments

In this section we survey some recent developments in the theory of modular formal specifications. Three topics are considered: general theory, genericity and sharing, and module systems for behavioural specifications.

2.1 General theory

Pivotal developments in the institutional theory of structured specifications (such as [38, 23] etc.) provide an abstract treatment of the underlying logic as an abstract institution but consider fixed sets of concrete structuring operators. To overcome this limitation, but also to achieve unification between the Goguen-Burstall [24, 23] and the Sannella-Tarlecki [38, 39] approaches to the semantics of structured specifications, the recent paper [15] introduces a second level of institution-independence by treating the class of the structured specifications together with their model theory as an abstract institution. The relationship between the level of structured specifications and the level of the underlying logic is axiomatized by a special kind of institution morphism. The following definition recalls from [15] the main concept of this theory.

Definition 3 (Structured institution) *An institution* $\mathcal{I}' = (\mathit{Sign}', \mathit{Sen}', \mathit{Mod}', \models')$ *is said to be* structured *over a base institution* $\mathcal{I} = (\mathit{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ *through a* structuring functor Φ*, or* $(\Phi, \mathcal{I})$-structured*, when*

- Φ *is a functor* $\mathit{Sign}' \to \mathit{Sign}$*,*
- *for every* $\mathcal{I}'$*-signature* Σ'*,* $\mathit{Sen}(\Phi(\Sigma')) = \mathit{Sen}'(\Sigma')$*, and similarly, for every* $\mathcal{I}'$*-signature morphism* φ'*,* $\mathit{Sen}(\Phi(\varphi')) = \mathit{Sen}'(\varphi')$*,*

– *for every $\mathcal{I}'$-signature Σ', $Mod'(\Sigma')$ is a full subcategory of $Mod(\Phi(\Sigma'))$ such that the diagram below commutes for every $\mathcal{I}'$-signature morphism $\varphi\colon \Sigma'_1 \to \Sigma'_2$, and*

$$\begin{tikzcd}
Mod'(\Sigma'_1) \arrow[r, "\subseteq"] & Mod(\Phi(\Sigma'_1)) \\
Mod'(\Sigma'_2) \arrow[r, "\subseteq"'] \arrow[u, "Mod'(\varphi)"] & Mod(\Phi(\Sigma'_2)) \arrow[u, "Mod(\Phi(\varphi))"']
\end{tikzcd}$$

– *for every $\mathcal{I}'$-signature Σ', Σ'-model M' and Σ'-sentence ρ,*

$$M' \models'_{\Sigma'} \rho \quad \textit{if and only if} \quad M' \models_{\Phi(\Sigma')} \rho.$$

Several examples of structured institutions are presented in some detail in [15]. One of the most important ones is that of the structured specifications of Sect. 1.3.2. In that example, *Sign'* is the category of the structured specifications, *Mod'*(SP) is *Mod*(SP) as defined in Sect. 1.3.2 and the satisfaction relation of $\mathcal{I}'$ is inherited from $\mathcal{I}$ in the obvious way.

Recent papers such as [15, 40, 11, 19, 17] develop elements of modularisation theory in this abstract framework.

2.2 Genericity and Sharing

Generic or parameterised modules represent one of the most important module composition techniques because they can be (re)used in various ways by appropriate instantiations of their parameters. In simple terms, a parameterised module (denoted SP(P)) can be regarded as a module import $P \to$ SP, with P being its *parameter* and SP being its *body.* Instantiation of parameterised modules is performed through interpretations of their parameters. In the specification literature they are usually called *views* and they are syntactic mappings $v\colon P \to \mathrm{SP}_1$ that satisfy two conditions:

1. they match consistently the signature of the parameter P to the signature of the value SP_1 (which is also a specification module);

technically this amounts to the fact that v is a *signature morphism*; and

2. any implementation[1] of SP_1 has to be a an implementation of the parameter P via the interpretation of the syntactic entities given by v.

Given a parameterised module $\mathrm{SP}(P)$ and a view $v\colon P \to \mathrm{SP}_1$ the *instance* $\mathrm{SP}(P \Leftarrow v)$ is commonly defined by the using the *pushout* technique (cf. [9, 39], etc.) from category theory [31], which informally can be explained as a 'user-defined' form of union. This is a two-steps process. First consider the underlying signatures of the specifications and compute a pushout in the category of signatures:

$$\begin{tikzcd}
\Phi(P) \arrow[r, "\subseteq"] \arrow[d, "\Phi(v)"'] & \Phi(\mathrm{SP}) \arrow[d, "v'"] \\
\Phi(\mathrm{SP}_1) \arrow[r, "i'"'] & \Sigma'
\end{tikzcd}$$

At the second stage we built the actual instance

$$\begin{tikzcd}
P \arrow[r] \arrow[d, "v"'] & \mathrm{SP} \arrow[d, "v'"] \\
\mathrm{SP}_1 \arrow[r, "i'"'] & \mathrm{SP}(P \Leftarrow v)
\end{tikzcd} \tag{2}$$

by defining

$$\mathrm{SP}(P \Leftarrow v) = (\mathrm{SP}_1 \star i') \cup (SP \star v').$$

This requires that, minimally, there are unions and translations available as building operators.

While this is the traditional approach to parameter instantiation which is widely employed by actual specification formalisms (e.g. [9, 2, 20], etc.) and also by programming languages such as ML and

[1]Mathematically speaking, 'implementations' are treated as models or interpretations in the underlying logic.

other languages influenced by it, it still raises several technical issues of important methodological significance and that can be summarised as follows:

1. Since the pushout construction is unique only up to isomorphic renaming, actual implementations of module systems involving parameters provide ad-hoc constructions for the results of parameter instantiations. But how can we ensure in general that there exists an instantiation such that $\mathrm{SP}_1 \to \mathrm{SP}(P \Leftarrow v)$ behaves like an import, and moreover would this be uniquely defined?
2. In order to avoid technical complications the actual specification systems commonly dismiss the sharing between the body (SP) and the instance (SP_1), a situation that in practice constitutes a real restriction, as for example it may lead to duplication of the same data.
3. In the case of multiple parameters, for example $\mathrm{SP}(P_1, P_2)$, we can instantiate them sequentially (first $(\mathrm{SP}(P_1 \Leftarrow v_1)(P_2)$ and next $\mathrm{SP}(P_1 \Leftarrow v_1)(P_2 \Leftarrow v_2)$, or the other way around) or in parallel by regarding SP as parameterised by a single parameter, $\mathrm{SP}(P_1 + P_2 \Leftarrow v_1 + v_2)$. Are the two sequential instantiations and the parallel one equivalent methods in the sense of yielding isomorphic results?

The main technicalities involved in the answer to these questions include both a reshape of the definition of parameter instantiation and a general property of the signatures of the specification language formulated. In brief the traditional pushout square (2) has to be redefined as

$$\begin{tikzcd}
P \cup \mathrm{SP}_1 \arrow[r, "\subseteq"] \arrow[d, "v \cup 1_{\mathrm{SP}_1}"'] & \mathrm{SP} \cup \mathrm{SP}_1 \arrow[d, "v'"] \\
\mathrm{SP}_1 \arrow[r, "i"'] & \mathrm{SP}(P \Leftarrow v)
\end{tikzcd} \tag{3}$$

(where $v \cup 1_{\mathrm{SP}_1}$ implies that v is identity on the part shared between P and SP_1)

and the signatures have to enjoy the following general property: for any signature morphism $\varphi\colon \Sigma \to \Sigma_1$ and any inclusion $\Sigma \subseteq \Sigma'$

$$\begin{tikzcd}
 & \Sigma\cap\Omega \arrow[r, "\subseteq"] \arrow[d, "\subseteq"] \arrow[dd, bend right=60, "\subseteq"'] & \Sigma'\cap\Omega \arrow[d, "\subseteq"'] \arrow[dd, bend left=60, "\subseteq"] & \\
\textcircled{1} & \Sigma \arrow[r, "\subseteq"] \arrow[d, "\varphi"'] \arrow[dr, phantom, "\textcircled{3}"] & \Sigma' \arrow[d, "\varphi'"] & \textcircled{2} \\
 & \Sigma_1 \arrow[r, "\subseteq"'] & \Sigma'_1 &
\end{tikzcd}$$

there exists a pushout $\textcircled{3}$ such that for any signature Ω if $\textcircled{1}$ holds then $\textcircled{2}$ holds too.

In the works [18, 15, 40] the concepts of inclusion ($\subseteq$), union ($\cup$), intersection ($\cap$), disjointness, etc. are treated abstractly within the framework of 'inclusion systems' as presented in Sect 1.3.3. In this way the solution proposed is general and can be applied to almost all existing specification formalisms (with the notable exception of behavioural specifications discussed below). The required property above holds naturally for all specification formalisms of interest, often in a stronger form than actually required (see [18, 40]). The generality of this solution implies also that it can be employed also by new specification formalisms to be defined in the future.

2.3 Module systems for behavioural specifications

Modern algebraic specification theory and practice has extended the traditional many-sorted algebra-based specification to several new paradigms. One of the most promising is behavioural specification, which originates from the work of Horst Reichel [33, 34] and can be found in the literature under names such as hidden algebra [25, 26], observational logic [5, 28], coherent hidden algebra [21] and hidden logic [35]. Behavioural specification characterises how objects (and systems) *behave*, not how they are implemented. This new form of abstraction can be very powerful for the specification and verification of software systems since it naturally embeds other useful paradigms such as concurrency, object-orientation, constraints, nondeterminism, etc. (see [26] for details). In the tradition of algebraic specification, the behavioural abstraction is achieved by using specification with hidden sorts and a behavioural concept of satisfaction based on the idea of indistinguishability of states that are observationally the same, which also generalizes process algebra and transition systems (see [26]). An important effort has been undertaken to develop languages and systems supporting the behavioural extension of conventional or less conventional algebraic specification techniques; these include CafeOBJ [20, 22], CIRC [37] and BOBJ [35]. In other situations, behavioural specification, although not directly realized at the level of the language definition, is employed as a mere methodological device [6]. In all cases there is the unavoidable need of a structuring mechanism for behavioural specifications.

However the modularisation of behavioural specifications poses specific challenges with respect to the standard modularisation techniques. Some important properties that in general are taken for granted do not hold in the case of behavioural specifications, for example the basic operation of union (aggregation) of behavioural specifications is only partial. The root cause of these problems lies in the 'encapsulation condition' on the signature morphisms, which prohibits new behavioural operations on old hidden sorts (i.e. that correspond to sorts of the source signature). Therefore the basic compositionality properties of behavioural specifications hold in a partial rather than total algebra style form. For example (see [19]) the associativity of union of behavioural specifications

$$(\mathrm{SP} \cup \mathrm{SP}') \cup \mathrm{SP}'' = \mathrm{SP} \cup (\mathrm{SP}' \cup \mathrm{SP}'')$$

means that either both members are defined and are equal semantically or else that neither of them is defined.

In the case of the instantiation of parameterised behavioural specifications the pushout square (2) is replaced with the following pushout square:

$$\begin{tikzcd}
P \cup (\mathrm{SP} \cap \mathrm{SP}_1) \arrow[r, "\subseteq"] \arrow[d, "v \cup \mathrm{id}"'] & \mathrm{SP} \arrow[d] \\
\mathrm{SP}_1 \arrow[r] & \mathrm{SP}_1'
\end{tikzcd} \tag{4}$$

when $P \cup \mathrm{SP}_1$ is defined. When $\mathrm{SP} \cup \mathrm{SP}_1$ is defined too, (4) can be replaced by the technically more convenient (3). However in this case the condition underlying (3) is stronger than that of (4) (note that according to [19] while unions of behavioural specifications are partial, intersections are total).

3 Conclusions and Future Research

We have presented some important elements of modularisation theory in formal specification, including important mathematical tools (institutions, inclusion systems) and methods (pushout-style parameterisation). In the second part of the paper we have discussed recent developments such as two-layered institution-independence, upgrade to pushout-style, multiple parameters, and behavioural specification. The latter three topics have been developed in close collaboration with Ionuţ Ţuţu.

At this moment there is ongoing work on parameterisation with sharing for behavioural specifications, on general views for parameter instantiation, etc.

References

[1] Marc Aiguier and Fabrice Barbier. An institution-independent proof of the Beth definability theorem. *Studia Logica*, 85(3):333–359, 2007.

[2] Edigio Astesiano, Michel Bidoit, Hélène Kirchner, Berndt Krieg-Brückner, Peter Mosses, Don Sannella, and Andrzej Tarlecki.

CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[3] Jan Bergstra, Jan Heering, and Paul Klint. *Algebraic Specification.* Association for Computing Machinery, 1989.

[4] Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

[5] Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioural and abstractor specifications. *Sci. Comput. Program.*, 25(2-3):149–186, 1995.

[6] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Observational interpretation of CASL specifications. *Mathematical Structures in Computer Science*, 18(2):325–371, 2008.

[7] Francis Borceux. *Handbook of Categorical Algebra.* Cambridge University Press, 1994.

[8] Tomasz Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286(2):197–245, 2002.

[9] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *1979 Copenhagen Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1980.

[10] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science.* Springer, 2007.

[11] Ionuţ Ţuţu. Comorphisms for structured institutions. *Information Processing Letters*, 113(894–900), 2013.

[12] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 7(2):195–206, 1997.

[13] Răzvan Diaconescu. Elementary diagrams in institutions. *Journal of Logic and Computation*, 14(5):651–674, 2004.

[14] Răzvan Diaconescu. *Institution-independent Model Theory.* Birkhäuser, 2008.

[15] Răzvan Diaconescu. An axiomatic approach to structuring specifications. *Theoretical Computer Science*, 433:20–42, 2012.

[16] Răzvan Diaconescu. Three decades of institution theory. In Jean-Yves Béziau, editor, *Universal Logic: an Anthology*, pages 309–322. Springer Basel, 2012.

[17] Răzvan Diaconescu. On the existence of translations of structured specifications. *Information Processing Letters*, 115(1):15–22, 2015.

[18] Răzvan Diaconescu and Ionuţ Ţuţu. On the algebra of structured specifications. *Theoretical Computer Science*, 412(28):3145–3174, 2011.

[19] Răzvan Diaconescu and Ionuţ Ţuţu. Foundations for structuring behavioural specifications. *Journal of Logical and Algebraic Methods in Programming*, 83(3–4):319–338, 2014.

[20] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[21] Răzvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *Universal Computer Science*, 6(1):74–96, 2000. First version appeared as JAIST Technical Report IS-RR-98-0017F, June 1998.

[22] Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285:289–318, 2002.

[23] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.

[24] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[25] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume

785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.

[26] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.

[27] Joseph Goguen and Grigore Roşu. Composing hidden information modules over inclusive institutions. In Olaf Owe, Stein Krogdahl, and Tom Lyche, editors, *From Object-Orientation to Formal Methods*, volume 2635 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 2004.

[28] Rolf Hennicker and Michel Bidoit. Observational logic. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology*, number 1584 in LNCS, pages 263–277. Springer, 1999. Proc. AMAST'99.

[29] Kestrel Institute. *Specware system and documentation*, 2003. `www.specware.org`.

[30] Oliver Kutz, Till Mossakowski, and Dominik Lücke. Carnap, Goguen, and the hyperontologies - logical pluralism and heterogeneous structuring in ontology design. *Logica Universalis*, 4(2):255–333, 2010.

[31] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, second edition, 1998.

[32] T. Mossakowski, C. Maeder, and K. Lütich. The heterogeneous tool set. In *Lecture Notes in Computer Science*, volume 4424, pages 519–522. 2007.

[33] Horst Reichel. Behavioural equivalence – a unifying concept for initial and final specifications. In *Proceedings, Third Hungarian Computer Science Conference*. Akademiai Kiado, 1981. Budapest.

[34] Horst Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras*. Clarendon, 1987.

[35] Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

[36] Grigore Roşu. Axiomatisability in inclusive equational logic. *Mathematical Structures in Computer Science*, 12(5):541–563, 2002.

[37] Grigore Roşu and Dorel Lucanu. Circular coinduction: A proof theoretical foundation. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science*, volume 5728 of *Lecture Notes in Computer Science*, pages 127–144, 2009.

[38] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Control*, 76:165–210, 1988.

[39] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specifications and Formal Software Development.* Springer, 2012.

[40] Ionuţ Ţuţu. Parameterisation for abstract structured specifications. *Theoretical Computer Science*, 517:102–142, 2014.

[41] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. Spass version 2.0. In *Proceedings of the 18th International Conference on Automated Deduction*, CADE-18, pages 275–279, London, UK, 2002. Springer-Verlag.

Răzvan Diaconescu

Received September 11, 2015

Simion Stoilow Institute of Mathematics of Romanian Academy
Romania
E–mail: `Razvan.Diaconescu@imar.ro`

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Fundamental theorems of extensional untyped λ-calculus revisited

Alexandre Lyaletsky

Abstract

This paper presents new proofs of three following fundamental theorems of the untyped extensional λ-calculus: the η-Postponement theorem, the $\beta\eta$-Normal form theorem, and the Normalization theorem for $\beta\eta$-reduction. These proofs do not involve any special extensions of the standard language of λ-terms but nevertheless are shorter and much more comprehensive than their known analogues.

Keywords: extensional untyped λ-calculus, $\beta\eta$-reduction, postponement of η-reduction, η-Postponement theorem, $\beta\eta$-Normal form theorem, Normalization theorem for $\beta\eta$-reduction.

1 Introduction

The untyped version of the λ-calculus is considered.

Its variables are denoted by symbols x, y and z, λ-terms by t, p, q, u and w, redeces by $\triangle$ (of cause, indices are sometimes used). All the other denotations used in the paper are completely standard or otherwise will be introduced separately.

Throughout the paper the variable convention is assumed to be satisfied; hence the conditions $x \notin t$ and $x \notin FV(t)$ say the same.

Recall some basic facts concerning η- and $\beta\eta$-reduction. By definition, the notion η of η-reduction is $\{ < \lambda x.wx, w > \mid x \notin w \}$, the notion $\beta\eta$ of $\beta\eta$-reduction is $\beta \cup \eta$. The notions of η- and $\beta\eta$-reduction induce, in the usual way, the dictionaries of their derivative notions (such as η- and $\beta\eta$-redeces, the relations of one-step and multi-step η-

and $\beta\eta$-reduction, η- and $\beta\eta$-reduction sequences, etc.). Remark that the notion of η-reduction is strongly normalizing since the contraction of an η-redex in any λ-term decreases its length.

The extensional untyped λ-calculus studies properties of the notion of $\beta\eta$-reduction as well as of its derivative relations, especially $\twoheadrightarrow_{\beta\eta}$ (multi-step $\beta\eta$-reduction) and $=_{\beta\eta}$ ($\beta\eta$-convertibility). Along with the Church-Rosser theorem for $\beta\eta$-reduction (and not taking the results on λ-representability into account), the most important results in the extensional λ-calculus are: 1) the η-Postponement theorem, 2) the $\beta\eta$-Normal form theorem, 3) the Normalization theorem for $\beta\eta$-reduction.

In the paper, new proofs of the theorems 1) – 3) are constructed. These proofs do not involve any special extensions of the standard language of λ-terms but nevertheless are shorter and much more comprehensive than their original or known analogues. (For example, the original proof of the theorem 3) by J.W.Klop takes over 20 pages and is technically very complicated.) The new proofs are arranged in the following logical order: 1) $\Rightarrow$ 2) $\Rightarrow$ 3).

2 Postponable binary relations

Our proof of the η-Postponement theorem exploits some general properties which are more convenient to be observed and studied in the general set-theoretic situation.

Definition 1. Given a set A and binary relations Q and R on A, then R is said to be *postponable* after Q if the following diagram holds:

$$\begin{tikzcd}
& \cdot \arrow[dl, dashed, "Q"'] \arrow[dr, "R"] & \\
\cdot \arrow[dr, dashed, "R"'] & & \cdot \arrow[dl, "Q"] \\
& \cdot &
\end{tikzcd}$$

(Here and in the sequel, the language of diagrams of binary relations is used. In the general case, such a diagram is a configuration on the plane consisting of points some of which may be labelled by elements of a fixed set A, and arrows between points, each obligatorily labelled by a binary relation on A. Each arrow can be of two sorts: usual or dotted. If a diagram contains a usual arrow from a to b and labelled by R, then the latter expresses that $a\,R\,b$; if a point has no label, then it is considered to be bounded by a universal quantifier (restricted by A). At that, precisely those arrows are dotted that lead to or start with the elements, the existence of which is being claimed (together with the conditions imposed by the labels). Thus, each diagram (containing at least one dotted arrow) determines a certain implicative statement. For example, the diagram from the last definition means the following:

$$\forall a,b,c\in A\left[\,a\,R\,b\ \ \&\ \ b\,Q\,c\ \ \Rightarrow\ \ \exists\,d\in A\,(a\,Q\,d\ \ \&\ \ d\,R\,c)\,\right],$$

i.e. that $R\circ Q\subseteq Q\circ R$, where $\circ$ denotes the usual composition of binary relations.)

Note that if binary relations $Q=f$ and $R=g$ are functions, then g is postponable after f if and only if $g\circ f=f\circ g$, that is f and g commute with each other. Therefore, in this case f is postponable after g as well. However in the general case the latter is not valid, i.e. the notion of postponability is not symmetric.

By R^+ and R^* denote, resp., the transitive and reflexive-transitive closures of a binary relation R.

It can easily be proved that if R is postponable after Q, then R^* is postponable after Q^* as well. Containing this statement as a particular case, the following result can be viewed as its natural generalization.

Postponement Lemma. *Let Q and R be binary relations on a set A such that for any triple of elements of A, at least one of the following diagrams holds:*

$$\begin{tikzcd}
& \cdot \arrow[dl, dashed, "Q"'] \arrow[dr, "R"] & \\
\cdot \arrow[dr, dashed, "R^*"'] & & \cdot \arrow[dl, "Q"] \\
& \cdot &
\end{tikzcd}
\qquad \textit{or} \qquad
\begin{tikzcd}
\cdot \arrow[dr, "R"] \arrow[dd, dashed, "Q^+"'] & \\
& \cdot \arrow[dl, "Q"] \\
\cdot &
\end{tikzcd}$$

Then R^* is postponable after Q^* and $(Q\cup R)^*=Q^*\circ R^*$.

Proof. Since $R^* \circ Q^* \subseteq (Q \cup R)^*$, for proving the postponability, it is sufficient to show that $(Q \cup R)^* \subseteq Q^* \circ R^*$, which also substantiates the second statement (the inclusion opposite to the latter holds trivially). Let a and c be any elements of A with $a\,(Q \cup R)^* c$. Obviously, it can be assumed that $a \neq c$. Then there is, for some elements $b_1, \ldots, b_{n-1} \in A$, a valid sequence of the form

$$b_0\, S_0\, b_1\, S_1\, b_2\, S_2 \ldots S_{n-2}\, b_{n-1}\, S_{n-1}\, b_n\,, \tag{1}$$

where $b_0 = a$, $b_n = c$, and $S_i \in \{Q, R\}$ for every $i \in \{0, n-1\}$.

Consider its leftmost "two-step" segment of the form $b_k R\, b_{k+1} Q\, b_{k+2}$ (if there is no such a segment, there is nothing to prove). If, for simplicity, the "Q-prefix" of (1) is empty, then (1) has the following form:

$$b_0 R\, b_1 R\, b_2 R \ldots R\, b_{k-1} R\, \underline{b_k\, R\, b_{k+1} Q\, b_{k+2}}\, S_{k+2}\, b_{k+3}\, S_{k+3} \ldots S_{n-1} b_n\,. \tag{2}$$

Applying one of the diagrams from the conditions of the lemma to the underlined segment, one of the following sequences will be obtained:

$$b_0 R \ldots R\, b_{k-1} R\, \underline{b_k\, Q\, b'_{k+1} R\, b'_{k+1,\,1} R \ldots R\, b'_{k+1,\,m} R\, b_{k+2}}\, S_{k+2} \ldots S_{n-1}\, b_n$$

(in the case of the left diagram), where m is a natural number, or

$$b_0 R \ldots R\, b_{k-1} R\, \underline{b_k\, Q\, b'_{k+1,\,1} Q \ldots Q\, b'_{k+1,\,m} Q\, b_{k+2}}\, S_{k+2} \ldots S_{n-1}\, b_n$$

(in the case of the right diagram), where m is a positive natural number.

Comparing the obtained sequences with the previous one, notice that in the both cases, the position of the leftmost occurrence of Q is one item to the left than that was in (2). Therefore, the proof can be completed by induction, applied to the set of sequences of the form (1) that is considered to be lexicographically ordered in accordance with the positions of all occurrences of Q in a sequence under consideration when reading it from left to right. $\square$

Remark. The Postponement lemma states less than that was proved. Actually, the above given proof determines an algorithm of reconstructing each sequence (1) to a form $b_0\, Q \ldots Q\, d\, R \ldots R\, b_n$ which will be referred to as *postponing* R after Q. (Of cause, this algorithm makes sense provided the conditions of the lemma are satisfied.)

3 Postponement of η-reduction

Note that when considering diagrams over the set of λ-terms some arrows of which are labelled by one-step reductions, it is often convenient to introduce additional labels for (some of) these arrows for indicating the contracted redex occurrences. (Examples are given below.)

Given a one-step $\beta\eta$-reduction sequence $\sigma\colon t_1 \xrightarrow{\triangle}_{\beta\eta} t_2$. If $\triangle_1$ is such a $\beta\eta$-redex occurrence in t_1 that has exactly one residual in t_2 (w.r.t. σ), then the latter will be denoted by $\overrightarrow{\triangle_1}$. If $\triangle_2$ is a $\beta\eta$-redex occurrence in t_2, then it can be trivially verified that there can be at most one $\beta\eta$-redex occurrence in t_1 for which $\triangle_2$ is a residual of (or belongs to the set of residuals of); in this case, it is denoted by $\overleftarrow{\triangle_2}$, i.e. $\overleftarrow{\triangle_2}$ is a "coresidual" of $\triangle_2$ w.r.t. σ.

η-Postponement Theorem. [1; 2] *Every finite $\beta\eta$-reduction sequence $\sigma\colon t_1 \twoheadrightarrow_{\beta\eta} t_2$ can be reconstructed into a sequence of the form $\sigma'\colon t_1 \twoheadrightarrow_\beta u \twoheadrightarrow_\eta t_2$, for some λ-term u.*

Proof. Let us verify the diagrams from the conditions of the Postponement lemma (with $Q = \longrightarrow_\beta$ and $R = \longrightarrow_\eta$). For a given two-step $\beta\eta$-reduction sequence of the form $t_1 \xrightarrow{\triangle_\eta}_\eta t_2 \xrightarrow{\triangle_\beta}_\beta t_3$, note that the coresidual $\overleftarrow{\triangle_\beta}$ always exists and is always a β-redex occurrence in t_1. Let $\triangle_\eta \equiv \lambda x.wx$, $\triangle_\beta \equiv (\lambda y.p')q'$ and $\overleftarrow{\triangle_\beta} \equiv (\lambda z.p)q$. Consider all the possible cases of mutual locations of the redeces $\triangle_\eta$ and $\overleftarrow{\triangle_\beta}$ in t_1:

1. $\triangle_\eta \cap \overleftarrow{\triangle_\beta} = \varnothing$		
2. $\triangle_\eta \supset \overleftarrow{\triangle_\beta}$		
3. $\triangle_\eta \subset \overleftarrow{\triangle_\beta}$	3.1. $\triangle_\eta \equiv \lambda z.p \equiv \lambda x.(\lambda y.p')x$	(hence $z \equiv x$ and $q \equiv q'$)
	3.2. $\triangle_\eta \subseteq p$	(hence $z \equiv y$ and $q \equiv q'$)
	3.3. $\triangle_\eta \subseteq q$	(hence $z \equiv y$ and $p \equiv p'$)

The cases 1, 2 and 3.2 are trivial: one only needs to reverse the order of contractions (first, to contract $\overleftarrow{\triangle_\beta}$ in t_1 and then, to contract the residual $\overrightarrow{\triangle_\eta}$ of $\triangle_\eta$ in the resulting term), which all lead to a diagram of the form:

$$\begin{tikzcd}
 & t_1 \arrow[dl, dashed, "\overleftarrow{\triangle}_\beta"', "\beta" near end] \arrow[dr, "\triangle_\eta", "\eta"' near end] & \\
\cdot \arrow[dr, dashed, "\overrightarrow{\triangle}_\eta"', "\eta" near end] & & t_2 \arrow[dl, "\triangle_\beta", "\beta"' near end] \\
 & t_3 &
\end{tikzcd}$$

Case 3.1:

$$\begin{tikzcd}
 & \ldots(\lambda x.(\lambda y.p')x)q'\ldots \arrow[dl, dashed, "(\lambda x.(\lambda y.p')x)q'"', "\beta" near end] \arrow[dr, "\lambda x.(\lambda y.p')x", "\eta"' near end] & \\
\ldots(\lambda y.p')q'\ldots \arrow[dr, dashed, "(\lambda y.p')q'"', "\beta" near end] & & \ldots(\lambda y.p')q'\ldots \arrow[dl, "(\lambda y.p')q'", "\beta"' near end] \\
 & \ldots p'[y := q']\ldots &
\end{tikzcd}$$

Case 3.3. First, consider the subcase when $q \equiv \triangle_\eta \equiv \lambda x.wx$:

$$\begin{tikzcd}
 & \ldots(\lambda y.p')(\lambda x.wx)\ldots \arrow[dl, dashed, "(\lambda y.p')(\lambda x.wx)"', "\beta" near end] \arrow[dr, "\lambda x.wx", "\eta"' near end] & \\
\ldots p'[y := \lambda x.wx]\ldots \arrow[dr, dashed, "\eta" near end] & & \ldots(\lambda y.p')w\ldots \arrow[dl, "(\lambda y.p')w", "\beta"' near end] \\
 & \ldots p'[y := w]\ldots &
\end{tikzcd}$$

Finally, in the general conditions of the case 3.3, one has $q \equiv \ldots\lambda x.wx\ldots$, which leads to a diagram similar to the latter. $\square$

4 $\beta\eta$-Normal forms

Note that for a λ-term of the form $(\lambda x.wx)q$, where $x \notin w$, the both contractions $(\lambda x.wx)q \xrightarrow{(\lambda x.wx)q}_{\beta} wq$ and $(\lambda x.wx)q \xrightarrow{\lambda x.wx}_{\eta} wq$ lead to the same result wq.

Definition 2. An η-redex occurrence $\triangle_\eta \equiv \lambda x.wx$ in a λ-term t is called *β-replaceable*, if $\triangle_\eta$ is a re-part of some β-redex occurrence in t, i.e. there is a term q such that $(\lambda x.wx)q$ is a β-redex occurrence in t.

Lemma 1. *Given a finite η-reduction sequence $\sigma\colon t \twoheadrightarrow_\eta t'$ in which neither of the contracted η-redeces is β-replaceable. If t' is a β-normal form, then so is t.*

Proof. Obviously, it is sufficient to prove the lemma for the case of a one-step η-reduction sequence $\sigma\colon t \xrightarrow{\triangle_\eta}_\eta t'$. If t is not a β-normal form, then it contains some β-redex occurrence $(\lambda x.p)q$ and hence $t \equiv \ldots(\lambda x.p)q\ldots$. Since $\triangle_\eta$ is not β-replaceable, it follows that $\triangle_\eta \not\equiv \lambda x.p$. Then, evidently, $(\lambda x.p)q$ has a nonempty residual in t' which is a β-redex occurrence. Thus, t' is not a β-normal form. $\square$

$\beta\eta$-Normal Form Theorem. [1; 3] *An arbitrary λ-term t has a $\beta\eta$-normal form $\Leftrightarrow$ t has a β-normal form.*

Proof. The sufficiency is obvious, since if t has a β-normal form t', then any η-redex contraction in t' does not create new β-redeces and decreases the length of t'.

Let us prove the necessity. Suppose a λ-term t has a $\beta\eta$-normal form t'. By the Church-Rosser theorem for $\beta\eta$-reduction ([1; 2]), then there is a $\beta\eta$-reduction sequence $\sigma\colon t \twoheadrightarrow_{\beta\eta} t'$. Moreover, it can be assumed without loss of generality that neither of the η-redeces being contracted in σ is β-replaceable (since any such an η-redex can be replaced in σ with the corresponding β-redex). Furthermore, from the analysis of the diagrams from the proof of the η-Postponement theorem it can be concluded that when postponing η-reduction in σ, no contracted β-replaceable η-redeces can emerge. Therefore, there exists a $\beta\eta$-reduction sequence of the form $\sigma'\colon t \twoheadrightarrow_\beta u \twoheadrightarrow_\eta t'$, for some λ-term u, that does not contract neither of the β-replaceable η-redeces. Lemma 1 finally implies that u is the needed β-normal form of t. $\square$

5 Normalization theorem for $\beta\eta$-reduction

Recall that for a given notion R of reduction and λ-term t, an R-leftmost redex is any such an R-redex occurrence in t, the position (in t) of the first symbol of which cannot be strictly to the right of the position of the first symbol of any other R-redex occurrence. Therefore, this notion is not deterministic in the general case, i.e. for some notion R of reduction, a term t may have two or more distinct leftmost R-redex occurrences. By this reason, the latter notion is sometimes strengthened to the notion of the R-leftmost-outermost redex occurrence which is always unique in every λ-term (if any).

As to the $\beta\eta$-reduction, the notion of a $\beta\eta$-leftmost redex occurrence is not deterministic as well. However this is a small problem, since the only possibility for the ambiguity in this case is when considering terms with subterm occurrences of the form $(\lambda x.px)q$, where $x \notin p$ (having two distinct leftmost $\beta\eta$-redex occurrences: $\lambda x.px$ and $(\lambda x.px)q$, the both contractions of which lead to the same result pq).

By $left_{\beta\eta}$ denote the so-called $\beta\eta$-leftmost strategy which in every λ-term always contracts the $\beta\eta$-leftmost-outermost redex occurrence (if any). For each term t, it determines a certain finite or infinite $\beta\eta$-reduction sequence starting with t which is also called $\beta\eta$-leftmost. Analogously, by $left_\beta$ denote the β-leftmost strategy.

We write $t \xrightarrow[\beta\eta\text{-}left]{} t'$ and $t \xrightarrow[\beta\text{-}left]{} t'$ instead of, resp., $t' = left_{\beta\eta}(t)$ and $t' = left_\beta(t)$. The notation $t \xrightarrow[\beta\eta\text{-}left]{\triangle}_\eta t'$ means that $t \xrightarrow[\beta\eta\text{-}left]{} t'$ and $t \xrightarrow{\triangle}_\eta t'$. Therefore, $\xrightarrow[\beta\eta\text{-}left]{}_\eta$ can be considered as a binary relation on the set of λ-terms; by $\underset{\beta\eta\text{-}left,\neq\varnothing}{\twoheadrightarrow}_\eta$ denote its transitive closure. The relations $\xrightarrow[\beta\eta\text{-}left]{}_\beta$ and $\underset{\beta\eta\text{-}left,\neq\varnothing}{\twoheadrightarrow}_\beta$ are introduced analogously, the same concerns $\xrightarrow[\beta\text{-}left]{}_\beta$ and $\underset{\beta\text{-}left,\neq\varnothing}{\twoheadrightarrow}_\beta$.

Lemma 2. *The following diagram holds:*

$$\begin{tikzcd}
& \cdot \arrow[dl, dashed, two heads, "\beta\text{-}left{,}\neq\varnothing"', "\beta" near end] \arrow[dr, two heads, "\beta\eta\text{-}left{,}\neq\varnothing", "\eta"' near end] & \\
\cdot \arrow[dr, dashed, two heads, "\beta\eta\text{-}left{,}\neq\varnothing"', "\eta" near end] & & \cdot \arrow[dl, two heads, "\beta\text{-}left{,}\neq\varnothing", "\beta"' near end] \\
& \cdot &
\end{tikzcd}$$

Proof. First consider the case of a two-step reduction sequence of the form $t_1 \xrightarrow[\beta\eta\text{-}left]{\Delta_\eta}_\eta t_2 \xrightarrow[\beta\text{-}left]{\Delta_\beta}_\beta t_3$. Since Δ_η is the $\beta\eta$-leftmost-outermost redex, it follows that Δ_η is strictly to the left of $\overleftarrow{\Delta}_\beta$ in t_1. Obviously, this is possible only in the cases 1 and 2 from the proof of the η-Postponement theorem, which, as noted there, leads to the following diagram:

$$\begin{tikzcd}
& t_1 \arrow[dl, dashed, "\beta\text{-}left"', "\overleftarrow{\Delta}_\beta"] \arrow[dr, "\beta\eta\text{-}left", "\Delta_\eta"'] & \\
t_2' \arrow[dr, dashed, "\beta\eta\text{-}left"', "\overrightarrow{\Delta}_\eta"] & & t_2 \arrow[dl, "\beta\text{-}left", "\Delta_\beta"'] \\
& t_3 &
\end{tikzcd}$$

(at that, evidently, $\overleftarrow{\Delta}_\beta$ is indeed the β-leftmost redex occurrence in t_1 and $\overrightarrow{\Delta}_\eta$ the $\beta\eta$-leftmost-outermost redex occurrence in t_2').

Now the general case can be concluded from the following typical example of a diagram can arise under such conditions (in which all the labels are omitted due to the triviality of its construction):

$$\begin{tikzcd}
\cdot \arrow[r] \arrow[d, dashed] & \cdot \arrow[r] \arrow[d, dashed] & \cdot \arrow[r] \arrow[d, dashed] & \cdot \arrow[r] \arrow[d, dashed] & \cdot \arrow[d] \\
\cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[d] \\
\cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[r, dashed] \arrow[d, dashed] & \cdot \arrow[d] \\
\cdot \arrow[r, dashed] & \cdot \arrow[r, dashed] & \cdot \arrow[r, dashed] & \cdot \arrow[r, dashed] & \cdot
\end{tikzcd}$$

□

Recall that a $\beta\eta$-strategy f is called normalizing if whenever a term t has a $\beta\eta$-normal form, $f^n(t)$ is a $\beta\eta$-normal form for some natural number n.

Normalization Theorem for $\beta\eta$-Reduction. [4; 5] *The strategy $left_{\beta\eta}$ is normalizing.*

Proof. Supposing the contrary, there is a term t having a $\beta\eta$-normal form, the $\beta\eta$-leftmost reduction sequence σ of which is infinite. It will be proved, by means of postponing η-reduction with the help of Lemma 2, that σ can be reconstructed into the infinite β-leftmost reduction sequence (starting with t), which leads to a contradiction: indeed, by the Normalization theorem for β-reduction ([1; 3]), then t does not have a β-normal form and by the $\beta\eta$-Normal form theorem, t does not have a $\beta\eta$-normal form as well.

It can be assumed without loss of generality that σ contracts at least one η-redex (otherwise, σ is already β-leftmost) and, moreover, that it contracts the infinite number of η-redeces (otherwise, exclude from σ such an initial segment that the resulting sequence contracts β-redeces only, which sends back to the previous case). On the other hand, σ should contract the infinite number of β-redeces, since the notion of η-reduction is strongly normalizing. In addition to all these conditions, it can also be assumed, for definiteness, that σ starts with an η-redex contraction (otherwise, exclude from σ its β-prefix). Then σ can be represented in a form of the following infinite sequence:

$$\sigma\colon t_0 \xrightarrow[\beta\eta\text{-}left,\neq\varnothing]{}\!\!\!\!\!\!\!\!\twoheadrightarrow_{\eta} u_0 \xrightarrow[\beta\eta\text{-}left,\neq\varnothing]{}\!\!\!\!\!\!\!\!\twoheadrightarrow_{\beta} t_1 \xrightarrow[\beta\eta\text{-}left,\neq\varnothing]{}\!\!\!\!\!\!\!\!\twoheadrightarrow_{\eta} u_1 \xrightarrow[\beta\eta\text{-}left,\neq\varnothing]{}\!\!\!\!\!\!\!\!\twoheadrightarrow_{\beta} \cdots,$$

where $t_0 \equiv t$ (i.e. σ is being divided into alternating η- and β-segments).

Finally, notice that every $\beta\eta$-leftmost reduction sequence that contracts β-redeces only is evidently the β-leftmost reduction sequence as well. Now the following infinite diagram can be constructed with the help of Lemma 2:

$$\begin{tikzcd}
t_0 \arrow[r, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] \arrow[d, dashed, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"'] & u_0 \arrow[d, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"] & & & \\
\cdot \arrow[r, dashed, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] \arrow[d, dashed, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"'] & t_1 \arrow[r, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] & u_1 \arrow[d, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"] & & \\
\cdot \arrow[rr, dashed, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] \arrow[d, dashed, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"'] & & t_2 \arrow[r, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] & u_2 \arrow[d, two heads, "\beta\text{-}left{,}\neq\varnothing\quad\beta"] & \\
\cdot \arrow[rrr, dashed, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] \arrow[d, dotted, two heads] & & & t_3 \arrow[r, two heads, "\beta\eta\text{-}left{,}\neq\varnothing\quad\eta"'] & u_3 \arrow[d, dotted, two heads] \\
\ldots & & & & \ldots
\end{tikzcd}$$

Its vertical line determines the β-leftmost reduction sequence starting with $t_0 \equiv t$. Thus, the latter is indeed infinite, just as expected. $\square$

Note that this proof is rather close to that from [5]. (Both the proofs are based on the idea to reduce the Normalization theorem for $\beta\eta$-reduction to its analogue for β-reduction by means of postponing η-reduction and taking into account that η-reduction is strongly normalizing, both of them are proceeded by contradiction and exploit Lemma 2, however in the other proof the contradiction is obtained in a different way.)

An interested reader is invited to compare the constructed proofs with their original or known analogues. The detailed references to the latter are contained in the following table compiled for his convenience:

η-Postponement theorem	[1, pp. 384 – 386] [2, pp. 132 – 135]
$\beta\eta$-Normal form theorem	[1, pp. 384 – 386] [3, pp. 313 – 314]
Normalization theorem for $\beta\eta$-reduction	[4, pp. 279 – 290] [5, pp. 529 – 537]

References

[1] H. Barendregt. *The Lambda Calculus. Its Syntax and Semantics* (revised ed.), Elsevier (1984).

[2] H. Curry, R. Feys, W. Craig. *Combinatory Logic* (vol. I), North-Holland: Amsterdam (1958).

[3] H. Curry, J. Hindley, J. Seldin. *Combinatory Logic* (vol. II), North-Holland: Amsterdam (1972).

[4] J.W. Klop. *Combinatory reduction systems* (PhD thesis), Utrecht university (1980).

[5] A. Lyaletsky. *New proofs of important theorems of extensional untyped λ-calculus*, Cybernetics and Systems Analysis, 50(4): pp. 529–537, 2014.

Alexandre Lyaletsky Received September 21, 2015

Alexandre Lyaletsky
Institution: Taras Shevchenko National University of Kyiv (Faculty of Cybernetics)
Phone: +38 067 4086768
E–mail: `foraal@mail.ru`

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Finite automata over algebraic structures: models and some methods of analysis

Volodymyr V. Skobelev, Volodymyr G. Skobelev

Abstract

In this paper some results of research in two new trends of finite automata theory are presented. For understanding the value and the aim of these researches some short retrospective analysis of development of finite automata theory is given. The first trend deals with families of finite automata defined via recurrence relations on algebraic structures over finite rings. The problem of design of some algorithm that simulates with some accuracy any element of given family of automata is investigated. Some general scheme for design of families of hash functions defined by outputless automata is elaborated. Computational security of these families of hash functions is analyzed. Automata defined on varieties with some algebra are presented and their homomorphisms are characterized. Special case of these automata, namely automata on elliptic curves, are investigated in detail. The second trend deals with quantum automata. Languages accepted by some basic models of quantum automata under supposition that unitary operators associated with input alphabet commute each with the others are characterized.

Keywords: finite automata, finite rings, varieties, simulation, hash functions, elliptic curves, quantum automata.

1 Introduction

It is well known that 'an automaton' is one of the basic notions of computer science. Its significance was established in the fundamental paper of A.M. Turing [1] where it has been used as some formal model for informal notion of 'an algorithm' (i.e. either a digital transducer, or an acceptor of a language). Foundations of finite automata (FA) theory were laid in the middle of XX century [2]. In its essence any finite automaton presents some formal model for processes that can be implemented on computers (under the subject to the limitation that the memory is finite).

Development of FA theory has been motivated not only by its internal problems, but also it has been carried out in close interaction with other areas of computer science. The last circumstance in many respects led to numerous applications of FA models. On the other hand, research of actual applied problems (including the ones in the area of modern information technologies) and emergence of some new paradigms for notion of 'computation' led to significant reconsideration problems in FA theory. As the result, the formation of some entirely new sections of this theory has been started.

In this paper we consider two of these new sections. The first one deals with FA defined via recurrence relations over some finite ring. In many ways this section owes its appearance to research in the field of information protection. Moreover, the necessity of investigation of these models is substantially caused by the problems of modern cryptography [3, 4]. The second section deals with quantum FA, i.e. with some section of quantum algorithms theory which is being developed intensively at present. In this context, an essential factor is that the notion 'quantum FA' is based on the new paradigm of computations called 'quantum computations' [5, 6].

2 Survey of finite automata theory

The following two stages can be naturally highlighted in the development of FA automata theory.

The first stage covers 50s–80s of the XX century.

Finite automaton considered as a transducer has been defined as a system $\mathtt{M} = (\mathtt{Q}, \mathtt{X}, \mathtt{Y}, \delta, \lambda)$ (where $\mathtt{Q}$, $\mathtt{X}$ and $\mathtt{Y}$ are respectively finite set of states, finite input alphabet and finite output alphabet, $\delta : \mathtt{Q} \times \mathtt{X} \to \mathtt{Q}$ is the transition function and $\lambda : \mathtt{Q} \times \mathtt{X} \to \mathtt{Y}$ is the output function). Moore and Mealy models of FA and some their variants associated with FA functioning in time have been determined. The problems of analysis and synthesis for FA [7, 8, 9], the problem of completeness for FA [10,11] and problems of theory of experiments with FA [12] have been investigated within these models. Analysis of transformations of free semigroups carried out by FA [13] had a significant influence on formation of algebraic theory of FA [14,15] and automata-algebraic approach to software engineering [16].

It should be noted investigation of information-lossless FA [17, 18] which (possibly with some additional information) carry-out injective transformations of input semigroup into output semigroup. Some important subclass of information-lossless FA form reversible FA. These FA are characterized in that the input alphabet coincides with the output alphabet, and in each state it is carried out some bijective conversion of input symbols into output symbols. Reversible FA forms some powerful mathematical tool which enables us to investigate the deep inner connection between the FA theory and the theory of groups. Thus, these FA can be applied successfully in resolving of a wide range of theoretical and applied problems, both. It should be emphasized that just information-lossless FA demonstrate possibility for using of FA as some mathematical model for stream ciphers. Also, numerous applications in resolving of theoretical and applied problems were found for group FA. In these FA transition function carries out some permutation of the set of states for every fixed input symbol value.

Finite automaton considered as an acceptor has been defined as a system $\mathtt{M} = (\mathtt{Q}, \mathtt{X}, \delta, \mathtt{q}_{in}, \mathtt{Q}_{acc})$ (where $\mathtt{Q}$ and $\mathtt{X}$ are respectively finite set of states and finite input alphabet, $\delta : \mathtt{Q} \times \mathtt{X} \to \mathtt{Q}$ is the transition function, $\mathtt{q}_{in} \in \mathtt{Q}$ is the initial state and $\mathtt{Q}_{acc} \subseteq \mathtt{Q}$ is the set of accepting states). An input string is accepted by $\mathtt{M}$ if it transforms the initial state into the set of accepting states. The set of all such input strings is the language accepted by $\mathtt{M}$. It has been proved that for any fixed finite alphabet the set of languages $\mathfrak{L}_{DFA}$ accepted by FA acceptors equals to the set of regular languages (Kleene's theorem). It should be noted that any FA acceptor is some 1-way 1-head Turing Machine (TM) with input tape, i.e. information can only be read (1-way means that at every step the head of TM moves one cell to the right).

Non-deterministic FA acceptors have been investigated under supposition that any subset $\mathtt{Q}_{in} \subseteq \mathtt{Q}$ of initial states could be chosen and any ternary relation $\delta \subseteq \mathtt{Q} \times (\{\Lambda\} \cup \mathtt{X}) \times \mathtt{Q}$ (Λ is the empty symbol) could define admissible transitions. Accepted language has been defined as the set of strings that transform at least one initial state into the set of accepting states. It has been proved that the set of all languages accepted by these acceptors equals to the set $\mathfrak{L}_{DFA}$. Although every non-deterministic FA acceptor can be effectively transformed into equivalent deterministic one, this transformation can lead to a significant increase in cardinality for the set of states (there are known some examples when non-deterministic FA acceptor has n states while equivalent deterministic one has 2^n states). Possibly, just this factor has grounded application of non-deterministic FA acceptors algebra [9] for formation of one of the main classes of discrete event systems intended to automate industrial process control.

Nontrivial generalization of non-deterministic FA acceptors was the emergence of probabilistic FA [19, 20]. In this model for each state and each input symbol the probability of transition into each state is defined (thus, there is some deep inner link between probabilistic FA and finite Markov chains [21]). Formally, probabilistic FA is a system $\mathtt{M} = (\mathtt{Q}, \mathtt{X}, \{M_{\mathtt{x}}\}_{\mathtt{x} \in \mathtt{X}}, \mathtt{u}_0, \mathtt{Q}_{acc})$, where $\mathtt{Q} = \{\mathtt{q}_1, \ldots, \mathtt{q}_n\}$ is the set of states, $\mathtt{X}$ is finite input alphabet, $M_{\mathtt{x}}$ ($\mathtt{x} \in \mathtt{X}$) is some stochastic $n \times n$-matrix of transitions, $\mathtt{u}_0 = (\alpha_1^{(0)}, \ldots, \alpha_n^{(0)})^T$ ($\alpha_i^{(0)} \in \mathbb{R}_+$ ($i \in \mathbb{N}_n$), $\sum\limits_{i=1}^{n} \alpha_i^{(0)} = 1$) is the initial distribution of states, and $\mathtt{Q}_{acc} \subseteq \mathtt{Q}$ is the set of accepting states. The evolution of $\mathtt{M}$ on input string $\mathtt{x}_1 \ldots \mathtt{x}_l$ ($l \in \mathbb{Z}_+$) is defined by identity $(\alpha_1^{(l)}, \ldots, \alpha_n^{(l)})^T = M_{\mathtt{x}_l} \ldots M_{\mathtt{x}_1} \mathtt{u}_0$. This string is accepted by $\mathtt{M}$ with probability $\mathtt{P}_{\mathtt{M}}(\mathtt{x}_1 \ldots \mathtt{x}_l) = \sum \alpha_i^{(l)}$, where the sum is over all i such that $\mathtt{q}_i \in \mathtt{Q}_{acc}$. It is defined that probabilistic FA $\mathtt{M}$ accepts the language $\mathtt{L} \subseteq \mathtt{X}^+$ with: 1) probability p ($0.5 \leq p \leq 1$) if it accepts every string $\mathtt{w}_1 \in \mathtt{L}$ with probability not less than p, while any string $\mathtt{w}_2 \notin \mathtt{L}$ is accepted with probability not exceeding $1-p$; 2) error $(p_1; p_2)$ ($0 \leq p_1 < p_2 \leq 1$) if it accepts every string $\mathtt{w}_1 \in \mathtt{L}$ with probability not less than p_2, while any string $\mathtt{w}_2 \notin \mathtt{L}$ is accepted with probability not exceeding p_1. It should be noted that any probabilistic FA is some

1-way 1-head probabilistic TM with input tape.

Progress in error-correcting codes development [22, 23] and linear systems analysis [24] has stimulated research of FA presented via recurrence relations over finite fields [25].

The second stage in the development of FA automata theory started in 90s of the XX century.

Development of models for cryptographic protection of information had a great influence on FA theory. The following problems became actual. Firstly, it is analysis of pre-images of output strings produced by FA [26]. Secondly, it is analysis of linear and poly-linear recurrences over finite rings [27, 28]. These recurrences define some class of autonomous automata intended for design of generators of pseudo-random sequences used in modern ciphers. Thirdly, it is analysis of experiments with linear and bilinear automata defined via recurrence relations over finite fields [29]. Fourthly, it is investigation of complexity of FA identification [30]. This problem is caused by application of FA for analysis of computational security for stream ciphers [31, 32]. Fifthly, it is investigation of families of FA defined via algebraic recurrence relations over finite rings [33, 34]. If these FA are reversible, they can be used as mathematical models for some stream ciphers.

The problems listed above show that formation of some new section of algebraic theory of FA is carried out at present. Essentially new factor for this section is the transition from transformations of free semigroups to transformations of algebraic structures performed by FA defined via recurrence relations over finite algebraic structures. These researches are presented in Section 3 of the current paper.

Since 1997 a variety of quantum FA (QFA) models different in computational capacity have been investigated. All of them are acceptors and they are defined in terms of 1-way k-head ($k \geq 1$) quantum TM (QTM) with input tape. Accepting of languages was analyzed both from the point 'with given probability' and 'with given error'.

Basic QFA models with measurement of a state only at the last step are listed below ($\mathtt{X}$ ($|\mathtt{X}| = m$) is input alphabet and with every letter $\mathtt{x} \in \mathtt{X}$ some unitary operator $U_{\mathtt{X}}$ acting in n-dimensional complex space $\mathbb{C}^n$ is associated).

The model MO-1QFA [35] is 1-way 1-head QTM $\mathtt{M} = (\mathtt{Q}, \mathtt{X}, |\varphi\rangle, \mathtt{Q}_{acc})$, where $\mathtt{Q} = \mathrm{B}_n$ ($\mathrm{B}_n = \{|i\rangle | i \in \mathbb{N}_n\}$) is the set of basic states, the unit vector $|\varphi\rangle \in \mathbb{C}^n$ is the pure initial state and $\mathtt{Q}_{acc} \subseteq \mathrm{B}_n$ is the set of accepting states. Probability that $\mathtt{M}$ accepts a string $w = x_1 \dots x_l \in X^+$ equals to $\mathrm{P}(|\varphi\rangle, w) = \| P_{acc} U_w |\varphi\rangle \|^2$, where $U_w = U_{x_l} \dots U_{x_1}$ and P_{acc} is the projection operator on the subspace spanned by $\mathtt{Q}_{acc}$.

The model L-QFA [36] differs from the model MO-1QFA only that it deals with some initial mixed state $\{(|\varphi_i\rangle, \alpha_i)\}_{i \in \mathbb{N}_n}$ such that $|\varphi_i\rangle \in \mathbb{C}^n$ ($i \in \mathbb{N}_n$) are pair-wise different unit vectors, $\alpha_i > 0$ ($i \in \mathbb{N}_n$), and $\sum\limits_{i \in \mathbb{N}_n} \alpha_i = 1$ (α_i ($i \in \mathbb{N}_n$) is referred to as probability that at initial instant QTM $\mathtt{M}$ exists in the state $|\varphi_i\rangle$).

Probability that L-QFA $\mathtt{M}$ accepts a string $w \in X^+$ equals to $\mathrm{P}(\{(|\varphi_i\rangle, \alpha_i)\}_{i \in \mathbb{N}_n}, w) = \sum\limits_{i \in \mathbb{N}_n} \alpha_i \mathrm{P}(|\varphi_i\rangle, w)$.

The model kQFA [37] is 1-way k-head QTM $\mathtt{M} = (\mathtt{Q}, \mathtt{T}, |\varphi\rangle, \mathtt{Q}_{acc})$ (at any instant all heads move simultaneously by one cell to the right), where $\mathtt{T} = \mathtt{X}^k \cup \bigcup\limits_{i=1}^{k-1} \mathtt{X}^i \{\Lambda\}^{k-i}$. It is worth to note that similarly to the case when the model L-QFA was defined as some generalization of the model MO-1QFA, in [38] the model L-kQFA was defined as some generalization of the model kQFA.

Currently, analysis of QFA models is focused on detailed study of the set of accepted languages, as well as on resolving of the problem of identification of equivalent states. Some research of languages accepted by above listed models of QFA is presented in Section 4 of the current paper.

3 Automata over algebraic structures

Let $\mathcal{K} = (K, +, \cdot)$ be fixed finite ring and $\mathcal{M} = \{\mathtt{M}_{\mathbf{a}}\}_{\mathbf{a} \in \mathbf{A}}$ ($\mathbf{A} \subseteq K^l$) be any family of FA

$$\mathtt{M}_{\mathbf{a}} : \begin{cases} \mathbf{q}_{t+1} = \mathbf{f}_1(\mathbf{q}_t, \mathbf{x}_{t+1}, \mathbf{a}) \\ \mathbf{y}_{t+1} = \mathbf{f}_2(\mathbf{q}_t, \mathbf{x}_{t+1}, \mathbf{a}) \end{cases} \quad (t \in \mathbb{Z}_+),$$

where $\mathbf{f}_1 : K^{n_1+n_2+l} \to K^{n_1}$ and $\mathbf{f}_2 : K^{n_1+n_2+l} \to K^{n_3}$ are fixed mappings, and $\mathbf{a}$ are parameters. It is known that via any experiment with an automaton $\mathtt{M}_{\mathbf{a}}$ the values of parameters $\mathbf{a} \in \mathbf{A}$ not always can be identified uniquely. So naturally arises the problem of design of some algorithm that simulates any $\mathtt{M}_{\mathbf{a}} \in \mathcal{M}$ with some accuracy (from the standpoint of cryptography this means 'an attack on the algorithm'). This problem has been resolved in [39, 40]. The essence of proposed solution is as follows.

We fix a set of parameters $\mathbf{B} \subseteq K^{l_1}$ and three families of mappings $\{\varphi^{(1)}_{\mathbf{b}} : K^{n_1+n_2} \to K^{n_3}\}_{\mathbf{b}\in\mathbf{B}}$, $\{\varphi^{(2)}_{\mathbf{b}} : K^{n_1} \times \bigcup\limits_{j=1}^{r-1} (K^{n_3})^j \times K^{n_2} \to K^{n_3}\}_{\mathbf{b}\in\mathbf{B}}$ and $\{\varphi^{(3)}_{\mathbf{b}} : K^{n_1+rn_3+n_2} \to K^{n_3}\}_{\mathbf{b}\in\mathbf{B}}$. Let $\mathcal{G}_{\mathbf{B}} = \{G_{\mathbf{b}}\}_{\mathbf{b}\in\mathbf{B}}$ be the set of mappings, such that $G_{\mathbf{b}}(\mathbf{q}_0, \mathbf{x}_1 \ldots \mathbf{x}_m) = \mathbf{y}_1 \ldots \mathbf{y}_m$ ($\mathbf{b} \in \mathbf{B}, m \in \mathbb{N}$), where

$$\mathbf{y}_i = \begin{cases} \varphi^{(1)}_{\mathbf{b}}(\mathbf{q}_0, \mathbf{x}_1), & \text{if } i = 1 \\ \varphi^{(2)}_{\mathbf{b}}(\mathbf{q}_0, \mathbf{y}_1 \ldots \mathbf{y}_{i-1}, \mathbf{x}_i), & \text{if } i = 2, \ldots, r \\ \varphi^{(3)}_{\mathbf{b}}(\mathbf{q}_0, \mathbf{y}_{i-r} \ldots \mathbf{y}_{i-1}, \mathbf{x}_i), & \text{if } r < i \leq m \end{cases} .$$

Let $H_{\mathbf{b},\mathbf{q}_0}(\mathbf{x}_1 \ldots \mathbf{x}_m) = G_{\mathbf{b}}(\mathbf{q}_0, \mathbf{x}_1 \ldots \mathbf{x}_m)$ ($\mathbf{b} \in \mathbf{B}, \mathbf{q}_0 \in K^{n_1}, m \in \mathbb{N}$). It is evident that each family $\mathcal{H}_{\mathbf{b}} = \{H_{\mathbf{b},\mathbf{q}_0}\}_{\mathbf{q}_0 \in K^{n_1}}$ ($\mathbf{b} \in \mathbf{B}$) defines some finite automaton over the ring $\mathcal{K}$. Fixing surjection $h : \mathbf{A} \to \mathbf{B}$ we associate some family $\mathcal{H}_{h(\mathbf{a})}$ with every automaton $\mathtt{M}_{\mathbf{a}} \in \mathcal{M}$.

The ordered pair $(\mathcal{G}_{\mathbf{B}}, h)$ is defined as simulation model for the family $\mathcal{M}$. It is supposed that equalities $H_{h(\mathbf{a}),\mathbf{q}_0}|_{\bigcup\limits_{i=1}^{r} (K^{n_2})^i} = F_{\mathbf{a},\mathbf{q}_0}|_{\bigcup\limits_{i=1}^{r} (K^{n_2})^i}$ ($\mathbf{a} \in \mathbf{A}, \mathbf{q}_0 \in K^{n_1}$) hold, where $F_{\mathbf{a},\mathbf{q}_0} : (K^{n_2})^+ \to (K^{n_3})^+$ is the mapping realized by initial automaton $(\mathtt{M}_{\mathbf{a}}, \mathbf{q}_0)$. Semantics of these equalities is that simulation model $(\mathcal{G}_{\mathbf{B}}, h)$, connected to the input and the output channels of an automaton $\mathtt{M}_{\mathbf{a}}$ ($\mathbf{a} \in \mathbf{A}$) passes the first r output symbols, and then blocks the output channel of an automaton $\mathtt{M}_{\mathbf{a}}$ and simulates its behavior on the remaining tail of input string.

On the base of standard techniques of algorithms theory accuracy of simulation model $(\mathcal{G}_{\mathbf{B}}, h)$ has been defined for all combinations of notions 'in the worst case' and 'in average'. Asymptotically exact simulation models have been extracted and some sufficient conditions for existence of these models have been established in [39, 40].

It is evident that any hash function is some mapping of input semigroup into the set of states realized by some finite initial automaton. From the standpoint of cryptography analysis of hash functions families defined by outputless FA over finite ring is actual. This problem has been investigated in [41]. The main results are as follows.

Let $\mathcal{F}_{k,m}$ $(k \le m)$ be the set of all mappings $\mathbf{f}: K^{k+m} \to K^k$, such that the following two equalities $|\{\mathbf{x} \in K^m | \mathbf{f}(\mathbf{q},\mathbf{x}) = \mathbf{q}''\}| = |K|^{m-k}$ and $\{\mathbf{x} \in K^m | \mathbf{f}(\mathbf{q},\mathbf{x}) = \mathbf{q}''\} \cap \{\mathbf{x} \in K^m | \mathbf{f}(\mathbf{q}',\mathbf{x}) = \mathbf{q}''\} = \emptyset$ hold for all $\mathbf{q}, \mathbf{q}', \mathbf{q}'' \in K^k$ $(\mathbf{q} \ne \mathbf{q}')$. It is evident that any mapping $\mathbf{f} \in \mathcal{F}_{k,m}$ defines strongly connected outputless automaton $\mathtt{M}_\mathbf{f}$, such that K^k is the set of states and K^m is input alphabet.

Let $H_{\mathbf{f},\mathbf{q}_0}$ be the mapping of input semigroup $(K^m)^+$ into the set of states K^k realized by initial automaton $(\mathtt{M}_\mathbf{f}, \mathbf{q}_0)$. Thus, automaton $\mathtt{M}_\mathbf{f}$ defines the family of hash functions $\{H_{\mathbf{f},\mathbf{q}_0}\}_{\mathbf{q}_0 \in K^k}$.

The following theorems are true:

Theorem 1. [41]. *For any mapping* $\mathbf{f} \in \mathcal{F}_{k,m}$ *if* $\mathbf{q}_0 \ne \mathbf{q}_0'$ $(\mathbf{q}_0, \mathbf{q}_0' \in K^k)$, *then* $H_{\mathbf{f},\mathbf{q}_0}(\mathbf{u}) \ne H_{\mathbf{f},\mathbf{q}_0'}(\mathbf{u})$ *for any input string* $\mathbf{u} \in (K^m)^+$.

Corollary 1. [41]. *For any* $\mathbf{f} \in \mathcal{F}_{k,m}$ *if* $\mathbf{q}_0 \ne \mathbf{q}_0'$ $(\mathbf{q}_0, \mathbf{q}_0' \in K^k)$, *then* $H^{-1}_{\mathbf{f},\mathbf{q}_0}(\mathbf{q}) \cap H^{-1}_{\mathbf{f},\mathbf{q}_0'}(\mathbf{q}) = \emptyset$ *for any* $\mathbf{q} \in K^k$.

Theorem 2. [41]. *For any mapping* $\mathbf{f} \in \mathcal{F}_{k,m}$ *and* $\mathbf{q}_0 \in K^k$ *equality* $|H^{-1}_{\mathbf{f},\mathbf{q}_0}(\mathbf{q}_t) \cap (K^m)^t| = |K|^{tm-k}$ $(\mathbf{q}_t \in K^k)$ *holds for all* $t \in \mathbb{N}$.

Let $p^{(1)}_{\mathbf{f},\mathbf{q}_0,t}(\mathbf{q})$ be probability that input string $\mathbf{u}$ randomly selected in the set $(K^m)^t$ is some solution of the equation $H(\mathbf{u}) = \mathbf{q}$, and $p^{(2)}_{\mathbf{f},\mathbf{q}_0,t}$ be probability that for two different input strings $\mathbf{u}$ and $\mathbf{u}'$ randomly selected in the set $(K^m)^t$ equality $H(\mathbf{u}) = H(\mathbf{u}')$ holds.

The following theorems are true:

Theorem 3. [41]. *For any mapping* $\mathbf{f} \in \mathcal{F}_{k,m}$ and $\mathbf{q}_0, \mathbf{q} \in K^k$ *equality* $p^{(1)}_{\mathbf{f},\mathbf{q}_0,t}(\mathbf{q}) = |K|^{-k}$ *holds for all* $t \in \mathbb{N}$.

Theorem 4. [41]. *For any mapping* $\mathbf{f} \in \mathcal{F}_{k,m}$ *and* $\mathbf{q}_0 \in K^k$ *equality* $p^{(2)}_{\mathbf{f},\mathbf{q}_0,t} = |K|^{-k}(1 - \frac{|K|^k - 1}{|K|^{mt} - 1})$ *holds for all* $t \in \mathbb{N}$.

Thus, the number $|K|^{-k}$ characterizes computing security for a family of hash functions $\{H_{\mathbf{f},\mathbf{q}_0}\}_{\mathbf{q}_0\in K^k}$. This implies some feasibility for using these families in resolving problems of information protection.

Applications of elliptic curves over finite fields for resolving problems of information transformation justify feasibility of research FA defined on varieties (i.e. on the sets of solutions of systems of algebraic equations) over finite ring. It allows to set internal connections between modern algebraic geometry, systems theory, FA theory and cryptology.

From standpoint of algebraic FA theory and its applications it is reasonable to deal with the set $\mathcal{V}_1(\mathcal{K})$ of all varieties $\mathbf{V} \subseteq K^n$ with some algebra $(\mathbf{V}, \mathcal{F}_1 \cup \mathcal{F}_2)$, where $\mathcal{F}_1 = \{\alpha_0, \alpha_1, \ldots, \alpha_{k_1}\}$ and $\mathcal{F}_2 = \{\beta_1, \ldots, \beta_{k_2}\}$ are the sets of unary and binary operations, correspondingly. For any variety $\mathbf{V} \in \mathcal{V}_1(\mathcal{K})$ the algebra $(\mathbf{V}, \mathcal{F}_1 \cup \mathcal{F}_2)$ gives possibility to define the set $\mathcal{A}^{(1)}(\mathbf{V})$ of Mealy FA

$$\begin{cases} \mathbf{q}_{t+1} = \beta_{j_1}(\alpha_{i_1}(\mathbf{q}_t), \alpha_{x_{t+1}}(\mathbf{v}_1)) \\ \mathbf{y}_{t+1} = \beta_{j_2}(\alpha_{i_2}(\mathbf{q}_t), \alpha_{x_{t+1}}(\mathbf{v}_2)) \end{cases} \qquad (t \in \mathbb{Z}_+)$$

and the set $\mathcal{A}^{(2)}(\mathbf{V})$ of Moore FA

$$\begin{cases} \mathbf{q}_{t+1} = \beta_{j_1}(\alpha_{i_1}(\mathbf{q}_t), \alpha_{x_{t+1}}(\mathbf{v}_1)) \\ \mathbf{y}_{t+1} = \beta_{j_2}(\alpha_{i_2}(\mathbf{q}_{t+1}), \mathbf{v}_2) \end{cases} \qquad (t \in \mathbb{Z}_+),$$

where $\mathbf{v}_1, \mathbf{v}_2 \in \mathbf{V}$ are fixed points, $i_1, i_2 \in \mathbb{Z}_{k_1+1}$ and $j_1, j_2 \in \mathbb{N}_{k_2}$ are fixed integers, $\mathbf{q}_0 \in \mathbf{V}$, and $x_{t+1} \in \mathbb{Z}_{k_1+1}$ $(t \in \mathbb{Z}_+)$. Thus, for any $\mathtt{M} \in \mathcal{A}^{(1)}(\mathbf{V}) \cup \mathcal{A}^{(2)}(\mathbf{V})$ values of x_t, $\mathbf{q}_t$ and $\mathbf{y}_t$ are, correspondingly, an input symbol, a state and an output symbol at instant t.

Let $\mathbf{V}, \mathbf{U} \in \mathcal{V}_1(\mathcal{K})$. We say that: 1) the variety $\mathbf{U}$ is a homomorphic image of the variety $\mathbf{V}$, if the algebra $(\mathbf{U}, \mathcal{F}_1^{(2)} \cup \mathcal{F}_2^{(2)})$ is a homomorphic image of the algebra $(\mathbf{V}, \mathcal{F}_1^{(1)} \cup \mathcal{F}_2^{(1)})$; 2) varieties $\mathbf{U}$ and $\mathbf{V}$ are isomorphic if algebras $(\mathbf{U}, \mathcal{F}_1^{(2)} \cup \mathcal{F}_2^{(2)})$ and $(\mathbf{V}, \mathcal{F}_1^{(1)} \cup \mathcal{F}_2^{(1)})$ are isomorphic.

The next theorem is true:

Theorem 5. [42]. *Let* $\mathbf{U}, \mathbf{V} \in \mathcal{V}_1(\mathcal{K})$. *If* $\mathbf{U}$ *is a homomorphic image of* $\mathbf{V}$, *then there exist mappings* $\Psi_j : \mathcal{A}^{(j)}(\mathbf{V}) \to \mathcal{A}^{(j)}(\mathbf{U})$ $(j = 1, 2)$, *such that homomorphic image of any automaton* $\mathtt{M}_j \in \mathcal{A}^{(j)}(\mathbf{V})$ *is the automaton* $\Psi_j(\mathtt{M}_j)$.

Corollary 2. [42]. *Let* $\mathbf{U},\mathbf{V}\in\mathcal{V}_1(\mathcal{K})$*. If* $\mathbf{U}$ *and* $\mathbf{V}$ *are isomorphic varieties, then there exist mappings* $\Psi_j:\mathcal{A}^{(j)}(\mathbf{V})\to\mathcal{A}^{(j)}(\mathbf{U})$ $(j=1,2)$*, such that automata* $\mathtt{M}_j\in\mathcal{A}^{(j)}(\mathbf{V})$ and $\Psi_j(\mathtt{M}_j)$ *are isomorphic.*

Any elliptic curve γ over a finite field $\mathcal{K}=(K,+,\cdot)$ defines the abelian group $(G_\gamma,+_\gamma)$, where G_γ is the set of all points of γ including specified point $\mathcal{O}$ (this point serves as the neutral element of the group). Setting $\mathcal{F}_1=\{\alpha_0,\alpha_1,\ldots,\alpha_{k_1}\}$ $(1\le k_1<|G_\gamma|)$, where $\alpha_0(P)=\mathcal{O}$ $(P\in G_\gamma)$ and $\alpha_i(P)=\underbrace{P+_\gamma\ldots+_\gamma P}_{i\ \text{times}}$ $(P\in G_\gamma)$ for all $i=1,\ldots,k_1$, and $\mathcal{F}_2=\{+_\gamma\}$, we get some algebra $(G_\gamma,\mathcal{F}_1\cup\mathcal{F}_2)$. Thus, any elliptic curve γ defines some variety of above considered type.

For any $P_1,P_2\in G_\gamma\backslash\{\mathcal{O}\}$ and $n,m\in\mathbb{N}_{k_1}$ recurrence relations

$$\begin{cases} q_{t+1}=nq_t+_\gamma x_tP_1 \\ y_{t+1}=mq_t+_\gamma x_tP_2 \end{cases} \quad (t\in\mathbb{Z}_+)$$

and

$$\begin{cases} q_{t+1}=nq_t+_\gamma x_tP_1 \\ y_{t+1}=mq_{t+1} \end{cases} \quad (t\in\mathbb{Z}_+),$$

where $x_{t+1}\in\mathbb{N}_{k_1}$, define the family $\mathcal{M}_{1,\gamma,k_1}$ of Mealy FA and the family $\mathcal{M}_{2,\gamma,k_1}$ of Moore FA, correspondingly.

The following theorems are true:

Theorem 6. [43]. *For any automaton* $\mathtt{M}_1\in\mathcal{M}_{1,\gamma,k_1}$ *identification of its initial state (with the accuracy to the set of equivalent states) is reduced to searching any solution of equation* $mu=a_0$*, where an element* $a_0\in G_\gamma$ *is determined as the result of some simple experiment of the length* 1 *with the automaton* $\mathtt{M}_1$*.*

Theorem 7. [43]. *For any automaton* $\mathtt{M}_2\in\mathcal{M}_{2,\gamma,k_1}$ *identification of its initial state (with the accuracy to the set of equivalent states) is reduced to searching any solution of equation* $mnv=b_0$*, where an element* $b_0\in G_\gamma$ *is determined as the result of some simple experiment of the length* 1 *with the automaton* $\mathtt{M}_2$*.*

Theorem 8. [43]. *Exact imitation model for the family* $\mathcal{M}_{1,\gamma,k_1}$ *of Mealy FA can be designed as the result of some multiple experiment of the multiplicity* 3 *and of the height not exceeding* $|G_\gamma|+1$*. The total* *length of all input strings applied to the investigated automaton in this experiment does not exceed* $|G_\gamma| + 1 + 0.5|G_\gamma| \cdot (|G_\gamma| + 3)$.

Theorem 9. [43]. *Exact imitation model for the family* $\mathcal{M}_{2,\gamma,k_1}$ *of Moore FA can be designed as the result of some multiple experiment of the multiplicity* 2 *and of the height not exceeding* $|G_\gamma|$. *The total length of all input strings applied to the investigated automaton in this experiment does not exceed* $|G_\gamma| + 0.5|G_\gamma| \cdot (|G_\gamma| + 1)$.

These results imply some feasibility for using the above considered families of FA in resolving problems of information protection.

4 Quantum Automata

QFA under supposition that unitary operators associated with input alphabet commute each with the others have been investigated in [44]. Languages accepted either with given probability, or with given error have been characterized as follows.

Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_m\}$ be the input alphabet of QFA. It is supposed that elements of the set $\mathcal{U} = \{U_i | i \in \mathbb{N}_m\}$ (U_i is unitary operator associated with $\mathtt{x}_i \in \mathtt{X}$) commute each with the others. With any input string $\mathtt{w} \in \mathtt{X}^l$ ($l \in \mathbb{N}$) the string $pr_{\mathcal{U}}(\mathtt{w}) = U_1^{r_1} \ldots U_m^{r_m}$ can be associated, where r_i ($i \in \mathbb{N}_m$) is the number of occurrences of $\mathtt{x}_i$ in $\mathtt{w}$.

Let $\equiv_{\mathtt{X},\mathcal{U}}$ be equivalence on the set $\mathtt{X}^+$ defined as follows: for any input strings $\mathtt{w}_1, \mathtt{w}_2 \in \mathtt{X}^+$

$$\mathtt{w}_1 \equiv_{\mathtt{X},\mathcal{U}} \mathtt{w}_2 \Leftrightarrow pr_{\mathcal{U}}(\mathtt{w}_1) = pr_{\mathcal{U}}(\mathtt{w}_2).$$

The following theorem is true:

Theorem 10. [44]. *Let* $\mathcal{U}$ *be any set of unitary operators that commute each with the others. Then any language accepted (either with given probability, or with given error) by the model MO-1QFA, as well as by the model L-QFA with measurement at final instant only is union of some elements of the factor-set* $\mathtt{X}^+/\equiv_{\mathtt{X},\mathcal{U}}$.

It is evident that with any element $B \in \mathtt{X}^+/\equiv_{\mathtt{X},\mathcal{U}}$ unique unitary operator U_B can be associated, such that $U_B = pr_{\mathcal{U}}(\mathtt{w})$ for all $w \in B$. Let $\equiv'_{\mathtt{X},\mathcal{U}}$ be any equivalence on the set $\mathtt{X}^+$, such that every element of the factor-set $\mathtt{X}^+/{\equiv'_{\mathtt{X},\mathcal{U}}}$ is union of some elements $B \in \mathtt{X}^+/{\equiv_{\mathtt{X},\mathcal{U}}}$ to which the same unitary operator U_B is associated.

The following corollary is true.

Corollary 3. [44]. *Let $\mathcal{U}$ be any set of unitary operators that commute each with the others. Then any language accepted (either with given probability, or with given mistake) by the model MO-1QFA, as well as by the model L-QFA with measurement at final instant only is union of some elements of the factor-set $\mathtt{X}^+/{\equiv'_{\mathtt{X},\mathcal{U}}}$.*

Important special case of equivalence $\equiv'_{\mathtt{X},\mathcal{U}}$ on the set $\mathtt{X}^+$ takes place in the following situation.

Let $\mathcal{U} = \{U_i | i \in \mathbb{N}_m\}$ ($U_i \neq I$ for all $i \in \mathbb{N}_m$) be some set of unitary operators that commute each with the others, such that for every $i \in \mathbb{N}_m$ there exists some positive integer a_i that satisfies the identity $U_i^{a_i} = I$. In what follows it is assumed that a_i ($i \in \mathbb{N}_m$) is the minimal positive integer that satisfies this identity. We define equivalence $\equiv^{(1)}_{\mathtt{X},\mathcal{U}}$ on the set $\mathtt{X}^+$ in the following way: for any input strings $\mathtt{w}_1, \mathtt{w}_2 \in \mathtt{X}^+$ ($pr_{\mathcal{U}}(\mathtt{w}_i) = U_1^{r_{i1}} \dots U_m^{r_{im}}$ $(i = 1, 2)$)

$$\mathtt{w}_1 \equiv^{(1)}_{\mathtt{X},\mathcal{U}} \mathtt{w}_2 \Leftrightarrow$$

$$\Leftrightarrow U_1^{r_{11} (mod\, a_1)} \dots U_m^{r_{1m} (mod\, a_m)} = U_1^{r_{21} (mod\, a_1)} \dots U_m^{r_{2m} (mod\, a_m)}.$$

It is evident that the equivalence $\equiv^{(1)}_{\mathtt{X},\mathcal{U}}$ is some special case of equivalence $\equiv'_{\mathtt{X},\mathcal{U}}$. Moreover, the following identity holds: $|\mathtt{X}^+/{\equiv^{(1)}_{\mathtt{X},\mathcal{U}}}| = \prod\limits_{i=1}^{m} a_i$.

Thus, the following corollary is true.

Corollary 4. [44]. *Let $\mathcal{U} = \{U_i | i \in \mathbb{N}_m\}$ ($U_i \neq I$ for all $i \in \mathbb{N}_m$) be some set of unitary operators that commute each with the others, such that for every $i \in \mathbb{N}_m$ there exists some positive integer a_i that satisfies the identity $U_i^{a_i} = I$. Then any language accepted (either with given probability, or with given mistake) by the model MO-1QFA, as well as by the model L-QFA with measurement at final instant only is union of some elements of the factor-set $\mathtt{X}^+/{\equiv^{(1)}_{\mathtt{X},\mathcal{U}}}$.*

Similar results can be established for models kQFA and L-kQFA under supposition that unitary operators associated with elements of the set $\mathtt{X}^k$ commute each with the others. However, some technical difficulties arise with definition of equivalence on the set $\mathtt{T}^+$ due to the presence of elements of the set $\bigcup\limits_{i=1}^{k-1} \mathtt{X}^i\{\Lambda\}^{k-i}$.

In [38] presented above approach has been worked out in detail for one of the most simple non-trivial models of QFA, namely 1-qubit QA under supposition that associated unitary operators are rotations of the Bloch sphere [5, 6] around the y-axe and measurement of a state is produced at final instant only. Criteria when investigated models MO-1QFA, L-QFA, kQFA and L-kQFA accept some language with given probability, as well as with given error has been established.

These results imply feasibility of investigation of the structure of the set of all finitely generated commutative semigroups of special unitary operators in $\mathbb{C}^2$ (the notion 'special' means that the determinant of a matrix that defines unitary operator equals to unit). This problem has been investigated in [45]. Main results are as follows.

Let $\mathcal{V}$ be the set of all special unitary operators $V : \mathbb{C}^2 \to \mathbb{C}^2$ and $\mathfrak{S}$ be the set of all finitely generated commutative semigroups $\mathcal{G} = (G, \cdot)$ $(G \subseteq \mathcal{V})$. The semigroup generated by elements $V_1, \dots, V_k \in \mathcal{V}$ is denoted $(\langle V_1, \dots, V_k\rangle, \cdot)$. Without loss of generality it can be suggested that for any semigroup $(\langle V_1, \dots, V_k\rangle, \cdot) \in \mathfrak{S}$ $(k \geq 2)$ the following condition holds: $(\forall r_1, r_2 \in \mathbb{N}_k)(r_1 \neq r_2 \Rightarrow (\forall n \in \mathbb{N})(V_{r_1}^n \neq V_{r_2}))$.

For any $\gamma \in [0, 4\pi)$ we denote $R_\gamma^{(1)}$, $R_\gamma^{(2)}$ and $R_\gamma^{(3)}$ rotations of the Bloch sphere through the angle 0.5γ around, correspondingly, the x-axe, the y-axe and the z-axe. It is worth to note that any special unitary operator $V \in \mathcal{V}$ can be presented as superposition $V = R_{\gamma_1}^{(3)} R_{\gamma_2}^{(2)} R_{\gamma_3}^{(3)}$ for some $\gamma_1, \gamma_2, \gamma_3 \in [0, 4\pi)$.

The set $\mathfrak{S}_1 = \{(\langle V\rangle, \cdot) | V \in \mathcal{V}\}$ consists of all commutative cyclic semigroups $\mathcal{G} \in \mathfrak{S}$. Setting $\mathfrak{S}_1^{(l)} = \{(\langle V\rangle, \cdot) | V \in \mathcal{V}^{(l)}\}$ $(l = 1, 2, 3)$, where $\mathcal{V}^{(l)} = \{R_\gamma^{(l)} | \gamma \in [0, 4\pi)\}$ $(l = 1, 2, 3)$, we extract in the set $\mathfrak{S}_1$ the sets of all commutative cyclic semigroups of rotation of the Bloch sphere through fixed angle around fixed coordinate axe. It is evident that $(\langle R_\gamma^{(l)}\rangle, \cdot)$ $(\gamma \in [0, 4\pi); l = 1, 2, 3)$ is finite semigroup if and only if $\gamma (mod\, \pi) \in \mathbb{Q}_+$.

For any integer $k \geq 2$ we set

$$\mathfrak{S}_{2l}^{(k)} = \{(\langle R_{\gamma_1}^{(l)}, \ldots, R_{\gamma_k}^{(l)} \rangle, \cdot) | R_{\gamma_1}^{(l)}, \ldots, R_{\gamma_k}^{(l)} \in \mathcal{V}^{(l)} \&$$

$$\&(\forall r_1, r_2 \in \mathbb{N}_k)(\forall n \in \mathbb{N})(r_1 \neq r_2 \Rightarrow (R_{\gamma_{r_1}}^{(l)})^n \neq R_{\gamma_{r_2}}^{(l)})\} \quad (l = 1, 2, 3).$$

The set $\mathfrak{S}_2 = \bigcup\limits_{l=1}^{3} \bigcup\limits_{k=2}^{\infty} \mathfrak{S}_{2l}^{(k)}$ consists of all finitely generated commutative non-cyclic semigroups of rotation of the Bloch sphere around fixed coordinate axe. It is evident that $(\langle R_{\gamma_1}^{(l)}, \ldots, R_{\gamma_k}^{(l)} \rangle, \cdot) \in \mathfrak{S}_{2l}^{(k)}$ $(k \in \mathbb{N}$ $(k \geq 2)$ and $l = 1, 2, 3)$ is finite semigroup if and only if $\gamma_r (mod\, \pi) \in \mathbb{Q}_+$ for all integers $r \in \mathbb{N}_k$.

Now we investigate conditions under which two different special unitary operators of general form commute. Let

$$V_j = \begin{pmatrix} e^{i\alpha_j} \cos 0.5\gamma_j & -e^{-i\beta_j} \sin 0.5\gamma_j \\ e^{i\beta_j} \sin 0.5\gamma_j & e^{-i\alpha_j} \cos 0.5\gamma_j \end{pmatrix} \in \mathcal{V} \quad (j = 1, 2),$$

where $\alpha_j, \beta_j \in [0, 2\pi)$ $(j = 1, 2)$ and $\gamma_j \in [0, 4\pi)$ $(j = 1, 2)$. Then

$$V_1 V_2 = V_2 V_1 \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} (e^{i2(-\beta_1+\beta_2)} - 1) \sin 0.5\gamma_1 \sin 0.5\gamma_2 = 0 \\ e^{i\beta_1} \sin \alpha_1 \sin 0.5\gamma_2 \cos 0.5\gamma_1 = e^{i\beta_2} \sin \alpha_2 \sin 0.5\gamma_1 \cos 0.5\gamma_2 \end{cases} \tag{1}$$

It is sufficient to set these or the others restrictions on the structure of special unitary operator V_1 and determine corresponding restrictions on the structure of special unitary operator V_2. Thus, we can analyze the following cases.

Case 1. Let $\sin 0.5\gamma_1 = 0$ $(\gamma_1 \in [0, 4\pi))$, i.e. $\gamma_1 \in \{0, 2\pi\}$ and $\cos 0.5\gamma_1 = \pm 1$. We get $V_1 \in \mathcal{V}_1(\alpha_1) = \{\widetilde{R}_{\alpha_1}^{(3)}, -\widetilde{R}_{\alpha_1}^{(3)}\}$ $(\alpha_1 \in [0, 2\pi))$, where

$$\widetilde{R}_{\alpha_1}^{(3)} = \begin{pmatrix} e^{i\alpha_1} & 0 \\ 0 & e^{-i\alpha_1} \end{pmatrix} \quad (\alpha_1 \in [0, 2\pi)),$$

i.e. $\widetilde{R}_{\alpha_1}^{(3)}$ is rotation of the Bloch sphere through the angle $-\alpha_1$ around the z-axe.

The second identity in (1) takes the form

$$\sin\alpha_1 \sin 0.5\gamma_2 = 0 \quad (\alpha_1 \in [0,2\pi), \gamma_2 \in [0,4\pi)). \tag{2}$$

The following cases can take place.

Case 1.1. Let $\sin 0.5\gamma_2 = 0$ $(\gamma_2 \in [0,4\pi))$, i.e. $\gamma_2 \in \{0, 2\pi\}$ and $\cos 0.5\gamma_2 = \pm 1$. We get $V_2 \in \mathcal{V}_1(\alpha_2)$.

Let $\mathfrak{S}_3 = \bigcup\limits_{k=2}^{\infty} \mathfrak{S}_3^{(k)}$, where

$$\mathfrak{S}_3^{(k)} = \{(\langle V_1, \ldots, V_k\rangle, \cdot) | V_1, \ldots, V_k \in \bigcup_{\omega\in[0,2\pi)} \mathcal{V}_1(\omega)\&$$

$$\&(\forall r_1, r_2 \in \mathbb{N}_k)(\forall n \in \mathbb{N})(r_1 \neq r_2 \Rightarrow V_{r_1}^n \neq V_{r_2})\}.$$

For any fixed numbers $\alpha_{r_1}, \alpha_{r_2} \in [0,2\pi)$ we get:

1) if $V_{r_1} = \widetilde{R}_{\alpha_{r_1}}^{(3)}$ and $V_{r_2} = \widetilde{R}_{\alpha_{r_2}}^{(3)}$ or $V_{r_1} = -\widetilde{R}_{\alpha_{r_1}}^{(3)}$ and $V_{r_2} = -\widetilde{R}_{\alpha_{r_2}}^{(3)}$, then identity $V_{r_1}^n = V_{r_2}$ holds for some integer $n \in \mathbb{N}$ if and only if relation $n\alpha_{r_1} - \alpha_{r_2} \equiv 0\,(mod\,2\pi)$ holds;

2) if $V_{r_1} = \widetilde{R}_{\alpha_{r_1}}^{(3)}$ and $V_{r_2} = -\widetilde{R}_{\alpha_{r_2}}^{(3)}$, then identity $V_{r_1}^n = V_{r_2}$ holds for some integer $n \in \mathbb{N}$ if and only if $\pi^{-1}(n\alpha_{r_1} - \alpha_{r_2})$ is some odd integer;

3) if $V_{r_1} = -\widetilde{R}_{\alpha_{r_1}}^{(3)}$ and $V_{r_2} = \widetilde{R}_{\alpha_{r_2}}^{(3)}$, then identity $V_{r_1}^n = V_{r_2}$ holds for some $n \in \mathbb{N}$ if and only if either n and $\pi^{-1}(n\alpha_{r_1} - \alpha_{r_2})$ are odd integers, or n is some even integer and relation $n\alpha_{r_1} - \alpha_{r_2} \equiv 0\,(mod\,2\pi)$ holds.

It is evident that the set $\mathfrak{S}_3$ consists of some finitely generated non-cyclic commutative semigroups and inclusion $\mathfrak{S}_3 \subset \mathfrak{S}_{23}^{(2)}$ holds.

Case 1.2. Let $\sin 0.5\gamma_2 \neq 0$ $(\gamma_2 \in [0,4\pi))$, i.e. $\gamma_2 \in [0,4\pi)\backslash\{0,2\pi\}$. Identity (2) takes the form $\sin\alpha_1 = 0$ $(\alpha_1 \in [0,2\pi))$, i.e. $\alpha_1 \in \{0,\pi\}$. We get $V_1 \in \{I, -I\}$.

Let $\mathcal{V}_2 = \{I, -I\}$ and $\mathcal{V}_3$ be the set of all special unitary operators $V_2 \in \mathcal{V}$, such that $\gamma_2 \in [0,4\pi)\backslash\{0,2\pi\}$ and $V_2^n \notin \mathcal{V}_2$ for all $n \in \mathbb{N}$. We get some set $\mathfrak{S}_4 = \{(\langle V_1, V_2\rangle, \cdot) | V_1 \in \mathcal{V}_2, V_2 \in \mathcal{V}_3\}$ of finitely generated non-cyclic commutative semigroups.

Case 2. Let

$$\begin{cases} \sin 0.5\gamma_1 \neq 0 & (\gamma_1 \in [0,4\pi)) \\ \sin 0.5\gamma_2 \neq 0 & (\gamma_2 \in [0,4\pi)) \end{cases},$$

i.e. $\gamma_j \in [0,4\pi)\backslash\{0,2\pi\}$ $(j=1,2)$. The first identity in (1) takes the form $e^{i2(-\beta_1+\beta_2)}-1=0$ $(\beta_1,\beta_2 \in [0,2\pi))$. Without loss of generality we can assume that $\beta_1 \le \beta_2$. We get that either $\beta_1=\beta_2$ and

$$V_j = \begin{pmatrix} e^{i\alpha_j}\cos 0.5\gamma_j & -e^{-i\beta_1}\sin 0.5\gamma_j \\ e^{i\beta_1}\sin 0.5\gamma_j & e^{-i\alpha_j}\cos 0.5\gamma_j \end{pmatrix} \quad (j=1,2),$$

or $\beta_2=\beta_1+\pi$ and

$$V_1 = \begin{pmatrix} e^{i\alpha_1}\cos 0.5\gamma_1 & -e^{-i\beta_1}\sin 0.5\gamma_1 \\ e^{i\beta_1}\sin 0.5\gamma_1 & e^{-i\alpha_1}\cos 0.5\gamma_1 \end{pmatrix},$$

$$V_2 = \begin{pmatrix} e^{i\alpha_2}\cos 0.5\gamma_2 & e^{-i\beta_1}\sin 0.5\gamma_2 \\ -e^{i\beta_1}\sin 0.5\gamma_2 & e^{-i\alpha_2}\cos 0.5\gamma_2 \end{pmatrix}.$$

The second identity in (1) takes the form

$$\sin\alpha_1 \sin 0.5\gamma_2 \cos 0.5\gamma_1 = \sin\alpha_2 \sin 0.5\gamma_1 \cos 0.5\gamma_2, \tag{3}$$

where $\gamma_1,\gamma_2 \in [0,4\pi)\backslash\{0,2\pi\}$. The following cases can take place.

Case 2.1. Let $\cos 0.5\gamma_1 = 0$ $(\gamma_1 \in [0,4\pi)\backslash\{0,2\pi\})$, i.e. $\gamma_1 \in \{\pi,3\pi\}$. We get $V_1 \in \mathcal{V}_4(\beta_1)=\{J_{\beta_1},-J_{\beta_1}\}$ $(\beta_1 \in [0,2\pi))$, where

$$J_{\beta_1} = \begin{pmatrix} 0 & -e^{-i\beta_1} \\ e^{i\beta_1} & 0 \end{pmatrix} \quad (\beta_1 \in [0,2\pi)).$$

Identity (3) takes the form

$$\sin\alpha_2 \cos 0.5\gamma_2 = 0 \quad (\gamma_2 \in [0,4\pi)\backslash\{0,2\pi\}, \alpha_2 \in [0,4\pi)).$$

The following cases can take place.

Case 2.1.1. Let $\cos 0.5\gamma_2 = 0$, i.e. $\gamma_2 \in \{\pi,3\pi\}$. Since $V_2 \in \mathcal{V}_4(\beta_1)$ and $V_2 \ne V_1$, then $V_2=-V_1$. For any $\beta_1 \in [0,2\pi)$ identity $J^2_{\beta_1}=-I$ holds. We get some set $\mathfrak{S}_5 = \{(\langle V,-V\rangle,\cdot) | V \in \bigcup\limits_{\beta_1\in[0,2\pi)} \mathcal{V}_4(\beta_1)\}$ of finite non-cyclic commutative semigroups.

Case 2.1.2. Let $\cos 0.5\gamma_2 \ne 0$, i.e. $\gamma_2 \in [0,4\pi)\backslash\{0\pi,2\pi,3\pi\}$. Then $\sin\alpha_2=0$, i.e. $\alpha_2 \in \{0,\pi\}$. Since $e^{i\alpha_2}=\pm 1$, then

$$V_2 \in \mathcal{V}_5(\beta_1) = \bigcup_{\gamma_2\in[0,4\pi)\backslash\{0\pi,2\pi,3\pi\}} \mathcal{V}_5(\gamma_2,\beta_1) \quad (\beta_1 \in [0,2\pi)),$$

where $\mathcal{V}_5(\gamma_2,\beta_1)=\{U_j(\gamma_2,\beta_1)|j=1,\ldots,4\}$, and

$$U_1(\gamma_2,\beta_1)=\begin{pmatrix}\cos 0.5\gamma_2 & -e^{-i\beta_1}\sin 0.5\gamma_2\\ e^{i\beta_1}\sin 0.5\gamma_2 & \cos 0.5\gamma_2\end{pmatrix},$$

$$U_2(\gamma_2,\beta_1)=\begin{pmatrix}\cos 0.5\gamma_2 & e^{-i\beta_1}\sin 0.5\gamma_2\\ -e^{i\beta_1}\sin 0.5\gamma_2 & \cos 0.5\gamma_2\end{pmatrix},$$

$U_3(\gamma_2,\beta_1)=-U_2(\gamma_2,\beta_1)$ and $U_4(\gamma_2,\beta_1)=-U_1(\gamma_2,\beta_1)$.

It is evident that:

1) if $\beta_1=0$, then $U_1(\gamma_2,\beta_1)$ is rotation of the Bloch sphere through the angle $0.5\gamma_2$ around the y-axe;

2) if $\beta_1=1.5\pi$, then $U_2(\gamma_2,\beta_1)$ is rotation of the Bloch sphere through the angle $0.5\gamma_2$ around the x-axe.

We get some set

$$\mathfrak{S}_6=\bigcup_{\beta_1\in[0,2\pi)}\ \bigcup_{\gamma_2\in[0,4\pi)\backslash\{0\pi,2\pi,3\pi\}}\mathfrak{S}_6(\gamma_2,\beta_1)$$

of finitely generated non-cyclic commutative semigroups, where

$$\mathfrak{S}_6(\gamma_2,\beta_1)=\{(\langle V_1,V_2\rangle,\cdot)|V_1\in\mathcal{V}_4(\beta_1)\&$$

$$\&V_2\in\mathcal{V}_5(\gamma_2,\beta_1)\&(\forall n\in\mathbb{N})(V_2^n\neq V_1)\}.$$

Case 2.2. Let

$$\begin{cases}\cos 0.5\gamma_1\neq 0 & (\gamma_1\in[0,4\pi))\\ \cos 0.5\gamma_2\neq 0 & (\gamma_2\in[0,4\pi))\end{cases},$$

i.e. (see case 2) $\gamma_1,\gamma_2\in[0,4\pi)\backslash\{0,\pi,2\pi,3\pi\}$. The following cases can take place.

Case 2.2.1. Let $\sin\alpha_1=0$ ($\alpha_1\in[0,2\pi)$), i.e. $\alpha_1\in\{0,\pi\}$. Identity (3) takes the form $\sin\alpha_2=0$ ($\alpha_2\in[0,2\pi)$), i.e. $\alpha_2\in\{0,\pi\}$. We get that:

1) if $\beta_2=\beta_1$, then

$$V_j=\begin{pmatrix}\pm\cos 0.5\gamma_j & -e^{-i\beta_1}\sin 0.5\gamma_j\\ e^{i\beta_1}\sin 0.5\gamma_j & \pm\cos 0.5\gamma_j\end{pmatrix}\quad (j=1,2);$$

2) if $\beta_2 = \beta_1 + \pi$, then

$$V_1 = \begin{pmatrix} \pm\cos 0.5\gamma_1 & -e^{-i\beta_1}\sin 0.5\gamma_1 \\ e^{i\beta_1}\sin 0.5\gamma_1 & \pm\cos 0.5\gamma_1 \end{pmatrix},$$

$$V_2 = \begin{pmatrix} \pm\cos 0.5\gamma_2 & e^{-i\beta_1}\sin 0.5\gamma_2 \\ -e^{i\beta_1}\sin 0.5\gamma_2 & \pm\cos 0.5\gamma_2 \end{pmatrix}.$$

It is evident that $V_1 \in \mathcal{V}_6(\gamma_1,\beta_1) = \{U_1(\gamma_1,\beta_1), U_3(\gamma_1,\beta_1)\}$ and $V_2 \in \mathcal{V}_5(\gamma_2,\beta_1)$. We get some set $\mathfrak{S}_7 = \bigcup\limits_{\beta_1\in[0,2\pi)} \mathfrak{S}_7(\beta_1)$ of finitely generated non-cyclic commutative semigroups, where

$$\mathfrak{S}_7(\beta_1) = \bigcup_{\gamma_1,\gamma_2\in[0,4\pi)\backslash\{0,\pi,2\pi,3\pi\}} \{(\langle V_1, V_2\rangle, \cdot) | V_1 \in \mathcal{V}_6(\gamma_1,\beta_1)\&$$

$$\&V_2 \in \mathcal{V}_5(\gamma_2,\beta_1)\&(\forall n\in\mathbb{N})(V_1^n \neq V_2 \& V_2^n \neq V_1)\}.$$

Case 2.2.2. Let

$$\begin{cases} \sin\alpha_1 \neq 0 & (\alpha_1 \in [0,2\pi)) \\ \sin\alpha_2 \neq 0 & (\alpha_2 \in [0,2\pi)) \end{cases},$$

i.e. $\alpha_j \in [0,2\pi)\backslash\{0,\pi\}$ $(j = 1,2)$. Identity (3) takes the form

$$\frac{\sin\alpha_1}{\sin\alpha_2} = \pm\tan 0.5\gamma_1 \cot 0.5\gamma_2,$$

where $\gamma_1,\gamma_2 \in [0,4\pi)\backslash\{0,\pi,2\pi,3\pi\}$ and $\alpha_1,\alpha_2 \in [0,2\pi)\backslash\{0,\pi\}$.

Let $\mathfrak{S}'(\beta_1)$ be the set of all subsets $\{V_1, V_2\}$ of special unitary operators, such that:

1) unitary operators V_j $(j = 1,2)$ are defined by formula

$$V_j = \begin{pmatrix} e^{i\alpha_j}\cos 0.5\gamma_j & -e^{-i\beta_1}\sin 0.5\gamma_j \\ e^{i\beta_1}\sin 0.5\gamma_j & e^{-i\alpha_j}\cos 0.5\gamma_j \end{pmatrix} \quad (j = 1,2),$$

where $\gamma_1,\gamma_2 \in [0,4\pi)\backslash\{0,\pi,2\pi,3\pi\}$ and $\alpha_1,\alpha_2 \in [0,2\pi)\backslash\{0,\pi\}$;

2) identity $\frac{\sin\alpha_1}{\sin\alpha_2} = \tan 0.5\gamma_1 \cot 0.5\gamma_2$ holds;

3) disequalities $V_1^n \neq V_2$ $(n\in\mathbb{N})$ and $V_2^n \neq V_1$ $(n\in\mathbb{N})$ hold.

Similarly, let $\mathfrak{S}''(\beta_1)$ be the set of all subsets $\{V_1, V_2\}$ of special unitary operators, such that:

1) unitary operators V_j $(j = 1, 2)$ are defined by formulae

$$V_1 = \begin{pmatrix} e^{i\alpha_1}\cos 0.5\gamma_1 & -e^{-i\beta_1}\sin 0.5\gamma_1 \\ e^{i\beta_1}\sin 0.5\gamma_1 & e^{-i\alpha_1}\cos 0.5\gamma_1 \end{pmatrix},$$

$$V_2 = \begin{pmatrix} e^{i\alpha_2}\cos 0.5\gamma_2 & e^{-i\beta_1}\sin 0.5\gamma_2 \\ -e^{i\beta_1}\sin 0.5\gamma_2 & e^{-i\alpha_2}\cos 0.5\gamma_2 \end{pmatrix}.$$

where $\gamma_1, \gamma_2 \in [0, 4\pi)\backslash\{0, \pi, 2\pi, 3\pi\}$ and $\alpha_1, \alpha_2 \in [0, 2\pi)\backslash\{0, \pi\}$;

2) identity $\frac{\sin\alpha_1}{\sin\alpha_2} = -\tan 0.5\gamma_1 \cot 0.5\gamma_2$ holds;

3) disequalities $V_1^n \neq V_2$ $(n \in \mathbb{N})$ and $V_2^n \neq V_1$ $(n \in \mathbb{N})$ hold.

We get some set

$$\mathfrak{S}_8 = \bigcup_{\beta_1 \in [0,2\pi)} \{(\langle V_1, V_2\rangle, \cdot) | \{V_1, V_2\} \in \mathfrak{S}'(\beta_1)\} \cup$$

$$\cup \bigcup_{\beta_1 \in [0,2\pi)} \{(\langle V_1, V_2\rangle, \cdot) | \{V_1, V_2\} \in \mathfrak{S}''(\beta_1)\}$$

of finitely generated non-cyclic commutative semigroups.

Summarizing all the above, we conclude that the following theorem is true:

Theorem 11. *The following inclusion holds:* $\mathfrak{S} \supseteq \bigcup_{j=1}^{8} \mathfrak{S}_j$.

Unfortunately, it is still unknown, if the identity $\mathfrak{S} = \bigcup_{j=1}^{8} \mathfrak{S}_j$ holds.

5 Conclusions

In the given paper some research in two new trends of FA theory has been presented.

The first trend deals with investigation of FA families defined on algebraic structures over finite rings. The presented results justify some feasibility for using these families in resolving problems of information protection. Based on this viewpoint, the following further research can be pointed. Firstly, searching non-trivial FA families for which any asymptotically accurate simulation model is much more complicated than a system of equations defining the family itself. Secondly, characterization of families of reversible FA for which transition to any simulation model results in essential loss of accuracy. Thirdly, detailed investigation into computational security of specific families of hash-functions determined by outputless automata over finite rings. Fourthly, detailed investigation into computational security of FA families defined on elliptic curves over finite fields.

The second trend deals with investigation of languages accepted by QFA models under supposition that unitary operators associated with input alphabet commute each with the others. In this direction, some progress in investigation of 1-qubit QFA have been achieved. However, no similar results are known for l-qubit QFA ($l \geq 2$). Possibly, the reason is that no visual geometric model which is similar to Bloch sphere is known for $l \geq 2$. Characterization of l-qubit QFA ($l \geq 2$) under supposition that unitary operators associated with input alphabet commute each with the others forms some trend for future research.

References

[1] A.M. Turing *On computable numbers, with an application to the Entscheidungsproblem.* Proc. London Math. Soc., ser. 2, vol. 42 (1936), pp. 230–265.

[2] *Automata studies* (Ed. by C.E. Shannon, J. McCarthy). Princeton University Press, 1956.

[3] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone. *Handbook of applied cryptography.* CRC Press, 2001.

[4] J. Kaz, Y. Lindell. *Introduction to modern cryptography.* CRC Press, 2007.

[5] M.A. Nielsen, I.L. Chuang. *Quantum computation and quantum information.* Cambrige University Press, 2010.

[6] C.P. Williams. *Explorations in quantum computing.* Springer-Verlag London Limited, 2011.

[7] V.M. Glushkov. *Synthesis of digital automata.* Moskow, Nauka, 1962. [in Russian]

[8] Z. Kohavi. *Switching and finite automata theory.* New York, McGraw-Hill, 1970.

[9] B.A. Trachtenbrot, Y.M. Barzdin. *Finite automata. Behavior and synthesis.* North-Holland, 1973.

[10] A.A. Letichevskii. *Completeness conditions for finite automata.* USSR Computational Mathematics and Mathematical Physics, vol. 1, issue 3 (1962), pp. 829–840.

[11] M.I. Kratko. *Undecidability of completeness for finite automata.* Doklady AN SSSR, vol. 155, No 1 (1964), pp. 35–37. [in Russian]

[12] A. Gill. *Introduction to the theory of finite-state machines.* New York, McGraw-Hill, 1962.

[13] V.M. Glushkov. *The abstract theory of automata.* Russian Mathematical Surveys, vol. 16, No 5 (1961) , pp.1–53.

[14] S. Eilenberg. *Automata, languages and machines. Vol. A.* New York, Academic Press, 1974.

[15] S. Eilenberg. *Automata, languages and machines. Vol. B.* New York, Academic Press, 1976.

[16] V.M. Glushkov, G.E. Tseitlin, E.L. Yushchenko. *Algebra, languages, programming.* Kiev, Naukova Dumka, 1978. [in Russian]

[17] D.A. Huffman. *Canonical forms for information-lossless finite state logical machines.* IRE Transactions Circuit Theory. Special Supplement, vol. CT-6 (1959), pp. 41–59.

[18] S. Even. *On information-lossless automata of finite order.* IEEE Transactions on Electronic Computers, vol. C-14, 4 (1965), pp. 561–569.

[19] M.O. Rabin *Probabilistic automata.* Information and Control, No 3 (1963), pp. 230–245.

[20] A. Paz A. *Introduction to probabilistic automata.* New York, Academic Press, 1971.

[21] J.G. Kemeny, T.L. Snell. *Finite Markov chains.* Princeton, NJ: D. Van Nostrand, 1960.

[22] E.R. Berlekamp. *Algebraic coding theory.* New York, McGraw-Hill, 1968.

[23] W.W. Peterson, E.J. Weldon, Jr. *Error-correcting codes.* The M.I.T. Press, Cambridge, MA, 1972.

[24] L.A. Zadeh, C.A. Desoer. *Linear system theory.* New York, McGraw-Hill, 1963.

[25] A. Gill. *Linear sequential circuits – analysis, synthesis, and applications.* New York, McGraw-Hill, 1966.

[26] B.A. Sevastyanov, V.P. Chistyakov. *On the number of input sequences corresponding to the output sequences of a finite automaton.* Review of Applied and Industrial Mathematics, vol. 1, Moskow, TVP (1994), pp. 96–107. [in Russian]

[27] V.L. Kurakin, A.S. Kuz'min, A.A. Nechaev. *Pseudo-random and polylinear sequences.* Memoires in Discrete Mathematics, vol. 1, Moskow, TVP (1997), pp. 139–202. [in Russian]

[28] V.L. Kurakin, A.S. Kuz'min, A.A. Nechaev. *Properties of linear and polylinear recurrencies over Galois rings (I).* Memoires in Discrete Mathematics, vol. 2, Moskow, TVP (1998), pp. 191–222. [in Russian]

[29] D.V. Speransky. *Experiments with linear and bilinear finite automata.* Saratov, Saratov State University, 2004. [in Russian]

[30] A.V. Babash. *Approximate models for finite automata.* Review of Applied and Industrial Mathematics, vol. 12 (2005), pp. 108–117. [in Russian]

[31] N. Courtois, W. Meier. *Algebraic attack on stream ciphers with linear feedback.* LNCS, vol. 2656 (2003), pp. 345–349.

[32] V.N. Trenkaev, R.G. Kolesnikov. *Automata approach to attack on symmetric ciphers.* Bulletin of Tomsk State University. Appendix, No 23, (2007), pp. 130–135.

[33] V.V. Skobelev, V.G. Skobelev. *Ciphersystems analysis.* Donetsk, IAMM of NASU, 2009. [in Russian].

[34] V.V. Skobelev, N.M. Glazunov, V.G. Skobelev. *Varieties over rings. Theory and applications.* Donetsk, IAMM of NASU, 2011. [in Russian].

[35] C. Moore, J. Crutchfield. *Quantum automata and quantum grammars.* Theor. Comput. Sci., vol. 237 (2000), pp. 257–306.

[36] A. Ambainis, M. Beaudry, M. Golovkins, at al. *Algebraic results on quantum automata.* LNCS, vol. 2996 (2004), pp. 93–104.

[37] A. Belovs, A. Rosmanis A., J. Smotrovs. *Multi-letter reversible and quantum finite automata.* LNCS, vol. 4588 (2007), pp. 60–71.

[38] V.G. Skobelev. *Analysis of finite 1-qubit quantum automata unitary operators of which are rotations.* Visn., Ser. Fiz.-Mat. Nauky, Kyïv. Univ. Im. Tarasa Shevchenka, No. 2 (2014), pp. 194–201.

[39] V.V. Skobelev. *Simulation of automata over a finite ring by the automata with a finite memory.* Journal of Automation and Information Sciences, vol. 44, issue 5 (2012), pp. 57–66.

[40] V.V. Skobelev. *Analysis of the problem of recognition of automaton over some ring.* Dopov. Nats. Akad. Nauk Ukr., Mat., Pryr., Tekh. Nauky, No 9 (2012), pp. 29–35.

[41] V.V. Skobelev. *Analysis of families of hash functions defined by automata over a finite ring.* Cybern. Syst. Anal., vol. 49, No. 2 (2013), pp. 209–216.

[42] V.V. Skobelev. *Analysis of automata models determined on varieties ovef finite ring.* Journal of Automation and Information Sciences, vol. 45, issue 8 (2013), pp. 21–31.

[43] V.V. Skobelev. *Automata on algebraic structures.* Donetsk, IAMM of NASU, 2013. [in Russian]

[44] V.G. Skobelev. *Quantum automata with operators that commutes.* Visn., Ser. Fiz.-Mat. Nauky, Kyïv. Univ. Im. Tarasa Shevchenka, Special Issue (2013), pp. 34–41.

[45] V.G. Skobelev. *On the structure of the set of all finitely generated semigroups of special unitary operators in the space* $\mathbb{C}^2$. Visn., Ser. Fiz.-Mat. Nauky, Kyïv. Univ. Im. Tarasa Shevchenka, No. 3 (2014), pp. 182–187.

Volodymyr V. Skobelev, Volodymyr G. Skobelev Received September 16 , 2015

Volodymyr V. Skobelev
V.M. Glushkov Institute of Cybernetics of NAS of Ukraine
40 Glushkova ave., Kyiv, Ukraine, 03187
Phone: +38 063 431 86 05
E-mail: `vvskobelev@incyb.kiev.ua`

Volodymyr G. Skobelev
V.M. Glushkov Institute of Cybernetics of NAS of Ukraine
40 Glushkova ave., Kyiv, Ukraine, 03187
Phone: +38 063 431 86 05
E-mail: `skobelevvg@mail.ru`

Communicative automata based programming. Society Framework*

Andrei Micu, Adrian Iftene

Abstract

One of the aims of this paper is to present a new programming paradigm based on the new paradigms intensively used in IT industry. Implementation of these techniques can improve the quality of code through modularization, not only in terms of entities used by a program, but also in terms of states in which they pass. Another aspect followed in this paper takes into account that in the development of software applications, the transition from the design to the source code is a very expensive step in terms of effort and time spent. Diagrams can hide very important details for simplicity of understanding, which can lead to incorrect or incomplete implementations. To improve this process communicative automaton based programming comes with an intermediate step. We will see how it goes after creating modeling diagrams to communicative automata and then to writing code for each of them. We show how the transition from one step to another is much easier and intuitive.

Keywords: Communicative Automata, XML Automata, Automata transfer, Distributed systems, Traveling code.

1 Introduction

The last decades of evolution for computing machines brought a significant increase in computing power and their diversity. The rise of parallel computing, the important foundations of modern computers,

*This work was supported by MUCKE (Multimedia and User Credibility Knowledge Extraction) project Reference No. 2 CHIST-ERA, three years from 01.10.2012.
* This paper represents an extended version of the paper presented at FOI2015.

has revolutionized the world of software and hardware making it possible to create artificial intelligent systems. Whether it is a desktop, mobile phone, mainframe, or any other computing system, it is able to simultaneously perform a number of tasks that sometimes depend on each other. Their synchronization is essential in most cases and it depends on the states in which the processes or threads are at a time. Synchronization is not easy to achieve if the source code is not structured in terms of states. Sometimes it happens that a seemingly stable code in terms of errors to work as expected for successive runs with the same input data, but at a certain running (with the same input data) to give a wrong result.

Automata have been used since before the beginning of modern computers to solve mathematical problems. Nowadays they have applications in many of the components of a software product such as lexical analyzers, parsers which use regular expressions or network communication protocols [1]. In 2003 Russian scientist Anatoly Shalyto published an article about automata based programming [2]. This paper presents a new way of programming mechanisms for simulation of states, transitions and input/output operations. In designing of large applications state charts and activity diagrams can be used, and they are very similar to automata. The problem arises when you have to translate these diagrams into source code. Shalyto senses this transposition and connects his theory with the association between diagram elements and automata elements.

Communicative automata based programming has elements from the object-oriented version of the Russian researcher and, additionally, it solves the problems mentioned above. It proposes an improved model of the application, dividing the tasks of the control automata in the object-oriented model to several independent machines that communicate with each other. Every communicative automata is self-contained and do not share information, the only way to exchange data is the transmission of messages. So the code of applications benefits from high cohesion without sacrificing coupling and it can be reused easily.

The main strength in the technology based on communicative automata is that the application can be easily distributed across multiple computing machines. Each system has its suite of communicative automata, which communicates with the rest of the automata by the same type of messages; in this case the communication channel is the network. Moreover, these systems can switch automata between them, which do not depend on a particular machine, providing task balancing. This is useful particularly in the client-server model, where the client is a computing machine with low capacities, such as a mobile phone. In this paper we will see how we created the premises for the communicative automata based programming paradigm, which is based on object-oriented programming concepts. This paradigm intensively uses the concept of automaton; code structure is given by the states and transitions. Novelty to classic automata based programming is to treat automata as atomic elements at application-level and the introduction of using transmission of messages between automata. Society Framework implements the basic elements and concepts described by communicative automata based programming. The framework allows creation of native and XML automata, the XML ones having an important advantage because they can be serialized locally and deserialized on another machine at runtime.

2 Communicative automata based programming

2.1 Automata based programming (classical version)

For the first time in software engineering, Shalyto describes an application model composed only from automata. Its technology uses intensive enumerations and switch-case instructions, so that is also called "Switch Technology" [2]. In this solution, although the code is easy to understand, there are big problems when the number of states increases. This is because the program code increases with each added state and transition. The solution appears later in the paradigm "object-oriented programming based on states" [2]. It combines the advantages of automata for easier understanding of the program behavior and advantages of object-oriented programming for easier understand-

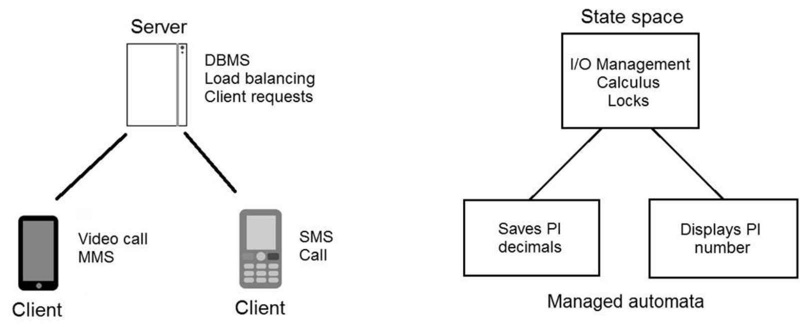


Figure 1. Comparison between client-server model and Shalyto model

ing of the structure. The new technique shifts from "switch-case" instructions to classes and objects to describe automata, thus avoiding nested switches. Shalyto takes in discussion the existence of several automata in one application. Their management is difficult when you have to synchronize certain transitions or when performing operations of reading/writing from the same memory location. Therefore the notion of "space of states" [2] arises, a set of conditions designed to control objects. Knowing which states control objects, we can program automata to have synchronous access to memory. Moreover, if we consider one object for each automaton, the space of states may act as a supervisor for the other automata. Thus, the application model begins to look like client-server model, as in Figure 1.

Another novelty in this technique is the capacity of system to respond to events, a necessary feature for communication between state space and the other automata. So, not only I/O operations can change the state of automata, but also other automata by generating its own events. Object-oriented programming based on states solves many of the initial problems presented by Shalyto, but even this option cannot be used in large projects. The main reason would be that as the code grows in size, it becomes more difficult to extend. State space must be rewritten for each new added automaton and becomes more and more complex, and harder to understand.

2.2 Communicative automata based programming

Programming based on interconnected automata expands object-oriented programming, inheriting all its elements and rules. On top of them there is the added notion of communicative automaton, with new rules related to its functionality. The major differences between the automata from object-oriented version and communicative automata are their atomic characteristic and the communication based on messages. In what follows, an automaton is a finite automaton with epsilon-transitions, without isolated states, with one or more final states and a single initial state [3].

Communicative automata bring new elements compared to the classical automata, leading to changing application architecture. The main elements are: (1) **State** – a series of instructions viewed as an atomic part. It contains code that performs the actual work of the automaton; (2) **Transition** – a series of instructions viewed as an atomic part. In contrast with states, transitions contain only the code needed to determine the next state where it will pass; (3) **Message** – an entity that contains data transmitted by an automaton to another automaton or by a code that is not part of an automaton; (4) **List of messages** – a comprehensive list of received messages by automaton.

A communicative automaton may contain, in addition to the base elements, other resources such as variables, operating system resources or references/pointers to other automata. In most cases, when a system runs more communicative automata, it is desirable to execute their code in parallel. The general solution in modern systems is to run the code for each automaton on a separate thread. Some programming languages such as JavaScript before HTML5 [4, 5], do not support working with multiple threads. If the parallel execution is not possible, the programmer will have to use an own method of allocation and arbitration of automata to the processor. This approach has an important advantage in that the programmer can choose the convenient moments when to deallocate an automaton to ensure a consistent state at each step of the execution. This approach has a disadvantage, though. At any point in execution it cannot run more than one automaton, so others have to wait.

Each automaton must provide ways to add messages to its message list. The problem occurs when an automaton should reference another automaton, to which it must send a message. A naive way to solve this problem is to keep a reference/pointer to every possible destination, which is set by the function that creates it. Such practices, however, are extremely hard to maintain since it requires changing the code of the function that instantiates automata each time when we add a new type of automaton. A better approach is to mediate communication with an object that keeps track of automata. This object, called "router", has an implementation similar with the Observer pattern. Automata must register using a unique identifier to this mediator and must be able to handle messages from any source, automaton or not. The router methods can send messages based on recipient identifier, thus obtaining a total decoupling of automata in the system. When the router handles messages sent through the network, security problems may appear which must be taken into account. A security system must provide a separation between user automata to avoid situations in which an automaton sends a compromising message to another automaton.

3 Society framework

3.1 Architecture

Society Framework is a project developed to demonstrate the advantages of communicative automata based programming. Its source code and examples are publicly available on https://code.google.com/p/soci ety-framework/. Its target is to facilitate the development of communicative automata based applications and to minimize the errors that can happen in such an application. There are two framework implementations, one written in Java language and one written in C# language, the reason being the demonstration of its interoperability.

The three major modules of this framework are the following: (1) **Base module for communicative automata** – contains interfaces,

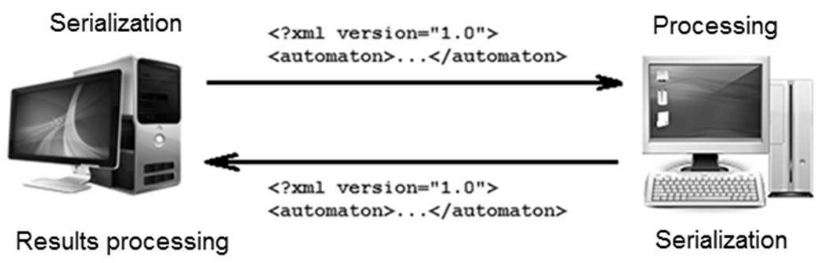


Figure 2. An efficient use of a communicative automaton

abstract classes and completely implemented classes to create a native automaton; (2) **XML communicative automata module** – contains interfaces, abstract classes and completely implemented classes to create an XML automaton; (3) **Communication module** – contains classes with role in automata communication, both locally and through the network, on different applications.

In Society the communicative automata are divided in two large categories: native automata, with Java or C# code based on the framework implementation and XML automata, for which the code is written using an extension of XML. The main reason is the fact that, unlike the native automata, the XML ones can be serialized, sent through the network to another application and re-instantiated on that machine, the process being intuitively described in Figure 2. The XML code inside the serialization is transformed in one more object trees representing the instruction that must be executed in the states and the transitions. For this reason we cannot say that the framework compiles the code and neither that it interprets it. It constructs its own code using objects corresponding to the instructions described by the XML.

XML automata can use only a restricted set of instructions, on which the framework can construct XML specifications. The reason is the fact that XML automata are not intended for complex or intense processing due to the great overhead compared to the native ones. Their utility is the fact that they can communicate with the na-

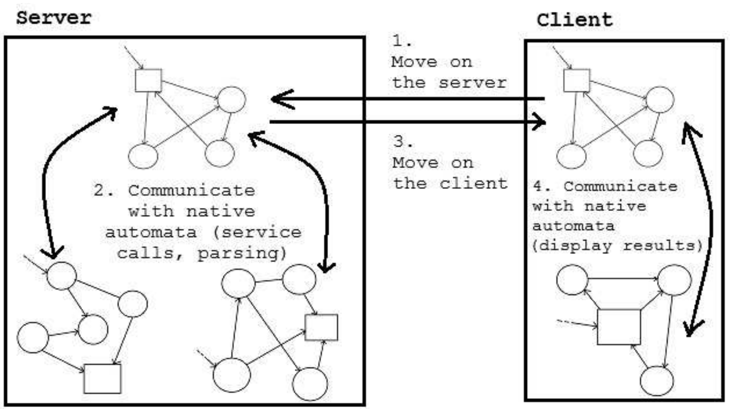

Server
Client
1. Move on the server
3. Move on the client
2. Communicate with native automata (service calls, parsing)
4. Communicate with native automata (display results)

Figure 3. Example of optimization using a native automaton and an XML automaton

tive ones through messages, the latter fulfilling the tasks described in the messages much more efficiently. The alternative usage of the two types enables optimizing the application processing, an example being described in Figure 3.

The efficiency relies on the fact that the only data sent between the server and the client is the automaton serializations. Thus, the transfer of data between server and client is minimized and it is replaced by service calls, the result being a constant number of connections to the server for executing a task. Fortunately an XML automaton plays the role of manager for the native automata in the system and they don't require much code, so traveling to another machine through network doesn't imply a lot of data transfer. Another major advantage in this approach is the reduced overhead of the native automata because, as we previously mentioned, the native automata are written directly in the language/platform of the framework implementation. The fact that a native automaton can use any instruction of this type raises security problems regarding the actions permitted to XML automata. XML automata can only execute instructions that don't compromise security, the only way of communicating with the machine it runs on being the

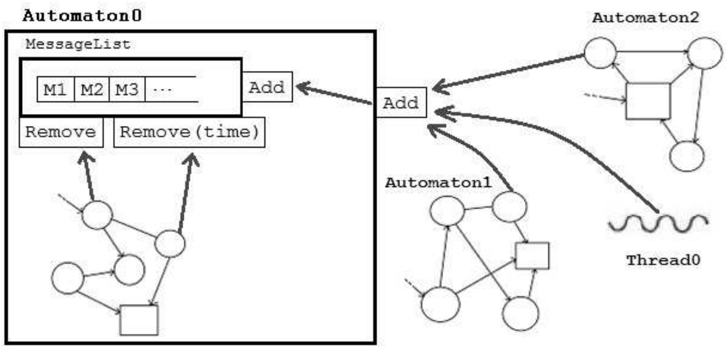


Figure 4. Interaction with the message list (queue)

message sending to other automata.

3.2 Base module for communicative automata

The base module contains the base class for all the automata created in a system (BaseAutomaton) and classes for automaton components. In both framework implementations these are: (1) **State** – the base class for the states in an automaton; (2) **Transition** – the base class for the transitions in an automaton; (3) **TransitionGroup** – class that contains transitions and acts like a normal transition; (4) **MessageList** – the class that implements the queue of messages received by an automaton.

The State, Transition and TransitionGroup classes are classes inside the BaseAutomaton class. The reason is the fact that these classes must have access to the fields and the methods in BaseAutomaton.

The MessageList class implements a special type of queue with synchronized add and remove methods (see Figure 4). The add operation is synchronized to ensure consistency to the list while more than one execution threads add messages simultaneously. The remove operation is synchronized to block the thread that calls it when there are no messages in the queue until another thread places a message in it.

The message queue is encapsulated in the automaton, the only permitted operation being the message addition by using the Add method at automaton. Inside the State, Transition and TransitionGroup classes there is direct access to the MessageList object, meaning that they can both add and remove messages.

Each state contains a transition or a transition group which indicates the next state and is kept in an indexed list (HashMap in Java and Dictionary in C#) with string identifiers. To create a new state the State class must be extended and the stateCode (or StateCode in C#) method must be overridden. Then the Transition class must be extended (or TransitionGroup if the state has more than one transitions) and the transitionCode (or TransitionCode in C#) method must be overridden, such that the new implementation returns a valid transition name. The next executed state is decided by the name returned by the transition. Societys communicative automata can run both on the current thread, by calling the run (or Run in C#) method, and on a newly created thread by calling start (or Start in C#). To stop the thread that runs the automaton without risking data corruption the stopSafely (or StopSafely in C#) method can be used. The run method contains a while loop that ends at the stopSafely call. This loop executes states one at a time and the order of execution is driven by each states transition or transition group.

Another important class in the base module is the SocietyManager which manages the automata inventory from the current application and the XML automata transfers. This can be extended to implement the saving and loading methods for the automata. For automata transfer SocietyManager runs a special native automaton called AutomataTransferAutomaton responsible with managing the connections and sending/receiving automata.

3.3 XML communicative automata module

XML communicative automata are an extension of the native ones, their base class being BaseAutomaton. The XML automaton name is given by the serialization and deserialization of this type, which is achieved by using the Society XML, an extension of XML.

The serialization of XML automata contains 6 major elements: (1) **Automaton name** – also named identifier, it is the character sequence attribute of the $< automaton >$ element with role in a possible subscribe of the automaton at the message router; (2) **Current state name** – current state identifier, also stored as a character sequence; (3) **Current message** – the serialization of the last message pulled from the message queue; (4) **Message list** – the serialization of the automaton's message queue; (5) **Variables** – the list of variables and their values; (6) **States** – the list of states in the automaton and their code.

The current state name must match the name of a state in the state list. If this rule is violated, then the automaton cannot start. Also, if the name which was provided for the automaton serialization differs from the name it had in the system before, when the automaton is re-instantiated, all entities which send messages to that automaton must be aware of the change and send message for the new name.

Variables are key-value pairs. In the Society framework automata work with 5 data types: Boolean, Integer, Double, String and Map. The Boolean type is represented using characters delimited by dots: .T. for true and .F. for false. Integer and Double types have the same representation as in Java or C#. String type is delimited by apostrophes and it can contain anything exception apostrophe characters. Map type can represent vectors with any number of dimensions; the only limit is the machine memory. To ensure this property, the framework uses a series of indexed lists (the type of list depends by framework implementation). The indexed values can be of any type, including Map type. Thanks to this flexibility we can create vectors that contain other vectors, any number of times and in any combination, the result being as much dimensions as the memory can hold.

The state serialization contains the state identifier (name attribute), the executed code ($< code >$ element) and the transition ($< transition >$ element) or the transition group ($< transition\_group >$ element). A transition group contains any number of transitions and a $< code >$ element which contains the instructions that manage the returned values for each transition. From the moment when the automaton is started, the initial state code is executed indicated by the value in the *current_state* attribute. Then the transition code from that state is executed (or the transition group code if it's a group of transitions) and the next state name is obtained. The process continues until the automaton is stopped, either from a state code, or from an execution thread outside it.

It should be noted the fact that inside a transition group each transition must have a name (specified in the name attribute), so that their code can be called from the $< code >$ element of the group. If a state has just a transition, and not a transition group, then that transition doesn't have to specify a name. The instructions represent the imperative part of Society XML and there are two types of them: (1) **Simple instructions** – instructions that don't contain other instructions inside them (empty elements); (2) **Compound instructions** – instructions that contain other instructions inside them (non-empty elements).

Simple instructions are the base for the code executed in the states and transitions. These are represented by XML elements without content, the only parameters being their attributes. The following simple instructions can be used in Society XML: *get_next_message*, *send_message*, *execute*, *continue*, *break* and *return*. Compound instructions are usually loops (*while*, $do - while$, for), but they can be other types, like the $if - else$ instruction or the $switch - case$. Their behavior is identical with the one in the framework's implementation language, with small differences to enable much more flexibility by using the expressions. The following compound instructions can be used in Society XML: *while*, *do_while*, for, if, *else*, *switch*, *case* and $default$.

Unlike the other elements, where the declaring order does not drive the code behavior, instructions must be written in the exact order in which they must be executed. Regarding the calculability of Society XML, it contains enough instructions to be Turing-complete: at least one assignation operation, one conditional operation and one jump instruction. This means that XML communicative automata can solve any problem which can be transformed in an algorithm. The input and the output are ensured by the router and the message queue inside the automaton. Native automata must provide communication methods with the user for the XML ones because the language of the latter does not allow native calls for reading, writing or displaying data. The advantage is the fact that XML automata don't have any security issues so long as the native ones verify and control the requests. The expressions have an important role in Society XML because they provide values for the attributes in the instructions presented earlier. An expression will always return a value, even if this value is null, and some expressions may even change the state of variables during the execution, like the assign operation or the function call. The expressions may also be used to access the values of the received messages.

XML communicative automata deserialization

The BaseAutomaton class incorporates methods to serialize and deserialize automata. For parsing the XML data the Society framework uses SAXParser in Java and XMLReader in C#. For constructing the objects that compose the XML automaton functionality the framework uses a special automaton called DeserializationAutomaton. For each beginning and ending tag the parser sends a message to the deserialization automaton containing the corresponding data (tag name, attributes, and tag type). According to the state and message data the automaton will create the objects and perform transitions.

The major components in the XML automaton serialization are the following: the variables, the messages, the states, the transitions and the transition groups. When the deserialization automaton encounters an expression inside an attribute it must analyse it and construct an instruction tree equivalent. The constructExpression method inside DeserializationAutomaton has a string representation of the expression as input and an instruction tree as output. It uses an algorithm inspired from the infix to prefix expression transformation algorithm [6]. Instead of constructing the list of prefix ordered symbols the algorithm was modified to construct the instruction tree. To extract the symbols (tokens) from the expression the deserialization automaton uses the LexerAutomaton. The lexer provides these symbols as instructions. The paranthesis (OpenedBracketInstruction and Closed-

BracketInstruction) are also considered instructions, though they only have functionality inside the deserialization process. Because both the expressions and the instructions implement the same interface, Instruction, they are treated in the same way: instructions are executed by calling their code method and, for their part; the instructions call the same method for the expressions they contain. The call of a state or transition is done by only calling their code method, as the other calls are done through call chaining.

The lexer automaton is similar to the automata constructed for regular expressions. Its implementation has a string as input and for each of its runs it provides an Instruction object in the final state. This object corresponds to the next symbol found in the expression string, therefore situations when the lexer automaton must run multiple times on the same expression are often. If the automaton reaches the final state and provides a null value it means no more symbols can be found in the current expression and the constructExpression method returns the created tree. Lexer Automaton holds data about the current parse at each run. These pieces of information are named accumulators and stored in a list. In the final states of the automaton these are used to construct the returned instruction. The first accumulator is always the string of the expression to parse. There are cases when the lexer will also call the constructExpression method to create a subtree of a substring in the expression. An example would be the function parsing: for each substring delimited by parenthesis and commas constructExpression is called to obtain the subtrees corresponding to the parameters.

XML communicative automata serialization

XML automata serialization is a much simpler process thanks to the tree structure of the instructions inside them. The serialization process implies the construction of the XML based on the objects inside the automaton. To achieve this it is necessary to inspect the variables, the message list, the special members (automaton name, current message, etc.) and a single BFS traversal of object trees in the states, transitions and transition groups.

The expression serialization is an exception from the BFS: an expression tree is traversed in-order. The reason for this is the fact that the expressions linear structure requires the left subtree of a node to be written before the operator and the right subtree to be written after the operator. To send serialization through the network, Society framework uses TCP connections which it tries to keep alive during the execution. The non-ASCII characters are encoded using UTF-8. This means that the values and names in an XML automaton can use any character from Unicode.

3.4 The communication module

The communication module includes the classes responsible with sending and routing the messages: (1) **Message** – the class which represents a message; (2) **MessageRouter** – the class responsible with message transfers.

The Message class contains two fields (or properties in C#): from and data. The from field holds the identifier with which the sender automaton has subscribed to the router or any other name if the message was not sent by an automaton. The data field is a reference to the sent object and it can be of any type. The MessageRouter class manages the message transfers for both automata in the same application and automata on different machines. It implements the Observer, Singleton and Lazy Initialization patterns and its unique instance can be accessed anywhere in the code. Automata can subscribe to receive messages using the subscribe(String name, BaseAutomaton automaton) method and they can unsubscribe through the unsubscribe(String name) method. The name parameter will have the value of a unique identifier for that automaton, usually its name. The sendLocal(String to, Message message) method sends the message provided as parameter to an automaton in the current application. It returns whether the automaton was found in the application and, in case it was found, the message is added to its message queue.

To send messages to an automaton outside the application the router uses an automaton which is responsible with the message transfer through the network called NetworkMessagingAutomaton. This looks similar to the DeserializationAutomaton inside the SocietyManager class, the only difference being the type of sent information. The network messaging automaton contains a message queue from which messages are sent starting from the moment when the connection is established with the application in the network. To place a message in this queue the sendRemote(String to, Message message) method is used. The send(String to, Message message) method first tries to send the message locally, then, if the recipient automaton is not found in the current application, it places the message in the NetworkMessagingAutomaton's queue.

4 Comparisons

4.1 Loose code vs. native communicative automata

Loose code means any object-oriented code written without the constraints of the communicative automaton based programming paradigm. By applying these constraints to loose code the native communicative automatons can be obtained, the performance difference being minimal.

To create a native automaton the following steps must be followed: (1) **Extending the BaseAutomaton or Automaton classes** – if the automaton state doesn't have to be persisted, then the Automaton class is used, otherwise the BaseAutomaton class is extended and the serialization/deserialization methods are implemented; (2) **Adding the member variables** – necessary for the automaton functionality; (3) **Creating the nested classes** – corresponding to the states, transitions and transition groups; (4) **Instantiating and adding the previously created classes at the current automaton** – these steps can be made in the automaton constructor.

The code executed in the stateCode and transitionCode methods by the automaton is the imperative (procedural) code corresponding to the language of the framework implementation. The automaton code, as a whole, has a structure enforced by the programming paradigm: it is grouped in states, transitions and transition groups.

The run method (or Run in C#), which was previously mentioned in this article, is responsible with the correct execution of the automaton regarding the order in which the states, the transitions and the transition groups are executed. The management instructions in this method have O(1) complexity, except the operation that searches a state in the state set. After a transition returns an identifier, in the run method, a search for the state corresponding to that identifier is attempted. This implies a get operation in a HashMap (Java) or Dictionary (C#) with a complexity in the worst case scenario of O(n) [7, 8], where n is the length of the identifier hash. Though the complexity in the general case is greater than when using normal vectors, the programmer can minimize the execution time by offering identifiers with a minimum number of characters. The execution time also depends on the hash algorithm on which the search is based (specific for the platform).

While the automata are designed, the programmers and architects must decide how to split the automaton tasks and what are their states and transitions. Fewer automata and states/transitions means less allocated memory and less time consumed by the context switches between automaton threads. This approach implies more code written per state or transition, therefore the imperative part of the automata is favored. On the other hand, more automata and states/transitions mean a better modularization of the code and the possibility to allocate the tasks to a greater number of processors or machines, favoring the declarative part of the automaton.

4.2 Native communicative automata vs. XML communicative automata

The XML communicative automata, as mentioned in the previous chapters, are an extension of the native communicative automata. Their states, transitions and transition groups are constructed in a certain manner to ensure they can be serialized and deserialized using the Society XML language. The code inside the XML automaton is composed of an object tree and the objects are implementing the Instruction interface. Executing its code means calling the code method of the root object which will trigger directly or indirectly the call of code in the other objects from the tree.

The overhead compared to the native ones is visibly greater because each instruction implies a method call at the level of implementation. Starting from the moment the automaton tries to assign a value to a Map object at an inexistent level, the algorithm creates the necessary levels based on the indexes from the left operand. If on one of the levels that must be created there is an object of a type different from Map, then it is replaced by a new Map object.

In the native automata the variable access has O(1) complexity thanks to the fact that they are direct members of the automaton class. XML automata keep all the variables in an indexed list, the same way the states are stored, therefore the access algorithm complexity is O(n), where n is the length of the variable name hash [7, 8].

When comparing XML automata and native automata it can be inferred that the native ones must be used for intense processing and system calls and XML ones must be used for the business logic, control and code that must be transferred between applications. This way we can take advantage of both without sacrificing execution time or code modularity. The connection between the two types of automata is the messaging which ensures a uniform communication. Because Society framework was written using just the base platform for each language it was implemented in, there are no additional dependencies for it to run. An advantage of this decision is the ease of extending the framework for the Android platform. In Android the Java code runs on a special virtual machine called Dalvik [9]. To run the intermediary Java code (Java byte-code) it is transformed into intermediary Dalvik code (Dalvik byte-code) for optimizations [10]. This way the Android extension for the Society framework was created which contains specialized classes for that environment in the society.framework.android package. The ActivityAutomaton class is an extension of the Automaton class which contains a reference to the current Activity in the application instance. The reference to the Activity is necessary for some actions on the application resources: user interface changes, system resources usage, service starting, etc. The AndroidSocietyManager class extends the SocietyManager class and implements the serialization and deserialization methods for saving and loading the automata state from a persistent environment.

5 Conclusions

Communicative automata based programming makes the process of moving from the design to the implementation easier and the state or the activity diagrams to be found directly in the source code, without the need of a detailed documentation. The paradigm combines both imperative programming elements and declarative elements for obtaining a higher quality code with less effort. The new features it brings on top of the classic automata based programming, automaton atomicity and messaging communication, enforce practical rules with an aim for minimizing the errors: code modularization, data encapsulation at automaton level and request verification. These advantages have a great impact in production, especially in large projects where the work is assigned to a great number of programmers and architects.

Society framework has reached its purpose: the demonstration of the advantages brought by the communicative automata based programming. The differentiation between native automata and XML automata resulted in the development of two categories of automata, each with its advantages and disadvantages. The native ones are intended to provide methods to access the machine resources in a controlled and efficient manner and the XML ones have the role to use these functionalities, to execute the business logic and to travel in the network at the most suitable place for a certain task. The alternative and efficient use of those has results superior to the current approaches for some problems: minimizing the number of connections and the amount of data sent through the network, efficient execution of a distributed application or providing universal processing services on a powerful machine.

The Society framework, though it is not the most efficient implementation of the mechanisms in the communicative automata based programming, it draws closer to the industry needs. Large distributed applications or the ones that intensively use network transfers can be boosted by Society framework. Nonetheless, based on the application necessities, different mechanisms can be implemented for the automata. The target would be code optimization, security (message encryption, authentication, error tolerance, etc.) or the sending of messages through other environments (embedded systems, Bluetooth).

References

[1] E. Gribko. *Applications of Deterministic Finite Automata.* (2013).

[2] A. Shalyto. it Technology of Automata-Based Programming. (2004).

[3] J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Addison-Wesley, (2000).

[4] J. Edwards. *Multi-threading in JavaScript.* (2012).

[5] R. Gravelle. *Introducing HTML 5 Web Workers: Bringing Multi-threading to JavaScript.* (2012).

[6] S. Singhal. *Infix to Prefix Conversion. Sharing ideas, Sharing experiences.* (2012).

[7] Arno. *HashMap vs. TreeMap.* (2010).

[8] K. Normark. *Generic Dictionaries in C#.* (2010).

[9] J. Hildenbrand. *Android A to Z: What is Dalvik.* (2012).

[10] Security Engineering Research Group, Institute of Management SciencesPeshawar, Pakistan, *Analysis of Dalvik Virtual Machine and Class Path Library.* November (2009).

Andrei Micu, Adrian Iftene,

Received September 20, 2015

Andrei Micu, Adrian Iftene
Institution: "Alexandru Ioan Cuza" University
Address: General Berthelot, No. 16
Phone: 004 - 0232 - 2011549
E–mail: `andrei.micu@info.uaic.ro, adiftene@info.uaic.ro`

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

The Law of Gravitation for Ontologies and Domains of Discourse*

Vadim Ermolayev

Abstract

The idea of the presented approach is to borrow a plausible analogy of a "system law"[1] from the field of Dynamics in Mechanics – the Newton's Law of Universal Gravitation. This analogy is exploited for building the law of gravitation in dynamic systems comprising a Domain of Discourse and knowledge representations (ontologies) describing this domain. As ontology elements do not possess physical mass, this component of the gravitation law is substituted by the property of *fitness* of an ontology to the requirements of the knowledge stakeholders characteristic for the described domain. It is also argued that the implementation of the developed theoretical framework is feasible as the supporting techniques, including some software tools, already exist. As the examples of the relevant component methods and tools, the paper presents concisely the OntoElect methodology, Ontology Difference Visualizer, and Structural Difference Discovery Engine. These instruments help solve some practical problems in eliciting domain requirements, developing structural contexts for

*This paper is a revised and extended presentation of the substance of the invited talk [1] given at the 2015 Workshop on Foundations of Informatics at Chisinau on the 25th of August, 2015.

[1]As remarked in [2], a system law is a rule which generalizes the behavior of some observed phenomenon within a concrete system and its given spatiotemporal context. A system law tells what behavior is expected within the system. Thus a system's law can cause change or represent a barrier to change. It can be used to predict certain aspects of the system behavior, which are based on the force, or influence it exerts on the internal environment of the system. In contrast to a natural law, a system law is neither universal nor does it need to be true, correct, etc.

the requirements, generating the mappings between these structural contexts and the target ontology, computing increments and decrements of ontology fitness based on these mappings. It is concluded that the presented framework has prospects to be applied practically for visualization and analysis of ontology changes in dynamics. Use cases for ontology refinement and anomaly detection are suggested for validation.

Keywords: Ontology, Domain, Dynamics, Gravitation, Fitness

1 Introduction

The world of knowledge representations, comprising ontologies, is by its nature a reflection of the world we live in. Dynamics in physical, social, biological contexts are the subject of study by several disciplines, where useful analogies can be sought. The findings hint about a way to identify and specify useful aspects and help offer the law to describe dynamics in ontological systems.

It is known for example from Mechanics, the branch of Physics and Engineering, that Kinematics studies the motion of an object without direct reference to the causes of this motion. Motion in this context is understood as a change of position, often compared to a reference point. In difference to Kinematics, Dynamics is concerned with forces and torques and their effect on the motion of objects. For example, in Dynamics it is analyzed why an object changes its position and due to which causes or influences the acceleration has this specific value function over time.

One of the particular kinds of forces of interest regarding a physical system is gravitation. Basically, gravitation forces are known to be expressed by the Newton's Law of Universal Gravitation [3] as proportional to the product of interacting masses and inverse to the square distance between these masses. In biological and social systems similar "forces" reflect the degree of "attraction" of a particular object to a group, habitat, etc. For knowledge representations, an analogy to the notions of mass, gravitation, force could be sought in terms of the fitness of a knowledge representation module to the requirements of the stakeholders in a Domain of Discourse or its similarity to the other modules which could be found regarding the Domain of Discourse.

This paper starts with the discussion of the notion of an ontology – one of the fundamental concepts in Knowledge Representation and Management. In this context, the property of being a “shared conceptualization” is explained in terms of the fitness to the requirements of the domain knowledge stakeholders, resulting in their commitment. The paper continues with an outline of the state of the play in the field of Ontology Change, putting a particular emphasis on ontology Dynamics versus Kinematics. Then, the fundamentals of the theory of Ontology Dynamics based on the analogy to the Newton’s Law of Universal Gravitation are presented. Yet further, the paper deliberates about the techniques for implementing this theoretical ontology gravitation framework. The paper concludes with the summary of the presented work and outlines the potential applications of the presented framework in Ontology Refinement and Anomaly Detection.

2 Ontologies, Domain Requirements, Fitness, and Dynamics

An ontology is often denoted as a “formal, explicit specification of a shared conceptualization” (c.f. [4]) and this paper follows this definition. In particular it is focused on describing and exploiting the properties of being “formal” and “explicit” regarding the representation of a conceptualization (specification), and – even more importantly – the property of being “shared” regarding the conceptualization itself. It is also emphasized that the completeness of an ontology has a straightforward impact on becoming a “shared conceptualization”

Being “formal” means that an ontology has to be specified using a formally defined ontology specification language such that logical inference is enabled with respect to this artifact. To enable logical inference, such a language needs to be based on logics – so an ontology is a logical theory. Ontology is also a descriptive theory as it is developed with the purpose to describe common sense, abstract high-level notions, or a Domain of Discourse.

Following [5], an ontology is a logical descriptive theory formally denoted as a tuple $O = \langle C, P, I, T, V, \leq, \perp, \in, = \rangle$, where C is the set of *concepts* (or *classes*); P is the set of *properties* (*object* and *datatype properties*); I is the set of *individuals* (or *instances*); T is the set of *datatypes*; V is the set of *values*; $\leq$ is a *reflexive*, *anti-symmetric*, and *transitive relation* on $(C \times C) \cup (P \times P) \cup (T \times T)$ called *specialization*, that helps form partial orders on C and P called *concept hierarchy* and *property hierarchy* respectively; $\perp$ is an *irreflexive* and *symmetric relation* on $(C \times C) \cup (P \times P) \cup (T \times T)$ called *exclusion*; $\in$ is a relation over $(I \times C) \cup (V \times P)$ called *instantiation*; $=$ is a relation over $I \times P \times (I \cup V)$ called *assignment.* The sets C, P, I, T, V are pairwise disjoint. It is also assumed (c.f. [6]), that an ontology O comprises its schema S and the assertional part A:

$$O = \langle S, A \rangle ; S = \langle C, P, T \rangle ; A = \langle I, V \rangle . \tag{1}$$

Ontology schema S is also referred to as a terminological component (TBox). It contains the statements describing the concepts of O, the properties of those concepts, and the axioms over the schema constituents. The set of individuals A, also referred to as assertional component (ABox), is the set of the ground statements about the individuals and their attribution to the schema – i.e. where these individuals belong.

This paper focuses on the ontologies that describe a particular well circumscribed Domain of Discourse – classified as domain ontologies. The reason for this emphasis is that any ontology development process, including its change management or refinement, takes as an input the requirements by the subject experts in the domain of interest and produces the ontology as its output – covering those requirements correctly and to the maximal possible extent. Straightforwardly, the set of methods shaping out this process needs to comprise the mechanisms for:

- Eliciting the (change[2]) requirements from the domain knowledge stakeholders as fully as possible

[2]Change requirements are elicited in the ontology Refinement phase. In the phase of Initial Development initial requirements are collected.

- Measuring how completely the requirements were captured
- Transforming the elicited requirements to the (changes in the) ontology
- Measuring how well the result fits to the intentions of the domain knowledge stakeholders

If a methodology fails to do any of the above sufficiently well, then the commitment of the knowledge stakeholders to the output ontology will be low. So, such a product cannot be regarded as a really "shared conceptualization".

Hence, a domain ontology O_D could be regarded as a harmonized formal and explicit representation of the union of the interpretations (K) by the knowledge stakeholders $s_i \in S$ of the subject domain D. So, naïvely, we may elicit all the K-s and build the ontology of those as:

$$O_D = hrm(\bigcup_S unf_{f_j}(K_{s_i})), \tag{2}$$

where hrm is a harmonization function and unf is the transformation that maps a knowledge interpretation represented in the form f_j to the knowledge representation formalism used by the knowledge engineer (unification). Even if so, harmonization and unification functions are not easy to perform. For example, a formalism f_j for K_{s_i} could be more expressive than the ontology specification language used for coding O_D; K_{s_m} and K_{s_n} could be mutually contradictory in some parts; etc. Reality introduces more complications – mainly influencing the properties of being explicit and complete:

- K-s are **subjective**. The stakeholders interpret their domain based on their individual background knowledge and experience.
- K-s are **tacit**. The views on the domain by the subject experts are often not stated explicitly. On the contrary, some parts of those K-s are assumed, taken as evident or default, subsuming that (all) the professional community regards these assumptions

in a similar way. The tacit parts are the cause for difference in interpretations, or even misinterpretations.

- K-s are **partial**. Subject experts focus on their narrow context of professional interest and expertise, and have only a shallow coverage of the broader area within the domain. The partiality and fragmentation of their K-s is the reason for (a) contradictions between different views on the overlapping contexts; and (b) gaps in the coverage of the domain.
- K-s are **not available**. The knowledge stakeholders are not readily willing to spend their time for materializing their K-s or revealing them to knowledge engineers in another form.

In Ontology Engineering and Management the degree of the conformance of an ontology to the requirements of the domain knowledge stakeholders is regarded as its *fitness*. Measuring ontology fitness is not an easy task as one has to have: the requirements; the ontology; these two compared and difference measured. Several approaches to ontology fitness measurement are known from the literature – e.g. [7, 8]. One of these approaches has been developed as a part of the OntoElect ontology engineering methodology [9]. In OntoElect, ontology fitness to domain stakeholder requirements is understood as proportional to the ratio of positive and negative votes of these stakeholders regarding the assessed ontology. These votes are collected indirectly [9], as for example in [10], by:

- Extracting a saturated set of multi-word key terms from the statistically representative document corpus
- Detecting the most influential key terms by applying weights to the most “important” documents in the corpus
- Transforming the natural language definitions of the selected key terms to formalized structural contexts in the ontology specification language; and
- Mapping the structural contexts to the ontology

Ontologies describing realistic domains could be substantially large and complex in their structures and properties. So, the development and management of these descriptive theories call for solving several interesting research problems. As profoundly surveyed in [11], ontology change – changing an ontology in response to a certain need – is one of the most important and challenging among them. Ontology Change as a field remains to be on the research and development agenda. For example, a Google Scholar search for "Ontology Dynamics" OR "Ontology Evolution" OR "Ontology Change" yields over 5 300 papers[3]. If the search is constrained by those published after 01 January 2015 it returns 224 hits.

The term of ontology change is often used broadly – to cover several interrelated facets of the problem and covering different kinds of changes to ontologies: in response to external events; caused by translations to a different language having different expressive power; caused by the evolution of stakeholder requirements; introduced by the ontology engineer according to the evolved understanding of the domain; etc. Several research sub-fields have emerged to cope with this broad variety of change aspects. The most prominent of those are:

- Ontology **evolution** (reactive response to a change in the domain or its conceptualization)
- Ontology **refinement** (goal-directed, proactive change)
- Ontology **versioning** (enable transparent access to different versions of an ontology)
- Ontology **mapping** (identify related vocabulary elements)
- Ontology **morphing** (map between vocabularies and axioms)
- Ontology **matching** (map and measure semantic distance between vocabularies and axioms)
- Ontology **alignment** (result of matching process)

[3]As of September 2, 2015

- Ontology **translation** (to a different representation language)
- Ontology **integration/ merging** (fuse knowledge from ontologies covering similar/ identical domains)
- Ontology **debugging** – **diagnosis** and **repair** (render an ontology consistent/coherent)

The plethora of these research facets, all looking at the phenomenon of change in ontologies, gave also the birth to the Ontology Dynamics community (http://ontologydynamics.org/). It may be noticed however, that the mainstream approach, also adopted by the aforementioned community, follows more Kinematics than Dynamics. Indeed, the term of "ontology change" is referred to "the problem of deciding the modifications to perform upon an ontology in response to a certain need for change as well as the implementation of these modifications and the management of their effects in depending data, services, applications, agents or other elements" (c.f.[11]). In simple words: given the need for a change, it is decided what is changed and to what extent – i.e. if following the analogy with Mechanics, how much the position, velocity, acceleration of the object changes.

It appears that the Ontology Change does not look sufficiently deeply into the causes of a change – which is in fact the task for Ontology Dynamics. In this paper some steps are made toward laying out a foundation for filling this gap based on analyzing the (changes in the) fitness of an ontology to a particular Domain of Discource.

3 Ontology Dynamics and the Law of Gravitation

Let us now think of a system, comprising a Domain of Discourse and several ontologies describing it, as of a closed "mechanical" system. For making this analogy plausible – i.e. to be able to propose usable dynamic laws – we have to find the proper analogies to the mechanical notions of: a coordinate grid and its origin; a position, a distance, a motion; a mass; and a force (gravitation).

Let us assume that a Domain of Discourse (D) is adequately modeled by the set of all relevant requirements (R), by its knowledge stakeholders, for representing knowledge in this domain. For building a grid based on these requirements it is assumed, as pictured in Fig. 1a, that:

- All the requirements are placed in the centre of the D; and
- They are not equal in their importance – i.e. have different spheres of influence around the centre of gravitation, which is quantified using normalized scores $ns \in [0, 1]$

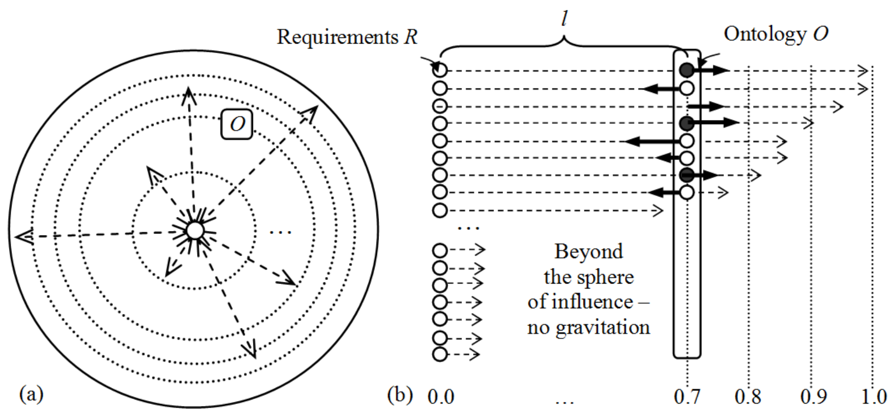


Figure 1. Domain requirements, their spheres of influence (a), and gravitation forces (b)

Let us suppose now that an ontology (O) is positioned in D at a (semantic) distance l from its centre (Fig. 1(b)). This can be any location on the circle of radius l around the centre of the grid. We are now interested in what might be the forces influencing O in this position.

Let us assume that O is checked against the requirements r from R which spheres of influence reach the position of O (i.e. $ns_r \geq l$). The following are the possible outcomes of these checks:

- A particular part of O, say a semantic context $o \in O$ (a white coloured circle in Fig. 1(b)), fulfils the requirement r. Therefore O becomes more fitting to R. In this case we will consider that the increase in fitness ($\Delta\Phi_o^+$) creates a positive gravitation force $\overrightarrow{G_o^+}$ applied to O and directed towards the centre of D, as pictured in Fig. 1(b). The absolute value of this force is computed using a direct analogy with the Newton's Law of Universal Gravitation [3]:

$$G_o^+ = \frac{1 \times \Delta\Phi_o^+}{(ns_r)^2}, \tag{3}$$

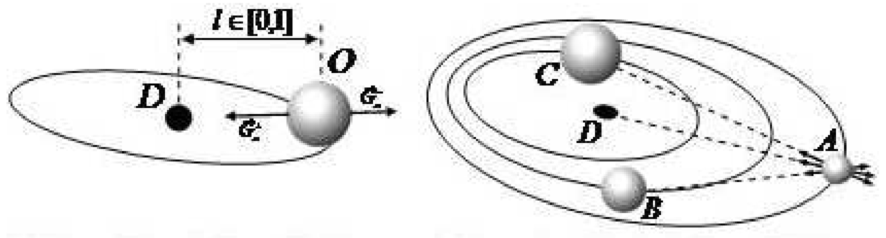
  where: "1" in the numerator is the fitness of r with respect to D – meaning that r fits D perfectly as one of its requirements; the value of $\Delta\Phi_o^+$ is within $[0, 1]$.

- There is no semantic context $o \in O$ that fulfills the requirement r (no circle on the ontology side in Fig. 1(b)) or there is an o that contradicts r (a black coloured circle in Fig. 1(b)). In both cases O becomes less fitting to R. Therefore we will consider that the decrease in fitness ($\Delta\Phi_O^-$ for a missing semantic context; $\Delta\Phi_o^-$ for a context contradictory to r) creates a negative gravitation force, $\overrightarrow{G_O^-}$ or $\overrightarrow{G_o^-}$ respectively, applied to O and directed towards the periphery of D, as pictured in Fig. 1(b). Similarly to (3), the absolute values of these forces are computed as:

$$\begin{aligned} G_O^- &= \frac{1 \times \Delta\Phi_O^-}{(ns_r)^2}, \\ G_o^- &= \frac{1 \times \Delta\Phi_o^-}{(ns_r)^2}. \end{aligned} \tag{4}$$

The overall gravitation force applied to O as an influence by D is computed as a vector sum:

$$\overrightarrow{G_O}\Big|_D = \sum_{r\in R:ns_r\geq l}\left(\overrightarrow{G_o^+} + \overrightarrow{G_O^=} + \overrightarrow{G_o^-}\right). \tag{5}$$



(a) One ontology in the gravitation field of *D*

(b) Several ontologies in the gravitation field of *D*, also influenced by mutual gravitation

Figure 2. Equilibrium states in the gravitation field of domain D: (a) the case of a single ontology; (b) – multiple ontologies

O is considered as *properly positioned* within D when it reaches its *equilibrium* state (Fig. 2a) with respect to the gravitation field in D, i.e. appears at a distance l from the centre of D at which $\overrightarrow{G_O}\Big|_D = \overrightarrow{0}$. This distance could be interpreted as an integral measure of the semantic difference between what does O describe and what is required to be described for D by its knowledge stakeholders. If O is not in an equilibrium state regarding D, $\overrightarrow{G_O}\Big|_D$ will cause it to move either towards the centre of D or towards its periphery. O also generates its gravitation field which affects D. However, we do not take into account the movement of D because the centre of the grid (and therefore a potential observer) is always located in the centre of D.

The gravitation field of O will come into effect in this grid if there are several ontologies positioned within D (Fig. 2b). This case is resolved similarly to the case of a single ontology described above. Ontology A reaches its *equilibrium state* within D and with respect to the ontologies B and C if $\overrightarrow{G_A}\Big|_D + \overrightarrow{G_A}\Big|_B + \overrightarrow{G_A}\Big|_C = \overrightarrow{0}$. So do the other ontologies B and C. In this equilibrium state the distances l_{AB},

l_{AC}, l_{BC} could be interpreted as the integral measures of the semantic difference in the respective pairs of ontologies, also under the influence of R in D. One topical difference for the case of multiple ontologies is that the differences and similarities in the pairs of ontologies are computed differently compared to the fitness in the pair O, D. For comparing ontologies, the use of matching techniques is the mainstream approach.

Let us now compare a mechanical system which is governed by the Newton's Gravitation Law and the proposed Domain – Ontology system using the proposed fitness-based gravitation. The comparison is summarized in Table 1.

The subsequent section focuses on the case of a single ontology in the gravitation field of D as the basic. It elaborates how the set of requirements R could be formed for D and also how ontology fitness changes could be computed.

4 Supporting Techniques

As outlined above, for making the theoretical framework based on the Law of Gravitation usable in practice several technical problems have to be solved and corresponding software tools to be developed. Let us unfold the workflow for computing gravitation forces from the outline given in Table 1.

As it may be seen in Fig. 3, the amount of work to be accomplished before the Law of Gravitation can be applied is quite high. The amount of human work may however be reduced due to:

- The re-use of the results of the previous iterations as the number of the newly coming requirements is normally much lower than of those already processed and still remaining valid for D
- The use of several instruments – methods and tools – that may help partially automate the process

The techniques and tools applicable in this context are presented in this section.

Table 1. Gravitation in a mechanical system versus domain – ontology system

	A Mechanical (e.g. Solar) System	**A Domain – Ontology System**
Coordinate grid	E.g. Helio-centric, 3 dimensional, Decartes	Domain-centric, 2 dimensional, normalized
Distance (l)	Meters, from point (0,0,0)	Normalized, semantic
Mass (m)	Kilogramms, measured using scales or other indirect methods	Fitness of O regarding R describing D
Force (G)	Newton's Law: $G = \gamma \frac{m_1 \times m_2}{l^2}$	$\left.\overrightarrow{G_O}\right\rvert_D = \sum_{r \in R: ns_r \geq l} \left(\overrightarrow{G_o^+} + \overrightarrow{G_O^-} + \overrightarrow{G_o^-} \right)$
Model type / granularity	Continuous	Discrete
To apply	• Measure masses • Measure distance	• Extract $r \in R$ with their ns • Create knowledge tokens (kt) for r • Map kt to O • Compute $\Delta\Phi_o^+$, $\Delta\Phi_o^-$, $\Delta\Phi_O^-$

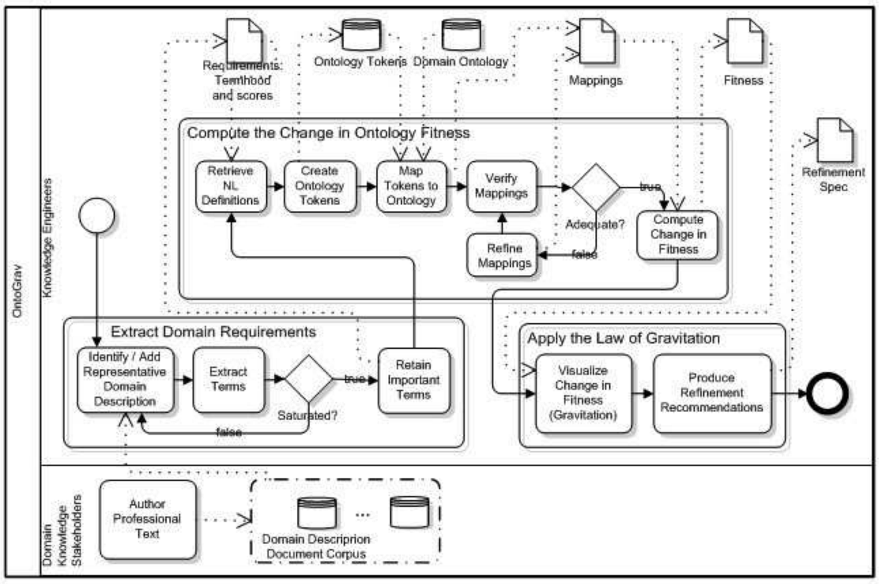

OntoGrav
Knowledge Engineers
Domain Knowledge Stakeholders
Requirements: Termhood and scores
Ontology Tokens
Domain Ontology
Mappings
Fitness
Refinement Spec
Compute the Change in Ontology Fitness
Retrieve NL Definitions
Create Ontology Tokens
Map Tokens to Ontology
Verify Mappings
Adequate?
true
false
Compute Change in Fitness
Refine Mappings
Extract Domain Requirements
Identify / Add Representative Domain Description
Extract Terms
Saturated?
true
false
Retain Important Terms
Apply the Law of Gravitation
Visualize Change in Fitness (Gravitation)
Produce Refinement Recommendations
Author Professional Text
Domain Descriprion Document Corpus

Figure 3. The workflow for applying the Law of Gravitation in the ontology refinement process

4.1 Extracting Domain Requirements

As explained in Section 2, a feasible way to make domain requirements explicit is to elicit those indirectly – by extracting multi-word key terms from a representative document corpus describing the domain. A document corpus could be considered as representative if it is sufficiently completely covers the description of the domain. One way to assess its completeness is to use the *saturation* metric proposed in OntoElect [9] as follows.

Let $Doc = Doc_1, ..., Doc_{i+1} = Doc_i \bigcup \Delta_{i+1}, ..., Doc_n$ be the sequence of the samples of the document corpus which are built incrementally – i.e. each subsequent sample Doc_{i+1} in the sequence is created by adding a number of new relevant documents (Δ_{i+1}) to the previous sample Doc_i. Let $T_i = \{(t^i_j, s^i_j, ns^i_j)\}$ be the bag of terms and their normalized scores extracted from the sample Doc_i. A normalized score

ns_j^i of a term t_j^i is computed as $ns_j^i = s_j^i / s_{\max}^i$, where $s_{\max}^i$ is the maximal score among all the terms in the bag. A bag of terms T_i is the termhood related to Doc_i if T_i contains only:

- Significant terms – i.e. those scored above the significance threshold ε_s; and
- Valid terms – i.e. those after filtering out the terms that are highly ranked, but have no substantial contribution to the semantics of the domain

One reasonable way to choose ε_s is to ensure that the terms in the termhood reflect the majority of the stakeholders' opinions. This could be done by taking in those terms from the top of the bag of terms, sorted by term score, having the sum of the scores slightly higher than the 50 per cent of the sum of all scores in the bag of terms. $Doc = Doc_1, ..., Doc_{n-1}, Doc_n$ is considered *saturated* if:

$$thd(T_{n-1}, T_n) < \varepsilon_{st}, \tag{6}$$

where: *thd* is the termhood difference function computed using the THD algorithm [9] which takes semantically equivalent and orphan terms in consideration; ε_{st} is the saturation threshold chosen empirically by a knowledge engineer for the given domain; T_{n-1}, T_n are the termhoods related to the two final document samples Doc_{n-1}, Doc_n of *Doc*.

It is assumed in our work that the sequence of *thd* values monotonically going down below ε_{st} indicates that Doc_n is a complete document corpus possessing sufficient representativeness. Non-monotonicity of *thd* values sequence signals that the corresponding Δ_{i+1} is either not very relevant to the domain or is a valuable addition containing the terminology not used in the previous samples (Doc_i). Anyhow, saturation indicates that the chosen document corpus is complete.

In order to apply semi-automated ontology mapping technique to compare these extracted requirements and ontology contexts, the requirements have to be represented similarly formally as the ontology. For achieving that:

- Natural language definitions for the terms in the final termhood are collected. An example is given in Figure 4. This activity is performed manually by a knowledge engineer.

- Formalized semantic contexts (knowledge tokens, *kt*) are built for the terms using the retrieved definitions. This activity could be facilitated by following the OntoElect methodology as described in [9, 10]. Knowledge tokens are in fact small ontological fragments coded in OWL DL and also visualized as UML class diagrams – as pictured in Fig. 4.

- The mappings of the constructed semantic contexts to the ontology are created. The mappings could be verified by a knowledge engineer using the visualization of the structural difference represented in an extended UML class diagram notation [14]. This activity could be done semi-automatically using the software tools for ontology alignment [13, 14].

4.2 Computing the Change in Ontology Fitness

For measuring the fitness of the entire ontology or its particular constituents with respect to the domain requirements the OntoElect methodology [9] recommends to use the metaphor of votes. Votes are computed based on:

- The scores of the respective terms t in R

- The mappings of the terms to the ontology elements

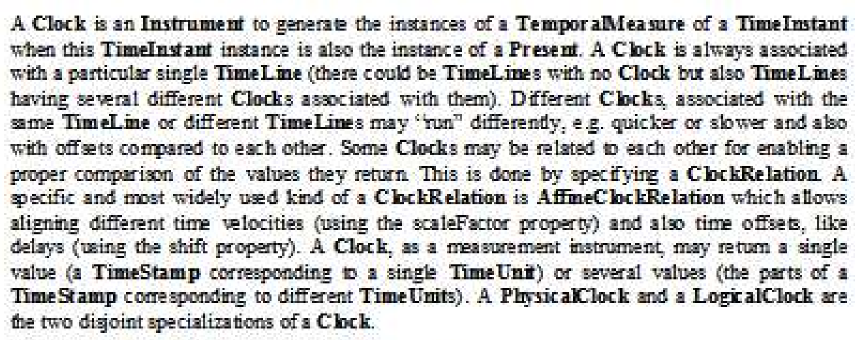
A **mapping** of the term t to ontology O is denoted as the function that establishes a relationship between t and the element of O: $\mu = (t, re, o, cf)$, where re is the relationship type – $re \in \{equivalence, membership, subsumption, meronymy, association\}$, o is the element in O, and cf is the confidence factor with a value from $[0, 1]$. Hence, $M_o = \{\mu\}$ is the set of all term mappings to the ontology element o.

A **positive vote** v_o for an ontology element $o \in O$ is denoted as a value reflecting the evidence of referring to o by the term t through the term mapping μ:

$$v_o = \sum_{\mu \in M_o} ns \times w(re) \times cf, \tag{7}$$

where: *ns* is the normalized score of t; cf is the confidence factor of the respective mapping μ; and $w(re)$ is the weight of the mapping based on the type of the relationship *re* of μ. The weights are introduced to reflect that different types of mappings could be regarded as the arguments of different strength in favour of this ontological element. Indeed, if a term is *equivalent* to the element, then it is a strong direct argument in favour of the element. However a statement about being an individual member of the element, a direct subsumption of an element, being a part of an element, or having an association to an element is considered as a weaker argument. So the weights are proposed as: *equivalence* – 1.0; *membership* – 0.7; *subsumption*, *meronymy* – 0.5; *association* – 0.3. These values may further be reconsidered if any experimental evidence is collected in this respect. Direct subsumption mappings to very abstract elements in the ontology should however be avoided. For example, all concepts, and therefore the terms categorized as concepts, subsume to the root concept of a **Thing** present in any OWL ontology. This subsumption mapping has indeed very little to do with domain semantics and therefore should not be counted as an argument for a vote. Valid direct subsumption mappings have to be sought to the most specific possible ontology elements. Indirect subsumption mappings could further be accounted for propagating votes up the concept hierarchy as described below. Propagated votes may be used to further clarify the distribution of the fitness upwards the subsumption hierarchy of the ontology.

So far only direct positive votes with respect to ontology elements have been discussed. So, the overall ontology fitness computed based on these votes reflects only the arguments focused on an element and without any influence on the surrounding of this element. This however might not be fully correct with respect to the fitness of the surround-



A **Clock** is an **Instrument** to generate the instances of a **TemporalMeasure** of a **TimeInstant** when this **TimeInstant** instance is also the instance of a **Present**. A **Clock** is always associated with a particular single **TimeLine** (there could be **TimeLines** with no **Clock** but also **TimeLines** having several different **Clocks** associated with them). Different **Clocks**, associated with the same **TimeLine** or different **TimeLines** may 'run" differently, e.g. quicker or slower and also with offsets compared to each other. Some **Clocks** may be related to each other for enabling a proper comparison of the values they return. This is done by specifying a **ClockRelation**. A specific and most widely used kind of a **ClockRelation** is **AffineClockRelation** which allows aligning different time velocities (using the scaleFactor property) and also time offsets, like delays (using the shift property). A **Clock**, as a measurement instrument, may return a single value (a **TimeStamp** corresponding to a single **TimeUnit**) or several values (the parts of a **TimeStamp** corresponding to different **TimeUnits**). A **PhysicalClock** and a **LogicalClock** are the two disjoint specializations of a **Clock**.

(a) A natural language definition of the concept of a Clock

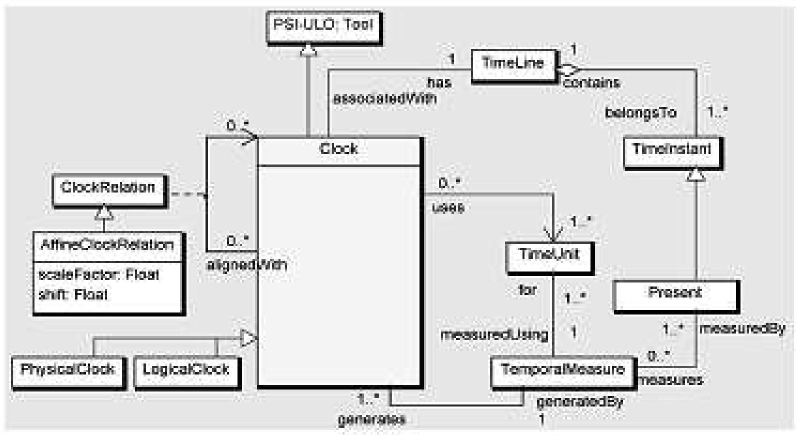


Figure 4. Term processing pipline by example. The term and semantic context of a Clock

ing elements. Indeed, let us for example assume that the concept of a **Clock** in a Time ontology gets a vote. Then it may be expected that the concept of an **Instrument**, subsuming **Clock** (See also Fig. 4), also qualifies for the part of the value of this vote. A straightforward reason is that, due to the subsumption relationship, the more specific concept inherits the properties of the more abstract concept in the subsumption hierarchy. So the vote has to be propagated up the hierarchy with attenuation – factored empirically or possibly aligned with the proportion of the inherited properties in each individual case.

A **propagated vote** v_o^p for an ontology element $o \in O$ is the value reflecting the contribution of o to the semantics of the ontology element o^{sub} subsumed by o:

$$v_o^p = att \times v_{o^{sub}}, \tag{8}$$

where att is the attenuation coefficient.

Positive and propagated votes provided by the term t are further used for computing the **fitness increments** $\Delta\Phi_o^+$ of the elements in O.

$$\Delta\Phi_o^+ = \sum_{\mu \in M_o} v_o + \sum_{O_o^{sub}} v_o^p, \tag{9}$$

where O_o^{sub} is the subset of the elements in O which are subsumed by o.

A **negative vote** provided by a term t ($v_t^- = -ns$) is:

- Either a vote based on the term $t \in T^{miss}$ pointing out that t is not described by O. In this case a fitness decrement for the whole ontology O could be computed as:

$$\Delta\Phi_O^- = v_t^- \mid_{t \in T^{miss}} ; \tag{10}$$

- Or a vote pointing out that the term t is in a contradiction with a particular ontology element o. In this case a fitness decrement for the ontology element $o \in O$ could be computed as:

$$\Delta\Phi_o^- = v_t^- . \tag{11}$$

The overall change in ontology fitness caused by the influence of the term t (requirement $r \in R$), being the sum of all positive, propagated, and negative votes could hence be computed as follows:

$$\Delta\Phi_O\,|_t = \sum_O (\Delta\Phi_o^+ + \Delta\Phi_o^-) + \Delta\Phi_O^- . \tag{12}$$

Consequently, the change in overall ontology fitness caused by R is:

$$\Delta\Phi_O\,|_R = \sum_R (\Delta\Phi_O\,|_t)\,, \tag{13}$$

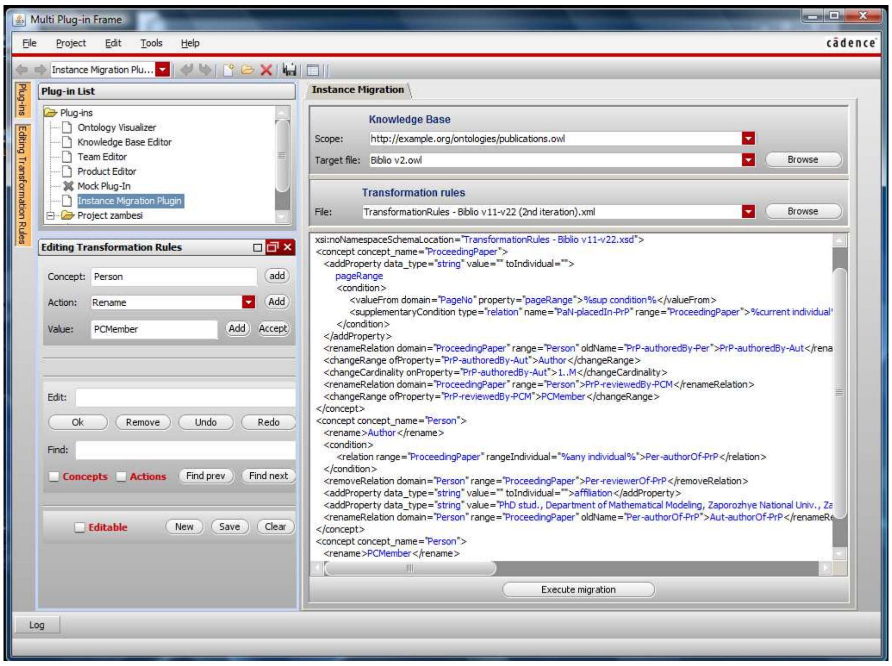
As already mentioned in Section 2.2, all these changes are taking effect if the sphere of influence ns of the requirement $r = (t, ns) \in R$ is more or equal to the distance l between O and D.

4.3 Computing Mappings between Ontologies

The creation of the mappings of the semantic contexts of the terms from the termhood (knowledge tokens, *kt*) could be done in a partially automated way using an appropriate ontology matching technique. One possible technique is meaning negotiation using argumentation based on the exchange of presuppositions [12]. This approach has been implemented in several software tools supporting different steps in the mapping generation process:

- Generation of the mappings between the TBoxes of two different ontologies in the ontology alignment format or as ABox transformation rules could be facilitated using the Structural Difference Discovery Engine (SDDE) [13]

SDDE uses an approach for ontology alignment based on the implementation of meaning negotiation [12] between intelligent software agents. Their negotiation strategy implies aligning ontologies by parts (conceptual subgraphs or contexts) that are relevant to a particular negotiation encounter. Negotiation is conducted in an iterative manner and is aimed at the reduction of a semantic distance between the contexts. Agents use propositional substitutions, expressed in a Type theory, which may reduce the distance, and support them with argumentation. The process is stopped when the distance reaches some commonly accepted threshold or the parties exhaust their propositions and arguments. The software produces a set of mappings between the ontology fragments either in the Ontology Alignment Format [15] or as transformation rules [16]. The mappings are produced as XML serializations of $\mu = (t, re, o, cf)$ – as explained in Section 4.2. These mappings, after been verified, may be refined using the Transformation Rule Editor of the OIM Tool [16] – as pictured in Fig. 5.



Concept: Person
Action: Rename
Value: PCMember

Scope: http://example.org/ontologies/publications.owl
Target file: Biblio v2.owl
File: TransformationRules - Biblio v11-v22 (2nd iteration).xml

```
xsi:noNamespaceSchemaLocation="TransformationRules - Biblio v11-v22.xsd">
<concept concept_name="ProceedingPaper">
  <addProperty data_type="string" value="" toIndividual="">
     pageRange
     <condition>
        <valueFrom domain="PageNo" property="pageRange">%sup condition%</valueFrom>
        <supplementaryCondition type="relation" name="PaN-placedIn-PrP" range="ProceedingPaper">%current individual'
     </condition>
  </addProperty>
  <renameRelation domain="ProceedingPaper" range="Person" oldName="PrP-authoredBy-Per">PrP-authoredBy-Aut</rena
  <changeRange ofProperty="PrP-authoredBy-Aut">Author</changeRange>
  <changeCardinality onProperty="PrP-authoredBy-Aut">1..M</changeCardinality>
  <renameRelation domain="ProceedingPaper" range="Person">PrP-reviewedBy-PCM</renameRelation>
  <changeRange ofProperty="PrP-reviewedBy-PCM">PCMember</changeRange>
</concept>
<concept concept_name="Person">
  <rename>Author</rename>
  <condition>
     <relation range="ProceedingPaper" rangeIndividual="%any individual%">Per-authorOf-PrP</relation>
  </condition>
  <removeRelation domain="Person" range="ProceedingPaper">Per-reviewerOf-PrP</removeRelation>
  <addProperty data_type="string" value="" toIndividual="">affiliation</addProperty>
  <addProperty data_type="string" value="PhD stud., Department of Mathematical Modeling, Zaporozhye National Univ., Za
  <renameRelation domain="Person" range="ProceedingPaper" oldName="Per-authorOf-PrP">Aut-authorOf-PrP</renameRe
</concept>
<concept concept_name="Person">
  <rename>PCMember</rename>
```

Figure 5. OIM tool Dashboard and Transformation Rule Editor, adopted from [16]

- Verifying the structural changes between the TBoxes of two different OWL ontologies by visualizing the difference using an extension to the UML class diagram language could be performed by the Ontology Difference Visualizer (ODV) tool [14]. ODV desktop is pictured in Fig. 6.

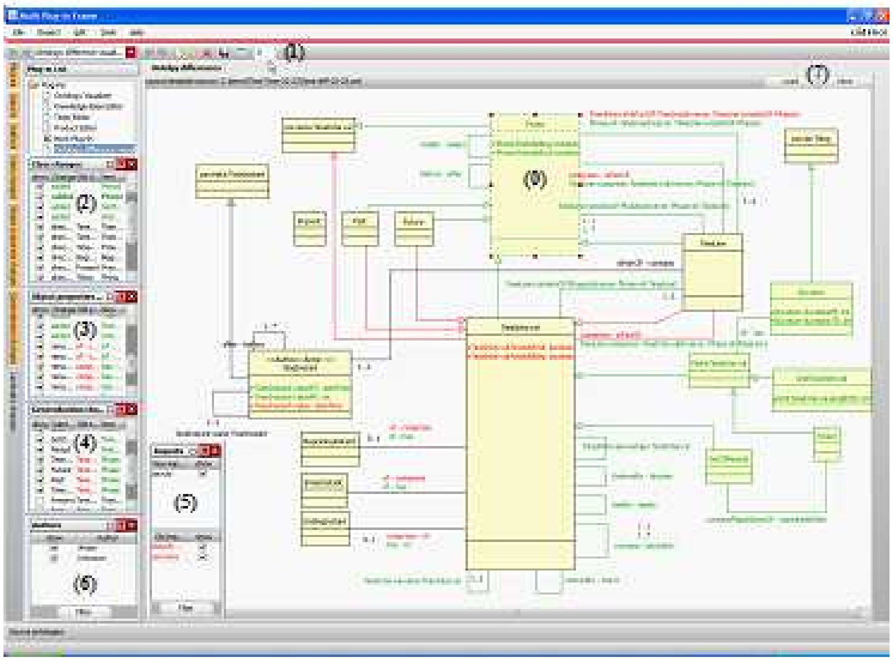


Figure 6. The ODV desktop. Visualized is the structural difference between the PSI Time Core ontologies v.2.2 and 2.3. Adopted from [14]

The composition of a semantic context of a concept (0), as implemented in the ODV, could be formed by specifying the radius of the neighborhood of this concept (1). Further it could be fine-tuned by manual inclusion or exclusion of the concepts (2), object properties (3), subsumption relationships (4). The analyzed ontological context may be placed on the wafer of the source (old) ontology by toggling the “show old” mode in the Tools menu. The context may be also altered by considering or filtering out the concepts belonging to the imported ontology modules (5). Finally the "owner" filter may be employed for concentrating on the changes that have been introduced by a particular ontology engineer in the team (6). The ODV implementation allows also editing and saving the layout of the visualized structural difference in the project file (7). Such a layout saves all the context settings and therefore allows personalized representations for different users. The release of the ODV proof of concept software prototype [14] has been implemented in Java as a plug-in to the Cadence ProjectNavigator software prototype. ODV uses OWL API 2 and therefore is capable of processing OWL ontologies coded in OWL DL.

5 Conclusive Remarks

This paper presented the approach to deal with dynamics in knowledge representations, in the form of ontologies, regarding the domains these ontologies are intended to describe. In order to place the reported research in the context of the scientific discipline, the basics of Ontology Engineering, Management, and Change have been concisely presented in Section 2.

The high-level idea followed in the presented work is to understand the dynamics of ontologies in a way similar to the other scientific disciplines – primarily answering the questions about the causes of a change and therefore offering the laws to compute forces and their effect on the motion of ontologies within the domain. Hence, the central part of the presented research deals with an attempt to exploit the analogy with the Newton's law of Universal Gravitation. This law has however to be applied to the objects that do not possess physical mass. Therefore, the proper analogues for a mass, a coordinate grid and its origin; a position, a distance, a motion; and a force (gravitation) have been elaborated – resulting in a theoretical Ontology Gravitation framework presented in Section 3. This framework is based on the notion and measurement of ontology fitness to the knowledge stakeholder requirements to the description of a particular Domain of Discourse.

It has also been described in Section 4 of the paper that the implementation of the presented theoretical framework is feasible as the supporting techniques, including some software tools already exist. The presentation focused on outlining the opportunities provided by the OntoElect methodology, Ontology Difference Visualizer, and Structural Difference Discovery Engine to help solve the practical problems in:

- Eliciting domain requirements without any direct involvement of the knowledge stakeholders
- Developing structural contexts for multiple word key phrases that indicate the requirements
- Generating the mappings between these structural contexts and the target ontology
- Computing increments and decrements of ontology fitness based on these mappings

The framework presented in the paper has prospects to be applied practically for visualization and analysis of ontology changes in dynamics. The following use cases could be of particular scientific, industrial, and societal value.

Ontology refinement is the implementation of the required changes in an ontology for making it fit the changed stakeholder requirements to the maximal possible extent – Fig. 7. In the terms of the Ontology Gravitation framework described above, stakeholder requirements are captured by R for D (Fig. 1), each having also its *ns*. So the changes in these requirements result in the changes to the gravitation field generated by D (Fig. 7a and 7b). These in turn will cause that the ontology O changes its position to reach a new equilibrium state in the changed gravitation field of D (Fig 7c and 7d). This new position of O may appear to be closer to the centre of D's gravitation – which indicates that the changes in the stakeholder requirements were favourable for the current implementation of O. It may also appear, as in Fig. 7d, that O will move further out from the gravitation centre of D – indicating that the changes in requirements hint about the necessity to refine O. A visualization tool showing the changes in ontology equilibrium state positions in response to the changes in the gravitation field of D may become a powerful instrument for a knowledge engineer to assess and justify the refinement of the particular fragments of the ontology. Such a justification will be based on the acquired knowledge, in a condensed and visualized form, about the causes triggering the needed change.

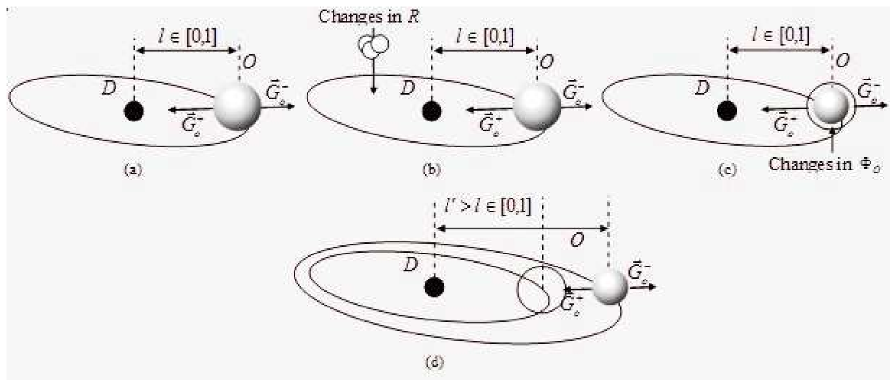


Figure 7. A way to visualize the changes in ontology fitness in ontology refinement

Anomaly detection in data analytics is about revealing the parts of data that change beyond normal values – hinting about a potential or developing problem in the system that is the source of these data. For example, if a system is a civil community and its environment (D), then it may be producing many diverse streams of observation data coming from various sorts of sensors – like outdoor temperature measurements, water levels, industrial emissions, share prices, cell phone activity, etc. Imagine that each sort of censor measurement is described by its individual ontology which is updated using knowledge extraction from the respective incoming data stream. From the other hand, community requirements R reflect the desire of the stakeholders to live in a comfortable (normal) environment: clean air and water; stable share prices, no traffic hold-ups, etc. If so, it is reasonable to expect that an equilibrium state, involving the abovementioned sensor data ontologies and D, will show how close (normal) or far (abnormal) each sort of sensor measurement is from the normal condition. This visual result may be made available in time sufficient for emergency response to the detected anomaly.

6 Acknowledgements

The theory of gravitation between ontologies and Domains of Discource, based on fitness, has been developed in the SemData project funded by the Marie Cure International Research Staff Exchange Scheme (IRSES) of the 7th Framework Programme of the European Union, grant No PIRSES-GA-2013-612551. OntoElect methodology has been developed as a part of the PhD project of Olga Tatarintseva. The tools for ontology mapping and alignment have been developed with Anton Copylov and Maxim Davidovsky.

References

[1] V. Ermolayev. *The Law of Gravitation in Ontology Dynamics.* In: S. Cojocaru and C. Gaindric (Eds.): Proc. W-shop on Foundations in Informatics (FOI 2015), pp. 246–267, Acad. of Sciences of Moldova, 2015.

[2] W.-E. Matzke. *Engineering Design Performance Management – from Alchemy to Science through ISTa (Invited Talk).* ISTA 2005, 23-25 May 2005, Palmerston North, New Zealand, pp. 154–179 (2005).

[3] I. Newton. *The Principia, Mathematical Principles of Natural Philosophy, a new Translation.* By I Bernard Cohen and Anne Whitman, preceded by "A Guide to Newton's Principia" by I Bernard Cohen, University of California Press, 1999, ISBN 978-0-520-08816-0, ISBN 978-0-520-08817-7.

[4] Rudi Studer, V. Richard Benjamins, Dieter Fensel. *Knowledge Engineering: Principles and Methods.* In: Data & Knowledge Engineering, 25(1-2):161–197, 1998.

[5] J. Euzenat, P. Shvaiko. *Ontology Matching*, Berlin Heidelberg (DE), Springer-Verlag, 2007.

[6] D. Nardi, R. J. Brachman. *An Introduction to Description Logics.* In The Description Logic Handbook, F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, Eds. Cambridge University Press New York, NY, USA, 2007.

[7] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann. *Modelling ontology evaluation and validation.* In: Sure, Y., Domingue, J. (eds.) ESWC 2006, LNCS 4011, pp. 140–154 (2006).

[8] Fabian Neuhaus, Amanda Vizedom, Ken Baclawski, Mike Bennett, Mike Dean, Michael Denny, Michael Grüninger, Ali Hashemi, Terry Longstreth, Leo Obrst, Steve Ray, Ram Sriram, Todd Schneider, Marcela Vegetti, Matthew West, Peter Yim. *Towards Ontology Evaluation Across the Life Cycle.* In: Applied Ontology. 8 (2013), pp. 179–194, DOI 10.3233/AO-130125.

[9] O. Tatarintseva, V. Ermolayev, B. Keller, W.-E. Matzke. *Quantifying Ontology Fitness in OntoElect Using Saturation- and Vote-Based Metrics.* In: Ermolayev et al. (eds.) ICT in Education, Research, and Industrial Applications. Revised Selected Papers of ICTERI 2013, CCIS **412**, pp. 136–162, Springer Verlag, Berlin – Heidelberg (2013).

[10] V. Ermolayev, S. Batsakis, N. Keberle, O. Tatarintseva, G. Antoniou. *Ontologies of Time: Review and Trends.* In: Int. J. of Computer Science & Applications, 11(3), pp. 57–115, 2014.

[11] Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, Grigoris Antoniou. *Ontology Change: Classification and Survey.* In: The Knowledge Engineering Review, 23(2), pp. 117-152, 2008. doi:10.1017/S0269888908001367.

[12] V. Ermolayev, N. Keberle, W.-E. Matzke, V. Vladimirov. *A Strategy for Automated Meaning Negotiation in Distributed Information Retrieval.* In: Y. Gil et al. (Eds.): ISWC 2005 Proc. 4th Int.

Semantic Web Conference (ISWC'05), 6-10 November, Galway, Ireland, LNCS 3729, pp. 201–215 (2005).

[13] M. Davidovsky, V. Ermolayev, V. Tolok. *Agent-Based Implementation for the Discovery of Structural Difference in OWL-DL Ontologies.* In: H.C. Mayr et al. (Eds.): UNISCON 2012, LNBIP 137, pp. 87–95 (2013).

[14] V. Ermolayev, A. Copylov, N. Keberle, E. Jentzsch, W.-E. Matzke. *Using Contexts in Ontology Structural Change Analysis.* In: Ermolayev, V., Gomez-Perez, J.-M., Haase, P., Warren, P, (eds.) CIAO 2010, CEUR-WS, vol. 626 (2010).

[15] J. David, J. Euzenat, F. Scharffe, C. Trojahn dos Santos. *The Alignment API 4.0,* Semantic Web, 2(1), 3-10 (2011).

[16] M. Davidovsky, V. Ermolayev, V. Tolok. *Instance Migration Between Ontologies having Structural Differences.* Int J on Artificial Intelligence Tools. 20(6) 1127–1156 (2011) DOI: 10.1142/S0218213011000553

Vadim Ermolayev

Received September 7, 2015

Department of IT, Zaporizhzhya National University,
66 Zhukovskogo st., 69063, Zaporizhzhya, Ukraine
E–mail: `vadim@ermolayev.com`

Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Solving Problem of Graph Isomorphism by Membrane-Quantum Hybrid Model

Artiom Alhazov Lyudmila Burtseva Svetlana Cojocaru
Alexandru Colesnicov Ludmila Malahov

Abstract

This work presents the application of new parallelization methods based on membrane-quantum hybrid computing to graph isomorphism problem solving. Applied membrane-quantum hybrid computational model was developed by authors. Massive parallelism of unconventional computing is used to implement classic brute force algorithm efficiently. This approach does not suppose any restrictions of considered graphs types. The estimated performance of the model is less then quadratic that makes a very good result for the problem of **NP** complexity.

1 Introduction

The present paper concerns application of new computational models based on hybrid of bio-inspired and quantum approaches. In computability theory, a model defines feasible computational operations with their execution time/space. There are many branches of bio-computation: evolution, DNA, swarm, etc. We took as our base the membrane computing formalism, also known as P systems [4].

A P system is a set of mutually inclusive membranes that contain multisets of objects (numbers, strings, or some abstract items) and evolve under some rules. All possible rules are applied in parallel to all possible membranes and objects. This is the membrane parallelism that makes this computation model very powerful.

In our model, membranes can additionally contain quantum machines that perform quantum computations (Fig. 1).

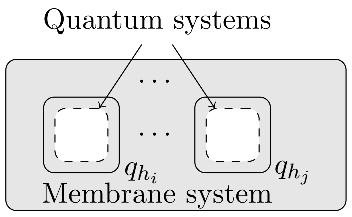


Figure 1. Structure of the hybrid model

The idea of such hybrid computation arises from needs of different domains delivering hard tasks, which are not always satisfactorily solved by existing high performance computational models.

To illustrate the proposed model, we present in this paper a solution of the graph isomorphism problem (GI). Due to its practical applications ranging from chemistry to social sciences, this problem has been solved by many algorithms, both classical and unconventional, still remaining under investigation. Unlike problems which are usually considered as suitable for unconventional computation, GI belongs to **NP**, but is not known to belong to its well-studied subsets like **P** or **NP**-complete. The best classical algorithm complexity is $O(c^{\sqrt{n}\log n})$, where n is the number of vertices in the graph.

Because of this uncertainty of general task, it is divided into several subtasks according to graph types (trees, planar graphs, poor-connected, etc.). The majority of mentioned subproblems are proved to be in the integer factorization class. So, existence of polynomial-time quantum algorithm for integer factoring makes GI a good candidate for speedup by a quantum computing [1]. Further developments of quantum computing solutions of GI mostly applied the quantum walk [6]. However, all issues attributable for quantum computation such as "probability" results or exponential growing of system size affect the proposed solutions.

As we said above, we choose the membrane computing formalism. The proposed hybrid model is the usual P system framework supplied by capability to perform quantum computations in its membranes.

Membranes, which are supposed to obtain quantum functionality, just have the specific marks in the P system description. In the marked membrane, the apparition of some specific objects (quantum data, or quantum triggers) starts quantum computation. Specified data become the initial state of the quantum registers. After finishing the quantum computation produces other specific objects (quantum results) in the external membrane.

To provide incorporated quantum functionality in the proposed hybrid model, standard scheme of quantum device [7] proves itself to be sufficient.

Both for membrane and quantum part, we use particular P system formalisms and quantum gates in dependence of the solved problem.

In this paper, the hybrid model solving GI is constructed over *decision P system with active membranes* implementing brute force algorithm. The quantum devices perform only comparison using CNOT, NOT and Toffoli gates.

2 Hybrid Computational Model

2.1 Membrane Subsystem

Membrane systems, or **P systems,** consist of a set of mutually inclusive membranes. The membrane structure μ is a rooted tree, traditionally represented by bracketed expression. For example, $[\ [\ [\ ]_4\ ]_2\ [\ ]_3\ ]_1$ denotes membranes 2 and 3 inside membrane 1, and membrane 4 inside membrane 2. There are many different variants of P systems. Some variants may use static membrane structure, others change it during calculations.

Membranes contain multisets of objects. Objects are numbers, strings, or some abstract items. Different operations over objects may be available. The initial state of a P system is always provided. The initial state is some membrane structure with some multisets inside.

The evolution of a P system is governed by a set of rules. Rules are applicable under certain conditions to change the objects in membranes. All possible rules are applied in parallel to all possible membranes and objects (membrane parallelism). The calculation stops when no rules can be applied.

We will use a decision P system with active membranes. *Decision* means that the alphabet of objects contains symbols `yes` and `no` that represent two possible results of calculation. *P system with active membranes* is defined as a tuple:

$$\Pi = \{O, E, \mu, w_1, \cdots, w_m, e_1, \cdots, e_m, R\}.$$

Here O is the alphabet of objects. $E = \{0, 1, ..., k\}$ is a set of membrane electrical charges, or polarizations. μ is a membrane structure of m membranes labeled by integers; we will denote $H = \{1, \ldots, m\}$ a set of membrane labels. $w_i \in O^*$ and $e_i \in E$, $i \in H$, represent initial content and initial polarization of the i-th membrane. Strings w_i over alphabet O (possibly empty) represent multisets of objects from O. R is a set of rules of the form:

(a) $[a \to v]_h^i$, $a \in O$, $v \in O$, $h \in H$, $i \in E$ (evolution rules, used in parallel in the region of the h-th membrane, provided that the polarization of the membrane is i);

(b) $a[]_h^i \to [b]_h^j$, $a, b \in O$, $h \in H$, $i, j \in E$ (communication rules, sending an object into a membrane and possibly changing the polarization of the membrane);

(c) $[a]_h^i \to []_h^j b$, $a, b \in O$, $h \in H$, $i, j \in E$ (communication rules, sending an object out of a membrane, possibly changing the polarization of the membrane);

(d) $[a]_h^i \to b$, $a, b \in O$, $h \in H$, $i \in E$ (membrane dissolution rules; in reaction with an object, the membrane is dissolved);

(e) $[a]_h^i \to [b]_h^j [c]_h^k$, $a, b, c \in O$, $h \in H$, $i, j, k \in E$ (division rules for elementary membranes not containing other membranes inside; in reaction with an object, the membrane is divided into two membranes with the same label, possibly of different polarizations, and the object specified in the rule is replaced in the two new membranes by possibly new objects).

2.2 Quantum Subsystem

The investigated hybrid model supposes additionally that in any membrane the apparition of some specific objects (quantum data, or quantum triggers) starts a quantum calculation. The said data are available as initial state of the quantum registers. After its termination the quantum calculation produces another specific objects (quantum results) inside the membrane. From the P system point of view, the quantum calculation is a step of the membrane calculation.

Quantum device. We suppose a standard quantum device available for quantum calculations. The quantum device contains qubits organized in quantum registers. It works in three steps: non-quantum (classical) initialization of qubits when they are set in base states; quantum transformation when the qubits are non-observable; non-quantum (classical) measurement that produces the observable result.

Several restrictions are imposed over the quantum device. Each qubit contains 0, or 1, or (during quantum calculation) superposition of both. Therefore, the initial data and the result may be regarded as non-negative integers in binary notation. The quantum transformation is linear and reversible. The general rule is that arguments and results are kept in different quantum registers. Another general condition is that the ancillary qubits were not entangled with the argument and the result after the calculation.

The construction of a quantum computer shown in Fig. 2 guarantees this.

3 Interface between Membrane and Quantum Sub-systems

Communications between membrane and quantum sub-systems are performed through input/output signals and triggering (Fig. 3).

We define the hybrid system formally as a tuple

$$\beta = (\Pi, T, T', H_Q, Q_N, Q_M, Inp, Outp, t, q_{h_1}, \cdots, q_{h_m}).$$

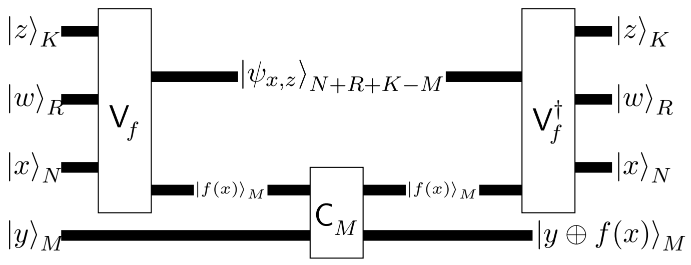


Figure 2. Quantum calculation; initialization and measurement are not shown

Here, Π is a P system, and $H_Q = \{h_1, \cdots, h_m\}$ is a subset of membrane labels in Π used for quantum calculations. T is a trigger and T' is the signal on obtaining the quantum result. Sub-systems $q_{h_1}, \cdots, q_{h_m}$ are the quantum sub-systems associated to the corresponding membranes from H_Q. The rest of the components of the tuple β specify the interaction between Π and q_{h_j}, $1 \le j \le m$ (Fig. 3).

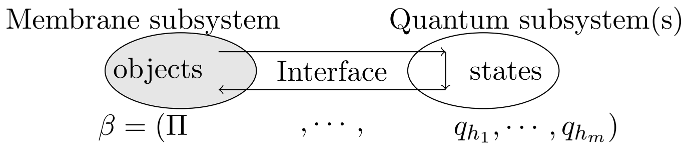


Figure 3. Subsystems and interface in the hybrid model

For simplicity, we assume that the running time of quantum subsystems of the same type is always the same. To keep this time general, we include a timing function $t : H_Q \to \mathbb{N}$: the quantum computation in a sub-system of type q_{h_j} takes $t(h_j)$ membrane steps. It is an open general question how to calculate the timing of quantum calculation with respect to the timing of membrane calculation. We could use as the first rough estimation that quantum calculation takes three steps of membrane calculation (initialization, quantum transformation, measurement).

The input size (in qubits) for quantum systems is given by Q_N :

$H_Q \to \mathbb{N}$. The output size (in bits) for quantum systems is given by $Q_M : H_Q \to \mathbb{N}$.

We would like to define the behavior of β in all possible situations, so we introduce the trigger $T \in O$, where O is the alphabet of Π. The work of a quantum sub-system of type q_{h_j} starts whenever T appears inside the corresponding membrane. Note that we said that q_{h_j} is a type of a quantum sub-system, because in general there may be multiple membranes with label h_j containing quantum sub-systems with the same functionality. The quantum state is initialized by objects from $Inp(h_j) = \{O_{k,h_j,b} \mid 1 \leq k \leq Q_N(h_j),\ b \in \{0,1\}\} \cup \{T\}$, so $Inp : H_Q \to 2^O$ is a function describing the input sub-alphabet for each type of quantum sub-system, the meaning of object $O_{k,h_j,b}$ being to initialize bit k of input by value b. We require that the set of rules satisfies the following condition: any object that may be sent into a membrane labeled h_j must be in $Inp(h_j)$.

The output of quantum sub-systems is returned to the membrane system in the form of objects from $Outp(h_j) = \{R_{k,h_j,b} \mid 1 \leq k \leq Q_M(h_j),\ b \in \{0,1\}\} \cup \{T'\}$, the meaning of object $R_{k,h_j,b}$ being that the output bit k has value b. In case of one-bit output, we often denote it `yes` and `no`.

The result of a quantum sub-system may be produced in the membrane together with object T'.

There are two possibilities to synchronize quantum and membrane levels. We can use a timing function and to wait for the quantum result by organizing the corresponding delay in membrane calculations, or we can wait for appearance of the resulting objects, or the trigger T'. For generality, our model provides both possibilities. The topic needs further investigations.

4 Graphs Isomorphism Problem

GI requires to decide whether two given graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are actually the same graph with relabeling of the vertices.

4.1 Graph Isomorphism: Hybrid Computation

The first graph is represented by objects $a^{(c)}_{i,j,0,0,0}$, where $c = 1$ if the graph has edge (i, j), and $c = 0$ if it does not, $0 \le i \le n-1$, $0 \le j \le n-1$. The second graph is similarly represented by objects $b^{(c)}_{i,j,0}$. Let $N = \lceil \log_2 n \rceil$ (hence, $n \le 2^N < 2n$). We construct the following hybrid system.

$$\begin{array}{rcl}
\beta & = & (\Pi, H_q = \{2\}, n, \mathit{Inp}, \mathit{Outp} = \{\mathtt{yes}, \mathtt{no}\}, q_2), \text{ where} \\
\mathit{Inp} & = & \{I_{k,b} \mid 0 \le k \le 2n^2 - 1,\ 0 \le b \le 1\}, \\
q_2 & & \text{is a quantum system comparing the first } n^2 \text{ bits with the} \\
 & & n^2 \text{ second bits in } 2N+1 \text{ steps, described later, and} \\
\Pi & = & (O, \Sigma, \mu = [\ [\ ]^0_2\ ]^0_1, w_1, w_2, R, 1)
\end{array}$$

is a decisional P system with active membranes, where

$$\begin{array}{rcl}
\Sigma & = & \{a^{(c)}_{i,j,0,0,0}, b^{(c)}_{i,j,0} \mid 0 \le i \le n-1,\ 0 \le j \le n-1, c \in \{0,1\}\}, \\
O & = & \{d_i \mid 1 \le i \le nN\} \cup \{p_i \mid 1 \le u \le (n+2)N + 5\} \\
 & \cup & \{x_{i,t,k,s} \mid 0 \le i < n,\ 0 \le t < n,\ 0 \le k \le N, \\
 & & 0 \le s \le \max(2^{k-1}, 0)\} \cup \{\mathtt{yes}, \mathtt{no}, X\} \\
 & \cup & \{a^{(c)}_{i,j,t,k,s} \mid -2^N \le i < n,\ -2^N \le j < n,\ 0 \le t \le n, \\
 & & 0 \le k \le N,\ 0 \le s \le \max(2^{k-1} - 1, 0),\ c \in \{0,1\}\} \cup \mathit{Inp} \\
 & \cup & \{b^{(c)}_{i,j,t} \mid 0 \le i < n,\ 0 \le j < n,\ 0 \le t \le nN + 1,\ c \in \{0,1\}\}, \\
w_1 & = & p_1,\ w_2 = d_1 x_{0,0,0,0} \cdots x_{n-1,0,0,0},
\end{array}$$

and the set R is the union of the following rule groups (together with their explanations): generation, checking, processing the input, and result. Note that the all four groups start working in parallel.

Generation

1 : $[\ d_i\ ]^e_2 \rightarrow [\ d_{i+1}\ ]^0_2\ [\ d_{i+1}\ ]^1_2,\ e \in \{0,1\},\ 1 \le i \le nN,$

2 : $[\ d_{nN+1}\ ]^e_2 \rightarrow [\ ]^0_2 d_{nN+1},\ e \in \{0,1\},$

1: creating 2^{nN} membranes and generating for each of them the corresponding nN bits defining the permutations candidates. Other objects may check these bits as membrane polarizations during nN steps (not considering the initial step, where the polarization was 0).

2: After the generation phase, set the polarization to 0.

Checking

3 : $[\ x_{i,t,k,s} \to x_{i,t,k+1,2s+e}\ ]_2^e,\ 0 \le i < n,\ e \in \{0,1\},$
$0 \le t < n,\ 0 \le k \le N-1,\ 0 \le s \le \max(2^{k-1}-1, 0),$

4 : $[\ x_{2s+e,t,N,s} \to \lambda\ ]_2^e,$
$0 \le s \le 2^{N-1}-1,\ 0 \le t < n,\ e \in \{0,1\},$

5 : $[\ x_{i,t,N,s} \to x_{i,t+1,1,0}\ ]_2^e,\ 0 \le i < n,$
$0 \le t < n-1,\ 0 \le s \le 2^{N-1}-1,\ e \in \{0,1\},\ i \ne 2s+e,$

6 : $[\ x_{i,n-1,N,s} \to X\ ]_2^e,\ 0 \le i < n,$
$0 \le s \le 2^{N-1}-1,\ e \in \{0,1\},\ i \ne 2s+e,$

7 : $[\ X\ ]_2^0 \to [\ \ ]_2^1 X,$

3: Compute the value $\sigma(t)$ of permutation σ for node label t.

4: Erase the label $\sigma(t)$.

5: Continue matching the label.

6: Rename unmatched node labels into X (to make the next step deterministic).

7: Invalid permutation detected. Cancel isomorphism check by setting polarization to 1.

Processing the input

8 : $[\ a^{(c)}_{i,j,t,k,s} \to a^{(c)}_{i,j,t,k+1,2s+e}\ ]_2^e,$
$-2^N \le i < n,\ -2^N \le j < n,\ 0 \le t < n,\ 0 \le k \le N-1,$
$c \in \{0,1\},\ 0 \le s \le \max(2^{k-1}-1, 0),$

9 : $[\ a^{(c)}_{i,j,t,N,s} \to a^{(c)}_{i',j',t+1,1,0}\ ]_2^e,\ -2^N \le i < n,\ -2^N \le j < n,$
$0 \le t < n,\ c \in \{0,1\},\ 0 \le s \le 2^{N-1}-1,$
$i' = -2s-e-1,\ i = t,\ i' = i$ otherwise,
$j' = -2s-e-1,\ j = t,\ j' = j$ otherwise,

10 : $[\ a^{(c)}_{-i-1,-j-1,n,1,0} \to I_{ni+j,c}\ ]_2^0,$
$0 \le i < n,\ 0 \le j < n,\ c \in \{0,1\},$

11 : $[\ b^{(c)}_{i,j,t} \to b^{(c)}_{i,j,t+1}\ ]_2^e,\ 0 \le i < n,\ 0 \le j < n,$
$0 \le t \le nN,\ e \in \{0,1\},\ c \in \{0,1\},$

12 : $[\ b^{(c)}_{i,j,nN+1} \to I_{n^2+ni+j,c}\ ]_2^0,$
$0 \le i < n,\ 0 \le j < n,\ c \in \{0,1\},$

8: Compute $\sigma(t)$ for matrix elements.

9: Perform row/column substitution if row/column is t. If so, store the result as a negative index, minus one. In either case, proceed with the next node.

11: The input symbols for the second graph wait while the permutations for the first graph are being generated.

10,12: Initialize the quantum subsystem.

Result

13 : $[\,\texttt{yes}\,]_2^0 \rightarrow [\ ]_2^1\texttt{yes}$,

14 : $[\,\texttt{yes}\,]_1^0 \rightarrow [\ ]_1\texttt{yes}$,

15 : $[\,p_i \rightarrow p_{i+1}\,]_1^0,\ 1 \le i \le (n+2)N+4$,

16 : $[\,p_{(n+2)N+5}\,]_1^0 \rightarrow [\ ]_1^1\texttt{no},\ 1 \le i \le (n+2)N+4$.

13: If the quantum subsystem detected a match, send this signal out to the skin.

14: Send the final answer `yes` out, also halting the computation.

15: Wait for the possible answer `yes` to appear.

16: If it did not appear in time, send the final answer `no`.

4.2 Quantum Comparison

Quantum comparison is shown in Fig. 4. It uses CNOT, NOT and Toffoli gates. The result is produced on the qubit initialized by $|0\rangle$ (the lowest in the diagram).

4.3 Notes on Complexity

The classical general algorithm solves GI for graphs of n vertices in time $O(c^{\sqrt{n}\log n})$, were c is a constant [3].

Quantum computation has been widely employed at GI solving during last decade. Initially, users of classic algorithms just applied the Grover method for search between relabeled candidates. But for graphs with n nodes a naive application of Grover search means $O(\sqrt{n!})$ queries, so some other quantum methods have been proposed to improve the efficiency. The most popular of these methods seems to be

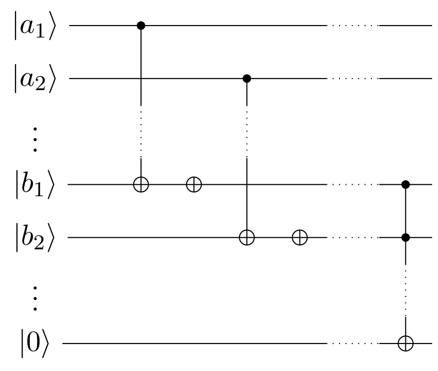


Figure 4. Quantum comparator

the quantum walk [6]. The computation complexity of GI solution applying quantum walk is declared for graph with n vertices as $O(n^7)$ for discrete quantum walk [2] and as $O(n^6)$ for continuous one [5].

In the presented GI solution the P system part of computation takes $2\lceil(\log_2 n)\rceil + 1$ steps. Supposing the pure P system computation the algorithm could execute the comparison of each pairs (candidate/pattern) by 2 steps. Totally the pure P system based comparison would take $2n^2$ steps.

We will count the quantum subsystem comparison as 3 steps. So, the whole work time is $(n + 2)\lceil(\log_2 n)\rceil + 4$.

5 Conclusions

This paper concerns the application of membrane-quantum hybrid computational model to speed up the classical brute force algorithm solving the problem of graph isomorphism.

Membrane-quantum hybrid computational model is the P system framework with additional quantum functionalities. With this approach, we obtained computation time advancement against both pure membrane and pure quantum solutions, namely: $O(n \log_2 n)$ (hybrid) against $O(n^2)$ (pure membrane) and $O(n^6)$ (pure quantum).

References

[1] S. Dorn. *Quantum Algorithms for Graph and Algebra Problems.* VDM Verlag (2008).

[2] B.L. Douglas, J.B. Wang. *Classical approach to the graph isomorphism problem using quantum walks.* Journal of Physics A: Mathematical and Theoretical 41(7) (2008).

[3] J. Kobler, U. Schoning, J. Toran. *The graph isomorphism problem: its structural complexity.* Birkhauser Verlag (1994).

[4] Gh. Păun. *Membrane Computing. An Introduction.* Springer (2002).

[5] X. Qiang, X. Yang, J. Wu, X. Zhu. *An enhanced classical approach to graph isomorphism using continuous-time quantum walk.* Journal of Physics: A Mathematical and Theoretical 45(4) (2012).

[6] K. Rudinger, J.K. Gamble, E. Bach, M. Friesen, R. Joynt, S.N. Coppersmith. *Comparing algorithms for graph isomorphism using discrete- and continuous-time quantum random walks.* Journal of Computational and Theoretical Nanoscience 10(7), 1653–1661(9) (2013).

[7] C.P. Williams. *Explorations in Quantum Computing.* Springer (2008).

Artiom Alhazov, Lyudmila Burtseva,
Svetlana Cojocaru, Alexandru Colesnicov,
Ludmila Malahov

Received September 19, 2015

Institute of Mathematics and Computer Science
Str. Academiei 5,
Chişinău, MD-2028,
Moldova
Phone: +373 22 72 59 82
E–mail:
{artiom.alhazov,lyudmila.burtseva,svetlana.cojocaru,kae,mal}@math.md