Theory of Clones and Formalized Design of Programs

Anatoliy Doroshenko¹, Georgiy Tseytlin¹, Olena Yatsenko¹, and Lubov Zachariya¹

¹ Institute of Software Systems of National Academy of Sciences of Ukraine, Glushkov prosp. 40, 03187 Kiyiv, Ukraine
dor@isofts.kiev.ua, tseytlin@vikno.relc.com, aiyat@i.com.ua, zlm56@web.de

Abstract. The paper is devoted to the theory of clones which is a promising approach to the challenge of program design automation in the framework of the algebraic algorithmics. A family of algebras of the same type, in particular, algebras of algorithms, constitutes a clone. Grammatical andalgorithmic clones as well as applied algebras belonging to various subject domains are described. The use of algebraic algorithmics facilities for formalized design of programs is illustrated with the examples of algorithms of symbolic multiprocessing.

Keywords: algebraic algorithmics, algebra of programs, clone, iteration and recursion, regular schemes.

1 Introduction

Algebraic treatment of programming abstracts has been always influential and insightful approach to programming due to its rigorous mathematical foundation and powerful facilities for program design. In recent years a promising approach of algebraic algorithmics is being intensively developed (see for instance [4, 8, 11, 14]). A particular direction of algorithmics, being developed in Ukraine and named as algebra of algorithms, dates back to fundamental works of academician V.M. Glushkov aimed to solve problems of design automation of computer logic structure and programming [7]. There systems of algebras of algorithms (SAA) have been constructed and investigated which were later called the Glushkov algebras (GA), and results on structural schematology and the multilevel structural program design (MSPD) technique have been obtained [14].

In this paper a review of some new results in algebras of algorithms is proposed that concerns constructing families of algebras (clones). Clones of algebras appear to have tight linkage with classic programming paradigms (structural, nonstructural, object-oriented) [16] and modern approach of generative programming [2]. They also unify usage of various discrete models (logic, automaton-theoretical, grammatical and algorithmic) to describe the programming paradigms.

The outline of the work is as follows. The concept of a clone is given in Section 2. Section 3 is devoted to the construction of the Post clone and its modifications. Main results on grammatical and algorithmic clones are presented in Section 4. The formalized design of programs in algorithm algebras illustrated with the examples of
parallel sorting and search algorithms are considered in Section 5. In Conclusion a number of directions are formulated whose development strongly depends on the algebra of algorithmics.

2 The Concept of a Clone

The concept of a clone belongs to one of fundamental concepts of a universal algebra [1]. As a clone, we usually imply a one-based universal algebra of the form:

\[ C = \langle A; \text{SUPER} \rangle, \]

where \( A \) is the base being a set of functions of the same type, \( \text{SUPER} \) is a signature consisting of a superposition of functions as well as of operations on their identification and variable relettering (in more detail, various definitions of the superposition can be found in [7, 10]). In [14–16] the concept of a clone is applied to construct various families of algebras, similar in the means used for description, analysis, and synthesis of the objects studied. In this case, the signatures of the algebras mentioned are systems of generatrices (SG) of the corresponding clones. The construction of algebra families is based on the solution of the functional completeness problem for the clones mentioned, that is known to be associated with the investigation of subalgebra lattices, in particular, construction of a set of maximal subalgebras for given clones.

An example of a one-based clone is the algebra of logic or the Post clone (PC) whose base is a set of all the Boolean functions (BF). This clone will be considered in details in Section 3. In this paper, along with the Post and Kleene clones and their context-free (CF) generalizations, algorithmic clones are also considered, which reflect the essence of the known programming paradigms (see Sections 4, 5).

3 The Post Clone and Its Modifications

The clones under consideration are associated with a two-valued algebra of logic and with some versions of superposition operations constituting the signatures of these clones. The results are reviewed on functional completeness and on constructing the lattices of subalgebras (closed classes) to characterize families of BF algebras. These algebras are, in particular, the means to describe logic components of algorithms (programs).

3.1 The Post Clone and Functional Completeness Theorem

The concept of a clone can be used to specify, construct, and investigate both various algebra families and concrete representatives of these families. The signature of operations of each algebra among those constituting the family under consideration is a signature of operations (SO) of the corresponding clone [16].
A classical example of a one-based clone is the two-valued algebra of logic, the Post algebra (PA) [14]:

\[ PA = \langle L(2); \text{SUPER} \rangle, \]

where \( L(2) \) is a set of all the BFs.

E. Post constructed for the PC a lattice of subalgebras (closed classes) of a countable power. In particular, a family of maximal subalgebras (precomplete classes) was constructed:

- \( T_0 \) is a class of all the BFs preserving constant of 0;
- \( T_1 \) is a class of all the BFs preserving constant of 1;
- \( C \) is a class of all self-dual BFs;
- \( M \) is a class of all monotonic BFs;
- \( L \) is a class of all linear BFs.

A famous Post theorem on functional completeness is formulated in terms of maximal subalgebras above-listed.

**Theorem 1.** An arbitrarily chosen system of BFs, \( \Sigma \), is an SO in PC if and only if it is not included into any of maximal subalgebras: \( T_0, T_1, C, M, L \).

The solution of the functional completeness problem is a medium to construct various algebras of BFs: The Boole algebra, the Zhegalkin algebra, etc. [14]. It should be reminded that the signatures of operations of these algebras are the SO of the Post clone, which is included into a number of multi-based clones associated with various families of algebras of algorithms (see Sections 4, 5). The paradigm about the isomorphism of \( PA \) and algebra of functional \( n \)-relations \( O/2 \) and the theorem of functional completeness for \( O/2 \) were formulated in [18].

### 3.2 Modified Post Clones and Synchronous Multiprocessing

Conventional applications of the Post clone and its modifications are associated with solving fundamental problems on design and synthesis of logic structures of computing machines and systems. Specifically, in connection with designing computing machines of “Mir” series, within the framework of automaton-theoretical constructions, a problem was set up to study a semigroup of periodically-determined (PD) transformations at abstract (infinite) registers [7]. These investigations allowed to construct a lattice of subalgebras for a modified Post clone (PC-M) [16].

A peculiarity of the PC-M is that its lattice of subalgebras, in contrast to that of PC, has a power of continuum. In connection with this fact, a surface, i.e. the upper lattice fragment containing all the finite-generated subalgebras was built. In particular, a family of all the maximal subalgebras (the surface upper level) was constructed, which allowed to prove the functional completeness theorem for PC-M. An interface zone was also investigated, that consists of infinite-generated subalgebras (with an infinite basis or without any basis). It should be noted that the means of PC-M are of practical interest as PD conversions at abstract registers and their non-uniform generalizations implement synchronous parallel register data processing.

The following binary operation can be included into the SO of logic clones:

\[ u = O \cdot u', \]

(3)
where $u$, $u'$ are pre and post conditions, respectively; $O$ is an object of different nature. One of the possible interpretations of this operation is a left multiplication of the condition by the operator, often called a prognostication operation [7, 14]. The functional completeness problem together with the corresponding operator clone was solved for a logic clone generated by a Boolean operations and a prognostication operation [16]. A description of the family of algebras of algorithms, whose typical representative is the Glushkov SAA, was also obtained [4, 14] (see Section 5).

It is worth mentioning, in particular, that the Post clone and its modifications have important applications associated with the description of logic conditions at constructing algorithms and programs as well as the objects processed by them within the framework of modern programming languages and computational environments [17].

4 Grammatical and Algorithmic Clones

This Section is devoted to the theory of grammatical and algorithmic clones being an important direction in the investigations of general algebra and a number of its applications aimed at forming the fundamentals of programming. A substantial attention is paid to classify recursive discrete models associated with formal languages and grammars.

4.1 Semigroup Clones and Their Properties

A clone, whose base contains at least one binary associative operation, will be called a semigroup clone. It follows from this definition that an associative operation can belong to SO of subalgebras of semigroup clones [15]. For example, such clones are semigroup ones, whose bases contain an operation of composition of operators (see Section 5) or an operation of multiplication of languages. A universal algebra having a continuum of subalgebras is called a continual type algebra.

**Lemma 1.** Semigroup clones are continual type algebras.

**Proof.**

Let $C = \langle A; \text{SUPER} \rangle$ be a semigroup clone, where $A$ is the base (a set of operations – functions) containing an associative binary operation $x_1 \ast x_2$. Closure $[Z]$, where $Z \subseteq A$, forms a subalgebra of the clone $C$, and system $Z$ is SO of this subalgebra. In particular, $[x_1 \ast x_2]$ forms a semigroup $SEM$ and is also a subalgebra of $C$; let $V/P = \{v/p ; p \in P\} \subseteq SEM$, where $v/p = vv...v (p \text{ times})$; $P$ is a set of all primes. By closing $V/P$ under superposition, we obtain a subalgebra $SEM/P = [V/P]$ of the semigroup $SEM$ of the clone $C$. The lemma is proved.

**Corollary 1.** $SEM/P$ has a countable basis $V/P$, any subalgebra from $C$ including $SEM/P$ has a continuum of subalgebras.

Let us introduce $(n+1)$-place iterations:

$$\{x_1[\alpha]x_2[\alpha] \ ... \ [\alpha]x_n[\alpha]x_{n+1}\} ,$$

as a combination of linear iterations on each of the points $[\alpha]$. 

331
This iteration is associated with the cycles having \( n \) exits; they were proposed while constructing algorithmic clones, in particular, generalized Dijkstra clones and their monotonically increasing sequence [16].

By extending SO of \( Z/0 \) with iterative operations introduced, we obtain a sequence of strict inclusions \( (M \subset M') \):

\[
Z/0 \subset Z/1 \subset ... \subset Z/n \subset Z/(n+1) ... ,
\]

(5)

where \( Z/(n+1) = Z/n \cup \{x_1[x_2][x_3] ...[x_n][x_{n+1}]; n = 0, 1, ...\} \).

Let us construct a sequence \( C = \{EK/n = [Z/n] | n = 1, 2, ...\} \), where \( EK \) is an extended Kleene clone.

Strict inclusions \( EK/n \subset EK/(n+1) \) hold for any \( n = 1, 2, ... \).

**Lemma 2.** A clone \( C \) is a set-theoretical limit \( C \lim_{n \to \infty} EK/n \), which forms a set-theoretical limit \( C \lim_{n \to \infty} EK/n \), does not have any basis.

Thus, the following statement holds.

**Theorem 2.** Semigroup clones are adequate by properties to \( k \)-valued logics \( (k > 2) \) [15].

### 4.2 The Kleene Clone and Its Context-Free Generalizations

A solution of the functional completeness problem for semigroup clones is associated with the property of isolation and its modifications.

Let \( C = <A, \text{SUPER}> \) be some semigroup clone, \( A \) is the base (a set of functions), \( h = f_1, f_2, ..., f_m \) is a superposition of arbitrarily chosen functions from the base \( A \).

A set \( M \subset A \) is isolated if superposition \( h \in M \) only if all the functions of the superposition \( h \) are included into \( M \); in other words, the superposition \( h \in \neg M \) if at least one function of the superposition \( H \) belongs to \( \neg M \), where \( \neg M = A \setminus M \) is a complement of \( M \).

The construction of the lattice of the clone \( A \) subalgebras is reduced to construction of lattices of subalgebras for \( B \) and \( \neg B = A \setminus B \), where \( B \) is an isolated subalgebra.

In particular, the solution of the functional completeness problem for the algebra \( A \) is reduced to the description of maximal subalgebras for \( B \) and extensions of \( B \) to subalgebras, maximal for \( A \).

Let us consider the Kleene algebra [15, 16]: \( KA = <RE, \text{SIGN}> \), where \( RE \) is the base, i.e. a set of all regular events (RE); \( \text{SIGN} \) is a signature consisting of operations: semigroup multiplication \( x_1 \ast x_2 \); unification \( x_1 \cup x_2 \); iterations \( IT(*) \) and \( IT(+) \).

As \( CL/A \) we will imply a one-based clone of languages: \( CL/A = <OP/A, \text{SUPER}> \), where \( OP/A \) is the base consisting of a set of various operations determined on \( CL/A \); \( \text{SUPER} \) is a signature including only a superposition of functions with a possible identification and relettering of function variables.

In particular, SO of \( Z \) can consist of operations included into the signature of \( KA \), then the clone \([Z]\) we will call the Kleene clone (\( KC \)) [15]. (Further, if an algebra \( A/N \) is a representative of the clone \( C \), then it will be denoted as \( CN \).)

Natural generalizations of the clone \( KC \) associated with the consideration of multiplace iterations are presented in [15].
Along with the algebra and clone \( KC \) and its iterative generalizations, algebras of context-free (CF) languages are of interest. Let \( G = (T, N, \alpha, P) \) be a CF grammar, where \( T \) is a terminal alphabet and \( N \) is a nonterminal alphabet; \( \alpha \in N \) is a start nonterminal; \( P \) is a set of products of the form \( p: n \to v \), such that \( n \in N \) is a nonterminal, \( v = u_1n_1u_2n_2\ldots u_mn_{m+1} \) is the second member of the product. According to every CF grammar \( G \) a multi-based algebra can be constructed: \( R/A = <\{ L/n | n \in N \}, SIGN> \), where \( L/n \) are bases (languages, whose strings are derived in \( G \) from the nonterminal \( n \)); \( SIGN \) is a signature consisting of CF-operations of the form \( n \to v \). The system of CF-operations constituting \( SIGN \) will be called a recursion operation corresponding to the grammar \( G \), and it will be denoted as \( R/G \). The recursion operations can be included into \( SO \) of the CF clone under consideration.

The basic idea for solving the functional completeness problem for the \( KC \) clone is associated with the isolation concept. Let us consider a set of all finite languages, which we will call \( FL \). The closure \( P = [x_1 \ast x_2, x_1 \cup x_2] \) under superposition forms a clone \( FL = <FL, SUPER> \); so that \( P \) is \( SO \) of the clone \( FL \). It is to be noted that the set \( FL \) is isolated for the clone \( KC \) and its generalizations mentioned above. In [15] a family of maximal subalgebras for \( KC \) was constructed, that solves the functional completeness problem for this clone. The following constant operations can be included into \( SO \) of \( KC \): an empty word \( e \) and an empty language \( F \), whose use at superposition allows to cut the strings in length. In [15] the functional completeness problem solution for a modified \( KC \) of a cut type was also obtained. Similar results can be obtained for above-mentioned generalizations of \( KC \) (including recursive CF-clones) as well as for algorithmic clones [16].

### 4.3 Algorithmic Clones

An algorithmic clone is the two-based system

\[
ALC = <\{OPER, L(2)\}; SUPER> ,
\]

where \( OPER \) is an operator base consisting of a set of noninterpreted operator schemes. The choice of \( SO \) of \( ALC \) determines the system of algorithmic constructions typical for one or another technique for constructing algorithms and programs. So, for the structural programming method such a system consists of constructions included into the signature of operations of Dijkstra algebra \( DA \) [4, 14]. The closure of the above system of algorithmic constructions under superposition generates the algorithmic clone \( DC = <\{SNS, L(2)\}, SUPER> \), where \( SNS \) is the algebra of structural noninterpreted schemes (the operator base of \( DC \)), \( L(2) \) is \( PA \) (the logical base of \( DC \)). \( SO \) of \( DC \) are adequate to signatures of operations of various algebras of algorithms, similar to \( DA \). The signatures of operations of these algebras can include diverse cyclic constructions, the prototypes of which are known language iterations (see Subsection 4.2).

Let us formulate the following main results on semigroup grammatical and algorithmic clones corresponding to known families of algebras of algorithms [4, 14]:

- the clones investigated are algebras of a continual type;
– each of them includes infinite-generated subalgebras (with an infinite basis and without any basis);
– families of maximal subalgebras were constructed for the Kleene clone and its generalizations, and the functional completeness problem was solved;
– a surface of DC and its generalizations having \( q \) exits from the cycle \( (q = 1, 2, ...) \) was built;
– a set of maximal subalgebras of CL clone was described, whose canonical representative is the Yanov algebra [14];
– the CL was found to be a set-theoretical limit of DC generalizations studied;
– a set of maximal subalgebras for the Glushkov clone (GC) and its logic component, whose SO includes a prognostication operation, was described.

The Glushkov algebras are considered in more detail in Section 5.

An analog of the investigations carried out can be the results obtained for a modified Post algebra [16]. The importance of the extension of the results listed to algorithmic clones of flowgraphs and their generalizations [4, 14] is caused by applications of the flowgraph theory while visualizing object-oriented programming. Criteria of functional completeness and scheme expressibility in the clones constructed were established. The establishment of such criteria is of great practical importance for implementing tool (software and hardware) facilities to design and synthesize algorithms and programs as well as their classes associated with topical subject domains [4, 14, 17].

It is worth mentioning that the considered results on iterative algorithmic algebras and their families (clones) are limited by a two-based case associated with the analysis of operator and logic constructions. At the same time, there is a possibility to extend them to multi-based clones associated with abstract data types.

## 5 Formalized Design of Programs in Algebras of Algorithms

The background for the development of multi-based algebras of algorithms and clones, which can generate families of such algebras, was a fundamental concept of an electronic computer abstract model [7]. This concept was later called a discrete transducer and finds nowadays various applications. It is based on the interaction between controlling and controlled components of any cybernetic system using a feedback principle. In terms of programming, such an interaction can be considered as functioning of an algorithm (or a program) while processing the data stored in the system memory.

The systems of algorithmic algebras and their extensions (SAA-M) are associated with the automaton model mentioned. They are oriented towards formalization of sequential and parallel (synchronous, asynchronous, and combined) computations [7]. Different types of such computations are considered in [3, 6, 9, 13, 17].

SAA-M is the two-based algebra \(<\{U, B\}; \Omega>\), where \( U \) is a set of logical conditions and \( B \) is a set of operators, defined on an informational set \( IS; \Omega = \Omega_1 \cup \Omega_2 \); is the signature of operations consisting of the systems \( \Omega_1, \Omega_2 \) of logical operations and operator operations respectively.
The following generalized Boolean operations constitute the system $\Omega$: disjunction $u_1 \lor u_2$; conjunction $u_1 \land u_2$; negation $\overline{u}$; prognostication operation $u = A \cdot u'$.

The main operator operations, which belong to the system $\Omega_2$, are the following: composition $A \ast B$ – the sequential execution of operators $A$ and $B$; alternative $([u] A, B)$ – the operation of the type “if $[u]$ then $A$ else $B$”; loop $\{[u] A\}$ – the iterative operation of the type “while not $[u]$ do $A$”, consisting in cyclic execution of operator $A$ until predicate $u$ is true. Signature $\Omega$ also contains operations intended for formalization of parallel computations. In particular, these operations include: filtration $u$; synchronous $A \land B$ and asynchronous $A \lor B$ disjunctions consisting in synchronous and asynchronous execution of operators $A$ and $B$ respectively. Control points and synchronizers are used to synchronize parallel processes represented in SAA-M. Control points $P(u)$ are fixed positions between operations in an algorithm scheme. Each control point $P$ has the condition $u$, which is false until the computation process reaches the point $P$ and becomes true at the moment of reaching $P$. Condition $u$ is called the synchronization condition associated with the point $P$. Synchronizers implement the delay of computations in processes. A synchronizer is defined by the cycle $S(u) = \{[u] E\}$, where $u$ is the logical function, which depends on synchronization conditions associated with control points of an algorithm scheme; $E$ is an empty operator.

Operator presentations of algorithms in SAA and SAA-M were called regular schemes (RS), and the algebras themselves were called the Glushkov algebras. The superpositions of signature operations of SAA-M specifying asynchronous operator interactions were called parallel regular schemes (PRS).

As the basis of $G\!A$ we fix a set $I = L \cup U$ of operator and logic variables. The following statements hold.

**Theorem 3** (the Glushkov theorem on regularization). An arbitrary algorithm (program or microprogram) $F$ is representable in SAA and SAA-M by an equivalent interpreted RS $F/I$, $F = F/I$; a constructive regularization procedure for an arbitrary algorithm is developed, i.e. reduction of $F/I$ to RS.

**Theorem 4.** Let $|A|$ be a set of operators generated by the algebra $A$, then the following strict inclusions hold:

$$|DA| \subset |YaA| \subset |GA|,$$

(7)

where $YaA$ is the Yanov algebra [14].

The proofs of the Theorems 3 and 4 are based on belonging of the prognostication operation to the signature of $DA$ and derivativeness of main $DA$ operations in $YaA$. It is important to mention a transformational reducedness of $YaA$ nonstructural schemes to equivalent RSs.

Algorithm flowgraphs belong to most common tools of algorithmic specifications. The problem of constructing the corresponding algebra of algorithms was solved in [14].

Let $AFG$ be an algebra of flowgraphs, whose SO consists of graph presentations of operations included into $GA$.

**Theorem 5** [14]. $AFG$ is isomorphic with $GA$, and any algorithm $A$ is representable by an equivalent interpreted flowgraph $G/1$, $A = G/1$. 

335
The results obtained for SAA-M and AFG are the basis of investigations in algebra of algorithms [4, 14]. This algebra is a two-level system. The upper level is a level of noninterpreted schemes, while the lower one is a level of interpretations associated with various applications [14]. Depending on the problem, the technique used to develop algorithm classes, technological environment for programming, the construction of a required algebra of algorithms (AA) is implemented from a family specified by the corresponding clone. The signature of the AA constructed satisfies the functional completeness theorem for a given clone and is thereby its SO. The AA constructed can be examined for axiomatization, development of systems of optimizing transformations, canonization of analytical representations etc. At the lower (interpreted) level of algebra of algorithmics a fully concrete applied algebra of algorithms is constructed (AAA) oriented towards presentation of algorithmic knowledge about a subject domain chosen taking into account known classification and inheritance mechanisms, typical for an object-oriented approach being the base of modern and promising informational technologies [12]. The formalism developed is universal as it is applied to design various algorithmic models of subject domains, specifically, to solve problems on symbolic multiprocessing (parallel sorting, search, syntactic analysis [4, 5, 9, 13, 14, 17]) and the corresponding language and tool facilities [14, 17]). The tools of algebra of algorithmics are integrated with Unified Modeling Language (UML) and its toolkit for object-oriented design [12].

A deep analogy should be emphasized to exist between the techniques of syntactic analysis, on the one hand, and formalized design of algorithms and programs, on the other. First and foremost, this analogy is determined by the fact that syntax structures are hierarchical and software projects are multilevel ones; basic strategies (descending, ascending and combined) have classification commonality. In [14] basic reasons for weak possibility to learn a software presented in programming languages of an imperative type were shown. A necessity to create algorithmic languages of quite new type, the languages of specification of schemes (models) of algorithms, is substantiated, and requirements that these languages should meet are established. A language SAA/1 was developed, which partially meets these requirements and is designed for a multilevel structural specification of sequential and parallel algorithms. The mathematical basis for the language SAA/1 is SAA-M.

Further, the examples of formalized design of algorithms of symbolic multiprocessing represented in SAA-M are given.

**Example 1.** Let us consider the asynchronous sorting algorithm of alternative insertions (SAI/A) represented as PRS in SAA-M. The array to be sorted is marked according to the conception of data marking, which is customary in algebraic algorithmics [14]. The conception consists in introduction of special markers fixing certain positions in data being processed and pointers moving over the data according to an algorithm. Special marking of data structures is necessary for formalization of data access. The initial state of marking of the array $A$ to be sorted is the following:

$$ A; B P_1 P_2 a_1 a_2 \ldots a_n F, \quad (8) $$

where $B, F$ are markers fixing the beginning and the end of the array $A$; $a_i$ is an element of $A$ ($i = 1, \ldots, n$); $P_1, P_2$ are pointers moving over the array $A$. SAI/A algorithm also uses the queue denoted as $BS$. The PRS of the $SAI/A$ algorithm is the following:
Part 3: Programming

\[ S_{A/A} = (\text{Start} \ast \text{Place}(P_1, P_2, B) \ast \{(l > r) P_2 \} \ast \{(l > r) P_2 \} \ast \{P(\text{ProcFin}(1)) \ast S(\text{ProcFin}(2)) \} \ast \{(l > r) P_2 \} \ast \{P(\text{ProcFin}(2)) \} \ast S(\text{ProcFin}(1)) \ast \text{Fin}, \]

\[ Alt = \text{BS.Get}(s) \ast \{(l > r) \{P_1 \leq s \} L(P_1)\} \ast \{P_1 \geq s \} R(P_1) \}, \]

where Start is the initialization operator; Place(P_1, P_2, B) is placing the pointers P_1, P_2 to the right of marker B; \(d(P_2, F)\) is the predicate being true if the pointer P_2 reaches the marker F and false otherwise; \(l > r\) P_2 is the predicate being true if this relation is true for the elements adjacent to the pointer P_2 and false if this condition is not met; BS.Put(r P_2) is the operator placing the element located to the right of the pointer P_2 into the queue BS; R(P_1) is the shift of the pointer P_1 by one element to the right; ProcFin(j) is the predicate being true if the j-th process has finished execution, \(j = 1, 2\); P(ProcFin(1)) and P(ProcFin(2)) are control points fixing the termination of first and second processes respectively; S(ProcFin(j)) is the synchronizer implementing the delay of the j-th process execution until the predicate ProcFin(j) is true; \('BS = \emptyset' and \('BS \neq \emptyset'\ are the predicates being true if the queue BS is empty and nonempty respectively; Alt is the compound operator, which searches the place to insert the element s extracted from the queue and places s to the left of P_1 with the help of the operator Insert(s, P_1); E is an empty operator; Fin is the operator printing the sorted array; BS.Get(s) is the operator, which removes the element from the queue BS and stores it in the variable s; L(P_1) is the shift of pointer P_1 by one element to the left.

The S/A/A scheme consists in asynchronous functioning of two processes. The first process searches all the disordered elements of the array to be sorted and places them to the queue BS. The second process implements the alternative insertion of elements extracted from the queue in the already sorted part of the array.

The schemes of algorithms in algorithmics can also be presented in natural-linguistic (i.e. in natural language) and graph form (flowgraphs of Kaluzhnin) [14]. It should be noted also that sorting algorithms can be obtained as a result of reinterpretation of search algorithms and vice versa with the use of corresponding metarules of algebraic algorithmics [14].

Example 2. Let us consider the algorithm S/A/A of asynchronous alternative search for records in a sorted file F_0. This algorithm can be obtained as a result of reinterpretation of the S/A/A sorting algorithm considered above. The initial state of marking of the file F_0 is the following:

\[ F_0 = B P_1 P_2 \ldots P_n a_1 a_2 \ldots a_n F, \]

where B, F are markers fixing the beginning and the end of F_0; \(a_i\) is a record of F_0, \(i = 1, \ldots, n\); P_j is a pointer moving over the file F_0, \(j = 1, \ldots, s\). The search is implemented according to the array of queries AQ:

\[ AQ = B' \pi_1 \pi_2 \ldots \pi_s F', \]

where B' and F' are markers fixing the beginning and the end of AQ; each query \(\pi_j: D \rightarrow E_j (j = 1, \ldots, s)\) is the predicate defined on a set D of records of the file F_0.
and possesses values in a set $E_2 = \{0, 1\}$. The record $a \in D$ satisfies the query $\pi_j$ if $\pi_j(a) = 1$.

The $AS/A$ algorithm is composed of $s$ parallel processes $SP_j$, $j = 1, \ldots, s$. Each of the processes searches for record satisfying the query $\pi_j$ in the file $F_0$. The PRS of the algorithm is the following:

$$AS/S = \bigvee_{j=1}^{s} SP_j,$$

$$SP_j = \text{PlaceMid}(P_j, B, F) \ast \{[r| P_j > x]\} \{[r| P_j = x]\} L(P_j),$$

$$\{[r| P_j = x]\} R(P_j) \ast \{([r] \text{ProcRec}, \text{NotFound}) \ast P(u_j) \ast\} \ast \{([j = s]\} S(u_1 \land u_2 \land \ldots \land u_{s-1}) \ast \text{PrintRes}, E\),$$

where $\text{PlaceMid}(P_j, B, F)$ is placing the pointer $P_j$ in the middle between markers $B$ and $F$; $[r| P_j > x]$ is the probable position of the record satisfying the query $\pi_j$ in the file $F_0$; $[r| P_j = x]$ is the predicate being true if a record to the right of the pointer $P_j$ is located to the right of $x$; $[r| P_j = x]$ is the predicate being true if a record $r| P_j$ is located at $x$ position; $\gamma$ is the predicate being true if a record to the right of $P_j$ satisfies the query $\pi_j$; $\text{ProcRec}$ is the operator processing the record found; $\text{NotFound}$ is the operator reacting on absence of records satisfying the query $\pi_j$ in the file $F_0$; $P(u_j)$ is the control point; $u_j$ is the predicate being true if the process $SP_j$ has finished execution; $S(u_1 \land u_2 \land \ldots \land u_{s-1})$ is the synchronizer implementing the delay of computations in the $s$-th process until all the other processes finish execution; $\text{PrintRes}$ is the operator printing the search results for all the queries processed.

The $AS/S$ algorithm consists in asynchronous functioning of $s$ processes $SP_j$. Each process moves the pointer $P_j$ from the middle of the file to the left or to the right depending on the probable position $x$ of a record looked for. If the record satisfying the query $\pi_j$ has been found the operator $\text{ProcRec}$ is executed, otherwise, operator $\text{NotFound}$ is signaling about the absence of such a record. After the $s$-th process has finished searching, it implements synchronization and prints the overall search results.

With the help of the tool facilities, based on algebra of algorithmics and developed in [17], the parallel programs in a selected object-oriented language can be synthesized for the algorithms considered.

### 6 Conclusion

In this review we presented the results on constructing the clones of algebras, unifying usage of logic, automaton-theoretical, grammatical and algorithmic discrete models to describe classic programming paradigms. The formalized program design associated with algorithmic clones was considered. The results received will be a strong incentive for further development of the following directions in informatics and modern informational technologies:

- the theory of clones, in particular, constructing subalgebra lattices and solving the functional completeness problem for clones associated with paradigms of modern programming;
– the development of algebraically-grounded classification of formal grammars (as a specification of the known Khomskiy hierarchy) on the basis of constructing the subalgebra lattices for recursive clones;
– construction and study of applied algebras at the level of interpretations of algebra of algorithmics in order to develop bases for knowledge belonging to topical subject domains;
– integrating of algebra of algorithmics facilities with tool systems of modern informational technologies for automated design and synthesis of algorithms and object-oriented programs (including parallel ones).

References