

УДК 519.682.1+519.683+519.7+519.1

OBJECT-ORIENTED DATA
AS PREFIX REWRITING SYSTEMS

A. E. Gutman

*To S. S. Kutatelaze on the occasion
of his 70th birthday*

A deterministic longest-prefix rewriting system is a rewriting system such that there are no rewriting rules $X \rightarrow Y$, $X \rightarrow Z$ with $Y \neq Z$, and only longest prefixes of words are subject to rewriting. Given such a system, analogs are defined and examined of some concepts related to object-oriented data systems: inheritance of classes and objects, instances of classes, class and instance attributes, conceptual dependence and consistency, conceptual scheme, types and subtypes, etc. A special attention is paid to the effective verification of various properties of the rewriting systems under consideration. In particular, algorithms are presented for answering the following questions: Are all words finitely rewritable? Do there exist recurrent words? Is the system conceptually consistent? Given two words X and Y , does X conceptually depend on Y ? Does the type of X coincide with that of Y ? Is the type of X a subtype of that of Y ?

Mathematics Subject Classification (2000): 68Q42, 68P05, 68N19, 68T30.

Key words: prefix rewriting, term rewriting, object-oriented data system, information system, consistency verification, ontology of a data model.

1. Introduction

The classical object-oriented approach to describing structured data employs the two primary relations, *has* (or “*has a*”) and *is* (or “*is a*”).

The *has* relation links objects (and classes) with their attributes. By saying “ X *has a* Y ” we mean that the object (or class) X possesses an attribute named Y , and we thus can speak of “*the* Y *of* X ” or “ X ’s Y ” as a property of X conventionally denoted by $X.Y$. For instance, if a web page has a submit button whose style assumes a border of a particular width, we can speak of “the width of the border of the style of the submit button of the page” and thus arrive at the object `page.submitButton.style.border.width`.

The *is* relation can be used for (1) instantiating objects from classes; (2) inheriting classes from classes; and (3) assigning values to attributes. By saying “40 *is an* Integer” we associate the object 40 with the class Integer and mean that 40 is an instance of Integer. The phrase “Integer *is* Number” means that the class Integer inherits from the class Number. By claiming that “John.age *is* 40” we assign the value 40 to the age attribute of John.

As is seen from the above examples, the wide interpretation of the *is* relation makes it possible to eliminate the difference between objects and classes. A single data system can syndicate the class declarations (“metadata”) and the object instantiations and initializations (“data”). We do not assert that data and metadata are worth more combined than separated;

nevertheless, this approach allows us to unify data analysis and develop a common tool for verifying conceptual and semantical consistency.

The *is* and *has* relations are naturally connected. By interpreting “*is*” as “inherits,” we assume that if “X is Y” then all the attributes of Y are inherited by X. In particular, if “X is Y” and “Y has a Z” then “X has a Z.” Moreover, if “X is Y” is the only explicit information on X, we can conclude that “X.Z is Y.Z.” (By the explicit information we mean the *is* rules which form the data system under consideration.) In doing so, we derive an *implicit* information on X and say that “X.Z is Y.Z *implicitly*.” Therefore, when evaluating the object X.Z, we rewrite its prefix X with Y according to the explicit rule “X is Y.” The same is applicable to objects of any length. For instance, if we know explicitly that “A.B is P.Q.R” then A.B.C.D rewrites implicitly to P.Q.R.C.D.

It is clear that the explicit rules supersede any implicit derivatives; therefore, if “X is Y” “Y has a Z,” and the data system contains the explicit rule “X.Z is A,” the latter wins over the implicit “X.Z is Y.Z.” However, a conflict of another kind is possible in case several explicit rules are simultaneously applicable. Consider the following fragment of a data system:

```

block.style.color  is blue
header.style.color is red
button             is block
button.style      is header.style

```

Let us try to evaluate the button’s style color, i.e., `button.style.color`. Since `button` is a `block`, we might conclude that `button.style.color` is `block.style.color`, which is blue. On the other hand, `button.style` is `header.style`; therefore, `button.style.color` is `header.style.color`, which is red. Intuitively, the latter evaluation should win, since “`button.style` is `header.style`” seems to take priority over “`button` is `block`.” The reason is not the fact that the former rule occurs next to the latter (we treat a data system as an unordered set of *is* rules). The key point is that the rule “`button.style` is `header.style`” is more *concrete* as it evaluates a longer object, `button.style` rather than `button`. Therefore, when evaluating an object, we should *rewrite the longest prefix* (i.e., use the most concrete rule applicable).

We will now dwell on *data consistency*. Obviously, when designing a set of definitions, conceptual cycles should be avoided. By saying “`man` is `man`” we define nothing, since evaluation of `man` results in a dead cycle. However, conceptual consistency in no way outlaws recursion. For instance, the rule “`man.son` is `man`” is quite legal. On the other hand, the rule “`man` is `man.son`” seems incorrect: we still do not know what *man* is unless *man’s son* is defined, while the latter is senseless prior to defining *man*. Furthermore, the rules “`man` is `Adam`” and “`Adam.rib` is `man.rib`” form an inconsistent pair, since `Adam.rib` is `man.rib`, while the latter implicitly rewrites to `Adam.rib`. Such examples justify the need for a formal definition of conceptual consistency and the search for the corresponding effective verification. (This is similar to analyzing the ontology of a data system as a set of concept definitions.)

It is clear that, prior to defining a set of concepts (classes or objects), we need at least one concept which does not require definition. In general, there can be several primary concepts; however, a single “generic object” is sufficient. We denote the latter by ω . Given a data system and a word X of the form `entity.attr1.attr2...attrn`, we rewrite X by applying the most concrete *is* rule, thus obtaining a new word, and continue rewriting the longest prefixes of the subsequent words until ω is reached. In this case we conclude that the initial word X is an *object* (or a *concept*). Otherwise, if the rewriting process either

(1) ends with a nonrewritable word other than ω or (2) never terminates, we claim that X is senseless. The possibility of (2) makes the analysis nontrivial and justifies the search for an effective verification if a given word makes sense. (This is close to analyzing the ontology of a concept within a data system.)

Given an object X and a word δ of the form $\text{attr}_1.\text{attr}_2 \cdots \text{attr}_n$, say that δ is a *detail* of X if $X.\delta$ makes sense. The set $\|X\|$ of all details of X can be regarded as the *type* of X . Whenever an algorithmic procedure assumes a formal argument A , the body of the procedure contains A along with some words $A.\delta_i$. For the procedure to operate correctly with X substituted for A , it is necessary (and probably sufficient) that all the words $X.\delta_i$ make sense. This results in the requirement that X be of an appropriate type. Therefore, we need an algorithm for comparing object types: given two objects X and Y , we should be able to effectively compare the types of X and Y , i.e., determine which of the relations $\|X\| = \|Y\|$, $\|X\| \subseteq \|Y\|$, $\|X\| \supseteq \|Y\|$ hold. The problem is not trivial if for no other reason than the fact that the type of an object can be infinite. (For instance, given the rule “**man.son** is **man**,” the type of **man** contains all the words **son**, **son.son**, **son.son.son**, ...)

In what follows we give formal definitions for the notions under consideration, state some results, and present algorithms for all the problems mentioned above. (The paper does not contain proofs of the theorems and justifications of the algorithms. All the details, including various examples, will be published elsewhere.)

To make notation less cumbersome, we treat the names of entities and attributes as single symbols (letters) of some alphabet \mathbb{A} and agree to write the property paths $\alpha_1.\alpha_2 \cdots \alpha_n$ as $\alpha_1\alpha_2 \cdots \alpha_n$ thus making them words over \mathbb{A} . The explicit rules “ X is Y ” will be written as $X \rightarrow Y$.

2. Definitions and Main Results

Throughout the paper, \mathbb{A} is a finite alphabet and \mathbb{A}^* (resp. \mathbb{A}^+) is the set of all (all nonempty) words over \mathbb{A} . The elements of \mathbb{A} are called *letters*. We conventionally identify the letters with the corresponding single-letter words. Say that X is a *prefix* (resp. a *proper prefix*) of $Y \in \mathbb{A}^+$ and write $X \sqsubseteq Y$ or $Y \supseteq X$ ($X \sqsubset Y$ or $Y \supset X$) if $X \in \mathbb{A}^+$ and $Y = XS$ for some $S \in \mathbb{A}^*$ ($S \in \mathbb{A}^+$). The length of a word X is denoted by $|X|$. Given an integer $n \geq 1$ and a word $X \in \mathbb{A}^+$ such that $|X| \geq n$, define $X \upharpoonright_n \in \mathbb{A}^+$ so that $X \upharpoonright_n \sqsubseteq X$ and $|X \upharpoonright_n| = n$. For brevity, in the sequel we say “word” instead of “nonempty word over \mathbb{A} .”

Given any binary relation \rightsquigarrow , we conventionally denote by \rightsquigarrow^+ the transitive closure of \rightsquigarrow and by \rightsquigarrow^* , the reflexive transitive closure of \rightsquigarrow .

Consider a finite binary relation \rightarrow on \mathbb{A}^+ (i.e., a finite subset of $\mathbb{A}^+ \times \mathbb{A}^+$) and a letter $\omega \in \mathbb{A}$. Say that the pair $\langle \rightarrow, \omega \rangle$ is a *deterministic longest-prefix rewriting system*, or a *system* for short, if \rightarrow is nonempty and the following hold:

- (a) $X \rightarrow Y$ and $X \rightarrow Z$ imply $Y = Z$;
- (b) there are no $S, Y \in \mathbb{A}^*$ such that $\omega S \rightarrow Y$.

Put $\mathbb{E} := \{X : X \rightarrow Y \text{ for some } Y\}$ and call the elements of \mathbb{E} *explicit words*. Say that E is an *explicit prefix* of X if $E \in \mathbb{E}$ and $E \sqsubseteq X$. Say that a word X is *rewritable* if X has an explicit prefix.

As is easily seen, condition (a) means that, for each $E \in \mathbb{E}$, there is a unique word E' such that $E \rightarrow E'$, while condition (b) amounts to the fact that all words of the form ωS , with $S \in \mathbb{A}^*$, are not rewritable.

Given a rewritable word X , consider the longest explicit prefix E of X , determine the suffix $S \in \mathbb{A}^*$ so that $X = ES$, and put $X' := E'S$. We call X' *the rewrite* of X . Introduce

the binary relation \Rightarrow on \mathbb{A}^+ by setting $X \Rightarrow Y$ if and only if X is a rewritable word and $Y = X'$.

By way of recursion, put $\mathbb{W}_0 := \mathbb{A}^+$, $X^{(0)} := X$ for $X \in \mathbb{W}_0$ and, for each $n \geq 1$, put $\mathbb{W}_n := \{X \in \mathbb{W}_{n-1} : X^{(n-1)} \text{ is rewritable}\}$ and $X^{(n)} := (X^{(n-1)})'$ for $X \in \mathbb{W}_n$. The word $X^{(n)}$ is called the n th *rewrite* of X . It is clear that, for each $X \in \mathbb{W}_n$, we have $X = X^{(0)} \Rightarrow X^{(1)} \Rightarrow \dots \Rightarrow X^{(n)}$, $X \stackrel{+}{\Rightarrow} X^{(n)}$ for $n > 0$, and $X \stackrel{*}{\Rightarrow} X^{(n)}$ for $n \geq 0$.

The elements of $\bigcap_{n=1}^{\infty} \mathbb{W}_n$ are called *infinitely rewritable words*. The other words are *finitely rewritable*. Given a word X , call the maximal (finite or infinite) sequence of the form $\langle X^{(0)}, X^{(1)}, X^{(2)}, \dots \rangle$ the *rewriting sequence* of X . Therefore, a word X is finitely (infinitely) rewritable if and only if the rewriting sequence of X is finite (infinite).

Say that $X \in \mathbb{A}^+$ is an *object* if $X \stackrel{*}{\Rightarrow} \omega$. Let \mathbb{O} be the set of all objects. Note that the rewriting sequence of every object $X \in \mathbb{O} \setminus \{\omega\}$ has the form $X = X^{(0)} \Rightarrow \dots \Rightarrow X^{(n)} \rightarrow \omega$, where $n \geq 0$.

From the above notation and definitions it is clear that we treat a system $\langle \rightarrow, \omega \rangle$ as a rewriting system and assume that only longest prefixes of words are subject to rewriting. The system is then regarded as a recognition device, with \mathbb{O} the accepted language (see [1]). We have called such a system “deterministic,” since every rewritable word has a unique rewrite.

We may regard the notion of an object as isolating “concepts” from “senseless words.” An object is a word X possessing a “meaning,” the rewrite $X' = X^{(1)}$, which also possesses a meaning, $(X')' = X^{(2)}$, and so on up to the final rewrite, the “generic object” ω , whose meaning is assumed predefined. The relation \rightarrow is thus treated as conceptual definition, and a rule $X \rightarrow Y$ is regarded as a definition of X via Y : “ X is a Y .” Next, a rule $X\alpha \rightarrow Z$ is an attribute definition, “the α of X is a Z ,” while $X\alpha\beta \rightarrow Z$ means “the β of the α of X is a Z ,” etc. In this respect, condition (a) imposed on the relation \rightarrow amounts to conceptual unambiguity (no concept can have several meanings).

We may also treat the relation \rightarrow as object-oriented inheritance or instantiation and regard a rule $X \rightarrow Y$ as an explicit indication of the fact that “class X directly inherits class Y ” or “object X is an instance of class Y .” Next, a rule $X\alpha \rightarrow Z$ may be regarded as an attribute declaration or property evaluation: “class X has attribute α of class Z ” or “the property $X\alpha$ has value Z ” or “the property $X\alpha$ is an instance of class Z .” In this respect, having imposed condition (a) on the relation \rightarrow , we thereby disallowed multiple inheritance (therefore, no object can belong to several incomparable classes).

Introduce the binary relation \Rightarrow_w on \mathbb{A}^+ by setting $X \Rightarrow_w Y$ if and only if $X = ES$ and $Y = E'S$ for some $E \in \mathbb{E}$, $S \in \mathbb{A}^*$. Therefore, \Rightarrow_w is the rewriting corresponding to the system $\langle \rightarrow, \omega \rangle$ regarded as an ordinary prefix rewriting system rather than a longest-prefix rewriting system. (It is clear that $X \rightarrow Y$ implies $X \Rightarrow Y$, and $X \Rightarrow Y$ implies $X \Rightarrow_w Y$. We may thus read the formulas $X \stackrel{+}{\Rightarrow} Y$ and $X \stackrel{\pm}{\Rightarrow}_w Y$ as “ X rewrites to Y ” and “ X weakly rewrites to Y .” The formula $X \rightarrow Y$ can be read as “ X explicitly rewrites to Y .”)

Introduce the binary relation \rightarrow on \mathbb{A}^+ by setting $X \rightarrow Y$ if and only if $X \supset Y$ or $X \Rightarrow_w Y$. As is easily seen, the transitive closure \rightarrow^{\pm} is the least transitive relation on \mathbb{A}^+ possessing the following three properties for all $X, Y, S \in \mathbb{A}^+$:

$$\text{if } X \rightarrow Y \text{ then } X \rightarrow^{\pm} Y; \quad \text{if } X \rightarrow Y \text{ then } XS \rightarrow^{\pm} YS; \quad XS \rightarrow^{\pm} X.$$

In case $X \rightarrow^{\pm} Y$ we say that X *depends on* Y . A word X is *well-defined* if X does not depend on X . Say that a system $\langle \rightarrow, \omega \rangle$ under consideration is *conceptually consistent* if all words are well-defined, i.e., no word depends on itself. For brevity, introduce the following

named condition:

The system is conceptually consistent. (Con)

The above terminology is justified by our informal treatment of a rewriting rule $X \rightarrow Y$ as a definition of X via Y (“ X is a Y ”) and regarding a rule $XS \rightarrow Z$ as a detail definition (“the S of X is a Z ”). Therefore, informally, the relation $X \overset{\pm}{\rightarrow} Y$ can be understood as follows: the definition of X explicitly or implicitly employs Y ; in particular, when subsequently describing a conceptual scheme, the concept Y should be introduced before X , otherwise X becomes ill-defined.

If \rightsquigarrow is a binary relation on \mathbb{A}^+ , put $|\rightsquigarrow| := \{X \in \mathbb{A}^+ : E \overset{*}{\rightsquigarrow} X \text{ for some } E \in \mathbb{E}\}$ and denote by $[\rightsquigarrow]$ the directed graph whose nodes are the words in $|\rightsquigarrow|$ and arcs are the pairs $\langle X, Y \rangle$ such that $X, Y \in |\rightsquigarrow|$ and $X \rightsquigarrow Y$.

Given a system, we call $[\rightarrow]$ the *conceptual scheme* and $[\Rightarrow_w]$ the *weak rewriting scheme*. Since $X \Rightarrow_w Y$ implies $X \rightarrow Y$, the weak rewriting scheme is a subgraph of the conceptual scheme.

Proposition 1. *Given $X, Y \in \mathbb{A}^+$, we have $X \overset{\pm}{\rightarrow} Y$ if and only if $X \sqsupset Y$ or $X \overset{\pm}{\Rightarrow}_w YS$ for some $S \in \mathbb{A}^*$.*

Say that a word X is *weakly recurrent* if $X \overset{\pm}{\Rightarrow}_w XS$ for some $S \in \mathbb{A}^*$.

Corollary 2. *A word is well-defined if and only if it is not weakly recurrent.*

Theorem 3. *The following properties of a system are equivalent:*

- (1) *all words are well-defined, i.e., (Con) holds;*
- (2) *each explicit word is well-defined;*
- (3) *there are no weakly recurrent explicit words;*
- (4) *there are no weakly recurrent words;*
- (5) *the conceptual scheme is acyclic;*
- (6) *the conceptual scheme is acyclic and finite;*
- (7) *the weak rewriting scheme is acyclic and finite.*

Say that a word X is *recurrent* if $X \overset{\pm}{\Rightarrow} XS$ for some $S \in \mathbb{A}^*$. Introduce the following named condition:

There are no recurrent words. (Rec)

Proposition 4. *(Con) implies (Rec).*

Put $\mathbb{A}_{\mathbb{E}} := \min\{A \subseteq \mathbb{A} : \mathbb{E} \subseteq A^+\}$, i.e., $\mathbb{A}_{\mathbb{E}}$ is the *explicit alphabet*, the set of all letters occurred in explicit words. In addition, put $\mu := \max\{|E| : E \in \mathbb{E}\}$.

Theorem 5. *If each word $X \in \mathbb{A}_{\mathbb{E}}^+$ with $|X| \leq \mu$ is not recurrent then all words are not recurrent, i.e., (Rec) holds.*

REMARK 6. Let $B(\mathcal{S})$ be a set of words defined via a system \mathcal{S} by some condition. Say that $B(\mathcal{S})$ is a *recurrence basis* if, given an arbitrary system \mathcal{S} , nonrecurrence of all words in $B(\mathcal{S})$ implies nonrecurrence of all words. Theorem 5 states that the set $\{X \in \mathbb{A}_{\mathbb{E}}^+ : |X| \leq \mu\}$ is a recurrence basis. Despite its finiteness, the set can be rather large. However, we are not aware of conditions which determine considerably smaller recurrence bases. (There are examples showing that neither the set \mathbb{E} of explicit words, nor the set of all prefixes of the explicit words can serve as a recurrence basis.)

Introduce the following named condition:

All words are finitely rewritable. (Fin)

Theorem 7. *If each explicit word is finitely rewritable then all words are finitely rewritable, i.e., (Fin) holds.*

Theorem 8. (Rec) implies (Fin).

Therefore, according to Proposition 4 and Theorem 8, we have the implications (Con) \Rightarrow (Rec) \Rightarrow (Fin). As examples show, the converse implications are not true in general.

Introduce the following two named conditions:

All explicit words are objects, i.e., $\mathbb{E} \subseteq \mathbb{O}$. (Obj)

If $X \in \mathbb{A}^+$, $\alpha \in \mathbb{A}$, and $X\alpha \in \mathbb{E}$ then $X \in \mathbb{O}$. (PreObj)

Proposition 9. (1) *Let $X, Y \in \mathbb{A}^+$, $X \xrightarrow{*} Y$. Then $X \in \mathbb{O}$ if and only if $Y \in \mathbb{O}$.*

(2) *Assume (Obj). Then, given $X \in \mathbb{A}^+$, we have $X \in \mathbb{O} \setminus \{\omega\}$ if and only if $X \xrightarrow{*} E$ for some $E \in \mathbb{E}$.*

(3) *Assume (PreObj). If $X, S \in \mathbb{A}^+$ and $XS \in \mathbb{O}$ then $X \in \mathbb{O}$.*

Let $X \in \mathbb{O}$ and $\alpha \in \mathbb{A}$. Say that α is an *attribute* of X if $X\alpha \in \mathbb{O}$. Denote by $\|X\|_1$ the set of all attributes of X . It is clear that $\|\omega\|_1 = \emptyset$.

Say that α is an *explicit attribute* (resp. *implicit attribute*) of X if $X\alpha \in \mathbb{O} \cap \mathbb{E}$ ($X\alpha \in \mathbb{O} \setminus \mathbb{E}$).

Say that α is an *overriding attribute* (resp. *added attribute*) of X if $X\alpha \in \mathbb{O} \cap \mathbb{E}$ and, in addition, $X'\alpha \in \mathbb{O}$ ($X'\alpha \notin \mathbb{O}$). If α is an overriding attribute of X , we say that $X\alpha$ *overrides* $X'\alpha$.

Therefore, every attribute is either explicit or implicit, and every explicit attribute is either overriding or added.

Proposition 10. *For all $X \in \mathbb{O} \setminus \{\omega\}$ and $\alpha \in \mathbb{A}$ the following hold:*

(1) *α is an implicit attribute of X if and only if $X\alpha \notin \mathbb{E}$ and $X'\alpha \in \mathbb{O}$;*

(2) *α is an added attribute of X if and only if $X\alpha \in \mathbb{O}$ and $X'\alpha \notin \mathbb{O}$.*

Proposition 11. *Assume (Obj). If $X, Y \in \mathbb{A}^+$, $\alpha \in \mathbb{A}$, $X \xrightarrow{*} Y$, and $Y\alpha \in \mathbb{O}$ then $X\alpha \in \mathbb{O}$.*

Proposition 12. *Assume (Obj). Consider the rewriting sequence $X = X^{(0)} \Rightarrow \dots \Rightarrow X^{(n)} = \omega$ of an object X . If $\alpha \in \|X\|_1$ then there is a number $0 \leq i \leq n$ such that $X^{(0)}\alpha, \dots, X^{(i)}\alpha \in \mathbb{O}$, $X^{(i+1)}\alpha, \dots, X^{(n)}\alpha \notin \mathbb{O}$, and α is an added attribute of $X^{(i)}$.*

Corollary 13. *Assume (Obj). A letter α is an attribute of an object X if and only if there is a number $n \geq 0$ such that $X^{(n)}\alpha \in \mathbb{E}$.*

Given an object X , say that δ is a *detail* of X if $\delta \in \mathbb{A}^+$ and $X\delta \in \mathbb{O}$. Denote by $\|X\|$ the set of all details of X and call $\|X\|$ the *type* of X . (It is clear that $\|\omega\| = \emptyset$.) Note that the set \mathbb{O} of all objects can be infinite and, moreover, some object types $\|X\|$ can be infinite. On the other hand, we will see that the set $\{\|X\| : X \in \mathbb{O}\}$ of all object types is always finite (see Theorem 27).

Proposition 14. *For all objects X and Y we have*

(1) *if $\|X\| = \|Y\|$ then $\|X\delta\| = \|Y\delta\|$ for all $\delta \in \|X\|$;*

(2) *if $\|X\| \subseteq \|Y\|$ then $\|X\delta\| \subseteq \|Y\delta\|$ for all $\delta \in \|X\|$.*

On assuming (PreObj), we also have

(3) *$\|X\| = \|Y\|$ if and only if $\|Y\|_1 = \|X\|_1$ and $\|X\alpha\| = \|Y\alpha\|$ for all $\alpha \in \|X\|_1$;*

(4) *$\|X\| \subseteq \|Y\|$ if and only if $\|X\|_1 \subseteq \|Y\|_1$ and $\|X\alpha\| \subseteq \|Y\alpha\|$ for all $\alpha \in \|X\|_1$.*

Proposition 15. *If $X, Y \in \mathbb{O}$, $X \Rightarrow Y$, $XS \notin \mathbb{E}$ for all $S \in \mathbb{A}^+$ then $\|X\| = \|Y\|$.*

If we informally interpret the relation $X \stackrel{\pm}{\Rightarrow} Y$ as “ X inherits Y ” (or “ X is a particular case of Y ” or “ X is a Y ”) and treat the objects $X\alpha$ as “properties of X ,” then in case $X \stackrel{\pm}{\Rightarrow} Y$

the object X should in a sense inherit the properties of Y and optionally make them more concrete and enlarge their totality. Formally, this requirement amounts to the following:

$$\text{if } X, Y \in \mathbb{O} \text{ and } X \overset{\pm}{\Rightarrow} Y \text{ then } \|X\| \supseteq \|Y\|. \quad (*)$$

Introduce the following named condition:

$$\begin{aligned} &\text{If } X, Y \in \mathbb{A}^+, \alpha \in \mathbb{A}, X\alpha \in \mathbb{E}, Y\alpha \in \mathbb{O}, \text{ and } X \Rightarrow Y \\ &\text{then } X\alpha \in \mathbb{O} \text{ and } \|X\alpha\| \supseteq \|Y\alpha\|. \end{aligned} \quad (\text{CoInh})$$

Theorem 16. *Assume (PreObj). The following are equivalent:*

- (1) *condition (CoInh) is satisfied;*
- (2) *if $X, Y \in \mathbb{O} \setminus \{\omega\}$ and $X \Rightarrow Y$ then $\|X\| \supseteq \|Y\|$;*
- (3) *if $X, Y \in \mathbb{O}$ and $X \overset{*}{\Rightarrow} Y$ then $\|X\| \supseteq \|Y\|$.*

Therefore, with (PreObj) satisfied, (*) is equivalent to (CoInh).

Corollary 17. *Assume (PreObj) and (CoInh). If $X, Y, S \in \mathbb{A}^+$, $X \overset{*}{\Rightarrow} Y$, and $YS \in \mathbb{O}$ then $XS \in \mathbb{O}$ and $\|XS\| \supseteq \|YS\|$. In particular, if $Y \in \mathbb{O}$ and $X \overset{*}{\Rightarrow} Y$ then $\|Y\|_1 \supseteq \|X\|_1$ and $\|X\alpha\| \supseteq \|Y\alpha\|$.*

Object systems with attribute value typing usually satisfy the following (or analogous) requirements: Suppose that a class Y has a declared attribute α with value type τ . If x is an instance of Y then x has attribute α whose value type is equal to or more concrete than τ . Similarly, if X is a class inherited from Y then X has the inherited attribute α whose value type is equal to or more concrete than τ . If we interpret the relation $X \Rightarrow Y$ as “the object X is an instance of the class Y ” or “the class X directly inherits the class Y ” and treat the relation $X\alpha \rightarrow V$ as “ V is the explicit value of the property $X\alpha$ ” or “ V is the declared value class of the attribute α within the class X ,” then the above requirements can be formalized by the following named condition:

$$\begin{aligned} &\text{If } X, Y \in \mathbb{A}^+, \alpha \in \mathbb{A}, Y\alpha \in \mathbb{O}, X\alpha \rightarrow V, \text{ and } X \Rightarrow Y \\ &\text{then } V \in \mathbb{O} \text{ and } \|V\| \supseteq \|Y\alpha\|. \end{aligned} \quad (\text{CoVal})$$

Theorem 18. *Conditions (PreObj) and (CoVal) imply (CoInh).*

The conceptual dependence was introduced above as a relation on the set of all words. As soon as non-object words are regarded as “senseless,” it is reasonable to describe dependence between objects involving objects only; namely, if a concept X depends on a concept Y , then there should be a chain of concepts (rather than arbitrary words) connecting X with Y . This principle is justified by the following theorem (see also Corollary 20).

Theorem 19. *Assume (Obj), (PreObj), and (CoInh). Given $X, Y \in \mathbb{O}$, X depends on Y if and only if there exist $X_1, \dots, X_n \in \mathbb{O}$, $n \geq 1$, such that $X \rightsquigarrow X_1 \rightsquigarrow \dots \rightsquigarrow X_n = Y$.*

Corollary 20. *Assume (Obj), (PreObj), and (CoInh). The restriction of $\overset{\pm}{\rightsquigarrow}$ onto \mathbb{O} is the least transitive relation on \mathbb{O} possessing the following three properties for all $X, Y \in \mathbb{O}$ and $\delta \in \mathbb{A}^+$:*

- if $X \rightarrow Y$ then $X \overset{\pm}{\rightsquigarrow} Y$;*
- if $X \rightarrow Y$ and $\delta \in \|X\| \cap \|Y\|$ then $X\delta \overset{\pm}{\rightsquigarrow} Y\delta$;*
- if $\delta \in \|X\|$ then $X\delta \overset{\pm}{\rightsquigarrow} X$.*

The last assertion shows that, with (Obj), (PreObj), and (CoInh) satisfied, the conceptual dependence relation between objects can be described in full conformity with the initial

definition of this relation on the set of all words. The only distinction consists in the fact that the latter description does not go beyond the set of objects.

REMARK 21. In Theorem 19 and Corollary 20, none of the conditions (Obj), (PreObj), or (CoInh) can be omitted.

3. Algorithmization

The rest of the paper is devoted to the effective verification of various properties of rewriting systems under consideration, and the following theorem is the main step in this direction.

Theorem 22. *Given a system, put $\mu := \max\{|E| : E \in \mathbb{E}\}$. A word X is infinitely rewritable if and only if one of the following two (mutually exclusive) conditions holds:*

- (a) *there are integers $n \geq 0$ and $r > 0$ such that $X^{(n)} = X^{(n+r)}$;*
- (b) *there are integers $n \geq 0$ and $r > 0$ such that*

$$\begin{aligned} \mu &\leq |X^{(n)}| \leq |X^{(n+1)}|, \dots, |X^{(n+r)}|, \\ X^{(n)} &\neq X^{(n+r)}, \quad X^{(n)} \downarrow_{\mu} = X^{(n+r)} \downarrow_{\mu}. \end{aligned}$$

In case (a) we have $X \xrightarrow{*} \underbrace{X^{(n)} \Rightarrow \dots \Rightarrow X^{(n+r-1)}}_{\text{period}} \Rightarrow X^{(n)} \Rightarrow \dots$. In case (b) put

$Y = X^{(n)} \downarrow_{\mu}$ and let $S \in \mathbb{A}^*$ be such that $X^{(n)} = YS$. Then there is a word $R \in \mathbb{A}^+$ such that $Y^{(r)} = YR$ and the rewriting sequence $\langle X^{(0)}, X^{(1)}, \dots \rangle$ contains a subsequence constituted by the words $X^{(n+mr)} = YR^m S$, $m \geq 0$, each of which starts a regular “growth period” of length r :

$$\begin{aligned} X &\xrightarrow{*} X^{(n)} = YS \Rightarrow Y^{(1)}S \Rightarrow \dots \Rightarrow Y^{(r-1)}S \\ &\Rightarrow X^{(n+r)} = YRS \Rightarrow Y^{(1)}RS \Rightarrow \dots \Rightarrow Y^{(r-1)}RS \\ &\Rightarrow X^{(n+2r)} = YR^2S \Rightarrow Y^{(1)}R^2S \Rightarrow \dots \Rightarrow Y^{(r-1)}R^2S \\ &\Rightarrow \dots \\ &\Rightarrow X^{(n+mr)} = YR^m S \Rightarrow Y^{(1)}R^m S \Rightarrow \dots \Rightarrow Y^{(r-1)}R^m S \\ &\Rightarrow \dots \end{aligned}$$

In particular, $\{X^{(n)}, X^{(n+1)}, \dots\} = \{Y^{(j)}R^m S : 0 \leq j < r, m \geq 0\}$.

Let \mathcal{P} be an arbitrary set of “constructive entities” (i.e., a set whose elements can be used as inputs for algorithms) and let $C(Y, p)$ be a condition imposed on words $Y \in \mathbb{A}^+$ with additional parameters in \mathcal{P} . Formally we may assume that C is a subset of $\mathbb{A}^+ \times \mathcal{P}$ and, for all $Y \in \mathbb{A}^+$, $p \in \mathcal{P}$, the expression $C(Y, p)$ means the containment $\langle Y, p \rangle \in C$.

Given $C(Y, p)$ as above, introduce the condition $C'(Y, R, S, p)$ for $Y, R \in \mathbb{A}^+$, $S \in \mathbb{A}^*$, $p \in \mathcal{P}$ as follows:

$$C'(Y, R, S, p) \text{ if and only if there is an } m \geq 1 \text{ such that } C(YR^m S, p).$$

Say that $C(Y, p)$ is *cyclically decidable* if the following two conditions hold:

- (a) there is an algorithm verifying $C(Y, p)$ for $Y \in \mathbb{A}^+$, $p \in \mathcal{P}$;
- (b) there is an algorithm verifying $C'(Y, R, S, p)$ for $Y, R \in \mathbb{A}^+$, $S \in \mathbb{A}^*$, $p \in \mathcal{P}$.

Note that (a) does not in general imply (b), which fact can be derived from existence of a recursive set $C \subseteq \mathbb{N}^2$ such that the set $\{n \in \mathbb{N} : (\exists m \in \mathbb{N}) \langle m, n \rangle \in C\}$ is not recursive (see, for instance, [2, Chapter C.1, § 6]).

Let $C(Y, p)$ be a cyclically decidable condition. Theorem 22 justifies the following simple algorithm which, given a system, a word $X \in \mathbb{A}^+$, and a parameter p , verifies existence of a word $Y \in \mathbb{A}^+$ such that $X \xrightarrow{*} Y$ and $C(Y, p)$.

Algorithm 23. [Is there a word Y such that $X \xrightarrow{*} Y$ and $C(Y, p)$?]

- If $C(X, p)$, return **Yes**. If X is not rewritable, return **No**.
- Otherwise, subsequently calculate the rewrites $X^{(1)}, \dots, X^{(i)}, \dots$ and, at each step $i \geq 1$, subsequently analyze the fragments $\langle X^{(n)}, X^{(n+1)}, \dots, X^{(n+r)} \rangle$ for $0 \leq n < n+r = i$, as follows:
 - If $C(X^{(n+r)}, p)$, return **Yes**. If $X^{(n)} = X^{(n+r)}$, return **No**.
 - If $\langle X^{(n)}, \dots, X^{(n+r)} \rangle$ satisfies condition (b) of Theorem 22, put $Y := X^{(n)} \downarrow_{\mu}$; let $S \in \mathbb{A}^*$ be such that $X^{(n)} = YS$; let $R \in \mathbb{A}^+$ be such that $Y^{(r)} = YR$ (such an R exists by Theorem 22); if $C'(Y, R, S, p)$, return **Yes**; otherwise return **No**.
 - If $X^{(n+r)}$ is not rewritable, return **No**. Otherwise proceed to the next step, $i+1$.

By Theorem 22, the above procedure terminates for every input.

As is easily seen, given a system \mathcal{S} , the condition $C(Y, \mathcal{S}) = \text{“}Y \text{ is not rewritable within } \mathcal{S}\text{”}$ is cyclically decidable. Therefore, specialized with this condition, Algorithm 23 verifies finite rewritability of a given word within a given system. A simplified version is presented below.

Algorithm 24. [Is a word X finitely rewritable?] Start subsequent calculation of the rewrites $X^{(0)}, X^{(1)}, \dots$. If a nonrewritable word $X^{(i)}$ occurs, return **Yes**. Otherwise, according to Theorem 22, a fragment $\langle X^{(n)}, X^{(n+1)}, \dots, X^{(n+r)} \rangle$ will occur which satisfies (a) or (b) of Theorem 22; in this case return **No**.

Since the set \mathbb{E} of explicit words is finite, Theorem 7 implies that (Fin) is effectively verifiable: it suffices to apply Algorithm 24 to all explicit words.

The specialization of Algorithm 23 with the condition $C(Y) = \text{“}Y = \omega\text{”}$ checks if a given word is an object. (This can be also verified by a slight modification of Algorithm 24.) It is now clear that (Obj) and (PreObj) are effectively verifiable.

Note that if (Fin) holds, the containment $X \in \mathbb{O}$ can be trivially verified: just check if the (finite) rewriting sequence of X ends with ω . In addition, if (Obj) holds, we can stop calculating the rewriting sequence of X if $X^{(n)} \in \mathbb{E}$ at some step $n \geq 0$; furthermore, if both (Obj) and (PreObj) hold, the calculation can be terminated if some $X^{(n)}$ becomes a prefix of any explicit word (see Proposition 9).

As is easily seen, the condition $C(Y, X) = \text{“}Y \sqsupseteq X\text{”}$ is cyclically decidable. Therefore a properly specialized version of Algorithm 23 checks if a given word X is recurrent. By Theorem 5, we conclude that (Rec) is effectively verifiable: to check if all words are not recurrent, it suffices to apply the algorithm to all words over $\mathbb{A}_{\mathbb{E}}$ of length at most μ . (However, the resultant verification occurs exponential-time; see Remark 6.)

Another approach to verifying (Rec) can be based on Theorem 8 which states that (Rec) implies (Fin). Check (Fin) first. If it fails then (Rec) also fails. If (Fin) holds true, condition (Rec) can be verified by processing all words X over $\mathbb{A}_{\mathbb{E}}$ of length at most μ and returning **Yes** whenever an X occurs such that $X \sqsubseteq X^{(n)}$ for some $n \geq 1$.

By Theorem 3, condition (Con) can be effectively verified by constructing the conceptual scheme and checking (during the construction) if the scheme is acyclic. The algorithm described below checks if a given system is conceptually consistent. If it is so, the algorithm returns the conceptual scheme of the system; otherwise it returns an example of a cycle in

the conceptual scheme. The algorithm uses a variable directed graph $\Gamma = \langle \Gamma_N, \Gamma_A \rangle$ (with Γ_N the nodes and Γ_A the arcs) and a variable \mathcal{A} whose values are finite subsets of $\mathbb{A}^+ \times \mathbb{A}^+$.

Algorithm 25. [Is a system conceptually consistent?]

- (1) Put $\Gamma_N := \mathbb{E}$, $\Gamma_A := \emptyset$.
- (2) Put $\mathcal{A} := \{\langle X, Y \rangle : X \text{ is a sink of } \Gamma \text{ and } X \succ Y\}$.
- (3) If $\mathcal{A} = \emptyset$, claim that the system is **conceptually consistent** and return Γ as the **conceptual scheme**.
- (4) Otherwise do the following for each pair $\langle X, Y \rangle \in \mathcal{A}$:
if $X = Y$ or Γ contains a path from Y to X ,
 claim that the system is **conceptually inconsistent**
 and return a **cycle**: $X \succ X$ or $Y \succ \dots \succ X \succ Y$;
 otherwise put $\Gamma_N := \Gamma_N \cup \{Y\}$, $\Gamma_A := \Gamma_A \cup \{\langle X, Y \rangle\}$.
- (5) Go to (2).

When applied to a conceptually inconsistent system, Algorithm 25 returns an example of a cycle $X_0 \succ X_1 \succ \dots \succ X_m = X_0$ in the conceptual scheme, but it is not guaranteed that all words X_i in the cycle are objects (even in case X_0 is an object). On the other hand, Theorem 19 implies that, with (Obj), (PreObj), and (CoInh) satisfied, every path $X_0 \succ X_1 \succ \dots \succ X_m$ between objects X_0, X_m can be transformed into a path of *objects* $Y_0 \succ Y_1 \succ \dots \succ Y_n$, with $Y_0 = X_0$ and $Y_n = X_m$. We note that such a transformation can be performed *effectively*.

By slightly modifying Algorithm 25, we can obtain a procedure for constructing the weak rewriting scheme rather than the conceptual scheme. The algorithm presented below checks in an arbitrary system if there are weakly recurrent words, and either returns an example of such a word or constructs the weak rewriting scheme of the system.

Algorithm 26. [Is there a weakly recurrent word?]

- (1) Put $\Gamma_N := \mathbb{E}$, $\Gamma_A := \emptyset$.
- (2) Put $\mathcal{A} := \{\langle X, Y \rangle : X \text{ is a sink of } \Gamma \text{ and } X \Rightarrow_w Y\}$.
- (3) If $\mathcal{A} = \emptyset$, claim that **there are no weakly recurrent words** and return Γ as the **weak rewriting scheme**.
- (4) Otherwise do the following for each pair $\langle X, Y \rangle \in \mathcal{A}$:
if $X \sqsubseteq Y$ or Γ contains a path from a prefix $Y_0 \sqsubseteq Y$ to X ,
 return a **weakly recurrent word**:
 $X \Rightarrow_w Y \sqsupseteq X$ or $Y_0 \Rightarrow_w \dots \Rightarrow_w X \Rightarrow_w Y \sqsupseteq Y_0$;
 otherwise put $\Gamma_N := \Gamma_N \cup \{Y\}$, $\Gamma_A := \Gamma_A \cup \{\langle X, Y \rangle\}$.
- (5) Go to (2).

According to Theorem 3, the weak rewriting scheme of a conceptually inconsistent system includes a path $X_0 \stackrel{\pm}{\Rightarrow}_w X_n \sqsupseteq X_0$ with $X_0 \in \mathbb{E}$. Given an inconsistent system, Algorithm 25 returns an example of a path $Y_0 \stackrel{\pm}{\Rightarrow}_w Y_n \sqsupseteq Y_0$, but the leading word Y_0 need not be explicit. In this connection it is worth noting that every path of the form $Y_0 \stackrel{\pm}{\Rightarrow}_w Y_n \sqsupseteq Y_0$ can be *effectively* transformed into a path $X_0 \stackrel{\pm}{\Rightarrow}_w X_n \sqsupseteq X_0$ (of the same length) with $X_0 \in \mathbb{E}$.

To the author's opinion, the most convincing indication of conceptual inconsistency is an example of some cycle $X_0 \succ \dots \succ X_n = X_0$ which is constituted by *objects* and starts with an *explicit* word X_0 . We note that, if an inconsistent system satisfies (Obj), (PreObj), and (CoInh), then a cycle of this kind can be found *effectively*.

We now turn to the effective analysis of object types.

Say that $C \in \mathbb{O}$ is a *character* if either $C = \omega$ or $C \sqsubset E$ for some $E \in \mathbb{E}$. Let \mathbb{C} be the set of all characters. It is clear that \mathbb{C} is finite.

Given an object X , consider the rewriting sequence $X^{(0)} \Rightarrow \dots \Rightarrow X^{(n)} = \omega$ and denote by $\text{ch}(X)$ the first character in the sequence $\langle X^{(0)}, \dots, X^{(n)} \rangle$. Call $\text{ch}(X)$ the *character* of X .

It is clear that $\text{ch}(C) = C$ for all $C \in \mathbb{C}$. Therefore, $\mathbb{C} = \{\text{ch}(X) : X \in \mathbb{O}\}$.

Theorem 27. *For every object X we have $\|X\| = \|\text{ch}(X)\|$. Consequently, $\{\|X\| : X \in \mathbb{O}\} = \{\|C\| : C \in \mathbb{C}\}$, and the set $\{\|X\| : X \in \mathbb{O}\}$ is finite.*

By the *type comparison problem* we mean the following: given two objects X and Y , effectively compare the types of X and Y , i.e., determine which of the relations $\|X\| = \|Y\|$, $\|X\| \subseteq \|Y\|$, $\|X\| \supseteq \|Y\|$ hold.

It is clear that the character of an object can be effectively determined. Therefore, due to Theorem 27, the type comparison problem reduces to effective comparison of the character types.

Given $X \in \mathbb{C}$ and $\alpha \in \|X\|_1$, introduce the notation $X_\alpha := \text{ch}(X\alpha)$ and, given an arbitrary function $\tau : \mathbb{C} \rightarrow \{1, \dots, N\}$, define the following equivalence relation on \mathbb{C} :

$$X \underset{\tau}{\sim} Y \quad \text{if and only if} \quad \|X\|_1 = \|Y\|_1 \text{ and } \tau(X_\alpha) = \tau(Y_\alpha) \text{ for all } \alpha \in \|X\|_1.$$

The following algorithm uses two variable functions $\tau, \tilde{\tau} : \mathbb{C} \rightarrow \{1, 2, \dots\}$ (each of which can be encoded as an array of naturals indexed by the characters).

Algorithm 28. [Compute a character typing.]

(1) Define τ by putting $\tau(X) := 1$ for all $X \in \mathbb{C}$.

(2) If $X \underset{\tau}{\sim} Y$ for all $X, Y \in \mathbb{C}$ such that $\tau(X) = \tau(Y)$, **return** τ .

(3) Assign $\tilde{\tau}$ a copy of τ .

(4) For each $k \in \{\tau(X) : (\exists Y \in \mathbb{C})(X \underset{\tau}{\not\sim} Y \ \& \ \tau(X) = \tau(Y))\}$ do the following:

arbitrarily enumerate the set $\{X \in \mathbb{C} : \tau(X) = k\}$ thus making it a sequence $\langle X_1, \dots, X_n \rangle$, $n \geq 2$, and, for each $i = 2, \dots, n$, do the following:

if $X_i \underset{\tau}{\sim} X_j$ for some $1 \leq j < i$, reassign $\tilde{\tau}(X_i) := \tilde{\tau}(X_j)$;

otherwise reassign $\tilde{\tau}(X_i) := \max\{\tilde{\tau}(X) : X \in \mathbb{C}\} + 1$.

(5) Assign $\tau := \tilde{\tau}$ and go to (2).

Theorem 29. *Algorithm 28 halts for any system and, if the system meets (PreObj), the resultant function $\tau : \mathbb{C} \rightarrow \{1, \dots, N\}$ is such that $\tau(X) = \tau(Y)$ if and only if $\|X\| = \|Y\|$.*

REMARK 30. Algorithm 28 is analogous to Vizing's algorithm of partitioning the vertex set of a graph into classes of similar vertices (see [3]). The author is grateful to S. V. Avgustinovich for discovering the analogy.

The following algorithm uses two variable sets $\Sigma, \Sigma_0 \subseteq \mathbb{C}^2$.

Algorithm 31. [Compute the character subtyping.]

(1) Put $\Sigma := \mathbb{C}^2$.

(2) Put $\Sigma_0 := \{\langle X, Y \rangle \in \Sigma : \|X\|_1 \subseteq \|Y\|_1 \ \& \ (\forall \alpha \in \|X\|_1) \langle X_\alpha, Y_\alpha \rangle \in \Sigma\}$.

(3) If $\Sigma_0 = \Sigma$, **return** Σ . Otherwise reassign $\Sigma := \Sigma_0$ and go to (2).

Theorem 32. *Given any system, Algorithm 31 always halts and, if the system meets (PreObj), the resultant set Σ equals $\{\langle X, Y \rangle \in \mathbb{C}^2 : \|X\| \subseteq \|Y\|\}$.*

Therefore, given a system subject to (PreObj), we can effectively verify the relation $\|X\| \subseteq \|Y\|$ for $X, Y \in \mathbb{O}$. (As is easily seen, this implies that (CoInh) and (CoVal) are effectively verifiable.) However, the mere claim " $\|X\| \not\subseteq \|Y\|$ " is not always sufficient, and one may require a particular reason why the inclusion fails. Within a bulky system, it may occur nontrivial to find a particular detail $\delta \in \|X\| \setminus \|Y\|$, and a corresponding algorithm would thus be a useful troubleshooting tool. Recall that, due to Theorem 27, it suffices to automate the solution for characters only.

The following algorithm uses a variable set $\Delta \subseteq \mathbb{C}^2$ and a variable function $\delta : \Delta \rightarrow \mathbb{A}^+$.

Algorithm 33. [Compute a diagnosis for subtyping failure.]

(1) Put $\Delta := \emptyset$ and $n := 1$.

(2) For all pairwise distinct $X, Y \in \mathbb{C}$ do the following:

if there is an $\alpha \in \|X\|_1 \setminus \|Y\|_1$, add $\langle X, Y \rangle$ to Δ and assign $\delta(X, Y) := \alpha$.

(3) For all pairwise distinct $X, Y \in \mathbb{C}$ such that $\langle X, Y \rangle \notin \Delta$ do the following:

if there is an $\alpha \in \|X\|_1$ such that $\langle X_\alpha, Y_\alpha \rangle \in \Delta$ and $|\delta(X_\alpha, Y_\alpha)| = n$,

add $\langle X, Y \rangle$ to Δ and assign $\delta(X, Y) := \alpha \delta(X_\alpha, Y_\alpha)$.

(4) If there were no assignments at step (3), **return** δ . Otherwise put $n := n + 1$ and go to (3).

Theorem 34. Given any system, Algorithm 33 always halts and, if the system meets (PreObj), the resultant function $\delta : \Delta \rightarrow \mathbb{A}^+$ is such that

$$\Delta = \{\langle X, Y \rangle \in \mathbb{C}^2 : \|X\| \not\subseteq \|Y\|\},$$

$$\delta(X, Y) \in \|X\| \setminus \|Y\| \text{ for all } \langle X, Y \rangle \in \Delta,$$

$$|\delta(X, Y)| = \min\{|\delta'| : \delta' \in \|X\| \setminus \|Y\|\} \text{ for all } \langle X, Y \rangle \in \Delta.$$

Acknowledgments. The author is grateful to Sergei Vladimirovich Avgustinovich for fruitful discussions.

References

1. Salmaa A. Formal Languages.—N. Y.: Academic Press, 1973.—336 p.
2. Barwise J. (ed.) Handbook of Mathematical Logic.—Amsterdam: North-Holland, 1977.—1165 p.
3. Vizing V. G. Distributive Coloring of Graph Vertices // Diskretn. Anal. Issled. Oper.—1995.—Vol. 2, № 4.—P. 3–12.

Received October 17, 2013.

GUTMAN ALEXANDER EFIMOVICH

Sobolev Institute of Mathematics,

Head of the Laboratory of Functional Analysis

Acad. Koptyug av. 4, 630090, Novosibirsk, Russia;

Novosibirsk State University, Professor

Pirogova 2, 630090, Novosibirsk, Russia

E-mail: gutman@math.nsc.ru

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ДАННЫЕ
КАК ПЕРЕЗАПИСЫВАЮЩИЕ СИСТЕМЫ

Гутман А. Е.

Рассматриваются перезаписывающие системы, не содержащие пар правил вида $X \rightarrow Y$, $X \rightarrow Z$, где $Y \neq Z$, в которых перезаписи подлежат только самые длинные префиксы. В рамках таких систем определяются и исследуются аналоги концепций, характерных для систем объектно-ориентированных данных: наследование классов и объектов, экземпляры классов, атрибуты экземпляров и классов, концептуальная зависимость и непротиворечивость, концептуальные схемы, типы, подтипы и др. Особое внимание уделяется эффективной проверке разнообразных свойств рассматриваемых перезаписывающих систем. В частности, приводятся алгоритмы для ответа на следующие вопросы: Все ли слова конечно переписываемы? Существуют ли рекуррентные слова? Является ли система концептуально непротиворечивой? Концептуально зависит ли данное слово X от слова Y ? Совпадают ли типы X и Y ? Является ли тип X подтипом типа Y ?

Ключевые слова: префиксная перезаписывающая система, полугуэвская система, система объектно-ориентированных данных, информационная система, проверка непротиворечивости, онтология модели данных.