ISSN 0037–4466, Siberian Mathematical Journal, 2026, Vol. 67, No. 3, pp. 531–547. © Pleiades Publishing, Ltd., 2026.
Russian Text © The Author(s), 2026, published in Sibirskii Matematicheskii Zhurnal, 2026, Vol. 67, No. 3, pp. 392–413.

OBJECT-ORIENTED DATA VIA PREFIX REWRITING. PART I: OVERVIEW AND MAIN RESULTS

A. E. Gutman

UDC 519.682.1+519.683+519.7+519.1

Abstract—A deterministic longest-prefix rewriting system is a string-rewriting system such that there are no rewriting rules $X \to Y$, $X \to Z$ with $Y \neq Z$, and only the longest prefixes of words are subject to rewriting. For such a system, analogs of some concepts related to object-oriented data systems are defined and studied: inheritance of classes and objects, instances of classes, class and instance attributes, conceptual dependence and consistency, conceptual scheme, types and subtypes, etc. Special attention is paid to the effective verification of various properties of the rewriting systems under consideration.

DOI: 10.1134/S003744662603002X

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

Introduction

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

The *has* relation links objects (and classes) to their attributes. By saying "`X` *has a* `Y`" we mean that the object (or class) `X` has an attribute named `Y`; thus, we 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 specifies 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 stating that "`Bob.age` *is* `40`" we assign the value `40` to the `age` attribute of `Bob`.

As the above examples show, the broad interpretation of the *is* relation makes it possible to eliminate the difference between objects and classes. A single data system can combine class declarations ("metadata") with object instantiations and initializations ("data"). We do not assert that data and metadata should be combined rather than separated; nevertheless, this approach allows us to unify data analysis and develop a common tool for verifying conceptual and semantic consistency.

The *is* and *has* relations are naturally connected. Interpreting *is* as inheritance or instantiation, we assume that if "`X` *is* `Y`" then all the attributes of `Y` are inherited by the class or object `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 explicit information we mean the *is* rules forming the data system under consideration.) In doing so, we derive *implicit* information on `X` and say that "`X.Z` *is* `Y.Z` implicitly." Therefore, when evaluating the property `X.Z`, we rewrite its prefix `X` with `Y` according to the explicit rule "`X` *is* `Y`." The same applies to constructions of arbitrary length. For instance, if we know explicitly that "`Bob.Address` *is* `Alice.Work.Address`" then `Bob.Address.City.Code` rewrites implicitly to `Alice.Work.Address.City.Code`.

It is clear that explicit rules supersede all 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 has priority over the implicit "`X.Z` *is* `Y.Z`." However, a conflict of another kind is possible when several explicit rules are simultaneously applicable.

Original article submitted January 15, 2026; accepted April 7, 2026.

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 color in the button's style, i.e., `button.style.color`. Since `button` is a `block`, one might expect 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 have priority, since "`button.style` *is* `header.style`" seems to have priority over "`button` *is* `block`." The reason is not 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* because 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 applicable rule).

We will now discuss *data consistency*. Obviously, when designing a system of definitions, one should avoid conceptual cycles. By saying "`man` *is* `man`" we define nothing, since evaluation of `man` results in a dead cycle. However, conceptual consistency in no way rules out recursion. For instance, the rule "`man.son` *is* `man`" is quite legitimate. 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, whereas the latter is senseless before `man` is defined. 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, before defining a set of concepts, classes, or objects, we need at least one concept that 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 $\texttt{entity}\,.\,\texttt{attr}_1\,.\,\texttt{attr}_2\,.\cdots.\,\texttt{attr}_n$, 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 say that X is senseless. The possibility of (2) makes the analysis nontrivial and justifies the search for an effective verification of whether a given word makes sense. (This is close to analyzing the ontology of a concept within a data system.)

Given an object X and an attribute path D of the form $\texttt{attr}_1\,.\,\texttt{attr}_2\,.\cdots.\,\texttt{attr}_n$, say that D is a *detail* of X if $X.D$ makes sense. The set $\|X\|$ of all details of X can be regarded as the *type* of X. (Such an approach is known as "duck typing.") Whenever an algorithmic procedure assumes a formal argument A, the body of the procedure usually contains occurrences of A along with some words $A.D_i$. For the procedure to operate correctly with X substituted for A, it is necessary (and probably sufficient) that all the words $X.D_i$ 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 nontrivial, if only because 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 of the notions under consideration, state the main results, and present algorithms for all the problems mentioned above. 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 attribute paths $\alpha_1\,.\,\alpha_2\,.\cdots.\,\alpha_n$ as $\alpha_1\alpha_2\ldots\alpha_n$, thus making them words over $\mathbb{A}$. The explicit rules "X *is* Y" will be written as $X \to Y$.

We rely below on standard background from several classical areas and therefore recall only those definitions and facts that are used explicitly in the sequel. For prefix rewriting and closely related regular and prefix systems, the reader may consult Büchi's foundational paper on regular canonical systems [1], Caucal's study of the regular structure of prefix rewriting [2], the monograph of Book and Otto on string-rewriting systems [3], and the characterization of regular languages by prefix grammars due to Frazier and Page [4]. For the object-oriented and object-database viewpoint, useful classical sources include the survey of Cardelli and Wegner [5], the OODBS manifesto of Atkinson et al. [6], and Dittrich's overview of object-oriented data model concepts [7]. For graph-theoretic terminology and elementary facts used throughout, we refer to Harary's classical monograph [8].

This paper expands the set of definitions and results announced in [9] and is the first part of a planned series of publications devoted to the representation of object-oriented data by prefix rewriting systems. The paper is preliminary and contains no proofs of the statements or justifications of the algorithms. All details, including various examples, will be published in subsequent parts of the series.

The material of the paper is grouped into six sections. Section 1 gives the basic definitions concerning the rewriting systems under consideration and shows how such classical object-oriented notions as inheritance, member declaration, object instantiation, and property evaluation are interpreted in this framework. Section 2 considers rewriting systems as systems of formal definitions and studies conceptual consistency of such systems. Section 3 contains an interpretation of overriding, added, and implicit attributes, as well as the notion of the declaring class (origin) of an attribute. Section 4 introduces and studies in detail the notions of type and subtype in their relation to class inheritance and attribute evaluation. Section 5 is devoted to the effective analysis of rewriting and includes algorithms that verify such properties of a system as the absence of infinitely rewritable words and conceptual consistency. Section 6, in turn, is devoted to the analysis of type structure and contains, in particular, algorithms for comparing object types and finding concrete details that distinguish one type from another.

1. Deterministic Longest-Prefix Rewriting System

1.1. Throughout the paper, $\mathbb{A}$ is a finite alphabet, and $\mathbb{A}^*$ (respectively, $\mathbb{A}^+$) is the set of all words (respectively, all nonempty words) over $\mathbb{A}$. For brevity, in the sequel we use "*word*" as shorthand for "nonempty word over $\mathbb{A}$." The elements of $\mathbb{A}$ are called *letters*. We conventionally identify a letter with the corresponding single-letter word. The length of a word X is denoted by $|X|$. In what follows, $\mathbb{N}$ is the set of positive integers $\{1, 2, \dots\}$.

1.2. Say that X is a *prefix* (respectively, a *proper prefix*) of $Y \in \mathbb{A}^+$ and write $X \sqsubseteq Y$ or $Y \sqsupseteq X$ (respectively, $X \sqsubset Y$ or $Y \sqsupset X$) if $X \in \mathbb{A}^+$ and $Y = XS$ for some $S \in \mathbb{A}^*$ (respectively, $S \in \mathbb{A}^+$).

Given a positive integer $n \in \mathbb{N}$ and a word $X \in \mathbb{A}^+$ such that $|X| \geqslant n$, let $X\restriction_n$ denote the prefix of X having length n; i.e., define $X\restriction_n \in \mathbb{A}^+$ so that $X\restriction_n \sqsubseteq X$ and $\left|X\restriction_n\right| = n$.

1.3. If $\rightsquigarrow$ is a symbol denoting a binary relation, we conventionally denote by $\overset{*}{\rightsquigarrow}$ the reflexive transitive closure of $\rightsquigarrow$ and by $\overset{+}{\rightsquigarrow}$ the transitive closure of $\rightsquigarrow$. Thus, $x \overset{*}{\rightsquigarrow} y$ (respectively, $x \overset{+}{\rightsquigarrow} y$) means that

$$x = z_0 \rightsquigarrow z_1 \rightsquigarrow \cdots \rightsquigarrow z_n = y$$

for some $z_1, \dots, z_n$, where $n \geqslant 0$ (respectively, $n \geqslant 1$). In particular, $x \overset{*}{\rightsquigarrow} y$ if and only if $x = y$ or $x \overset{+}{\rightsquigarrow} y$.

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

(a) $X \to Y$ and $X \to Z$ imply $Y = Z$;

(b) there are no $S, Y \in \mathbb{A}^*$ such that $\omega S \to Y$.

In what follows, when giving definitions and formulating statements, we assume by default that a deterministic longest-prefix rewriting system $\langle \to, \omega \rangle$ is fixed, and all introduced notions, as well as the terms and notation used, are interpreted with respect to this system.

1.5. Put

$$\mathbb{E} := \{X : X \to Y \text{ for some } Y\}$$

and call the elements of $\mathbb{E}$ *explicit words*. Denote by $\mathbb{A}_{\mathbb{E}}$ the *explicit alphabet*, i.e., the set of all letters occurring in explicit words:

$$\mathbb{A}_{\mathbb{E}} := \min\{A \subseteq \mathbb{A} : \mathbb{E} \subseteq A^+\}.$$

We also denote by μ the length of the longest explicit word:

$$\mu := \max\{|E| : E \in \mathbb{E}\}.$$

1.6. Say that E is an *explicit prefix* of a word X if $E \in \mathbb{E}$ and $E \sqsubseteq X$. Say that X is *rewritable* if X has an explicit prefix.

As is easily seen, condition 1.4(a) means that, for each $E \in \mathbb{E}$, there is a unique word E' such that

$$E \to E',$$

while condition 1.4(b) means that all words of the form ωS, with $S \in \mathbb{A}^*$, are nonrewritable.

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

$$X' := E'S,$$

where $E \to E'$ (see 1.6). We call X' the *rewrite* of X. Introduce the binary relation $\Rightarrow$ on $\mathbb{A}^+$ by setting

$$X \Rightarrow Y \quad \text{if and only if} \quad X \text{ is a rewritable word and } X' = Y.$$

1.8. By recursion, put

$$\begin{aligned} &\mathbb{W}_0 := \mathbb{A}^+, && X^{(0)} := X \text{ for } X \in \mathbb{W}_0;\\ &\mathbb{W}_n := \{X \in \mathbb{W}_{n-1} : X^{(n-1)} \text{ is rewritable}\}, && X^{(n)} := (X^{(n-1)})' \text{ for } X \in \mathbb{W}_n, \quad n \geqslant 1. \end{aligned}$$

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

$$\begin{aligned} &X = X^{(0)} \Rightarrow X^{(1)} \Rightarrow \cdots \Rightarrow X^{(n)};\\ &X \overset{*}{\Rightarrow} X^{(n)} \text{ for } n \geqslant 0, \quad X \overset{+}{\Rightarrow} X^{(n)} \text{ for } n > 0. \end{aligned}$$

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

$$\langle X^{(0)}, X^{(1)}, X^{(2)}, \ldots \rangle$$

the *rewriting sequence* of X. Thus, a word X is finitely (infinitely) rewritable if and only if the rewriting sequence of X is finite (infinite).

1.9. Say that a word $X \in \mathbb{A}^+$ is an *object* if $X \overset{*}{\Rightarrow} \omega$. Let $\mathbb{O}$ be the set of all objects:

$$\mathbb{O} := \{X \in \mathbb{A}^+ : X \overset{*}{\Rightarrow} \omega\}.$$

Note that the rewriting sequence of every object $X \in \mathbb{O}\backslash\{\omega\}$ has the form

$$X = X^{(0)} \Rightarrow \cdots \Rightarrow X^{(n)} \to \omega, \quad n \geqslant 0.$$

According to 1.4(b), the generic object ω is not rewritable, and ω is the only nonrewritable object.

1.10. From the notation and definitions above it is clear that we treat a system $\langle \to, \omega \rangle$ as a prefix-rewriting system and assume that only the longest prefixes of words are subject to rewriting. The system is then regarded as a recognition device, with $\mathbb{O}$ as the accepted language (see [10]). We call such a system "deterministic," since every rewritable word has a unique rewrite.

We may regard the notion of an object as separating "concepts" from "senseless words." An object is a word X possessing a "meaning," namely 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 to be predefined. The relation $\to$ is thus treated as a means of conceptual definition, and a rule $X \to Y$ is regarded as a definition of the concept X: "X is a Y." Next, a rule $X\alpha \to Z$ is a property definition, "the α of X is a Z," while $X\alpha\beta \to Z$ gives the definition "the β of the α of X is a Z," etc. In this respect, condition 1.4(a) imposed on the relation $\to$ amounts to conceptual unambiguity (no concept can have several meanings). Condition 1.4(b) implies that the generic object ω has no meaningful attributes.

We may also treat the relation $\to$ as object-oriented inheritance or instantiation and regard a rule $X \to Y$ 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 $X\alpha \to Z$ 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, by imposing condition 1.4(a) on the relation $\to$, we thereby disallow multiple inheritance (and, consequently, no object can belong to several incomparable classes). Moreover, according to 1.4(b), the generic object ω does not possess declared properties.

1.11. Let $\langle \mathbb{O}, \Leftarrow \rangle$ be the directed graph whose vertices are the objects and whose arcs are the pairs $\langle X, Y \rangle$ such that $X, Y \in \mathbb{O}$ and $Y \Rightarrow X$.

Theorem. *The graph $\langle \mathbb{O}, \Leftarrow \rangle$ is a tree with finite levels.*

We call $\langle \mathbb{O}, \Leftarrow \rangle$ the *inheritance tree*. It is clear that its root is the generic object ω. We point out that the inheritance tree may be infinite and may even have no leaves.

1.12. Introduce the binary relation $\Rightarrow_{\mathrm{w}}$ on $\mathbb{A}^+$ by setting

$$X \Rightarrow_{\mathrm{w}} Y \text{ if and only if } X = ES \text{ and } Y = E'S \text{ for some } E \in \mathbb{E},\ S \in \mathbb{A}^*.$$

Thus, $\Rightarrow_{\mathrm{w}}$ is the rewriting corresponding to the system $\langle \to, \omega \rangle$ regarded as an ordinary prefix-rewriting system rather than a longest-prefix rewriting system. It is clear that

$$X \to Y \text{ implies } X \Rightarrow Y, \text{ and } X \Rightarrow Y \text{ implies } X \Rightarrow_{\mathrm{w}} Y.$$

We may thus read the formulas $X \overset{+}{\Rightarrow} Y$ and $X \overset{+}{\Rightarrow}_{\mathrm{w}} Y$ as "X *rewrites to* Y" and "X *weakly rewrites to* Y." The formula $X \to Y$ can be read as "X *explicitly rewrites to* Y."

1.13. Introduce the binary relation $\rightarrowtail$ on $\mathbb{A}^+$ by setting

$$X \rightarrowtail Y \text{ if and only if } X \sqsupset Y \text{ or } X \Rightarrow_{\mathrm{w}} Y.$$

As is easily seen, the transitive closure $\overset{+}{\rightarrowtail}$ is the least transitive relation on $\mathbb{A}^+$ possessing the following three properties for all $X, Y, S \in \mathbb{A}^+$:

(a) if $X \to Y$ then $X \overset{+}{\rightarrowtail} Y$;

(b) if $X \to Y$ then $XS \overset{+}{\rightarrowtail} YS$;

(c) $XS \overset{+}{\rightarrowtail} X$.

In case $X \overset{+}{\rightarrowtail} Y$ we say that X *conceptually depends on* Y. A word X is *well-defined* if X does not conceptually depend on X.

2. Conceptual Consistency

2.1. Say that a system $\langle \to, \omega \rangle$ under consideration is *conceptually consistent* if all words are well-defined, i.e., no word conceptually 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 $X \to Y$ as a definition of X in terms of Y ("X is a Y") and of a rule $XS \to Z$ as a definition of a detail ("the S of X is a Z"). Therefore, informally, the relation $X \stackrel{+}{\rightarrowtail} Y$ can be understood as follows: the definition of X explicitly or implicitly uses Y; in particular, when subsequently describing a conceptual scheme, the concept Y should be introduced before X; otherwise, X becomes ill-defined.

2.2. Say that a word X is *recurrent* (respectively, *weakly recurrent*) if $X \stackrel{+}{\Rightarrow} XS$ (respectively, $X \stackrel{+}{\Rightarrow}_{\mathrm{w}} XS$) for some $S \in \mathbb{A}^*$.

2.3. Proposition. *Given $X, Y \in \mathbb{A}^+$, we have*

$$X \stackrel{+}{\rightarrowtail} Y \text{ if and only if } X \sqsupset Y \text{ or } X \stackrel{+}{\Rightarrow}_{\mathrm{w}} YS \text{ for some } S \in \mathbb{A}^*.$$

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

2.5. If $\rightsquigarrow$ is a binary relation on $\mathbb{A}^+$, put

$$|\rightsquigarrow| := \{X \in \mathbb{A}^+ : E \stackrel{*}{\rightsquigarrow} X \text{ for some } E \in \mathbb{E}\}$$

and denote by $[\rightsquigarrow]$ the directed graph whose vertices are the words in $|\rightsquigarrow|$ and whose arcs are the pairs $\langle X, Y \rangle$ such that $X, Y \in |\rightsquigarrow|$ and $X \rightsquigarrow Y$.

For a given system, we call $[\rightarrowtail]$ the *conceptual scheme* and $[\Rightarrow_{\mathrm{w}}]$ the *weak rewriting scheme*. Since $X \Rightarrow_{\mathrm{w}} Y$ implies $X \rightarrowtail Y$, the weak rewriting scheme $[\Rightarrow_{\mathrm{w}}]$ is a subgraph of the conceptual scheme $[\rightarrowtail]$.

2.6. Theorem. *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.*

2.7. Introduce the following named condition:

There are no recurrent words. (**Rec**)

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

2.9. Theorem. *If no word $X \in \mathbb{A}^+_{\mathbb{E}}$ of length $|X| \leqslant \mu$ is recurrent, then no word is recurrent, i.e.,* (Rec) *holds.*

2.10. Let $B(\mathcal{S})$ be a set of words defined by a condition in terms of a system $\mathcal{S}$. Say that $B(\mathcal{S})$ is a *recurrence basis* if, for every system $\mathcal{S}$, nonrecurrence of all words in $B(\mathcal{S})$ implies nonrecurrence of all words. Theorem 2.9 states that the set $\{X \in \mathbb{A}^+_{\mathbb{E}} : |X| \leqslant \mu\}$ is a recurrence basis. Although finite, this set can be rather large. However, we do not know of conditions that 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.)

2.11. Introduce the following named condition:

All words are finitely rewritable. (**Fin**)

2.12. Theorem. *If all explicit words are finitely rewritable, then all words are finitely rewritable, i.e.,* (Fin) *holds.*

2.13. Theorem. (Rec) *implies* (Fin).

Therefore, by Proposition 2.8 and Theorem 2.13, (Con) implies (Rec), and (Rec) implies (Fin). As examples show, the converse implications are not true in general.

2.14. Introduce the following two named conditions:

$$\text{All explicit words are objects, i.e., } \mathbb{E} \subseteq \mathbb{O}. \tag{**Obj**}$$

$$\text{If } X \in \mathbb{A}^+,\ \alpha \in \mathbb{A}, \text{ and } X\alpha \in \mathbb{E}, \text{ then } X \in \mathbb{O}. \tag{**PreObj**}$$

2.15. Theorem. *Assume* (Obj). *Let $X, Y \in \mathbb{A}^+$, let $X \to Y$, and suppose that a letter $\alpha \in \mathbb{A}\backslash\{\omega\}$ occurs in Y. Then $\alpha \in \mathbb{A}_{\mathbb{E}}$.*

Thus, under (Obj), all letters other than ω that occur in the rules of a system belong to the explicit alphabet.

2.16. Proposition.
(1) *Let $X, Y \in \mathbb{A}^+$ and $X \overset{*}{\Rightarrow} Y$. Then $X \in \mathbb{O}$ if and only if $Y \in \mathbb{O}$.*
(2) *Assume* (Obj). *Then, given $X \in \mathbb{A}^+$, we have $X \in \mathbb{O}\backslash\{\omega\}$ if and only if $X \overset{*}{\Rightarrow} E$ for some $E \in \mathbb{E}$.*
(3) *Assume* (PreObj). *If $X, S \in \mathbb{A}^+$ and $XS \in \mathbb{O}$, then $X \in \mathbb{O}$.*

3. Attributes

3.1. Let $X \in \mathbb{O}$ and $\alpha \in \mathbb{A}$. Say that α is an *attribute* of X if $X\alpha \in \mathbb{O}$. In this case, the object $X\alpha$ is called a *property* of X (more precisely, the *property* α or the *α-property* of X). Denote by $\|X\|_1$ the set of all attributes of X. From 1.4(b) it is clear that $\|\omega\|_1 = \varnothing$.

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

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

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

3.2. Proposition. *For all $X \in \mathbb{O}$ and $\alpha \in \mathbb{A}$ the following assertions hold:*
(1) *α is an implicit attribute of X if and only if X is rewritable, $X\alpha \notin \mathbb{E}$, and $X'\alpha \in \mathbb{O}$;*
(2) *α is an added attribute of X if and only if X is rewritable, $X\alpha \in \mathbb{O}$, and $X'\alpha \notin \mathbb{O}$.*

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

3.4. Proposition. *Assume* (Obj). *Consider the rewriting sequence*

$$X = X^{(0)} \Rightarrow \cdots \Rightarrow X^{(n)} = \omega$$

of an object X. If $\alpha \in \|X\|_1$, then there is an integer $0 \leqslant i \leqslant n$ such that

$$X^{(0)}\alpha, \ldots, X^{(i)}\alpha \in \mathbb{O}, \quad X^{(i+1)}\alpha, \ldots, X^{(n)}\alpha \notin \mathbb{O}, \quad \text{and } \alpha \text{ is an added attribute of } X^{(i)}.$$

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

3.6. Programming culture usually includes the recommendation to give declared entities names that are as descriptive as reasonably possible, so that every identifier helps the reader guess the role of the corresponding object in the given code fragment. In particular, one should avoid homonymy within a local context: for instance, one should not declare a variable or class having the same name as an attribute (a property or method) but different semantics, nor, conversely, should one declare attributes whose names are occupied by system objects, classes, or variables lying in the same local context and having a different intended meaning.

We call the letter ω, as well as the letters forming explicit one-letter words, *autonomous letters*. Within the formalism considered here, autonomous letters play the role of system and declared names, whereas attribute names are represented by letters occurring in the second or a later position in words. Taking this analogy into account, the above-mentioned absence of homonymy can be formulated as the following (somewhat stronger) condition: an autonomous letter may occupy only the first position in an explicit word:

$$\begin{array}{l}\text{if } \alpha \in \mathbb{A},\ X \in \mathbb{A}^+,\ Y \in \mathbb{A}^*, \text{ and } X\alpha Y \in \mathbb{E},\\ \text{then } \alpha \neq \omega \text{ and the word } \alpha \text{ is not explicit.}\end{array} \qquad (\textbf{NoClash})$$

3.7. Theorem. *Assume* (NoClash). *Then an autonomous letter may occupy only the first position in an object:*

$$\begin{array}{l}\textit{if } \alpha \in \mathbb{A},\ X \in \mathbb{A}^+,\ Y \in \mathbb{A}^*, \textit{ and } X\alpha Y \in \mathbb{O},\\ \textit{then } \alpha \neq \omega \textit{ and the word } \alpha \textit{ is not explicit.}\end{array}$$

3.8. Proposition.

(1) *Every one-letter object is formed by an autonomous letter.*

(2) *If* (Obj) *holds, then the autonomous letters are precisely the letters forming one-letter objects.*

(3) *If* (PreObj) *holds, then the first letter of every object is autonomous.*

3.9. Let $X \in \mathbb{O}$ and $\alpha \in \mathbb{A}$. We call X an *origin* of the attribute α if α is an added attribute of X, i.e., if $X\alpha \in \mathbb{O} \cap \mathbb{E}$ and $X'\alpha \notin \mathbb{O}$, or, equivalently, if $X\alpha \in \mathbb{O}$ and $X'\alpha \notin \mathbb{O}$ (see 3.1 and 3.2).

The notion of an origin serves as an analog of the notion of the declaring class for a given member (property, method, etc.): the most abstract of the classes possessing this member or, more precisely, the most distant ancestor among such classes in one of the inheritance chains.

We note that it is quite admissible, and does not contradict practice, for an attribute to have several distinct origins incomparable with respect to inheritance. For instance, in .NET languages, origins of the attribute `Length` include the classes `System.Array`, `System.String`, `System.IO.Stream`, and some others.

3.10. Theorem.

(1) *If $X, Y \in \mathbb{O}$, $X \Rightarrow Y$, and $\|Y\|_1 = \varnothing$, then X is an origin of all its attributes.* (*For instance, this is the case if $X \to \omega$.*)

(2) *If* (NoClash) *holds, then an autonomous letter cannot be an attribute of an object and, in particular, has no origin.*

(3) *If* (Obj) *and* (PreObj) *hold, then each nonautonomous letter of the explicit alphabet has an origin.*

(4) *If* (Obj), (PreObj), *and* (NoClash) *hold, then, for each nonautonomous letter $\alpha \in \mathbb{A}_{\mathbb{E}}$, the following assertions are equivalent:*

   (a) *for all explicit α-properties $X\alpha$ and $Y\alpha$, there exists an explicit α-property $Z\alpha$ such that $X \overset{*}{\Rightarrow} Z$ and $Y \overset{*}{\Rightarrow} Z$;*

   (b) *for all α-properties $X\alpha$ and $Y\alpha$, there is an explicit α-property $Z\alpha$ such that $X \overset{*}{\Rightarrow} Z$ and $Y \overset{*}{\Rightarrow} Z$;*

   (c) *α has a unique origin.*

(5) *Every attribute of every object has an origin. Moreover, if α is an attribute of an object X, then $X \overset{*}{\Rightarrow} Y$ for some origin Y of the attribute α.*

(6) *Assume* (Obj). *A letter α is an attribute of an object X if and only if $X \overset{*}{\Rightarrow} Y$ for some origin Y of the attribute α.*

4. Object Types

4.1. Say that a word $D \in \mathbb{A}^+$ is a *detail* of an object X if $XD \in \mathbb{O}$. Denote by $\|X\|$ the set of all details of X and call $\|X\|$ the *type* of X. (From 1.4(b) it is clear that $\|\omega\| = \varnothing$.)

Note that the set $\mathbb{O}$ of all objects may be infinite and, moreover, there may exist objects X with infinite type $\|X\|$. Nevertheless, the set $\{\|X\| : X \in \mathbb{O}\}$ of all object types is always finite (see Theorem 6.2).

4.2. Say that $\|X\|$ is a *subtype* of $\|Y\|$ if $\|X\| \supseteq \|Y\|$. Note that this definition involves reverse set-theoretic inclusion. This emphasizes the fact that $\|X\|$, being a subtype of $\|Y\|$, is more concrete and richer, whereas $\|Y\|$ is more abstract and poorer.

4.3. Proposition. *For all objects X and Y, the following assertions hold:*
(1) *if $\|X\| = \|Y\|$, then $\|XD\| = \|YD\|$ for all $D \in \|X\|$;*
(2) *if $\|X\| \subseteq \|Y\|$, then $\|XD\| \subseteq \|YD\|$ for all $D \in \|X\|$.*

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 $\alpha \in \|X\|_1$.*

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

4.5. If we informally interpret the relation $X \overset{+}{\Rightarrow} Y$ as "X inherits from Y" (or "X is a special case of Y" or "X is a Y") and treat the objects $X\alpha$ as properties of X, then, in case $X \overset{+}{\Rightarrow} Y$, the object X should, in a sense, inherit the properties of Y and possibly make them more concrete and extend their totality. Formally, this requirement amounts to the following:

$$\text{if } X, Y \in \mathbb{O} \text{ and } X \overset{+}{\Rightarrow} Y, \text{ then } \|X\| \supseteq \|Y\|. \qquad \textbf{(CoInh+)}$$

The facts stated below refine condition (CoInh+).

4.6. Introduce the following named condition:

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

4.7. Theorem. *Assume* (PreObj). *The following assertions are equivalent:*
(1) *condition* (CoInh) *is satisfied;*
(2) *if $X, Y \in \mathbb{O}\backslash\{\omega\}$ 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, under (PreObj), conditions (CoInh+) and (CoInh) are equivalent.

4.8. Corollary. *Assume* (PreObj) *and* (CoInh).
(1) *If $Y \in \mathbb{O}$ and $X \overset{*}{\Rightarrow} Y$, then $\|X\|_1 \supseteq \|Y\|_1$ and $\|X\alpha\| \supseteq \|Y\alpha\|$ for all $\alpha \in \|Y\|_1$.*
(2) *If $X, Y, D \in \mathbb{A}^+$, $X \overset{*}{\Rightarrow} Y$, and $YD \in \mathbb{O}$, then $XD \in \mathbb{O}$ and $\|XD\| \supseteq \|YD\|$.*
(3) *If $X, Y \in \mathbb{A}^+$, $X \overset{+}{\rightarrowtail} Y$, and $Y \in \mathbb{O}$, then $X \in \mathbb{O}$ and $\|X\| \supseteq \|Y\|$.*

4.9. 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 an object 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 inheriting from a class Y that has an attribute α of type τ, 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 from the class Y" and treat the relation $X\alpha \to V$ 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:

$$\begin{array}{l}\text{If } X, Y, V \in \mathbb{A}^+,\ \alpha \in \mathbb{A},\ Y\alpha \in \mathbb{O},\ X\alpha \to V, \text{ and } X \Rightarrow Y,\\ \text{then } V \in \mathbb{O} \text{ and } \|V\| \supseteq \|Y\alpha\|.\end{array} \qquad \textbf{(CoVal)}$$

4.10. Theorem. *The conjunction of* (PreObj) *and* (CoVal) *implies* (CoInh).

4.11. As the following assertion shows, under (PreObj), condition (CoVal) implies a substantially stronger variant of itself.

Proposition. *Assume* (PreObj) *and* (CoVal).

$$\text{If } X, Y, V \in \mathbb{A}^+,\ \alpha \in \mathbb{A},\ Y\alpha \in \mathbb{O},\ X\alpha \Rightarrow V,\ X \stackrel{*}{\Rightarrow} Y, \text{ and } X \neq Y,$$
$$\text{then } V \in \mathbb{O} \text{ and } \|V\| \supseteq \|Y\alpha\|.$$

4.12. We define the *object types graph* to be the graph whose vertex set is the set $\mathbb{O}$ of all objects and whose arcs are the pairs $\langle X, Y\rangle$ such that $X, Y \in \mathbb{O}$ and $\|X\| \subseteq \|Y\|$.

For every system, the inheritance graph $\langle \mathbb{O}, \Leftarrow\rangle$ is a tree (see 1.11); moreover, under (PreObj) and (CoInh), the inheritance tree embeds into the object types graph (see 4.7). Nevertheless, the object types graph is not always a tree. Moreover, there are examples of systems satisfying all named conditions considered in this paper for which there exist four objects A, B, C, D that form a subtree in the inheritance graph and a "diamond configuration" in the types graph:

$$D \Rightarrow B \Rightarrow A \Leftarrow C, \qquad \begin{array}{l} \|B\| \supset \|A\| \subset \|C\|, \\ \|B\| \subset \|D\| \supset \|C\|, \\ \|B\| \text{ and } \|C\| \text{ are incomparable.} \end{array}$$

4.13. Conceptual dependence was introduced in 1.13 as a relation on the set of all words. Since non-object words are regarded as "senseless," it is reasonable to describe dependence between objects by involving only objects; namely, if a concept X conceptually 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 4.15).

4.14. Theorem. *Assume* (Obj), (PreObj), *and* (CoInh). *An object X conceptually depends on an object Y if and only if there exist $X_1, \ldots, X_n \in \mathbb{O}$, $n \geqslant 1$, such that*

$$X \rightarrowtail X_1 \rightarrowtail \cdots \rightarrowtail X_n = Y.$$

4.15. Corollary. *Assume* (Obj), (PreObj), *and* (CoInh). *The restriction of $\stackrel{+}{\rightarrowtail}$ to $\mathbb{O}$ is the least transitive relation on $\mathbb{O}$ possessing the following three properties for all $X, Y \in \mathbb{O}$ and $D \in \mathbb{A}^+$:*

(a) *if $X \to Y$ then $X \stackrel{+}{\rightarrowtail} Y$;*

(b) *if $X \to Y$ and $D \in \|X\| \cap \|Y\|$ then $XD \stackrel{+}{\rightarrowtail} YD$;*

(c) *if $D \in \|X\|$ then $XD \stackrel{+}{\rightarrowtail} X$.*

The above assertion shows that, under (Obj), (PreObj), and (CoInh), conceptual dependence between objects can be described in full conformity with the initial definition of this relation on the set of all words. The only difference is that the latter description does not go beyond the set of objects.

4.16. In Theorem 4.14 and Corollary 4.15, none of the conditions (Obj), (PreObj), or (CoInh) can be omitted. If even one of these three conditions is dropped, examples of systems arise in which conceptual dependence between objects cannot be characterized within the set of objects: in these systems, there are objects X and Y such that X conceptually depends on Y, but every chain conceptually connecting X with Y necessarily involves senseless words.

4.17. Suppose that, for some word (concept, object, or class) X, we want to know the "history" of its definition. The history of X should contain X itself and the history of its parent X'; if, in addition, X has the form $Y\alpha$, where α is an overriding attribute of Y, then it is natural to include the history of the overridden word $Y'\alpha$ in the history of X. Moreover, when reflecting the relation between X and X', it is useful to include in the history of X some indication of whether X has explicit differences from X', i.e., to indicate whether X has details absent from X' and whether there are objects XD whose values override $X'D$.

Formally, a history will be a directed graph whose vertices are objects and whose arcs are triples of the form $\langle A, r, B\rangle$, where $A, B \in \mathbb{O}$ and $r \in \{0, 1, 2\}$, with each arc directed from A to B. We agree to call every such graph an *h-graph*. The *history* of an object X is an h-graph Γ satisfying the following conditions:

(a) X is a vertex of Γ;
(b) every vertex of Γ distinct from X is reachable from X;
(c) Γ contains an arc $\langle A, 0, B\rangle$ (respectively, $\langle A, 1, B\rangle$) if and only if $A \Rightarrow B$ and A has no explicit details (respectively, has at least one explicit detail);
(d) Γ contains an arc $\langle A, 2, B\rangle$ if and only if there exist an object C and an overriding attribute α of C such that $A = C\alpha$ and $B = C'\alpha$.

Theorem.
(1) *Every object has a unique history.*
(2) *The history of every object is finite.*
(3) *If the system is conceptually consistent, then the history of every object is acyclic.*

5. Effective Analysis of Rewriting

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

5.1. Theorem. *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 $n \geqslant 0$ and $r > 0$ such that $X^{(n)} = X^{(n+r)}$;*
(b) *there are integers $n \geqslant 0$ and $r > 0$ such that*

$$\mu \leqslant \left|X^{(n)}\right| \leqslant \left|X^{(n+1)}\right|, \ldots, \left|X^{(n+r)}\right|,$$
$$X^{(n)} \neq X^{(n+r)}, \quad X^{(n)}\restriction_{\mu} = X^{(n+r)}\restriction_{\mu}.$$

In case (a) *the rewriting sequence of X contains a cycle:*

$$X \stackrel{*}{\Rightarrow} \underbrace{X^{(n)} \Rightarrow \cdots \Rightarrow X^{(n+r-1)}}_{\text{period}} \Rightarrow X^{(n)} \Rightarrow \cdots.$$

In case (b) *put $Y = X^{(n)}\restriction_{\mu}$ 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 $\langle X^{(0)}, X^{(1)}, \ldots\rangle$ contains a subsequence consisting of the words*

$$X^{(n+mr)} = YR^mS, \quad m \geqslant 0,$$

each of which starts a regular "growth period" of length r:

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

In particular,

$$\left\{X^{(n)}, X^{(n+1)}, \ldots\right\} = \left\{Y^{(j)}R^mS : 0 \leqslant j < r,\ m \geqslant 0\right\}.$$

5.2. 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 that can be imposed on words Y with additional parameters $p \in \mathcal{P}$. 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 membership $\langle Y,p\rangle \in C$.

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:

$$C'(Y,R,S,p) \text{ if and only if } C(YR^mS,p) \text{ for some } m \geqslant 1.$$

Say that $C(Y,p)$ is *cyclically decidable* if the following two assertions 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). This can be derived from the 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 [11, Chapter C.1, §6]).

5.3. Let $C(Y,p)$ be a cyclically decidable condition. Theorem 5.1 justifies the following simple algorithm which, given a system, a word $X \in \mathbb{A}^+$, and a parameter p, verifies the existence of a word $Y \in \mathbb{A}^+$ such that $X \overset{*}{\Rightarrow} Y$ and $C(Y,p)$.

Algorithm. Is there a word Y such that $X \overset{*}{\Rightarrow} Y$ and $C(Y,p)$?

- If $C(X,p)$, return **Yes**.
  If X is not rewritable, return **No**.
- Otherwise, successively calculate the rewrites $X^{(1)},\dots,X^{(i)},\dots$ and, at each step $i \geqslant 1$, successively analyze the fragments $\langle X^{(n)}, X^{(n+1)}, \dots, X^{(n+r)}\rangle$ for $0 \leqslant n < n+r = i$, as follows:
  - If $C\big(X^{(n+r)},p\big)$, return **Yes**.
  - If $X^{(n)} = X^{(n+r)}$, return **No**.
  - If $\langle X^{(n)}, \dots, X^{(n+r)}\rangle$ satisfies condition (b) of Theorem 5.1,
    put $Y := X^{(n)}\restriction_\mu$; let $S \in \mathbb{A}^*$ be such that $X^{(n)} = YS$;
    let $R \in \mathbb{A}^+$ be such that $Y^{(r)} = YR$ (such a word R exists by Theorem 5.1);
    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, $i+1$.

5.4. 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, Algorithm 5.3, specialized to this condition, verifies finite rewritability of a given word within a given system. A simplified version is presented below.

Algorithm. Is a word X finitely rewritable?

- Start the successive calculation of the rewrites $X^{(0)}, X^{(1)}, \dots$.
- If a nonrewritable word $X^{(i)}$ occurs, return **Yes**.
- Otherwise, by Theorem 5.1, a fragment $\langle X^{(n)}, X^{(n+1)}, \dots, X^{(n+r)}\rangle$ will occur that satisfies (a) or (b) of Theorem 5.1; in this case return **No**.

Since the set $\mathbb{E}$ of explicit words is finite, Theorem 2.12 implies that (Fin) *is effectively verifiable*: it suffices to apply Algorithm 5.4 to all explicit words.

5.5. The specialization of Algorithm 5.3 to the condition

$$C(Y) = \text{"}Y = \omega\text{"}$$

checks *whether a given word is an object*. (This can also be verified by a slight modification of Algorithm 5.4.) It is now clear that (Obj) *and* (PreObj) *are effectively verifiable*.

5.6. Note that if (Fin) holds, the containment $X \in \mathbb{O}$ can be trivially verified: just check whether 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 $n \geqslant 0$; furthermore, if both (Obj) and (PreObj) hold, the calculation can be terminated if some $X^{(n)}$ becomes a prefix of an explicit word (see Proposition 2.16).

5.7. As is easily seen, the condition

$$C(Y, X) = \text{“}Y \sqsupseteq X\text{”}$$

is cyclically decidable. Therefore, a properly specialized version of Algorithm 5.3 checks *whether a given word X is recurrent*. By Theorem 2.9, we conclude that (Rec) *is effectively verifiable*: to check that no word is recurrent, it suffices to apply the algorithm to all words over $\mathbb{A}_{\mathbb{E}}$ of length at most μ. (However, the resulting verification is exponential-time; see 2.10.)

Another approach to verifying (Rec) can be based on Theorem 2.13, which states that (Rec) implies (Fin). Check (Fin) first. If it fails, then (Rec) also fails. If (Fin) holds, condition (Rec) can be verified by processing all words X over $\mathbb{A}_{\mathbb{E}}$ of length $|X| \leqslant \mu$ and returning **No** whenever a word X occurs such that $X \sqsubseteq X^{(n)}$ for some $n \geqslant 1$.

5.8. By Theorem 2.6, condition (Con) can be effectively verified by constructing the conceptual scheme and checking, during the construction, whether the scheme is acyclic. The algorithm described below checks whether a given system is conceptually consistent. If 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 $\Gamma = \langle \Gamma_{\mathrm{V}}, \Gamma_{\mathrm{A}} \rangle$ (with Γ_{V} the vertices and Γ_{A} the arcs) and a variable $\mathcal{A}$ whose values are finite subsets of $\mathbb{A}^+ \times \mathbb{A}^+$.

Algorithm. Is a system conceptually consistent?
(1) Put $\Gamma_{\mathrm{V}} := \mathbb{E}$, $\Gamma_{\mathrm{A}} := \varnothing$.
(2) Put $\mathcal{A} := \{\langle X, Y \rangle : X \text{ is a sink of } \Gamma \text{ and } X \rightarrowtail Y\}$.
(3) If $\mathcal{A} = \varnothing$, 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**: $X \rightarrowtail X$ or $Y \rightarrowtail \cdots \rightarrowtail X \rightarrowtail Y$;
    otherwise put $\Gamma_{\mathrm{V}} := \Gamma_{\mathrm{V}} \cup \{Y\}$, $\Gamma_{\mathrm{A}} := \Gamma_{\mathrm{A}} \cup \{\langle X, Y \rangle\}$.
(5) Go to (2).

5.9. When applied to a conceptually inconsistent system, Algorithm 5.8 returns an example of a cycle

$$X_0 \rightarrowtail X_1 \rightarrowtail \cdots \rightarrowtail X_m = X_0$$

in the conceptual scheme, but it is not guaranteed that all words X_i in the cycle are objects (even when X_0 is an object). On the other hand, Theorem 4.14 implies that, under (Obj), (PreObj), and (CoInh), every path

$$X_0 \rightarrowtail X_1 \rightarrowtail \cdots \rightarrowtail X_m$$

between objects X_0 and X_m can be transformed into a path of *objects*

$$Y_0 \rightarrowtail Y_1 \rightarrowtail \cdots \rightarrowtail Y_n, \quad \text{with } Y_0 = X_0 \text{ and } Y_n = X_m.$$

We note that such a transformation can be performed *effectively*.

5.10. By slightly modifying Algorithm 5.8, 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, whether there are weakly recurrent words and either returns an example of such a word or constructs the weak rewriting scheme of the system.

Algorithm. Is there a weakly recurrent word?
(1) Put $\Gamma_{\mathrm{V}} := \mathbb{E}$, $\Gamma_{\mathrm{A}} := \varnothing$.
(2) Put $\mathcal{A} := \{\langle X, Y\rangle : X \text{ is a sink of } \Gamma \text{ and } X \Rightarrow_{\mathrm{w}} Y\}$.
(3) If $\mathcal{A} = \varnothing$, 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**:
        $X \Rightarrow_{\mathrm{w}} Y \sqsupseteq X$ or $Y_0 \Rightarrow_{\mathrm{w}} \cdots \Rightarrow_{\mathrm{w}} X \Rightarrow_{\mathrm{w}} Y \sqsupseteq Y_0$;
    otherwise put $\Gamma_{\mathrm{V}} := \Gamma_{\mathrm{V}} \cup \{Y\}$, $\Gamma_{\mathrm{A}} := \Gamma_{\mathrm{A}} \cup \{\langle X, Y\rangle\}$.
(5) Go to (2).

5.11. According to Theorem 2.6, the weak rewriting scheme of a conceptually inconsistent system includes a path $X_0 \overset{+}{\Rightarrow}_{\mathrm{w}} X_n \sqsupseteq X_0$ with $X_0 \in \mathbb{E}$. Given an inconsistent system, Algorithm 5.10 returns an example of a path $Y_0 \overset{+}{\Rightarrow}_{\mathrm{w}} Y_n \sqsupseteq Y_0$, but the leading word Y_0 need not be explicit. In this connection it is worth noting that every path of the form $Y_0 \overset{+}{\Rightarrow}_{\mathrm{w}} Y_n \sqsupseteq Y_0$ can be *effectively* transformed into a path $X_0 \overset{+}{\Rightarrow}_{\mathrm{w}} X_n \sqsupseteq X_0$ (of the same length) with $X_0 \in \mathbb{E}$.

5.12. The most convincing evidence of conceptual inconsistency appears to be the occurrence of a cycle $X_0 \rightarrowtail \cdots \rightarrowtail X_n = X_0$ consisting of *objects* and starting 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*.

6. Effective Analysis of Object Types

6.1. 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

$$X^{(0)} \Rightarrow \cdots \Rightarrow X^{(n)} = \omega$$

and denote by $\mathrm{ch}(X)$ the first character occurring in this sequence. Call $\mathrm{ch}(X)$ the *character* of X.

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

6.2. Theorem. *For every object X we have $\|X\| = \|\mathrm{ch}(X)\|$. In particular,*

$$\{\|X\| : X \in \mathbb{O}\} = \{\|C\| : C \in \mathbb{C}\},$$

and the set $\{\|X\| : X \in \mathbb{O}\}$ is finite.

6.3. 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

$$\|X\| = \|Y\|, \quad \|X\| \subseteq \|Y\|, \quad \|X\| \supseteq \|Y\|$$

hold. It is clear that the character of an object can be effectively determined. Therefore, by Theorem 6.2, the type comparison problem reduces to effective comparison of the character types.

6.4. Given an arbitrary function $\tau : \mathbb{C} \to \mathbb{N}$, define the following equivalence relation on $\mathbb{C}$:

$$X \underset{\tau}{\sim} Y \text{ if and only if } \|X\|_1 = \|Y\|_1 \text{ and } \tau\big(\mathrm{ch}(X\alpha)\big) = \tau\big(\mathrm{ch}(Y\alpha)\big) \text{ for all } \alpha \in \|X\|_1.$$

We call a function $\tau : \mathbb{C} \to \mathbb{N}$ a *typing* if, for all $X, Y \in \mathbb{C}$,

$$\tau(X) = \tau(Y) \text{ implies } X \underset{\tau}{\sim} Y.$$

The simplest example of a typing is an arbitrary injection $\tau : \mathbb{C} \to \mathbb{N}$. Say that a typing τ is *minimal* if its image $\operatorname{im}\tau := \tau[\mathbb{C}]$ has the least number of elements $|\operatorname{im}\tau|$ among all possible typings:

$$|\operatorname{im}\tau| \leqslant |\operatorname{im}\sigma| \text{ for every typing } \sigma : \mathbb{C} \to \mathbb{N}.$$

We call a function $\tau : \mathbb{C} \to \mathbb{N}$ an *exact typing* if, for all $X, Y \in \mathbb{C}$,

$$\tau(X) = \tau(Y) \text{ if and only if } \|X\| = \|Y\|.$$

Assertion 6.5(2) of the following theorem justifies the use of the term "typing" in the latter definition.

6.5. Theorem. *Let* $\tau : \mathbb{C} \to \mathbb{N}$.
(1) *If* $\|X\| = \|Y\|$ *implies* $\tau(X) = \tau(Y)$ *for all* $X, Y \in \mathbb{C}$, *then* $\|X\| = \|Y\|$ *implies* $X \underset{\tau}{\sim} Y$ *for all* $X, Y \in \mathbb{C}$.
(2) *Every exact typing is a typing.*
(3) *Assume* (PreObj). *If* τ *is a typing, then* $\tau(X) = \tau(Y)$ *implies* $\|X\| = \|Y\|$ *for all* $X, Y \in \mathbb{C}$.
(4) *The converse to* (3) *is false: there exist a system satisfying* (PreObj) *and a function* $\tau : \mathbb{C} \to \mathbb{N}$ *such that* $\tau(X) = \tau(Y)$ *implies* $\|X\| = \|Y\|$ *for all* $X, Y \in \mathbb{C}$, *but* τ *is not a typing.*
(5) *Assertion* (3) *does not remain true without the requirement* (PreObj).

6.6. Theorem. *Assume* (PreObj). *A function* $\tau : \mathbb{C} \to \mathbb{N}$ *is an exact typing if and only if* τ *is a minimal typing.*

6.7. The following algorithm uses two variable functions $\tau, \sigma : \mathbb{C} \to \mathbb{N}$ (each of which can be encoded as an array of positive integers indexed by the characters).

Algorithm. Compute an exact typing.
(1) Define τ by putting $\tau(X) := 1$ for all $X \in \mathbb{C}$.
(2) If, for all $X, Y \in \mathbb{C}$, $\tau(X) = \tau(Y)$ implies $X \underset{\tau}{\sim} Y$, **return** τ.
(3) Assign σ a copy of τ.
(4) For each $k \in \big\{\tau(X) : X \in \mathbb{C} \text{ and } (\exists Y \in \mathbb{C})\big(X \underset{\tau}{\nsim} Y \text{ and } \tau(X) = \tau(Y)\big)\big\}$ do the following:
  arbitrarily enumerate the set $\{X \in \mathbb{C} : \tau(X) = k\}$,
  thus making it a sequence $\langle X_1, \ldots, X_n\rangle$, $n \geqslant 2$,
  and, for each $i = 2, \ldots, n$, do the following:
    if $X_i \underset{\tau}{\sim} X_j$ for some $1 \leqslant j < i$, reassign $\sigma(X_i) := \sigma(X_j)$;
    otherwise reassign $\sigma(X_i) := \max\{\sigma(X) : X \in \mathbb{C}\} + 1$.
(5) Assign $\tau := \sigma$ and go to (2).

6.8. Theorem. *Algorithm* 6.7 *halts for every system and, if the system satisfies* (PreObj), *the resulting function* $\tau : \mathbb{C} \to \mathbb{N}$ *is an exact typing:* $\tau(X) = \tau(Y)$ *if and only if* $\|X\| = \|Y\|$.

Algorithm 6.7 is analogous to Vizing's algorithm for partitioning the vertex set of a graph into classes of similar vertices (see [12]). The author is grateful to S.V. Avgustinovich for discovering this analogy.

6.9. Given an arbitrary binary relation $\lhd$ on $\mathbb{C}$ (i.e., a subset $\lhd \subseteq \mathbb{C}^2$), define the binary relation $\widetilde{\lhd}$ on $\mathbb{C}$ as follows:

$$X \mathrel{\widetilde{\lhd}} Y \text{ if and only if } \|X\|_1 \subseteq \|Y\|_1 \text{ and } \mathrm{ch}(X\alpha) \lhd \mathrm{ch}(Y\alpha) \text{ for all } \alpha \in \|X\|_1.$$

A relation $\lhd \subseteq \mathbb{C}^2$ is called a *subtyping* if $\lhd \subseteq \widetilde{\lhd}$, i.e., if, for all $X, Y \in \mathbb{C}$,

$$X \lhd Y \text{ implies } X \mathrel{\widetilde{\lhd}} Y.$$

The simplest example of a subtyping is the equality relation on $\mathbb{C}$. The relation $\{\langle X, Y\rangle \in \mathbb{C}^2 : \|X\| \subseteq \|Y\|\}$ will be called the *exact subtyping*; i.e., this is the relation $\lhd \subseteq \mathbb{C}^2$ such that, for all $X, Y \in \mathbb{C}$,

$$X \lhd Y \text{ if and only if } \|X\| \subseteq \|Y\|.$$

Assertion 6.10(2) of the following theorem justifies the use of the term "subtyping" in the latter definition.

6.10. Theorem. *Let $\lhd$ be an arbitrary binary relation on $\mathbb{C}$.*
(1) *If $\lhd$ includes the exact subtyping, then so does $\widetilde{\lhd}$. In other words, if $\|X\| \subseteq \|Y\|$ implies $X \lhd Y$ for all $X, Y \in \mathbb{C}$, then $\|X\| \subseteq \|Y\|$ implies $X \widetilde{\lhd} Y$ for all $X, Y \in \mathbb{C}$.*
(2) *The exact subtyping is a subtyping.*
(3) *Assume* (PreObj). *Every subtyping is included in the exact subtyping: if $\lhd$ is a subtyping, $X, Y \in \mathbb{C}$, and $X \lhd Y$, then $\|X\| \subseteq \|Y\|$.*

6.11. The following algorithm uses two variable sets $\lhd, \lhd_0 \subseteq \mathbb{C}^2$.

Algorithm. Compute the exact subtyping.
(1) Put $\lhd := \mathbb{C}^2$.
(2) Put $\lhd_0 := \lhd \cap \widetilde{\lhd}$.
(3) If $\lhd_0 = \lhd$, **return** $\lhd$.
Otherwise reassign $\lhd := \lhd_0$ and go to (2).

6.12. Theorem. *Given an arbitrary system, Algorithm* 6.11 *always halts and, if the system satisfies* (PreObj)*, the resulting relation $\lhd$ is the exact subtyping: for all $X, Y \in \mathbb{C}$, $X \lhd Y$ if and only if $\|X\| \subseteq \|Y\|$.*

6.13. 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*.) Nevertheless, the mere assertion "$\|X\| \nsubseteq \|Y\|$" is not always sufficient, and a more delicate analysis may require a specific reason why the inclusion $\|X\| \subseteq \|Y\|$ fails. In a large system, finding a particular detail $D \in \|X\| \backslash \|Y\|$ may be nontrivial, and a corresponding algorithm would thus be a useful troubleshooting tool. Recall that, by Theorem 6.2, it suffices to solve the problem algorithmically for characters only.

6.14. A *subtyping diagnosis* is a function $\Delta : \mathcal{D} \to \mathbb{A}^+$ defined on the set

$$\mathcal{D} = \{\langle X, Y\rangle \in \mathbb{C}^2 : \|X\| \nsubseteq \|Y\|\}$$

and assigning to each pair $\langle X, Y\rangle \in \mathcal{D}$ some detail

$$\Delta(X, Y) \in \|X\| \backslash \|Y\|.$$

6.15. The following algorithm uses a variable set $\mathcal{D} \subseteq \mathbb{C}^2$ and a variable function $\Delta : \mathcal{D} \to \mathbb{A}^+$.

Algorithm. Compute a subtyping diagnosis.
(1) Put $\mathcal{D} := \varnothing$.
(2) For all pairwise distinct $X, Y \in \mathbb{C}$ do the following:
if there is a letter $\alpha \in \|X\|_1 \backslash \|Y\|_1$, add $\langle X, Y\rangle$ to $\mathcal{D}$ and assign $\Delta(X, Y) := \alpha$.
(3) For all pairwise distinct $X, Y \in \mathbb{C}$ such that $\langle X, Y\rangle \notin \mathcal{D}$ do the following:
if there is a letter $\alpha \in \|X\|_1$ such that $\big\langle \mathrm{ch}(X\alpha), \mathrm{ch}(Y\alpha)\big\rangle \in \mathcal{D}$,
add $\langle X, Y\rangle$ to $\mathcal{D}$ and assign $\Delta(X, Y) := \alpha\, \Delta\big(\mathrm{ch}(X\alpha), \mathrm{ch}(Y\alpha)\big)$.
(4) If there were no assignments at step (3), **return** Δ.
Otherwise go to (3).

6.16. Theorem. *Given an arbitrary system, Algorithm* 6.15 *always halts and, if the system satisfies* (PreObj)*, the resulting function $\Delta : \mathcal{D} \to \mathbb{A}^+$ is a subtyping diagnosis: $\mathcal{D} = \{\langle X, Y\rangle \in \mathbb{C}^2 : \|X\| \nsubseteq \|Y\|\}$ and $\Delta(X, Y) \in \|X\| \backslash \|Y\|$ for all $\langle X, Y\rangle \in \mathcal{D}$.*

6.17. A subtyping diagnosis $\Delta : \mathcal{D} \to \mathbb{A}^+$ is called *laconic* if, for all $\langle X, Y\rangle \in \mathcal{D}$,

$$|\Delta(X, Y)| = \min\{|D| : D \in \|X\| \backslash \|Y\|\},$$

i.e., the detail $\Delta(X, Y) \in \|X\| \backslash \|Y\|$ has the least length among all possible words $D \in \|X\| \backslash \|Y\|$. In this sense, $\Delta(X, Y)$ gives the shortest answer to the question of why the inclusion $\|X\| \subseteq \|Y\|$ fails.

The subtyping diagnosis returned by Algorithm 6.15 is not always laconic. Moreover, for some systems satisfying all named conditions considered in the paper, this algorithm gives correct but not shortest answers.

6.18. The following algorithm is a modification of Algorithm 6.15. The main difference consists in the use of an auxiliary integer variable n, whose value is increased by one after each passage through step (3).

Algorithm. Compute a laconic subtyping diagnosis.

(1) Put $\mathcal{D} := \varnothing$ and $n := 1$.

(2) For all pairwise distinct $X, Y \in \mathbb{C}$ do the following:
if there is a letter $\alpha \in \|X\|_1 \backslash \|Y\|_1$, add $\langle X, Y\rangle$ to $\mathcal{D}$ and assign $\Delta(X, Y) := \alpha$.

(3) For all pairwise distinct $X, Y \in \mathbb{C}$ such that $\langle X, Y\rangle \notin \mathcal{D}$ do the following:
if there is a letter $\alpha \in \|X\|_1$ such that $\big\langle \mathrm{ch}(X\alpha), \mathrm{ch}(Y\alpha)\big\rangle \in \mathcal{D}$ and $\big|\Delta\big(\mathrm{ch}(X\alpha), \mathrm{ch}(Y\alpha)\big)\big| = n$, add $\langle X, Y\rangle$ to $\mathcal{D}$ and assign $\Delta(X, Y) := \alpha\, \Delta\big(\mathrm{ch}(X\alpha), \mathrm{ch}(Y\alpha)\big)$.

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

6.19. Theorem. *Given an arbitrary system, Algorithm* 6.18 *always halts and, if the system satisfies* (PreObj)*, the resulting function* $\Delta : \mathcal{D} \to \mathbb{A}^+$ *is a laconic subtyping diagnosis:* $\mathcal{D} = \{\langle X, Y\rangle \in \mathbb{C}^2 : \|X\| \not\subseteq \|Y\|\}$ *and, for all* $\langle X, Y\rangle \in \mathcal{D}$, $\Delta(X, Y) \in \|X\| \backslash \|Y\|$ *and* $|\Delta(X, Y)| = \min\{|D| : D \in \|X\| \backslash \|Y\|\}$.

Acknowledgment

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

FUNDING

The work was carried out in the framework of the State Task to the Sobolev Institute of Mathematics (Project FWNF–2026–0022).

CONFLICT OF INTEREST

The author of this work declares that he has no conflicts of interest.

References

1. Büchi J.R., "Regular canonical systems," Arch. Math. Logik Grundlag., vol. 6, 91–111 (1964).
2. Caucal D., "On the regular structure of prefix rewriting," Theoret. Comput. Sci., vol. 106, no. 1, 61–86 (1992).
3. Book R.V. and Otto F., *String-Rewriting Systems*, Springer, New York (1993).
4. Frazier M. and Page C.D., "Prefix grammars: An alternative characterization of the regular languages," Inform. Process. Lett., vol. 51, no. 2, 67–71 (1994).
5. Cardelli L. and Wegner P., "On understanding types, data abstraction, and polymorphism," ACM Comput. Surv., vol. 17, no. 4, 471–523 (1985).
6. Atkinson M.P., Bancilhon F., DeWitt D.J., Dittrich K.R., Maier D., and Zdonik S.B., "The object-oriented database system manifesto," in: *Deductive and Object-Oriented Databases*, North-Holland, Amsterdam (1990), 223–240.
7. Dittrich K.R., "Object-oriented data model concepts," in: *Advances in Object-Oriented Database Systems*, Springer, Berlin (1994), 29–45 (NATO ASI Series F: Computer and Systems Sciences, vol. 130).
8. Harary F., *Graph Theory*, Addison-Wesley, Reading, Massachusetts (1969).
9. Gutman A.E., "Object-oriented data as prefix rewriting systems," Vladikavkaz. Mat. Zh., vol. 17, no. 3, 23–35 (2015).
10. Salomaa A., *Formal Languages*, Academic, New York (1973).
11. *Handbook of Mathematical Logic*, Barwise J. (ed.), North-Holland, Amsterdam (1977).
12. Vizing V.G., "Distributive coloring of graph vertices," in: *Operations Research and Discrete Analysis*, Kluwer, Dordrecht (1997), 311–319 (Mathematics and Its Applications, vol. 391).

A. E. Gutman
Sobolev Institute of Mathematics, Novosibirsk, Russia
https://orcid.org/0000-0003-2030-7459
gutman@math.nsc.ru