АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ СЛОВАРЯ ТИПИЗИРОВАННЫХ КАНОНИЧЕСКИХ ШАБЛОНОВ УПРАВЛЯЮЩЕГО КОДА ПО ПРОМЫШЛЕННОМУ КОРПУСУ

С. Е. Щигал[1,а], **В. И. Соловьев**[2,б]

[1] *Российский государственный социальный университет, Москва, Российская Федерация*

[2] *Государственный университет по землеустройству, Москва, Российская Федерация*

[а] *ORCID: https://orcid.org/0009-0004-0295-2934,* ✍ *swimanshigal@mail.ru*

[б] *ORCID: https://orcid.org/0000-0003-0338-1227, vs@guz.ru*

Аннотация: буквальное сравнение управляющих программ чувствительно к именам переменных и не всегда выделяет существенные изменения. Цель — формализовать построение словаря типизированных канонических шаблонов языка Structured Text и определить ограничения статического сравнения. Материал — промышленный проект вентиляции из 189 программных единиц. Операторы преобразуются в сигнатуры, сохраняющие вид конструкции, ограниченную структуру выражений, роли и типы переменных, классы литералов. Частые сигнатуры образуют словарь, редкие заменяются общими категориями. Исправленный синтаксический анализ дал 7794 вхождения; при глубине один и минимальной частоте пять выделено 238 шаблонов с покрытием 54,0 %. Полные и пороговые последовательности сохранились после лексической замены идентификаторов в 188 измененных единицах. На шести процедурах расстояние до переименованного варианта равно нулю, до контролируемого изменения — от одного до трех. Во внешнем архиве из 110 вариантов полные представления различались в 65 и 75 случаях при глубинах один и два относительно модального эталона. Эти результаты не доказывают функциональной корректности. Основные ограничения — потеря тождества переменных, укрупнение литералов, неполный охват синтаксиса и отсутствие независимой экспертной проверки реально сгенерированных программ.

Ключевые слова: Structured Text, статический анализ, канонический шаблон, корпусный словарь, расстояние Левенштейна, промышленная вентиляция.

Для цитирования: Щигал С. Е., Соловьев В. И. Автоматическое построение словаря типизированных канонических шаблонов управляющего кода по промышленному корпусу. *Успехи кибернетики.* 2026;7(3):133–138.

Поступила в редакцию: 16.08.2026. *В окончательном варианте: 21.09.2026.*

TYPED CANONICAL PATTERN DICTIONARY FOR STATIC COMPARISON OF STRUCTURED TEXT PLC PROGRAMS

S. E. Shchigal[1,a], **V. I. Soloviev**[2,b]

[1] *Russian State Social University, Moscow, Russian Federation*

[2] *State University of Land Use Planning, Moscow, Russian Federation*

[a] *ORCID: https://orcid.org/0009-0004-0295-2934,* ✍ *swimanshigal@mail.ru*

[b] *ORCID: https://orcid.org/0000-0003-0338-1227, vs@guz.ru*

Abstract: we investigated a method for comparing numerical-control (NC) programs that reduces the sensitivity of literal comparison to variable names while retaining syntactic and semantic information relevant to program structure. We formalized a dictionary of typed canonical Structured Text patterns and evaluated its use for static program comparison. An industrial ventilation project comprising 189 program units was used to construct the dictionary. Statements were converted into signatures that retained statement types, selected expression structures, variable roles and data types, and literal classes. Frequent signatures were included as individual patterns, whereas rare signatures were grouped into generic categories. After correcting syntactic analysis errors, the dataset contained 7,794 statement occurrences. At a pattern depth of one and a minimum frequency of five, the dictionary contained 238 patterns covering 54.0% of all occurrences. Replacing lexical identifiers in 188 modified program units did not change either the full or thresholded pattern sequences. Across six procedures, comparison distances were zero for renamed variants and ranged from one to three for

controlled modifications. In an external archive containing 110 program variants, full representations differed from the modal reference in 65 cases at depth one and 75 cases at depth two. The method does not establish functional correctness. Its limitations include loss of variable identity, coarse representation of literals, incomplete syntax coverage, and the absence of independent expert validation of the generated programs.

Keywords: Structured Text, static analysis, canonical pattern, corpus-based dictionary, Levenshtein distance, industrial ventilation.

Cite this article: Shchigal S. E., Soloviev V. I. Typed Canonical Pattern Dictionary for Static Comparison of Structured Text PLC Programs. *Russian Journal of Cybernetics*. 2026;7(3):133–138.

Original article submitted: 16.08.2026. *Revision submitted: 21.09.2026.*

Введение

В вентиляции управляющий код реализует пуск, регулирование и защиты. Генерация из текстовых требований может уменьшить трудоемкость проектирования. Большие языковые модели применяются к программам контроллеров [1, 2], но правильный синтаксис не гарантирует сохранения защит. Статическое сравнение с эталоном, без выполнения кода на контроллере или модели объекта, выделяет варианты для проверки.

Одинаковая операция записывается с разными именами переменных; устойчивость к переименованию важна и для моделей кода [3]. Например, y := a AND b и out := in1 AND in2 имеют одинаковую форму при согласованном соответствии переменных. Напротив, небольшая замена команды открытия заслонки командой закрытия существенна для управления. Нормализация должна отделять произвольные имена от связей и назначения сигналов.

Обнаружение клонов — поиск одинаковых или сходных фрагментов [4]. CCFinder сравнивает нормализованные лексемы — минимальные значимые элементы текста [5]. Aroma использует структурный поиск [6], извлечение идиом выявляет повторяющиеся формы [7], TSED сравнивает синтаксические деревья [8], где операции — узлы, вложенность — связи. Корпусные закономерности служат основой обучаемых представлений кода [9], но готового словаря типизированных форм нашего проекта эти работы не дают.

Расстояние Левенштейна измеряет минимальное число вставок, удалений и замен элементов последовательности [10]. Набор STMutants позволяет исследовать реакцию представления на искусственные изменения программ Structured Text [11], но отличие последовательностей не равнозначно обнаружению функционального дефекта.

Цель — построить частотный словарь типизированных канонических шаблонов по неразмеченному корпусу и проверить покрытие, устойчивость к лексической замене и реакцию на изменения. Домен здесь — управление вентиляционным оборудованием. Корпусный словарь еще не является полной моделью инженерного смысла.

Корпус и правила представления

Экспорт одного проекта CODESYS содержит 189 единиц: 41 функциональный блок, 138 методов, 8 функций и 2 программы. Функциональный блок сохраняет состояние между вызовами; методы анализировались с декларациями родительских блоков. Корпус включает управление вентиляцией, обмен по Modbus и преобразования данных. Словарь строился и покрытие оценивалось на одних данных, без независимой тестовой выборки.

Лексический анализатор выделяет лексемы, синтаксический — строит дерево. Обход фиксирует текущий оператор, затем вложенные тела в исходном порядке. Литерал — записанная константа, например TRUE, 0, T#5s. Канонический шаблон — строковая сигнатура, определяемая правилами нормализации данной работы, а не отраслевым стандартом.

Переменная заменяется ролью и типом: вход I, выход O, вход-выход IO, локальная L, глобальная G. Поля экземпляров, имена функций и типы блоков сохраняются. Логические литералы кодируются как BT и BF, ноль — N0, прочие числа, время и строки — N, T, S. Операнды AND, OR, XOR, +, *, =, <> упорядочиваются; глубокие выражения заменяются знаком $\sim$. Выходное присваивание aoSpeed := 0.0 получает сигнатуру A O:REAL:=N0.

Для IF сохраняются число ветвей, наличие ELSE и условия. Для CASE и циклов — вид конструкции и вложенные операторы, но не метки выбора, условия и границы циклов. Аргумен-

ты вызовов упорядочиваются без имен формальных параметров. Эти упрощения ограничивают различимость.

Формирование словаря и расстояние

Пусть P — программа, s_i — поддерживаемый оператор, n — их число, m — глубина выражений, σ_m — функция сигнатуры. Полная последовательность задается формулой (1):

$$B_m(P) = (\sigma_m(s_1), \ldots, \sigma_m(s_n)). \tag{1}$$

Для корпуса C совокупность $S(C)$ учитывает все вхождения операторов. Частота и словарь при пороге K определяются формулой (2):

$$f_{C,m}(\sigma) = \sum_{s \in S(C)} \mathbf{1}[\sigma_m(s) = \sigma], \qquad D_{K,m} = \{\sigma : f_{C,m}(\sigma) \geq K\}. \tag{2}$$

Индикатор **1** равен единице при истинности условия, нулю иначе. Пороговое кодирование задается формулой (3):

$$b_{K,m}(s) = \begin{cases} \sigma_m(s), & \sigma_m(s) \in D_{K,m}, \\ \mathrm{GEN}(\mathrm{kind}(s)), & \text{иначе.} \end{cases} \tag{3}$$

Здесь kind — вид оператора, GEN — обобщенная категория. Применение (3) в исходном порядке дает $B_{K,m}(P)$. Конечный алфавит при m=1, K=5 содержит 238 шаблонов и пять категорий хвоста: присваивание, вызов, условие, выбор, цикл; всего 243 символа. Покрытие задается формулой (4):

$$\mathrm{Cov}(D_{K,m}) = \frac{\sum_{s \in S(C)} \mathbf{1}[\sigma_m(s) \in D_{K,m}]}{|S(C)|}. \tag{4}$$

Для последовательностей $(b_1, \ldots, b_n)$ и $(c_1, \ldots, c_r)$ расстояние определяется рекуррентно по формуле (5):

$$L_{i,j} = \min\{L_{i-1,j} + 1,\ L_{i,j-1} + 1,\ L_{i-1,j-1} + \mathbf{1}[b_i \neq c_j]\}. \tag{5}$$

Начальные условия: $L_{0,j} = j$, $L_{i,0} = i$; итог $d = L_{n,r}$. Стоимость каждой операции — единица. Полное расстояние сравнивает B_m, пороговое — $B_{K,m}$. Условие $d \geq 1$ означает различие представлений, не опасность изменения. Значения K=5, $m \in \{0,1,2,3\}$ не оптимизированы по экспертной разметке.

Протокол и воспроизводимость

Сопоставлены версии анализа, глубины, лексическая замена, шесть изменений и внешний архив. Расчеты выполнены в разрешающем режиме с контекстом родительского блока. Сравнивались последовательности, не множества. Единица с неизменившимся текстом исключена из переименования.

Архив Figshare версии 3 содержит по десять вариантов 11 программ: всего 110, а не 108 окончательно отобранных мутантов STMutants [11]. Эталоном семейства служила наиболее частая последовательность; при равенстве частот — первая в детерминированном порядке файлов. Модальный эталон не является подтвержденной исходной программой.

Артефакт v0.2.0 содержит код, сценарии, тесты, синтетический пример, контрольные суммы и манифест агрегатов (https://github.com/stchigalsergey/st-functional-canonizer/releases/tag/v0.2.0). Промышленный корпус и словарь закрыты: точные показатели требуют законного доступа к совпадающему снимку 189 файлов. Открытый алгоритм проверяется на собственных и синтетических данных.

Исправление анализа и покрытие

Первая версия при m=1, K=5 дала 7819 вхождений, 233 частых шаблона, покрытие 54,5 %, включая 183 неразобранных фрагмента RAW. Исправление меток CASE дало 7794 вхождения, 2754 различные сигнатуры, 238 шаблонов и 54,0 % покрытия без RAW. Далее используются исправленные показатели.

Отсутствие RAW не доказывает полноту анализа: разрешающий режим отбрасывал 339 знаков разыменования ∧ в 58 файлах до разбора. Строгий режим выпуска останавливается с диагностикой, но разыменование не поддерживает. Агрегаты относятся к разрешающему расчету, а не к успешному строгому анализу всего корпуса.

В таблице 1 и на рисунке показана зависимость от глубины: 7794 вхождения, порог пять. При глубине один частые шаблоны покрывают 4209 вхождений, остальные 3585 переходят в общие категории.

Таблица 1

Размер словаря и точное покрытие при разных глубинах

Глубина	Различных сигнатур	Частых шаблонов	Покрытие, %
0	1585	181	72,0
1	2754	238	54,0
2	3181	219	47,2
3	3384	214	44,8

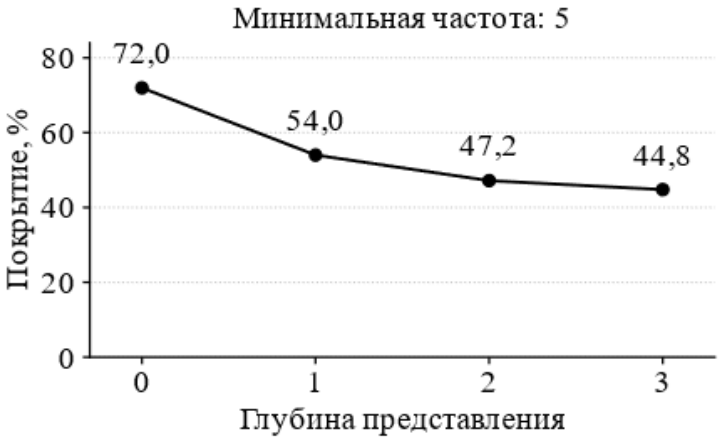


Рис. *Точное покрытие при разных глубинах и минимальной частоте пять*

Словарь включает логические присваивания, условия, циклы, таймеры и чтение регистров, отражая управление и обмен данными.

Лексическая замена и контролируемые изменения

При m=1 полные и пороговые последовательности сохранились в 188 из 188 измененных единиц. Варианты не компилировались и не исполнялись. Замена затронула заголовки 109 единиц и 3089 меток именованных аргументов в 51 единице без сопоставления с декларациями вызываемых блоков. Это нечувствительность к конкретной лексической замене, не доказательство сохранения поведения.

На шести процедурах при m=1, K=5 пороговое расстояние до переименованного варианта равно нулю, до измененного — от одного до трех (таблица 2). Защелка сохраняет аварию после исчезновения причины: один IF устанавливает флаг, отдельные IF выполняют автоматический и ручной сброс. Удалялся ручной сброс. Ограничение хода удерживает значение в диапазоне 0–100 %.

Правило $d \geq 1$ выделило шесть подготовленных изменений; малая выборка известных преобразований не определяет чувствительность на реальных ответах модели.

Внешний архив

От модального эталона отличаются 31 из 110 представлений ручного словаря примерно из 17 элементов, 38 из 110 порогового словаря глубины один и 36 из 110 глубины два. Полные

Таблица 2

Контролируемые изменения и расстояние до измененного варианта

Процедура	Изменение	Расстояние
FB_AirFilter	Удален ручной сброс защелки	2
FB_ActuatorSim	Удалено ограничение хода	3
FanController. DetectFlowTransitions	Удалена фиксация таймаута	3
LiquidStageBase. DecideState	Удалено таймерное подтверждение пуска	2
RecuperatorStageBase. DecideState	Удалена защитная ветвь	2
RecirculationStage. DecideState	Инвертирована аварийная команда прямотока	1

сигнатуры различаются в 65 и 75 случаях: 59 и 68 % против 35 и 33 % для пороговых. Это различия последовательностей, не обнаружение функциональных дефектов.

Обобщение редких сигнатур снижает различимость. Выбор эталона при равенстве частот менял показатели по типам мутаций: без исходных программ и разметки они не приводятся.

Обсуждение результатов

Частотное построение заменяет ручное перечисление. Большая глубина сохраняет структуру, уменьшая повторяемость; порог объединяет редкие формы. Покрытие не измеряет переносимость и безопасность.

Теряется тождество переменных одинаковой роли и типа: y := a AND b и y := a AND a, а также doOpen := TRUE и doClose := TRUE могут совпасть. Нужны канонические номера с сохранением связей между вхождениями и признаки назначения выходов из интерфейса, привязки канала или спецификации.

Укрупнение литералов может скрыть изменение выдержки с пяти до пятисот секунд: критичные параметры требуют точных значений или обоснованных классов. Сортировка аргументов теряет позиционное значение, пропуск условий циклов и меток выбора — сведения об управлении. Эти потери нельзя трактовать как семантическую эквивалентность.

Редактирующее расстояние штрафует перестановку независимых действий. Для отличения допустимой перестановки от опасной замены нужны зависимости и доменные последствия. Словарь следует фиксировать до оценки новых программ и проверять на независимых проектах с разделением управляющего и транспортного кода.

Экспертная разметка допустимых реализаций и ошибок реально сгенерированного кода необходима для оценки чувствительности, специфичности и порогов. Шесть изменений и корпусное покрытие этого не заменяют. STMutants не представляет ошибки вентиляции и здесь не имеет подтвержденного эталона поведения.

Заключение

Получены 238 частых шаблонов с покрытием 54,0 % и пять категорий хвоста. Устойчивость к лексической замене и различимость изменений установлены в пределах протокола, без доказательства функциональной корректности. Для профиля ошибок нужны сохранение тождества сигналов и критичных литералов, расширение анализа и экспертная проверка.

ЛИТЕРАТУРА

1. Koziolek H., Gruener S., Ashiwal V. ChatGPT for PLC/DCS Control Logic Generation. *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA).* 2023:1–8. DOI: 10.1109/ETFA54631.2023.10275411.
2. Fakih M., Dharmaji R., Moghaddas Y., Quiros Araya G., Ogundare O., Al Faruque M. A. LLM4PLC: Harnessing Large Language Models for Verifiable Programming of PLCs in Industrial

Control Systems. *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 2024:192–203. DOI: 10.1145/3639477.3639743.

3. Yefet N., Alon U., Yahav E. Adversarial Examples for Models of Code. *Proceedings of the ACM on Programming Languages*. 2020;4(OOPSLA):1–30. DOI: 10.1145/3428230.
4. Roy C. K., Cordy J. R., Koschke R. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Science of Computer Programming*. 2009;74(7):470–495. DOI: 10.1016/j.scico.2009.02.007.
5. Kamiya T., Kusumoto S., Inoue K. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Transactions on Software Engineering*. 2002;28(7):654–670. DOI: 10.1109/TSE.2002.1019480.
6. Luan S., Yang D., Barnaby C., Sen K., Chandra S. Aroma: Code Recommendation via Structural Code Search. *Proceedings of the ACM on Programming Languages*. 2019;3(OOPSLA):1–28. DOI: 10.1145/3360578.
7. Allamanis M., Sutton C. Mining Idioms from Source Code. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2014:472–483. DOI: 10.1145/2635868.2635901.
8. Song Y., Lothritz C., Tang D., Bissyandé T., Klein J. Revisiting Code Similarity Evaluation with Abstract Syntax Tree Edit Distance. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. 2024;2:38–46. DOI: 10.18653/v1/2024.acl-short.3.
9. Allamanis M., Barr E. T., Devanbu P., Sutton C. A Survey of Machine Learning for Big Code and Naturalness. *ACM Computing Surveys*. 2018;51(4):81. DOI: 10.1145/3212695.
10. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. *Доклады Академии наук СССР*. 1965;163(4):845–848. Режим доступа: https://www.mathnet.ru/rus/dan31411.
11. Kabir M. H., Islam M. R., Lou H. H. STMutants: A Mutation Testing Dataset for Structured Text Programs in Industrial Automation. 2026. arXiv:2606.05499v1. DOI: 10.48550/arXiv.2606.05499.