

КОНЕЧНЫЙ АВТОМАТ ДЛЯ ПОИСКА СТРОКОВЫХ ЗНАЧЕНИЙ В ИСХОДНОМ КОДЕ PERL-ПРОГРАММ

А. А. Мышенков

FINITE-STATE MACHINE FOR SEARCHING STRING VALUES IN PERL SOURCE CODE

A. A. Myshenkov

Аннотация. *Актуальность и цели.* При оптимизации или обфускации кода одним из главных требований является сохранение функциональных возможностей программы. Это зависит от точности результатов лексического анализа. В данной работе рассматривается решение проблемы поиска строковых значений в исходном тексте на языке программирования Perl. Основной сложностью проведения такого поиска является особенность языка Perl, позволяющая записывать один и тот же алгоритм разными способами, в том числе и объявление строковых значений. *Материалы и методы.* Лексический анализ строковых значений осуществляется с помощью конечного автомата, на вход которого подаются символы строк кода. *Результаты.* Разработан алгоритм конечного автомата, который может быть использован в методах обфускации. *Выводы.* В результате исследования разработан алгоритм, позволяющий получить точные результаты лексического анализа, связанные с нахождением строковых значений. Благодаря этому соблюдаются условия сохранения функциональности программы и уменьшается вероятность возникновения ошибок при проведении обфускации.

Ключевые слова: лексический анализ, Perl, конечный автомат.

Abstract. *Background.* During the optimization or obfuscation of source code one of the major requirement is saving of program functionality. It depends on the accuracy of the result of lexical analysis. In this paper decision of string values search problem in Perl source code is researched. The main complexity of such searching is a feature of Perl, which allows to code one algorithm by different ways including string values. *Materials and methods.* Lexical analysis for searching string values is carried with finite-state machine. Characters of code line are input to finite-state machine. *Results.* Finite-state machine algorithm was designed. It can be used in obfuscation methods. *Conclusions.* Because of research, developed algorithm, which allows get accurate results of lexical analysis for search string values. Thereby a condition is observed to save program functionality and decrease probability of errors.

Key words: lexical analysis, Perl, finite-state machine.

Введение

Лексический анализ текстов на языках программирования наряду с синтаксическим и семантическим анализом имеет очень обширную область применения. Кроме использования в компиляторах и интерпретаторах методы лексического анализа применяются также для построения сред разработки, оптимизации и обфускации кода. При применении методов оптимизации или обфускации кода одним из главных требований является функциональность, т.е. полное сохранение функциональных возможностей программы [1].

Главным образом функциональность зависит от точности результатов лексического анализа. В данной статье рассмотрим решение проблемы поиска строковых значений на примере анализа текста на языке программирования Perl. Естественно, лексический анализ проводится в самом интерпретаторе Perl, а также существуют стандартные модули форматирования, такие как Deparse, но данных средств недостаточно для получения точных данных о местоположении строковых значений в коде. Однако эти средства могут использоваться в качестве вспомогательных.

Постановка задачи

Одной из особенностей языка Perl является принцип «There's More Than One Way To Do It» («Есть больше одного способа сделать это») [3]. Это подразумевает возможность записи одного и того же алгоритма множеством способов. Данный принцип применим и к записи строковых значений.

В Языке Perl строковые значения определяются операторами кавычек (“ и ‘), операторами q, qq, в которых ограничителями могут выступать как парные символы ((), {}, <>, []), так и повторяющиеся одиночные символы из множества букв и спецсимволов [2]. Операторы кавычек и символы ограничителей строковых значений сами могут находиться внутри строкового значения при использовании оператора экранирования. Таким образом, строковые значения являются частным случаем контекстно-свободной грамматики вложенных скобок. Это означает, что лексический анализ с применением шаблонов регулярных выражений ко всей строке невозможен, так как необходимо учитывать операторы экранирования и вложенность скобок. В связи с этим лексический анализ строковых значений осуществляется с помощью конечного автомата, на вход которого подаются символы строк кода. Состояния конечного автомата меняются при обнаружении операторов кавычек, ограничителей строковых значений, операторов скобок со счетчиком вложенности скобок, операторов экранирования.

Для избегания ошибок при интерполяции переменных необходимо провести форматирование текста с помощью стандартного модуля Perl Deparse [3]. В процессе форматирования интерполяция переменных заменяется конкатенацией. Это позволяет точно определить, что внутри строковых значений не будет интерполированных имен переменных. Все переменные в этом случае соединяются со строковыми значениями оператором конкатенации (.).

Для избегания путаницы будем называть искомые строки PERL-строками.

Алгоритм работы

Рассмотрим подробнее алгоритм конечного автомата M для поиска PERL-строк

$$M = (S, I^*, Q, F_q, F_i),$$

где S – входная строка, множество входных символов длиной l , подаваемая на вход автомата; i – индекс элемента множества входных символов; I – множество индексов:

$$S = \{s_1, \dots, s_l\}, i \in I = \{1, \dots, l\}, l = |S|.$$

Состояния Q автомата определяются несколькими триггерами T , счетчиком C^B :

$$Q = \{q_0, q_1, \dots, q_n\}, j = \{1, \dots, n\},$$

$$q_j = \{T^S, T', T'', T^Q, T^{QQ}, T^B, T^D, T^E, C^B\},$$

$$T \in \{0, 1\}, C \in \mathbb{Z}.$$

I^* – множество индексов, которые соответствуют найденным символам PERL-строк:

$$I^* = \{i_1^*, \dots, i_k^*\}, I^* \subset I, S^* \subset S, v_{i^*} \in S^*, i^* \in I^*.$$

Таким образом, S^* – подстрока с найденными PERL-строками, а I^* – массив индексов для нахождения подстроки PERL-строк в основной строке.

Триггер пробела T^S определяет состояние автомата, при котором возможно появление операторов начала строкового значения. Например, после форматирования кода операторы кавычек могут располагаться после пробела или скобок:

```
$x = 'string';
$y = $hash{"key"};
@a = ('a', 'b', 'c');
$string = 'concatenated' . 'string' . "\n";
```

В данном случае PERL-строками будут являться значения: string, key, a, b, c, concatenated, string, \n. Символы, располагающиеся перед операторами кавычек в данном примере, – это символ пробела, символы открывающихся скобок { и (.

Триггер одинарной кавычки T' определяет состояние автомата после открытия одинарной кавычки.

Триггер двойной кавычки T'' определяет состояние автомата после открытия двойной кавычки.

Триггеры операторов q и qq – T^Q и T^{QQ} соответственно.

Триггер скобок в качестве ограничителей строковых значений – T^B , B^O и B^C – символы открывающей и закрывающей скобки соответственно: $B^O \in \{ (, \{, [, < \}$ и $B^C \in \{), \},], > \}$.

Для проверки сбалансированности скобок используем счетчик $C^B \in \mathbb{Z}$. При $C^B = 0$ скобки считаются сбалансированными, т.е. количество открывающих и закрывающих скобок равно. При появлении открывающей скобки значение счетчика увеличивается на 1, соответственно, при появлении закрывающей скобки значение счетчика уменьшается на 1. Счетчик сбалансированности скобок работает для символов B^O и B^C , которые были определены оператором q или qq. Символы других скобок игнорируются. Экранированные символы скобок также игнорируются.

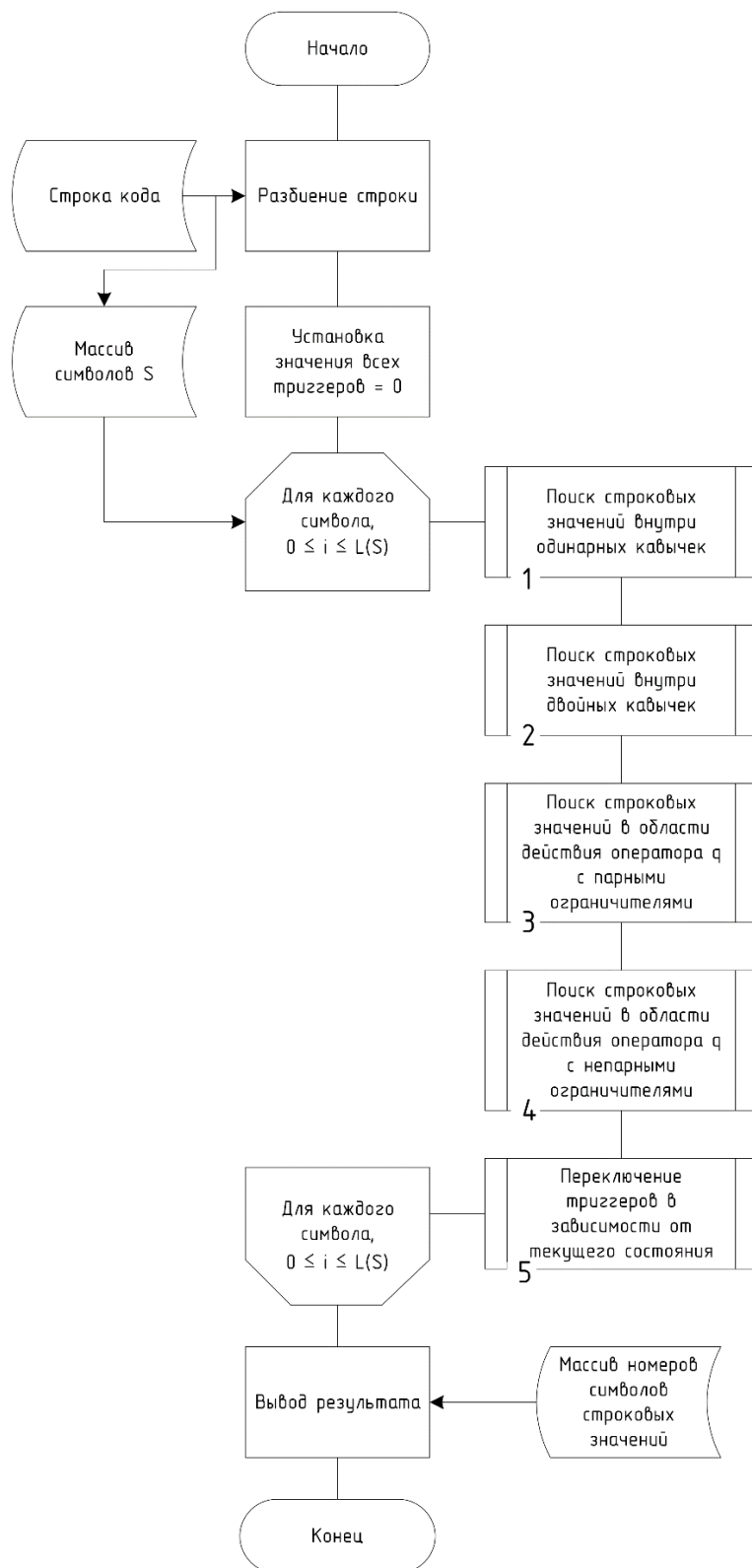


Рис. 1. Блок-схема алгоритма поиска строковых значений

В основном цикле выполняется пять подпрограмм, отмеченных на рисунке цифрами от 1 до 5. Блоки 1–4 реализуют функцию F_i , т.е. непосредственно поиск символов, соответствующих строковому значению. Блок 5 реализует функцию F_q и переключает триггеры, таким образом обеспечивая переход состояний.

На рис. 2 изображен алгоритм подпрограммы 1. В данном примере показан поиск строковых значений внутри одинарных кавычек. Состояния определяются триггерами одинарной кавычки и экранирования:

$$Q \in \{q_i, q_j\}, q_i = \{T', T^E\}, \text{ где } s \in \{', \backslash\}, q_j = \{T'\}, \text{ где } s \neq '.$$

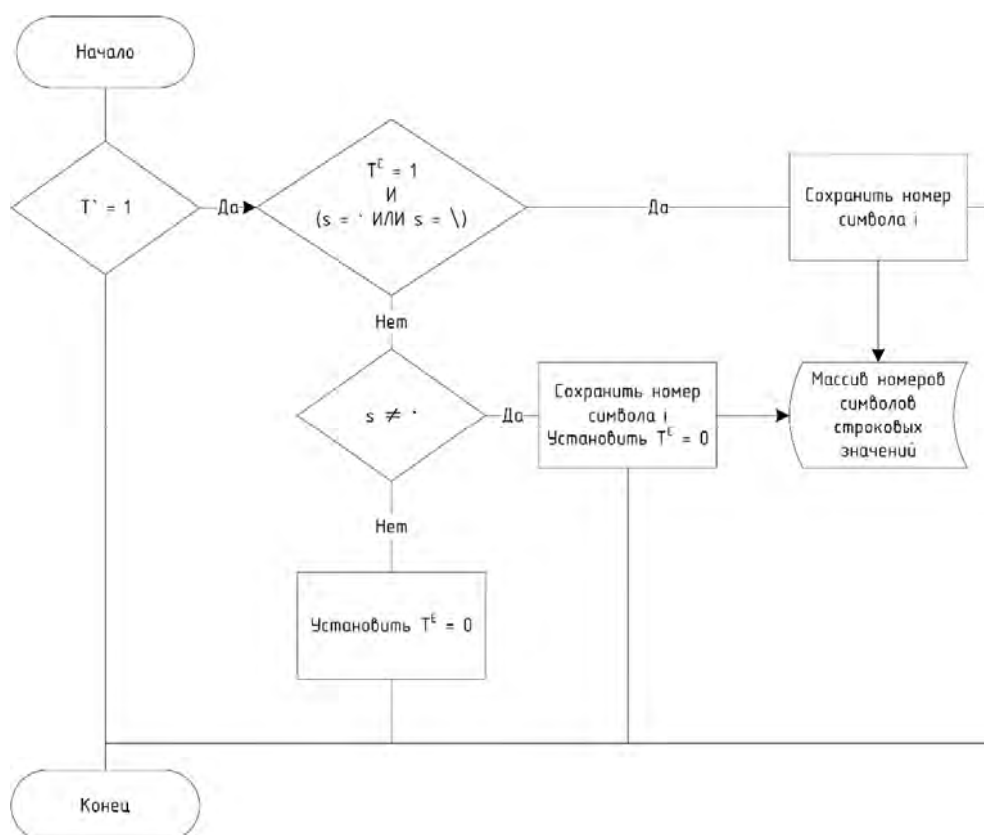


Рис. 2. Блок-схема подпрограммы поиска значений внутри одинарных кавычек

Пример работы алгоритма

Рассмотрим пример работы автомата на следующей строке кода:
\$x = 'abc';

i	s_i		T^S	T'	T''	T^Q	T^{QQ}	T^B	T^D	T^E	C^B		B^O	B^C	I^*
0	“\$”		0	0	0	0	0	0	0	0	0		∅	∅	∅
	↓		↓										↓		
Перейти без изменений															
1	“x”		0	0	0	0	0	0	0	0	0		∅	∅	∅
	↓		↓										↓		

Перейти без изменений															
2	“ “		0	0	0	0	0	0	0	0		∅	∅	∅	
	↓		↓										↓		
Обнаружен символ пробела. Установить $T^S = 1$ и перейти.															
3	“ = “		1	0	0	0	0	0	0	0		∅	∅	∅	
	↓		↓										↓		
Установить $T^S = 0$ и перейти.															
4	“ “		0	0	0	0	0	0	0	0		∅	∅	∅	
	↓		↓										↓		
Обнаружен символ пробела. Установить $T^S = 1$ и перейти.															
5	“ “ “		1	0	0	0	0	0	0	0		∅	∅	∅	
	↓		↓										↓		
Обнаружен символ одинарной кавычки и $T^S = 1$. Установить $T' = 1$, установить $T^S = 0$ и перейти.															
6	“ а “		0	1	0	0	0	0	0	0		∅	∅	∅	
	↓		↓										↓		
$T' = 1$, добавить индекс в I^* , перейти.															
7	“ Ъ “		0	1	0	0	0	0	0	0		∅	∅	{6}	
	↓		↓										↓		
$T' = 1$, добавить индекс в I^* , перейти.															
8	“ с “		0	1	0	0	0	0	0	0		∅	∅	{6, 7}	
	↓		↓										↓		
$T' = 1$, добавить индекс в I^* , перейти.															
9	“ “ “		0	1	0	0	0	0	0	0		∅	∅	{6, 7, 8}	
	↓		↓										↓		
Обнаружен символ одинарной кавычки и $T' = 1$. Установить $T' = 0$, перейти.															
10	“ , “		0	0	0	0	0	0	0	0		∅	∅	{6, 7, 8}	
	↓		↓										↓		
Конец строки.															

Таким образом, в результате выполнения получаем множество найденных индексов $I^* = \{6, 7, 8\}$. Отсюда следует, что найденная PERL-строка равна $S^* = \{s_6, s_7, s_8\} = abc$.

Заключение

В результате проведенного исследования получен алгоритм конечного автомата для поиска строковых значений в тексте исходного кода программ на языке Perl. В результате работы конечного автомата можно получить точное местоположение найденных символов в строке, что невозможно при использовании стандартных средств интерпретатора Perl. Полученные данные могут быть использованы для применения методов обфускации, например, метода шифрования строковых данных с соблюдением условия сохранения

функциональности программы и уменьшением вероятности возникновения ошибок, связанных с неточным определением строковых значений.

Список литературы

1. Варновский, Н. П. Математические проблемы обфускации / Н. П. Варновский, В. А. Захаров, Н. Н. Кузюрин // Математика и безопасность информационных технологий : материалы конф. в МГУ (28–29 октября 2004 г.). – М. : МЦНМО, 2005. – С. 65–91.
2. Шохирев, М. В. Язык программирования Perl 5 / М. В. Шохирев. – М. : Интернет-университет информационных технологий, 2007. – 192 с.
3. B::Deparse. – URL: <http://perldoc.perl.org/B/Deparse.html> (дата обращения: 14.02.2016).

Мышенков Артем Алексеевич
аспирант,
Самарский государственный технический
университет
E-mail: artem.myshenkov@gmail.com

Myshenkov Artem Alekseevich
postgraduate student,
Samara State Technical University

УДК 004.423.2

Мышенков, А. А.

Конечный автомат для поиска строковых значений в исходном коде PERL-программ / А. А. Мышенков // Модели, системы, сети в экономике, технике, природе и обществе. – 2016. – № 2 (18). – С. 223–230.