СИСТЕМА ПОИСКА И ЗАМЕНЫ ТЕКСТА С ПОМОЩЬЮ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Черемисинов Д. И.
Объединённый институт проблем информатики Национальной академии наук Беларуси
Минск, Республика Беларусь
E-mail: {cher@newman.bas-net.by

В теоретической информатике и в теории формальных языков, регулярный язык - формальный язык, заданный регулярным выражением. Регулярный язык может быть определен как язык, распознаваемый конечным автоматом. В программировании регулярными выражениями называются несколько языков программирования, распознающих языки, которые не могут быть выражены регулярным выражениям теоретической информатики. Предлагается язык и система программирования с синтаксисом регулярных выражении теоретической информатики и детерминированной семантикой.

Введение

Поиск информации — одно из основных использований компьютера. Одна из простейших задач поиска информации — поиск точно заданной подстроки в строке. Поиск подстрок - очень важная тема в задачах обработки текста. Текстовое представление остается главной формой, используемой при информационном обмене в системах автоматизированном проектировании. Соответствующие алгоритмы поиска подстрок – основные компоненты практического программного обеспечения. Кроме того, эти алгоритмы демонстрируют программирование методов, которые служат парадигмами других задач разработки программ. Наконец, поиск подстрок играет важную роль в теоретической информатике, обеспечивая модель ключевых проблем. На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки.

Операция поиска подстроки с заменой ее вхождения другой строкой играет главную роль в теории нормальных алгоритмов Маркова [1]. В теории нормальных алгоритмов Маркова эта операция называется подстановкой слов. Система подстановок задает нормальный алгоритм Маркова. Система применяется к любому слову y по следующему правилу: рассматривается первая подстановка, если левая ее часть входит в y, то первое вхождение заменяется правой частью первой подстановки. Результат – слово $y1$ – теперь будет являться исходным и к нему снова применяется первая подстановка. Если первая подстановка неприменима, то берется вторая и так далее.

Формально языковую подстановку можно рассматривать как упорядоченную тройку (L_1, L_2, θ) состоящую из двух языков L_1, L_2 и отображения $\theta : L_1 \to L_2$. Операция применения языковой подстановки состоит в построении слова W из слова V. Если существует $x \in L_1$ и x - вхождение в V , то W получается заменой в V вхождения x словом $y = \theta(x) \in L_2$. Если $x \in L_1$ не содержится в V, то подстановка (L_1, L_2, θ) к V не применима.

I. Регулярные выражения в языках программирования

Известные методы описания языковых подстановок различаются способами задания образца L_1. Широкоизвестным способом задания образцов являются регулярные выражения [2-3]. Наибольшее развитие регулярные выражения получили в Perl, где их поддержка встроена непосредственно в интерпретатор. В других языках, как правило, используются реализующие регулярные выражения дополнения и модули сторонних разработчиков. В VBScript и JScript используется объект RegExp, в C/C++ можно использовать библиотеки Regex++ и PCRE (Perl Compatible Regular Expression), а также ряд менее известных библиотек, для Java существует целый набор расширений – ORO , RegExp, Rex и gnu.regexp. Синтаксис регулярных выражений до сих пор не полностью стандартизован. Еще меньше стандартизована семантика. Семантика в разных языках регулярных выражений базируется на следующих типах виртуальных машин.

DFA (Deterministic Finite-state Automaton – детерминированные конечные автоматы) машины работают линейно по времени, поскольку не нуждаются в откатах (и никогда не проверяют один символ дважды). Они могут гарантированно найти самую длинную строку из возможных. Однако, поскольку DFA содержит только конечное состояние, он не может найти образец с обратной ссылкой и, из-за отсутствия конструкций с явным расширением, не ловит подвыражений. Они используются, например, в awk, egrep или lex.

Традиционные NFA-машины (NonDeterministic Finite-state Automaton – недетерминированные конечные автоматы) используют «жадный» алгоритм отката, проверяя все возможные расширения регулярного выражения в определенном порядке и выбирая первое подходящее значение. Поскольку традиционный NFA конструирует определенные расширения регулярного выражения для поиска соответствий, он может искать подвыражения и backreferences. Но из-за

откатов традиционный NFA может проверять одно и то же место несколько раз. В результате работает он медленнее. Поскольку традиционный NFA принимает первое найденное соответствие, он может и не найти самое длинное из вхождений. Именно такие механизмы регулярных выражений используются в Perl, Python, Emacs, Tcl и .Net.

POSIX NFA - машины похожи на традиционные NFA-машины, за исключением «терпеливости» – они продолжают поиск, пока не найдут самое длинное соответствие. Поэтому POSIX NFA-машины медленнее традиционных, и поэтому же нельзя заставить POSIX NFA предпочесть более короткое соответствие длинному. Одно из главных достоинств POSIX NFA-машины – наличие стандартной реализации.

II. Алгебра образцов

Будем рассматривать строку символов как динамический объект, состояния, которого характеризуются парами значений *(y,i)*, где i - указатель - целое число, $1 < i < (|y| + i)$.

Образцом f называется функция преобразования состояний строк, такая, что полученное в результате ее применения состояние строки отличается от исходного только значением указателя [4], причем оно в результате применения образца может только увеличиться. Функция f - образец, если для любых состояний $\alpha = (w, j), \beta = (v, i)$ и $\alpha = f(\beta), w = v, j \geq i$. Состояние (v, i) задает представление слова y в виде конкатенации $y = ab$, где $|a| = i - 1$.

Если $\alpha = f(\beta), \alpha = (y, i), \beta = (y, j)$, то пара состояний α, β задают представление слова y в виде $axc, |ax| = j - 1, |a| = i$. В этом случае образец f распознает слово x. Пусть $p(f, \alpha)$ - предикат, обозначающий утверждение, что образец f определен на состоянии α.

Введем следующие операции над образцами. Катенацией fg образцов f и g называется операция, определяемая условным выражением Мак-Карти [5] $fg(\alpha) = P(f, \alpha) \rightarrow g(f(\alpha))$. Альтернация $f \vee g$ образцов f и g определяется условным выражением $f \vee g(\alpha) = P(f, \alpha) \rightarrow (f(\alpha)); P(g, \alpha) \rightarrow (g(\alpha))$. Итерация образца $f*$ определяется через степень следующим выражением с бесконечным числом членов $f * (\alpha) = \neg P(f^1, \alpha) \rightarrow \alpha; \neg P(f^2, \alpha) \rightarrow f^1(\alpha); \ldots; \neg P(f^i, \alpha) \rightarrow f^i - 1(\alpha); \ldots$. Степень определяется рекурсивно выражениями $f^1(\alpha) = f(\alpha)$ и $f^n(\alpha) = f^{n-1}(f(\alpha))$. Отрицание $f\neg$ - операция, задаваемая выражением $f\neg(\alpha) = \neg P(f, \alpha) \rightarrow \alpha$.

Множество образцов, порождаемых этой алгеброй совпадает с множеством регулярных выражений Клини [3]. Между образцами алгебры образцов и регулярными выражениями Клини существует функциональное сходство.

Теорема. Для любого образца f язык $L[f]$, распознаваемый этим образцом, является подмножеством языка, определяемого регулярным выражением f.

III. Система программирования образцов

Для того чтобы образец задавал всю операцию подстановки, нужно «встроить» в него процедуру построения слова W. Анализируемая структура V отображена в структуре образца, и наиболее простой способ использования информации о структуре V состоит в разбиении процедуры построения W на операции, выполняемые после обнаружения подструктур в V в той последовательности, в которой анализируются эти подструктуры. Встраивание действий по построению W делается аналогично языку JACC.

Система программирования образцов представляет собой транслятор с языка обозначений образцов в язык C, и в системе программирования C является препроцессором компилятора C. В программе, построенной этим препроцессором, для представления состояния строк используются структура из двух переменных v, i. Значения v являются строками, i — числами. Схема программы, реализующей заданный образец, представляет собой сеть с одним входом и двумя выходами, помеченными *Failure* и *Success* соответственно. Операции этой сети являются операциями применения компонентных образцов. Последовательность применения операций соответствует порядку просмотра схемы, который всегда начинается от входа. При окончании просмотра на выходе с пометкой *Success* текущее значение v, i задает строку, являющуюся результатом применения образца, в противном случае образец не применим.

Для построения схемы программы используется обратная польская запись образца. При обнаружении в польской записи обозначения примитивного образца в магазин заносится схема программы, реализующей образец. Обнаружение символов операций над образцами вызывает выборку из магазина одной или двух подсхем и формирование новой подсхемы комбинированием выбранных схем. Эта схема затем заносится в магазин. При синтаксически правильной записи обозначений исходного образца в конце просмотра магазин содержит единственную схему, которая и является схемой программы, реализующей данный образец.

1. Марков, А. А. Теория алгорифмов / А. А. Марков, Н. М. Нагорный //М. : Наука, Гл. ред. физ.-мат. лит., 1984. — 376 с.
2. Фридл, Дж. Регулярные выражения / Дж. Фридл // СПб.: «Питер», 2001. — 352 с.
3. Kleene, S. C. Representation of Events in Nerve Nets and Finite Automata / S. C. Kleene // Automata Studies. Princeton University Press. 1956. – pp. 3–42.
4. Черемисинов, Д. И. Программирование подстановок языков / Д. И. Черемисинов // Программирование, – 1981. – № 5. – С. 30-–37.