Свойства регулярных языков, лексические анализаторы

Содержание

Свойства регулярных языков

Итак, мы с вами увидели, что регулярные языки можно задать тремя разными способами: 1. с помощью порождающих регулярных (автоматных) грамматик; 2. с помощью конечных автоматов; 3. с помощью регулярных выражений.
В последнем случае регулярные выражения фактически обозначают регулярные множества. А регулярные множества оказываются замкнутыми относительно многих операций: объединения, пересечения, дополнения. Это означает, что если L_1, L_2 - регулярные языки, то выполнение соответствующей операции, для которой регулярный язык замкнут, над словами языков L_1, L_2 дает также регулярный язык. В частности справедливо следующее утверждение:

Теорема

Класс автоматных языков замкнут относительно итерации, конкатенации и объединения.

Доказательство этого утверждения прямо следует из определения регулярного множества: если имеем два регулярных множества, то их объединение, конкатенация, итерация также являются регулярными множествами.
Очевидно, что не все языки регулярны, и существует мощный механизм доказательства нерегулярности некоторых языков, известный как лемма о накачке (или лемма о разрастании).

Лемма о накачке

Для начала рассмотрим на некотором примере, как доказать, что язык не является регулярным. Для этого прежде всего рассмотрим довольно простое утверждение, которое известно как **принцип Дирихле**:

Теорема

Если кролики рассажены в клетки, причём число кроликов больше числа клеток, то хотя бы в одной из клеток находится более одного кролика.

Пусть имеется язык $L = \{0^n 1^n : n \geq 1\}$, докажем, что он не является регулярным. Этот язык состоит из всех цепочек вида 01, 0011, 000111 и так далее, содержащих один или несколько нулей, за которыми следует точно такое же количество единиц.

Если бы данный язык был регулярным, то для него существовал бы соответствующий детерминированный конечный автомат. Заметим, что автомат должен иметь только конечное множество состояний, допустим это число M. Поскольку n может быть любым числом, превосходящим конечное число состояний автомата M, то всегда существуют два разных $n_1 \neq n_2$, для которых на входных цепочках 0^{n_1} и 0^{n_2} автомат вынужден находиться в одном и том же состоянии, обозначим его q_x (принцип Дирихле). Если далее для такого случая на входе 1, то детерминированный автомат переходит из состояния q_x в какое-то новое состояние q_y, независимо от того, сколько до этого было нулей на входе. Но теперь из состояния q_y автомат должен уметь различить входные цепочки 1^{n_1} и 1^{n_2}, что очевидно невозможно. Поскольку ясно, что автомат уже не может различить, какое количество нулей n_1 или n_2 было на входе.

Лемма о накачке

Несмотря на представленное доказательство этого частного случая, существует более универсальное утверждение, которое известно как теорема о накачке.

Теорема (теорема о накачке)

Пусть L — регулярный бесконечный язык. Существует константа n (зависящая от L), для которой каждую цепочку w из языка L, удовлетворяющую неравенству $|w| \geq n$, можно разбить на три цепочки $w = xyz$ так, что выполняются следующие условия.

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. *Для любого $k \geq 0$ цепочка $xy^k z$ также принадлежит L.*

Это значит, что всегда можно найти такую цепочку y недалеко от начала цепочки w, которую можно “накачать”. Таким образом, если цепочку y повторить любое число раз или удалить (при $k = 0$), то результирующая цепочка все равно будет принадлежать языку L.

Доказательство леммы о накачке

Доказательство.

Пусть L какой-то регулярный язык. Тогда рассмотрим соответствующий ему автомат. Данный автомат принимает конечное множество состояний, обозначим их количество n. Тогда рассмотрим такую цепочку $w = a_0 a_1 \dots a_m, m \geq n$, которая принадлежит этому языку. Это означает, что на цепочке w автомат переходит в заключительное состояние. А поскольку цепочка длиннее количества состояний n, то для цепочки w существует хотя бы одно состояние, через которое мы проходим несколько раз. Обозначим такое состояние q_x.

Разобъем цепочку w на три части $w = xyz, x = a_0 a_1 \dots a_i, y = a_{i+1} a_{i+2} \dots a_j, z = a_{j+1} a_{j+2} \dots a_m, 0 \leq i \leq j \leq n$. Предположим, что на цепочке x мы как раз переходим в состояние q_x, тогда существует такое j, что на цепочке y мы тоже перейдем в состояние q_x. Поскольку по окончанию цепочки y мы опять перейдем в состояние q_x , тогда ясно, что повторение цепочки y приведет опять в состояние q_x. Поэтому повторение цепочки y возможно сколько угодно раз и при этом получаемая цепочка $xy^k z$ будет все равно принадлежать языку L. □

Использование леммы о накачке

Лемма о накачке позволяет с общих позиций доказывать нерегулярность некоторых языков. Например, нерегулярность того же языка $L = \{0^n 1^n : n \geq 1\}$ можно доказать следующим образом. Возможно три отдельных случая применения леммы о накачке в цепочках $0^n 1^n$:

- Можно попробовать накачать какую-то цепочку из нулей, т.е. если $0^n 1^n$ принадлежит языку, то и $0^{n+m} 1^n$ принадлежит языку, что очевидно неверно.
- Можно попробовать накачать какую-то цепочку из единиц, т.е. $0^n 1^{n+m}$ должно принадлежать языку, что очевидно также неверно.
- Можно попробовать накачать какую-то цепочку из нулей и единиц, т.е. $0^n (0^k 1^l)^m 1^n$ должно принадлежать языку, что очевидно также неверно.

Таким образом, нет ни одной возможности накачать цепочку $0^n 1^n$, принадлежащую языку L. Однако выполнение леммы о накачке автоматически не означает регулярность соответствующего языка.Так есть нерегулярные языки, которые также допускают накачку. Есть конечные регулярные языки, которые, очевидно, не допускают накачку. Лемма лишь утверждает, что если бесконечный язык регулярен, то он обязан допускать накачку. Т.е. это условие является необходимым, но не является достаточным. Поэтому лемма о накачке обычно используется именно для доказательства нерегулярности языка, если удается показать, что строки никак не могут накачиваться.

Задание 1.

Доказать нерегулярность языка $L = \{0^k 1^m : k \leq m\}$.
Заметим, что этот язык допускает накачку. В частности можно взять любую подстроку 1^m и накачивать ее. Таким образом, прямое использование леммы о накачке не проходит.
Для доказательства заметим, что в лемме о накачке утверждается существование некоторой константы n, определяемой количеством состояний соответствующего детерминированного автомата, начиная с которой все подцепочки длины больше n можно накачивать. Возьмем цепочку $0^{n+k}1^m, 0 \leq k, m \geq n + k$, тогда подцепочку 0^{n+k} можно пытаться накачивать, но до бесконечности этого следать не получится, так как мы быстро достигнем условия $m \leq n + k$, а в этом случае цепочка уже не будет принадлежать L.

Задание 2.

Доказать нерегулярность языка правильных скобочных выражений.
Данный язык ранее мы описывали с помощью следующей порождающей грамматики $S \to ()|(S)|SS$. По виду правил, эта грамматика конечно не является регулярной. Однако возможно существует и регулярная грамматика, задающая тот же язык.
Но, оказывается, можно доказать, что такой грамматики не существует.
Используем для этого следующее утверждение: если L_1, L_2 - регулярные языки, то $L = L_1 \cap L_2$ - тоже регулярный язык. Для доказательства нерегулярности языка правильных скобочных выражений L_1, рассмотрим регулярный язык $L_2 = (^*)^*$ (он является регулярным, поскольку мы используем только операцию итерации над скобками), данный язык можно задать $L_2 = \{(^m,)^n : n \geq 0, m \geq 0\}$. Предположим, что L_1 является регулярным, тогда $L_1 \cap L_2 = (^n)^n$ (поскольку L_1 содержит только сбалансированное количество скобок, то от L_2 остаются только цепочки $(^n)^n$.)Значит язык $(^n)^n$ тоже должен быть регулярным. Но это не так, поскольку на языке 0^n1^n мы показали, что он не является регулярным.

Проблемы, разрешимые для регулярных языков

Регулярные языки представляют собой очень удобный тип языков. Для них разрешимы многие проблемы, которые оказываются неразрешимыми для других типов языков.

Ниже представлен перечень таких разрешимых для регулярных языков проблем:

- *Проблема эквивалентности.* Для двух регулярных языков L_1, L_2 требуется проверить, являются ли они эквивалентными.
- *Проблема принадлежности.*Для данного регулярного языка L и цепочке символов α требуется проверить, принадледит ли эта цепочка α данному языку L.
- *Проблема пустоты языка.* Требуется проверить, содержит ли заданный язык хотя бы одну непустую цепочку.
- *Проблема однозначности.* Для заданной регулярной грамматики построить эквивалентную ей однозначную регулярную грамматику.

Понятие о лексических анализаторах

Определение

Лексема - структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка.

Лексемами естественных языков являются слова. Лексемами языков программирования являются идентификаторы, константы, ключевые слова, знаки операций и т.д.

Определение

Лексический анализатор - это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка.

На вход лексического анализатора поступает текст исходной программы, а выходная информация передается другим компонентам компилятора (в частности синтаксическому анализатору).

Понятие о лексических анализаторах

Перечень лексем, полученных лексическим анализатором записывается в так называемую **таблицу лексем**. Каждой лексеме в таблице лексем соответствует некий уникальный условный код, зависящий от типа лексем. Информация о некоторых типах лексем также заносится в **таблицу идентификаторов**.

Таблица лексем содержит весь текст программы, разбитый на лексемы. Один идентификатор может встречаться в этой таблице много раз, в тех местах, где он встречается в самой программе. В таблице идентификаторов каждый идентификатор употребляется только один раз, а сама таблица идентификаторов служит для хранения текущего значения идентификатора.

Как правило, язык констант, идентификаторов и т.д. является регулярным. Следовательно, основой для реализации лексических анализаторов служат регулярные грамматики и конечные автоматы.

При работе лексический анализатор должен, кроме распознавания лексем, определять границы лексем, которые в тексте явно не указаны, выполнять действия по сохранению информации об обнаруженной лексеме, или выдавать сообщение об ошибке, если лексема неверна.

Понятие о лексических анализаторах

Нужно отметить, что определение границ лексем является для лексического анализатора наиболее сложной задачей. В ряде случаев он не может с ней справиться. Например, в коде на языке C возможна такая конструкция $k = i + + + + + j$. Это означает, что постинкремент переменной i складывается с прединкрементом переменной j. Увидеть это лексический анализатор сможет только просмотрев всю строчку и перебрав все возможные неправильные варианты. Но это можно сделать только на этапе синтаксического анализа. Поэтому в таких случаях лексический анализатор вынужден взаимодействовать с синтаксическим анализатором. Возможно две стратегии такого взаимодействия: последовательное взаимодействие, параллельное взаимодействие.

При последовательной стратегии лексический анализатор просматривает весь код от начала до конца и преобразует его в таблицу лексем. Если лексический анализатор не смог распознать какую-то лексему, то она считается ошибочной.

При параллельном взаимодействии лексический анализатор выделяет очередную лексему и передает ее синтаксическому анализатору. Синтаксческий анализатор, выполнив разбор очередной конструкции языка, может подтвердить правильность найденной лексемы и обратиться за следующей или же отвергнуть найденную лексему.

Понятие о лексических анализаторах

Последовательная работа лексического анализатора является более простой. Поэтому разработчики языков программирования стремятся организовать взаимодействие именно таким образом. Для большинства языков программирования границы лексем распознаются по заданным терминальным символам - пробелам, знакам операций, разделителям - точкам с запятой, запятым.
На практике разработчики компиляторов сознательно идут на то, что исключают некоторые правильные, но неоднозначные варианты. Попытки усложнить лексический анализ неизбежно приведут к необходимости взаимодействовать с синтаксическим анализатором, а это неизбежно снизит эффективность работы всего компилятора. Возникающие накладные расходы никак не оправдаются достигаемым эффектом - распознаванием строк с сомнительными лексемами.

Практическая реализация лексических анализаторов

При практической реализации компилятор может иметь в своем составе не один, а несколько лексических анализаторов, каждый из которых предназначен для распознавания своего типа лексем. Тогда алгоритм работы может выглядеть следующим образом:

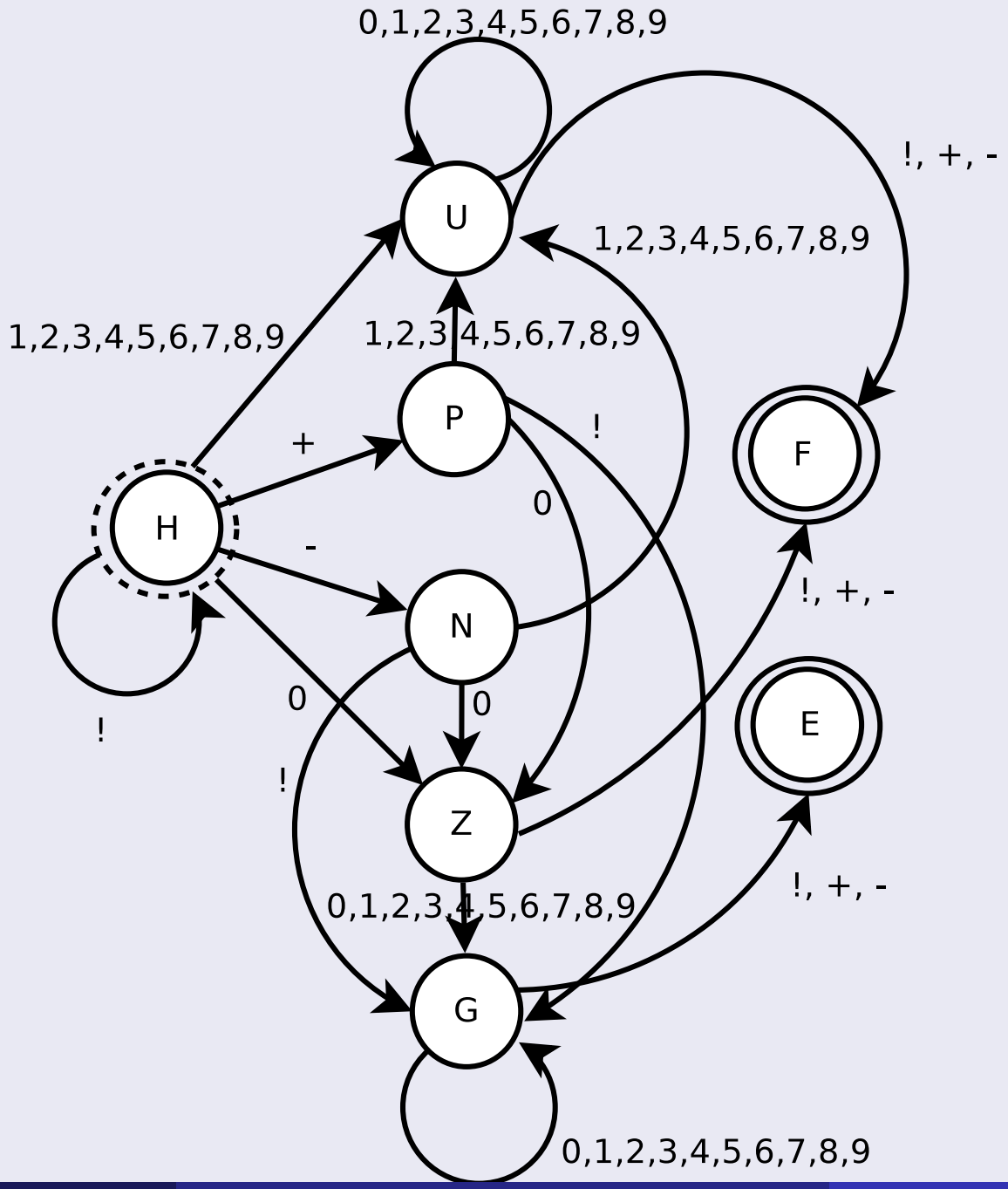
- из входного потока выбирается очередной символ, в зависимости от которого запускается нужный лексический анализатор.
- запущенный сканер просматривает входной поток, выделяя символы, которые входят в лексему, до обнаружения очередного символа, который может ограничить лексему, либо до обнаружения ошибочного символа.
- при успешном разпознавании лексемы, она заносится в таблицу лексем и идентификаторов, алгоритм возвращается к начальному этапу с того места, на котором остановился сканер.
- если выявлена ошибка, то в зависимости от реализации лексического анализатора, он либо прекращает работу или пытается проанализировать и выделить следующую лексему.

В целом техника построения лексических анализаторов основывается на синтезе детерминированного или недетерминированного конечного автомата с дополнением функциями распознавания ошибок и заполнения таблиц лексем и идентификаторов.

Практическая реализация лексических анализаторов

Рассмотрим пример синтеза лексического анализатора на примере выделения из текста лексем, представляющих собой целочисленные константы.
Каждая целочисленная константа должна состоять из цифр $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, она может начинаться со знаков плюс $+$ или минус $-$ и не может начинаться с незначащих нулей.
Например, такие варианты допустимы: $+24210, -2197, 975, 0, +0, -0$, а такие варианты нет: $+001233, -0344, 000328, 021$.
Будем обозначать все символы отличные от смысловых из алфавита $+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ одним символом !. Такое преобразование над входным текстом всегда можно без труда выполнить. Разделителями идентификаторов является символ !, а также символы $+, -$, если они встречены в середине числа.
Будем считать, что при работе конечного автомата после прихода его в конечное состояние он либо распознал идентификатор, либо выявил ошибку. Далее его можно опять запускать с текущей позиции для выявления следующей лексемы.
Построим детерминированный конечный автомат, отвечающий данной задаче лексического анализа.

Синтез лексического анализатора в виде конечного автомата



Реализация вспомогательных функций

Функция формирования текста из файла

```
def read_file(f):
    ff = open(f,'r')
    result=""
    for i in ff:
        result+=i
    return result
```

Реализация вспомогательных функций

Функция заменяет все символы, кроме символов алфавита, на заданный символ

```
def replace(text, abc, sep):
    lst=list(text)
    for i in range(len(lst)):
        if not(lst[i] in abc):
            lst[i]=sep
    result=""
    for i in lst:
        result=result+str(i)
    return result
```

Реализация лексического анализатора

Модифицируйте программный код детерминированного конечного автомата для реализации лексического анализатора целочисленных констант.