ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ
Практикум 1
27.10.2014

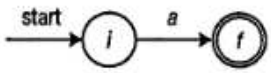
Андрей Саутин, группа 395

Задача 11. Даны регулярное выражение α в обратной польской записи и слово $u \in \{a, b, c\}^*$. Найти длину наибольшего подслова v слова u, такого что v принадлежит $L(\alpha)$.

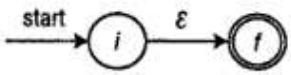
Алгоритм 1. ***Построение НКА на основе регулярного выражения.***

Для построения НКА на основе регулярного выражения воспользуемся построением Томпсона, описанным в Dragon Book (стр. 132). Заведем стек автоматов (изначально пустой), уже построенных по некоторым частям регулярного выражения. Будем бежать по регулярному выражению и выполнять различные действия в зависимости от символа, который только что считали:

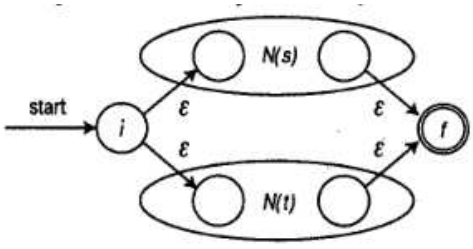
- считали a, b, c (символ алфавита). Строим автомат, состоящий только из одного начального и одного терминального состояний и добавляем переход из начального в терминальное по считанному символу. Построенный автомат кладем в стек.



- считали 1. Строим автомат, состоящий только из одного начального и одного терминального состояний и добавляем ε-переход из начального в терминальное. Построенный автомат кладем в стек.



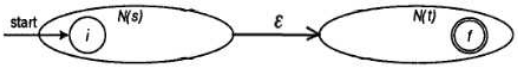
- считали $+$. Достаем из стека два автомата. Пусть первый был построен по регулярному выражению s, а второй — по t. Объединяем их в один автомат следующим образом:



Создаем новое начальное состояние i, из которого проводим ε-переходы в начальные состояния извлеченных из стека автоматов. Также создаем новое терминальное состояние f, в которое проводим ε-переходы из терминальных состояний извлеченных автоматов.

Заметим, что любой путь из i в f должен проходить либо через первый автомат $N(s)$, либо через второй автомат $N(t)$. Поэтому построенный автомат распознает $L(s) \cup L(t) = L(s + t)$.

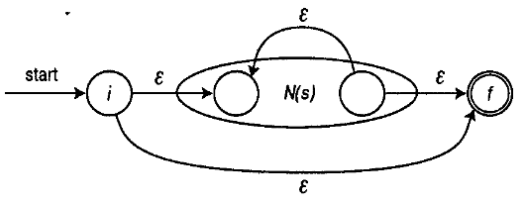
- считали $\cdot$. Достаем из стека два автомата. Пусть первый был построен по регулярному выражению s, а второй — по t. Объединяем их в один автомат следующим образом:



Начальное состояние i автомата $N(s)$ будет начальным состоянием объединенного автомата, а терминальное состояние f автомата $N(t)$ — его терминальным состоянием. Сделаем ε-переход от терминального состояния $N(s)$ к начальному состоянию $N(t)$.

Заметим, что путь из i в f должен проходить сначала через первый автомат $N(s)$, а затем через второй автомат $N(t)$. Поэтому построенный автомат распознает $L(s) \cdot L(t) = L(s \cdot t)$.

- считали *. Достаем из стека автомат. Пусть он был построен по регулярному выражению s. Построим автомат $N(s^*)$ следующим образом:



Создаем новое начальное состояние i, из которого проводим ε-переход в начальное состояние автомата $N(s)$. Также создаем новое терминальное состояние f, в которое проводим ε-переход из терминального состояния автомата $N(s)$, а также из состояния i. После чего добавляем ε-переход из терминального состояния автомата $N(s)$ в его начальное состояние.

Заметим, что в новом автомате из i в f мы можем пройти по ε-переходу (тем самым разрешаем слово ε), а также можем пройти через $N(s)$ любое количество раз. Поэтому построенный автомат распознает $L(s^+) \cup L(\varepsilon) = L(s^*)$.

Таким образом, если исходное регулярное выражение α было корректно, то после прохода по нему и выполнения вышеописанных действий в стеке будет лежать только один автомат — соответствующий регулярному выражению α.

Время работы алгоритма: $O(|\alpha|)$, т.к. мы пробегаем по всему выражению и для каждого его символа делаем константное число операций.

Замечание. Построенный автомат обладает следующими свойствами, которые целиком вытекают из алгоритма его построения:

- Количество состояний автомата $= O(|\alpha|)$, т.к. на каждом шаге построения мы создаем не более 2-х новых состояний.
- Построенный автомат имеет ровно одно начальное и ровно одно терминальное состояния, кроме того, в начальное состояние нет входящих переходов, а из терминального нет исходящих.
- Каждое состояние имеет либо один исходящий переход по символу алфавита, либо не более 2-х исходящих ε-переходов.

Алгоритм 2. ***Определение, принимается ли входное слово u НКА, построенным в алгоритме 1.***

Для ответа на вопрос, допускается ли входное слово u автоматом, воспользуемся алгоритмом, описанным в *Dragon Book*, стр. 136.

Заведем два массива: *unprocessedStates* и *processedStates*. В *unprocessedStates* будем хранить состояния автомата, в которых мы можем оказаться после очередной итерации алгоритма. В *processedStates* будем добавлять состояния, в которые мы можем попасть из какого-либо состояния из *unprocessedStates* по очередному символу слова u.

Определение. *ε-замыкание множества состояний S автомата N — это множество состояний автомата N, в которые можно попасть из какого-либо состояния множества S по произвольному (возможно нулевому) числу ε-переходов.*

Сначала добавим в *unprocessedStates* начальное состояние автомата и в том же массиве вычислим его ε-замыкание. Затем будем бежать по буквам слова и для каждого символа выполнять следующее:

- Для каждого состояния из массива *unprocessedStates* вычислим множество состояний, достижимых из него одним переходом по текущему символу слова. Все состояния из полученного множества будем добавлять в массив *processedStates*.
- Переместим содержимое массива *processedStates* в *unprocessedStates*. Массив *processedStates* очистим. Тем самым подготовили массивы для следующей итерации цикла.
- Наконец, вычислим в массиве *unprocessedStates* его ε-замыкание. Итерация цикла закончена, переходим к следующей букве слова.

В итоге, получим множество состояний, в которые мы могли попасть, последовательно переходя по символам нашего слова. Если среди этих состояний есть терминальное, то наше слово u разрешается автоматом, иначе — не разрешается.

Заметим, что алгоритм является online-алгоритмом, т.к. в конце каждой итерации мы имеем корректно посчитанное множество всех состояний, в которые мы могли попасть, последовательно переходя по символам некоторого префикса нашего слова. Тогда можно на каждой итерации алгоритма проверять, лежит ли терминальное состояние в этом множестве, и отвечать на вопрос, принимается ли некоторый префикс нашего слова u автоматом.

Кроме того, алгоритм можно останавливать, если в конце очередной итерации массив *unprocessedStates* оказался пустым. Этот случай соответствует ситуации, когда нет ни одного состояния, в которое мы могли бы прийти, последовательно переходя в НКА по символам уже обработанного префикса слова u.

Время работы алгоритма: $O(|u| \cdot |\alpha|)$, т.к. мы делаем $|u|$ итераций и на каждой из них делаем константное число действий с каждым состоянием в массиве *unprocessedStates*. Состояний в массиве не может быть больше, чем общее количество состояний в автомате (дважды одно и то же состояние в массив мы добавлять не будем, для этого у каждого состояния храним флаг, было ли оно уже добавлено в массив *unprocessedStates* на этой итерации), поэтому их $O(|\alpha|)$.

Алгоритм 3. *Поиск длины наибольшего подслова v слова u, такого что $v \in L(\alpha)$.*

(a) Построим НКА, соответствующий заданному регулярному выражению α, используя алгоритм 1.

(b) Модифицируем алгоритм 2 так, чтобы он возвращал длину наибольшего префикса v слова u, такого что $v \in L(\alpha)$. Модифицированный алгоритм 2 назовем 2'.

(c) Известно, что каждая подстрока слова u является префиксом некоторого его суффикса. Будем перебирать все суффиксы слова u. На каждой итерации будем с помощью алгоритма 2' вычислять длину наибольшего префикса $v \in L(\alpha)$ текущего суффикса u' и обновлять переменную-ответ, беря максимум из значения, уже лежащего в ней, и результата работы алгоритма 2' для текущего суффикса.

Итоговое время работы: $O(|u|^2 \cdot |\alpha|)$. Пункт 1 выполняется за $O(|\alpha|)$. Пункт 2 не портит асимптотики алгоритма 2, поэтому алгоритм 2' выполняется за $O(|u| \cdot |\alpha|)$. В пункте 3 мы перебираем суффиксы строки u за $O(|u|)$ и для каждого из них запускаем алгоритм 2'. Итого: $O(|\alpha| + |\alpha| \cdot |u| \cdot |u|) = O(|\alpha| \cdot |u|^2)$.

Итоговые затраты по памяти: $O(|\alpha| + |u|)$. Храним строку u, подаваемую на вход, и состояния НКА, количество которых — $O(|\alpha|)$.