Домашнее задание № 3, Формальные языки

Шестаков Денис Владиславович

28.09.2020

1. 1) $\{uabv \mid u, v \in \{a, b\}*,\ |u| = |v|,\ u \neq v^R\}$

   Возьмем строку $b^n aba^n$. Тогда $y = bb*$, пусть $|y| = l$. Подставляем $i = 0$, тогда слово имеет вид $b^{n-l}aba^n$. $|u| \neq |v| \Rightarrow b^{n-l}aba^n \notin$ языку, т.е. язык нерегулярный

   2) $\{a^k c^m e^n | k \geq 0,\ n \geq 0, m = k + n + 1\}$

   Пусть нам сказали число N. Тогда подставим $k = n = N \Rightarrow m = 2N + 1$. Получаем слово $a^N c^{2N+1} e^N$. Пусть $|y| = l$, берем $i = 0:\ a^{N-l} c^{2N+1} e^N$. Т.к. $2N + 1 \neq 2N - l$, то слово не принадлежит языку $\Rightarrow$ язык нерегулярный.

   3) $\{a^n | \exists p : p \geq n, p \in \mathbb{P}, p + 2 \in \mathbb{P}\}$

   Поймем, что в независимости от существования такого p для любого n наш язык будет регулярный. Если такое p существует для любого n, тогда такой язык просто описывается регулярным выражением $a*$. Если такое p существует не для всех n, тогда возьмем $max n$: $\exists p : p \geq n, p \in \mathbb{P}, p + 2 \in \mathbb{P}$. Но такой язык будет конечен $\Rightarrow$ регулярным.

2. Обычная реализация парсера находится в файле *PythonParcer.py*. Замерим на нескольких тестах до оптимизаций время работы:

РВ	Входящая строка	Время, сек
(bb|aa)*	bbbbbaaaaa	8.345379
1(0|1)*0	1000001011	12.004702
(ab)*a	abababababa	30.688751
(a|b)*c(a|b)*	aaabbcaabbb	>100

   Видно, что работает довольно долго. Теперь оптимизируем парсер, новый вариант находится в файле *PythonParcerUpdated.py*. После оптимизаций получаем:

РВ	Входящая строка	Время, сек
(bb|aa)*	bbbbbaaaaa	0.000075
1(0|1)*0	1000001011	0.000096
(ab)*a	abababababa	0.000095
(a|b)*c(a|b)*	aaabbcaabbb	0.000076

   Видно, что время уменьшилось очень сильно (порядка 1000000 раз на данных тестах). Чтобы оптимизированный парсер работал дольше 2 секунд, нужно потрудиться. Сильнее всего в рекурсию уходят *concat* и *alt*, поэтому будем строить регулярное выражения в основном используя их. Описание построения регулярных выражений находится в конце *PythonParcerUpdated.py*. Первый пример отрабатывает за 2.118025 сек, второй за 2.828463. В данных случаях, оптимизации не так сильно отрабатывают, а чтобы получить символ, нужно пройти достаточно глубоко по рекурсии. В первом случае время увеличивается из-за конкатенации и проверки выражения для случая $R*$. Во втором случае

alt расположены таким образом, что нужно пройти через все символы, чтобы определить, принадлежит ли символ выражению, что также сильно увеличивает количество опреаций.