Лабораторная работа №4

1 Контекстная свобода

Для начала заметим, что возможно лишь 2 перехода по S, при том эти переходы рекурсивно не содержат ее, потому подставим в правило $T \to aST$ 2 возможных S. Тогда вместо этого правило будет 2 новых:

$$T \to aabT, \quad T_1.a < 2, \quad T_0.a := T_1.a + 3,$$
$$T \to abTaaTT, \quad T_1.a + T_2.a + 2 > T_3.a, \quad T_0.a := T_1.a + T_2.a + T_3.a + 3.$$

Также заметим, что у нас значение атрибута становится не больше двух только лишь при использовании правил 3 и 5, исходя из преобразованного правила 2. Также в новом правиле $T \to aabT$ можно отдельно рассмотреть случай с равенством атрибута 0 и 1.

Далее заметим, что в S атрибуты перестали играть какую-то роль, потому допустимо их и вовсе убрать из него. В конечном счете получим эквивалентную грамматику:

$$S \to ab$$
$$S \to bTaaT$$
$$T \to b^*ba, \quad T.a := 0$$
$$T \to a, \quad T.a := 1$$
$$T \to aabbb^*a, \quad T.a := 3$$
$$T \to aaba, \quad T.a := 4$$
$$T \to bT, \quad T_1.a \geq 2, \quad T_0.a := T_1.a + 1$$
$$T \to abTaaTT, \quad T_1.a + T_2.a + 2 > T_3.a, \quad T_0.a := T_1.a + T_2.a + T_3.a + 3$$

Покажем, что последнее правило способствует порождению не КС-языка. По факту достаточно рассмотреть слова вида

$$abaabaaabT_1'aaaT_3'T_3$$

(атрибуты подберем погранично, чтобы они удовлетворяли погранично неравенству и при накачках можно было в нужную сторону их изменить), тогда накачка отдельных компонент со штрихами выведет T_2 из языка, если синхронно, то нулевой накачкой выводим все слово из языка. Отдельные накачки T_1 и T_3 очевидно выводят из языка.

Тогда, исходя из этого, слово для накачки будем таким:

$$abaabaaabb^naabaaaab^{(n+2)}aabab^{(2n+11)}aaba$$

Пересечем с соответствующим регулярным выражением и тогда, накачивая, как описано выше, можно во всех случаях вывести слово из языка.

Поэтому язык не является КС.

2 Оптимизированный парсер

Будем работать с новой грамматикой. Так как возможно только 2 перехода по S, один из которых тривиален, то задача сводится к рассмотрению возможных T.

Посмотрим на префикс: он будет определять правило, по которому надо делать разбор. Приоритет отдаем регулярным выражениям, так как они были ранее выделены в отдельные случаи, которые никак не пересекаются с остальными.

Далее, если слово начинается не на `ab`, то по предпоследнему правилу убираем слева `b`, не забывая про условия на атрибут. Если начинается на `ab`, то исходя из возможных расположений `aa` выбираем один T слева и два справа, которые рекурсивно обрабатываем.

3 Оценка вычислительной сложности

В худшем сценарии цепочки вызовов образуют полное дерево разбора:

- Внешний цикл в `parse_S` перебирает $O(n)$ позиций для поиска `"aa"`
- Для каждой позиции выполняются два вызова `parse_T` сложностью $T(k)$ и $T(m)$
- Каждый вызов `parse_T` в худшем случае (правило $T \to abTaaTT$) имеет:
  - Цикл по позициям `"aa"`: $O(n)$
  - Вложенный цикл по разбиениям: $O(n)$
  - Три рекурсивных вызова: $T(k) + T(m) + T(p)$

Верхняя граница сложности определяется произведением:

$$O(n) \times O(n) \times O(n) = O(n^3)$$

где:

- Первый множитель $O(n)$ — внешний цикл в `parse_S`
- Второй множитель $O(n)$ — поиск `"aa"` в `parse_T`
- Третий множитель $O(n)$ — перебор разбиений в правиле $T \to abTaaTT$

Таким образом, оптимизированный парсер имеет кубическую сложность $O(n^3)$ в худшем случае.

4 Тестирование

Для проверки эффективности предложенного алгоритма было проведено тестирование на двух типах данных:

- словах, принадлежащих языку;
- словах, не принадлежащих языку.

Для каждой длины измерялось время выполнения двух алгоритмов:

- обычного парсера;
- оптимизированного парсера, описанного в разделе 2.

Результаты представлены на графиках ниже (значения времени приведены в миллисекундах).

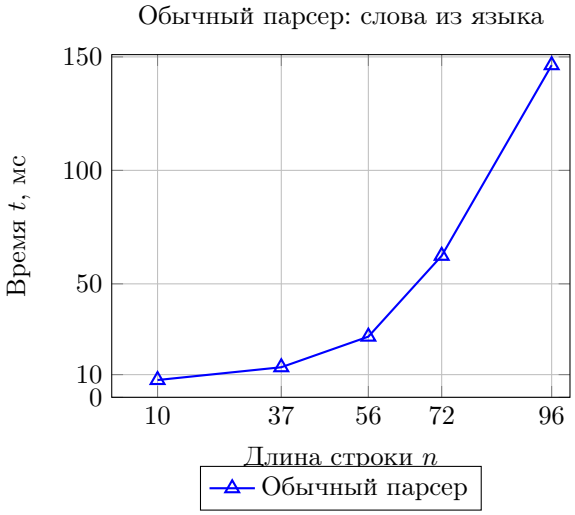

Рис. 1: Время разбора для слов из языка (обычный парсер)

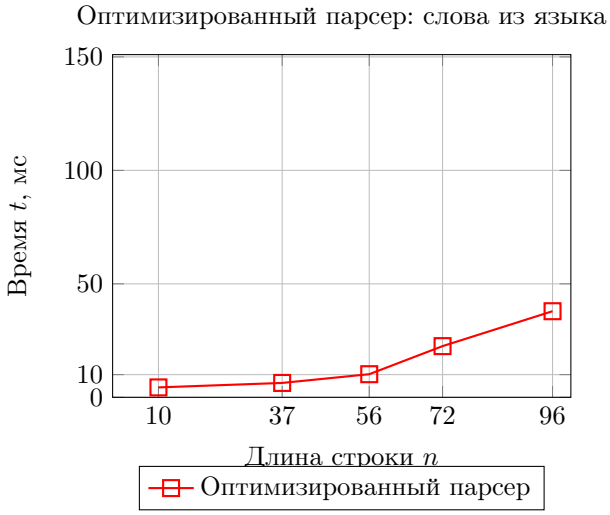

Рис. 2: Время разбора для слов из языка (оптимизированный парсер)

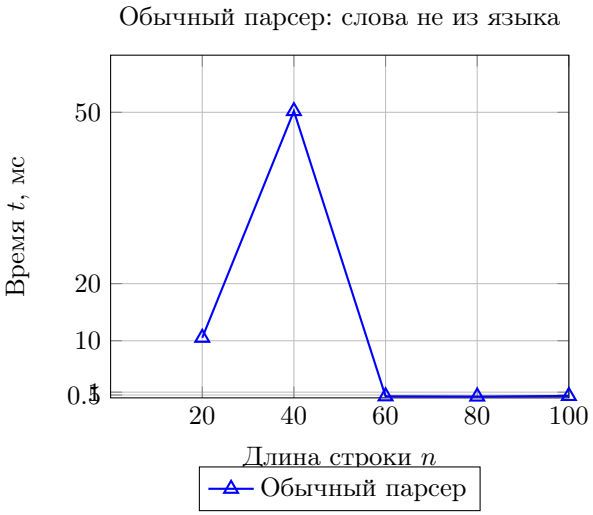

Рис. 3: Время разбора для слов не из языка (обычный парсер)

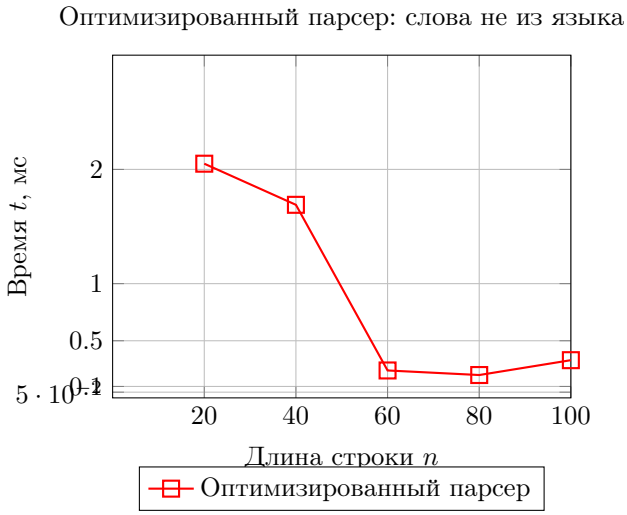

Рис. 4: Время разбора для слов не из языка (оптимизированный парсер)