# Лабораторная работа №1

Лиганкина Анна
Вариант 20

7 октября 2025 г.

## Содержание

**1 Задание** **2**

**2 Исследование SRS** **3**
2.1 Завершимость . . . 3
2.2 Классы эквивалентности . . . 3
2.3 Локальная конфлюэнтность . . . 3
2.4 Пополняемость по Кнуту-Бендиксу . . . 4
2.5 Построение SRS $\mathcal{T}'$ . . . 5

**3 Тестирование** **6**
3.1 Фазз-тестирование эквивалентности . . . 6
3.2 Метаморфное тестирование . . . 8

# 1 Задание

Дана SRS:

| | | | |
|---|---|---|---|
| $cb \to ba$ | $bba \to ba$ | $cac \to cc$ | $baca \to cabba$ |
| $aaa \to aa$ | $bbb \to b$ | $bab \to cac$ | $caab \to bb$ |
| $aba \to ba$ | $bbc \to c$ | $ccc \to c$ | $caac \to bc$ |
| $ac \to cc$ | $bcc \to cc$ | $babb \to ba$ | $aabcaa \to a$ |
| $baa \to ba$ | $ba \to cab$ | $babc \to \varepsilon$ | |

По имеющейся SRS определить:

— завершимость;

— конечность классов эквивалентности по НФ (для построения эквивалентностей считаем, что правила могут применяться в обе стороны). Если их конечное число, то построить минимальную систему переписывания, им соответствующую;

— локальную конфлюэнтность и пополняемость по Кнуту-Бендиксу.

По SRS $\mathcal{T}$ тем самым (исключая случай, когда она сразу локально конфлюэнтна или конечна и минимальна) строится другая SRS $\mathcal{T}'$, которая должна сохранять те же классы эквивалентности. Если исходная SRS завершима, то правила в $\mathcal{T}'$ должны удовлетворять условию убывания левой части относительно правой по выбранному фундированному порядку $\preceq$.

Провести автоматическое тестирование предполагаемой эквивалентности указанных SRS.

**Фазз-тестирование эквивалентности**: строится случайное слово $\omega$ и случайная цепочка переписываний его в $\omega'$ по $\mathcal{T}$. Проверить, можно ли получить $\omega'$ из $\omega$ (или наоборот) в рамках правил $\mathcal{T}'$.

**Метаморфное тестирование**: выбрать инварианты, которые должны сохраняться (либо монотонно изменяться) при переписывании в рамках $\mathcal{T}$. Породить случайную цепочку переписываний над случайным словом в $\mathcal{T}'$ и проверить выполнимость инвариантов. Как минимум два разных инварианта.

# 2 Исследование SRS

## 2.1 Завершимость

Исходная SRS $\mathcal{T}$ незавершима, т.к. существует цикл:

$$caba \xrightarrow{aba \to ba} cba \xrightarrow{cb \to ba} baa \xrightarrow{ba \to cab} caba$$

## 2.2 Классы эквивалентности

Нормальных форм в SRS $\mathcal{T}$ бесконечное число. Для доказательства можно привести пример строки, состоящей лишь из подряд идущих подстрок $abc$, которые ни во что не редуцируются.

Рассмотрим, лежат ли все НФ длины 1 и пустая строка в одном классе эквивалентности. Т.к. стрелки рассматриваются в обе стороны, имеем такие цепочки:

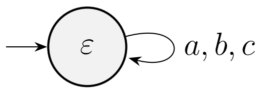
$$\varepsilon \xleftrightarrow{babc \leftrightarrow \varepsilon} babc \xleftrightarrow{aba \leftrightarrow ba} ababc \xleftrightarrow{babc \leftrightarrow \varepsilon} a$$

$$\varepsilon \xleftrightarrow{babc \leftrightarrow \varepsilon} babc \xleftrightarrow{bba \leftrightarrow ba} bbabc \xleftrightarrow{babc \leftrightarrow \varepsilon} b$$

$$\varepsilon \xleftrightarrow{babc \leftrightarrow \varepsilon} babc \xleftrightarrow{bab \leftrightarrow cac} cacc \xleftrightarrow{cac \leftrightarrow cc} ccc \xleftrightarrow{ccc \leftrightarrow c} c$$

Т.е. НФ $a, b, c$ лежат в одном классе эквивалентности $\varepsilon$. Т.к. алфавит состоит из этих букв, то все слова принадлежат одному классу эквивалентности '$\varepsilon$'.

Т.к. число классов эквивалентности конечно, то построим автомат по данной SRS:




Минимальной SRS, описывающей этот автомат, будет $a \to \varepsilon$, $b \to \varepsilon$, $c \to \varepsilon$.

## 2.3 Локальная конфлюэнтность

SRS $\mathcal{T}$ не является локально конфлюэнтной, т.к. для слова $babc$ в один шаг преобразований может быть переписана в слова $cacc$ (по правилу $bab \to cac$) и к слову $\varepsilon$, являющемуся НФ. При этом слово $cacc$ невозможно привести к $\varepsilon$.

Таким образом, система $\mathcal{T}$ не является локально конфлюэнтной.

## 2.4 Пополняемость по Кнуту-Бендиксу

Для применения алгоритма Кнута-Бендикса введем фундированный порядок — length-lexicographic order $\prec$ — и, в соответствии с ним, переориентируем правила (что сохранит классы эквивалентности исходной SRS $\mathcal{T}$).

Теперь наша SRS выглядит следующим образом:

| | | | |
|---|---|---|---|
| $aaa \to aa$ | $babc \to \varepsilon$ | $caab \to bb$ | $cac \to bab$ |
| $aabcaa \to a$ | $bba \to ba$ | $caac \to bc$ | $cb \to ba$ |
| $aba \to ba$ | $bbb \to b$ | $cab \to ba$ | $cc \to ac$ |
| $baa \to ba$ | $bbc \to c$ | $cabba \to baca$ | $ccc \to c$ |
| $babb \to ba$ | $bcc \to cc$ | $cac \to cc$ | |

Найдем множество всех критических пар:

1. Правила $aaa \to aa$ и $aabcaa \to a$ порождают критическую пару $\langle aabcaa, aa\rangle$ из слов $aaabcaa$ и $aabcaaa$. НФ для этой пары равны соответственно $\langle a, aa\rangle$. Поэтому, т.к. $a \prec aa$, введем правило $aa \to a$. Теперь можно редуцировать правила $aaa \to a$ и $baa \to ba$;

2. Правила $aa \to a$ и $aabcaa \to a$ порождают критические пары $\langle abcaa, a\rangle$ и $\langle aabca, a\rangle$ из слова $aabcaa$. НФ для этой пары равны соответственно $\langle abca, a\rangle$ и $\langle abca, a\rangle$. Поэтому, т.к. $a \prec abca$, введем правило $abca \to a$. Теперь можно редуцировать правило $aabcaa \to a$;

3. Правила $aa \to a$ и $caab \to bb$ порождают критическую пару $\langle cab, bb\rangle$ из слова $caab$. НФ для этой пары равны соответственно $\langle ba, bb\rangle$. Поэтому, т.к. $ba \prec bb$, введем правило $bb \to ba$. Теперь можно редуцировать правила $bba \to ba$ (т.к. $bba \xrightarrow{bb \to ba} baa \xrightarrow{aa \to a} ba$) и $babb \to ba$ (т.к. $babb \xrightarrow{bb \to ba} baba \xrightarrow{aba \to ba} bba \xrightarrow{\text{переход выше}} ba$);

4. Т.к. существуют 2 различных правила переписывания строки $cac$ рассмотрим критическую пару $\langle cc, bab\rangle$. НФ для этой пары равны $ac$ и $bab$. Поэтому, т.к. $cc \prec bab$, введем правило $bab \to ac$. Теперь можно редуцировать правило $cac \to bab$;

5. Правила $bab \to ac$ и $babc \to \varepsilon$ порождают критическую пару $\langle acc, \varepsilon\rangle$ из слова $babc$. НФ для этой пары равны соответственно $\langle ac, \varepsilon\rangle$. Поэтому, т.к. $\varepsilon \prec ac$, введем правило $ac \to \varepsilon$. Теперь можно редуцировать правило $babc \to \varepsilon$, т.к. можно переписать $babc \xrightarrow{bab \to ac} acc \xrightarrow{cc \to ac} aac \xrightarrow{aa \to a} ac \to \varepsilon$;

Промежуточная SRS:

| | | | |
|---|---|---|---|
| $aa \to a$ | $bb \to ba$ | $caac \to bc$ | $cc \to ac$ |
| $ac \to \varepsilon$ | $bbb \to b$ | $cab \to ba$ | $ccc \to c$ |
| $abca \to a$ | $bbc \to c$ | $cabba \to baca$ | |
| $aba \to ba$ | $bcc \to cc$ | $cac \to cc$ | |
| $bab \to ac$ | $caab \to bb$ | $cb \to ba$ | |

6. Правила $aa \to a$ и $ac \to \varepsilon$ порождают критическую пару $\langle ac, a\rangle$ из слова $aac$. НФ для этой пары равны соответственно $\langle \varepsilon, a\rangle$. Поэтому, т.к. $\varepsilon \prec a$, введем правило $a \to \varepsilon$. Теперь можно редуцировать правила $aa \to a$, $aba \to ba$, $caab \to bb$ и заменить $bb \to ba$ и $bbb \to b$ на $bb \to b$;

7. Правила $a \to \varepsilon$ и $ac \to \varepsilon$ порождают критическую пару $\langle c, \varepsilon\rangle$ из слова $ac$. Добавляем правило $c \to \varepsilon$ и редуцируем правила $ac \to \varepsilon$, $cab \to ba$, $cabba \to baca$, $cac \to cc$, $cb \to ba$, $cc \to ac$, $ccc \to c$

8. Правила $bcc \to cc$ и $c \to \varepsilon$ порождают критическую пару $\langle cc, bc\rangle$ из слова $bcc$. НФ для этой пары равны соответственно $\langle \varepsilon, b\rangle$, поэтому добавим правило $b \to \varepsilon$. Теперь можно редуцировать все правила, кроме $a \to \varepsilon$, $b \to \varepsilon$ и $c \to \varepsilon$.

Таким образом, пополненная SRS имеет вид:

$a \to \varepsilon$
$b \to \varepsilon$
$c \to \varepsilon$

## 2.5 Построение SRS $\mathcal{T}'$

На основе исследования SRS $\mathcal{T}$, эквивалентной ей SRS будем считать $\mathcal{T}'$:

$$\begin{aligned} a &\to \varepsilon \\ b &\to \varepsilon \\ c &\to \varepsilon \end{aligned}$$

т.к. она сохраняет те же классы эквивалентности.

# 3 Тестирование

## 3.1 Фазз-тестирование эквивалентности

Фазз-тестирование проводится следующим образом:

1. генерируется случайная строка $\omega$;
2. строится случайная цепочка по правилам SRS $\mathcal{T}$ в $\omega'$;
3. выбираем наименьшее из $\omega$ и $\omega'$, согласно выбранному фундированному порядку при построении $\mathcal{T}'$;
4. смотрим множество слов, которые можно получить из меньшего слова в SRS $\mathcal{T}'$;
5. проверяем, можно ли из большего слова получить элемент ранее найденного множества по SRS $\mathcal{T}'$ — если да, то $\omega$ и $\omega'$ эквивалентны.

Исходный код программы представлен в файле «fuzzer.hs».

Ниже представлен псевдокод самых важных функций:

```
findAllOccurrences(pattern, text):
    positions ← пустой список
    for i ← 0 to |text| − |pattern| do
        if подстрока text[i..] начинается с pattern then
            добавить i в positions
    return positions

replaceSubstring(idx, len, replacement, str):
    before ← str[0 : idx]
    after ← str[idx + len :]
    return before + replacement + after

getNextStates(str, rules):
    result ← ∅
    foreach (lhs, rhs) ∈ srs do
        indices ← findAllOccurrences(lhs, str)
        foreach idx ∈ indices do
            newStr ← replaceSubstring(idx, |lhs|, rhs, str)
            добавить newStr в result
    return result
```

```
findAllReachable(startStr, rules):
    visited ← {startStr}
    queue ← очередь со startStr
    while queue ≠ ∅ do
        current ← queue.Dequeue()
        newStrings ← getNextStates(current, rules)
        unvisited ← newStrings \ visited
        visited ← visited ∪ unvisited
        queue.Enqueue(unvisited)
    return visited

checkIntersection(startStr, rules, targetSet):
    if startStr ∈ targetSet then
        return True
    visited ← {startStr}
    queue ← очередь со startStr
    while queue ≠ ∅ do
        current ← queue.Dequeue()
        newStrings ← getNextStates(current, rules)
        if ∃s ∈ newStrings, такое что s ∈ targetSet then
            return True
        unvisited ← newStrings \ visited
        visited ← visited ∪ unvisited
        queue.Enqueue(unvisited)
    return False

main():
    ω ← generateRandomString(10, 20)
    ω′ ← randomlyTransform(w)
    if ω ≺ ω′ then
        shorterStr ← ω; longerStr ← ω′
    else
        shorterStr ← w′; longerStr ← w
    reachableFromShorter ← findAllReachable(shorterStr, srs𝒯′)
    areEquivalent ← checkIntersection(longerStr, srs𝒯′,
        reachableFromShorter)
    вывести areEquivalent
```

## 3.2 Метаморфное тестирование