Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 1

по курсу «Теория формальных языков»

Студент группы ИУ9-51Б Винокурова Е. С.

Преподаватель Непейвода А. Н.

Москва 2025

1 Задание

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 $\mathcal{T}$ 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 вами фундированному порядку $>$.

Провести автоматическое тестирование предполагаемой эквивалентности двух указанных SRS.

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

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

Вариант 4

fgh $\to$ fff

fgh $\to$ ggg

fgh $\to$ hhh

hh $\to$ hfhgh

gggg $\to$ ε

2 Проверка завершимости

Рассмотрим два инварианта. Первый инвариант — это количество пар ‘hh‘ в строке с учётом перекрытий. Например, для строки ‘hhh‘ количество пар будет

2. Второй инвариант — это сумма весов букв ‘g‘ в строке в зависимости от контекста. Будем считать, что если ‘g‘ находится в подстроке ‘fhgh‘, то вес такого ‘g‘ равен 0; если ‘g‘ стоит рядом с другим ‘g‘ (то есть хотя бы с одной стороны от него находится другой ‘g‘), то вес каждого такого ‘g‘ равен 2; во всех остальных случаях вес ‘g‘ равен 8.

Теперь рассмотрим сумму этих двух инвариантов для каждого правила и покажем, что она всегда строго убывает. Обозначим эту сумму через M.

Рассмотрим правило ‘fgh $\to$ fff‘. Для левой части ‘fgh‘ значение M = 8, а для правой части ‘fff‘ M = 0. Вне зависимости от контекста, в котором они находятся, M всегда уменьшается минимум на 8.

Рассмотрим правило ‘fgh $\to$ ggg‘. Для ‘fgh‘ значение M = 8, а для ‘ggg‘ M = 6. Контекст также не уменьшает разницу между левой и правой частями правила, поэтому для этого правила M уменьшается как минимум на 2.

Рассмотрим правило ‘fgh $\to$ hhh‘. Для ‘fgh‘ значение M = 8, а для ‘hhh‘ M = 2. Однако контекст может уменьшить разницу между левой и правой частями. Например, если взять строку ‘hfgh‘, которая переходит в ‘hhhh‘, получим изменение M: 8 $\to$ 3, то есть M уменьшается хотя бы на 5.

Рассмотрим правило ‘hh $\to$ hfhgh‘. Поскольку граничные буквы подстрок не меняются, контекст не влияет на разницу значений M. Таким образом, для ‘hh‘ значение M = 1, а для ‘hfhgh‘ M = 0, то есть M убывает на 1.

Рассмотрим правило ‘gggg $\to$ ε‘. Для ‘gggg‘ значение M = 8, а для ‘ε‘ M = 0. Однако при применении этого правила может возникнуть новая подстрока ‘fgh‘, где вес ‘g‘ будет 8. Проверим, что это не увеличит M. Пусть есть строка ‘fgggggh‘, для неё M = 10. Применив правило, получим ‘fgh‘, для которой M = 8. Таким образом, даже в этом случае ‘M‘ строго уменьшается.

Итак, для всех правил системы M строго убывает, и при этом M всегда неотрицательно. Следовательно, невозможно бесконечно применять правила, и система переписываний является завершимой.

3 Локальная конфлюэнтность и пополняемость по Кнуту-Бендиксу

SRS $\mathcal{T}$ локально не конфлюэнтна, так как из слова ‘fgh‘ можно получить 3 разных нормальных формы: ‘fff‘, ‘ggg‘ и ‘hfhghfhgh‘.

Построим по алгоритму Кнута-Бендикса систему $\mathcal{T}'$, которая сохраняет те же классы эквивалентности. Рассмотрим систему

fgh $\rightarrow$ fff

ggg $\rightarrow$ fgh

fgh $\rightarrow$ hhh

hfhgh $\rightarrow$ hh

gggg $\rightarrow$ ε

Здесь слова сначала сравниваются по длинне, если длина одинаковая, то они упорядочиваются лексикографически, где ‘h‘ = 1, ‘f‘ = 2, ‘g‘ = 3. Рассмотрим критические пары.

1. Из ‘fgh‘ можно получить ‘fff‘ и ‘hhh, поэтому добавим правило ‘fff‘ $\rightarrow$ ‘hhh‘. Можно убрать правило ‘fgh‘ $\rightarrow$ ‘hhh‘, так как это можно получить так ‘fgh‘ $\rightarrow$ ‘fff‘ $\rightarrow$ ‘hhh‘.

2. Из ‘ffff‘ можно получить ‘fhhh‘ и ‘hhhf‘, добавим правило ‘fhhh‘ $\rightarrow$ ‘hhhf‘.

3. Из ‘gggg‘ можно получить ε, ‘ghhh‘ и ‘hhhg‘, добавим правила ‘ghhh‘ $\rightarrow$ ‘hhhg‘ и ‘hhhg‘ $\rightarrow$ ε. Можно убрать правило ‘gggg‘ $\rightarrow$ ε, так как это можно получить так ‘gggg‘ $\rightarrow$ ‘gfgh‘ $\rightarrow$ ‘gfff‘ $\rightarrow$ ‘ghhh‘ $\rightarrow$ ‘hhhg‘ $\rightarrow$ ε.

4. Из ‘ffffg‘ можно получить ‘f‘ и ‘hhhfg‘, добавим правило ‘hhhfg‘ $\rightarrow$ ‘f‘.

5. Из ‘fffgh‘ можно получить ‘h‘ и ‘hhhff‘, добавим правило ‘hhhff‘ $\rightarrow$ ‘h‘.

6. Из ‘fgggg‘ можно получить ‘f‘ и ‘hhhhh‘, добавим правило ‘hhhhh‘ $\rightarrow$ ‘f‘.

7. Из ‘fffff‘ можно получить ‘fh‘ и ‘hf‘, добавим правило ‘fh‘ $\rightarrow$ ‘hf‘. Можно убрать правило ‘fhhh‘ $\rightarrow$ ‘hhhf‘, так как это можно получить так ‘fhhh‘ $\rightarrow$ ‘hfhh‘ $\rightarrow$ ‘hhfh‘ $\rightarrow$ ‘hhhf‘.

8. Из ‘hfhgh‘ можно получить ‘f‘ и ‘hh‘, добавим правило ‘hh‘ $\rightarrow$ ‘f‘. Можно убрать правило ‘fhhh‘ $\rightarrow$ ‘hhhf‘, так как это можно получить так ‘fhhh‘ $\rightarrow$ ‘hfhh‘ $\rightarrow$ ‘hhfh‘ $\rightarrow$ ‘hhhf‘.

9. Из ‘fghh‘ можно получить ‘ff‘ и ‘fgf‘, добавим правило ‘fgf‘ $\rightarrow$ ‘ff‘.

10. Из ‘gggg‘ можно получить ε, ‘ghf‘ и ‘hfg‘, добавим правило ‘ghf‘ $\to$ ‘hfg‘ и ‘hfg‘ $\to$ ε. Можно убрать правило ‘hhhfg‘ $\to$ ‘f‘, так как это можно получить ‘hhhfg‘ $\to$ ‘hh‘ $\to$ ‘f‘. Также можно убрать правило ‘hfhgh‘ $\to$ ‘hh‘, так как это можно получить так ‘hfhgh‘ $\to$ ‘hhfgh‘ $\to$ ‘hh‘. Также можно убрать правило ‘hhhg‘ $\to$ ε, так как это можно получить так ‘hhhg‘ $\to$ ‘hfg‘ $\to$ ε.

11. Из ‘fgfh‘ можно получить ‘f‘, ‘hff‘ и ‘hffg‘, добавим правило ‘hffg‘ $\to$ ‘hff‘ и ‘hff‘ $\to$ ‘f‘.

12. Из ‘fgfh‘ можно получить ‘f‘ и ‘fg‘, добавим правило ‘fg‘ $\to$ ‘f‘.

13. Из ‘hfg‘ можно получить ε и ‘hf‘, добавим правило ‘hf‘ $\to$ ε. Можно убрать правило ‘hhhff‘ $\to$ ‘h‘, так как это можно получить ‘hhhff‘ $\to$ ‘hhf‘ $\to$ ‘h‘. Также можно убрать правило ‘hff‘ $\to$ ‘f‘, так как это можно получить так ‘hff‘ $\to$ ‘f‘, применив добавленное правило. Можно убрать правило ‘hhhhh‘ $\to$ ‘f‘, так как это можно получить ‘hhhhh‘ $\to$ ‘hhhf‘ $\to$ ‘fhf‘ $\to$ ‘f‘.

14. Из ‘fhh‘ можно получить ‘ff‘ и ‘h‘, добавим правило ‘ff‘ $\to$ ‘h‘.

15. Из ‘ghf‘ можно получить ε и ‘g‘, добавим правило ‘g‘ $\to$ ε. Можно убрать правила ‘fgf‘ $\to$ ‘ff‘, ‘hfg‘ $\to$ ε, ‘hffg‘ $\to$ ‘hff‘ и ‘fg‘ $\to$ ‘f‘, так как их можно получить применив добавленное правило.

Критических пар больше нет, то есть система конфлюэнтна, также все действия сохраняли классы эквивалентности, то есть полученная система $\mathcal{T}'$ эквивалентна исходной и имеет вид после привидения к минимамльной:

fh $\to$ ε

hh $\to$ f

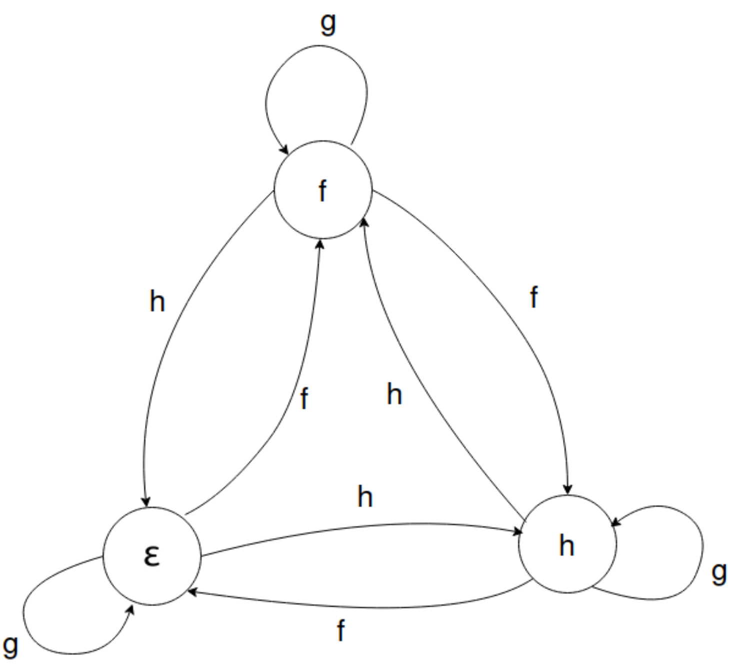
hf $\to$ ε

ff $\to$ h

g $\to$ ε

4 Классы эквивалентности

Рассмотрим классы эквивалентности полученной системы $\mathcal{T}'$, так как они совпадают с калссами эквивалентности исходной SRS, то $\mathcal{T}$ имеет такие же классы эквивалентности. Построим автомат по этой системе.



Таким образом в SRS $\mathcal{T}$ ровно 3 класса эквивалентности ε, ‘f‘ и ‘h‘.

5 Фазз-тестирование эквивалентности

Код программы представлен в Листинге 1.

```
import random
T = [
    ("fgh", "fff"),
    ("fgh", "ggg"),
    ("fgh", "hhh"),
    ("hh", "hfhgh"),
    ("gggg", "")
]
T1 = [
    ("fh", ""),
    ("hh", "f"),
    ("hf", ""),
    ("ff", "h"),
    ("g", "")
]
alphabet = ["f", "g", "h"]

def random_word():
    return "".join(random.choice(alphabet) for _ in range(17))

def apply_random_rules(word):
    for _ in range(8):
```

```python
        applicable = []
        for lhs, rhs in T:
            positions = []
            for i in range(len(word) - len(lhs) + 1):
                substring = word[i:i+len(lhs)]
                if substring == lhs:
                    positions.append(i)
            if positions:
                applicable.append((lhs, rhs, positions))
        if not applicable:
            continue
        lhs, rhs, positions = random.choice(applicable)
        pos = random.choice(positions)
        word = word[:pos] + rhs + word[pos+len(lhs):]
    return word

def apply_rules(word, rules):
    results = set()
    for lhs, rhs in rules:
        for i in range(len(word) - len(lhs) + 1):
            if word[i:i+len(lhs)] == lhs:
                new_word = word[:i] + rhs + word[i+len(lhs):]
                results.add(new_word)
    return results

def words(start, rules):
    seen = set([start])
    s = [start]
    while s:
        new_s = []
        for w in s:
            new_words = apply_rules(w, rules)
            for nw in new_words:
                if nw not in seen:
                    seen.add(nw)
                    new_s.append(nw)
        s = new_s
    return seen

w0 = random_word()
print("w =", w0)
w1 = apply_random_rules(w0)
print("w' = ", w1)

words1 = words(w0, T1)
words2 = words(w1, T1)
```

```
intersect = words1 & words2
if intersect:
    print("True")
else:
    print("False")
```

6 Метаморфоное тестирование

Были рассмотрены два инварианта.

1. $(|\omega|_f$ - $|\omega|_h)\text{mod}_3$ этот инвариант не меняется при применении любых правил из систем переписывания $\mathcal{T}'$ и $\mathcal{T}$.

2.

$$M(\omega) = \frac{1}{12}(|\omega|_f + |\omega|_g + |\omega|_h) - \frac{1}{12}(|\omega|_{ff} + |\omega|_{fg} + |\omega|_{fh} + |\omega|_{hf} + |\omega|_{hh} + \\ + |\omega|_{hg} + |\omega|_{gf} + |\omega|_{gg} + |\omega|_{gh})$$

Однако если $\omega = \varepsilon$ то $M(\omega) = \frac{1}{12}$. Этот инвариант не убывает при применении любых правил из систем переписывания $\mathcal{T}'$ и $\mathcal{T}$. Коэффициенты этого инварианта было подобраны программно и проверены на контекстах разной длины.

Для проверки была написана программ представленная в Листинге 2.

```
import random
from fractions import Fraction

alphabet = ["f", "g", "h"]
rules = {
    "fh": "",
    "hh": "f",
    "hf": "",
    "ff": "h",
    "g": ""
}

def generate_word(min_len, max_len):
    length = random.randint(min_len, max_len)
    return "".join(random.choice(alphabet) for _ in range(length))

def apply_rules_fixed_steps(word, rules, steps):
    for _ in range(steps):
```

```python
        applicable = [(lhs, rhs) for lhs, rhs in rules.items() if lhs in word
            ]
        if not applicable:
            break
        lhs, rhs = random.choice(applicable)
        start_idx = random.choice([i for i in range(len(word)) if word.
            startswith(lhs, i)])
        word = word[:start_idx] + rhs + word[start_idx + len(lhs):]
    return word

def diff_mod3(word):
    return (word.count("f") - word.count("h")) % 3

def invariant_M(w):
    S = ['f','g','h','ff','fg','fh','gf','gg','gh','hf','hg',
        'hh']
    alpha = [
        Fraction(1,12), Fraction(1,12), Fraction(1,12),
        Fraction(-1,12), Fraction(-1,12),Fraction(-1,12),Fraction(-1,12),
            Fraction(-1,12),Fraction(-1,12),Fraction(-1,12),Fraction(-1,12),
            Fraction(-1,12)
    ]
    def count_substring(s, w):
        count = 0
        L = len(s)
        for i in range(len(w) - L + 1):
            if w[i:i+L] == s:
                count += 1
        return count
    M = Fraction(0,1)
    for s,a in zip(S, alpha):
        M += a * count_substring(s, w)
    if w == "":
        M = Fraction(1, 12)
    return M

start = generate_word(15, 20)
start_val1 = diff_mod3(start)
start_val2 = invariant_M(start)

end = apply_rules_fixed_steps(start, rules, steps=15)
end_val1 = diff_mod3(end)
end_val2 = invariant_M(end)
a1 = start_val1 == end_val1
a2 = start_val2 == end_val2
if not (a1 and a2):
```

```
    print(start_val1 == end_val1)
    print(start_val2 <= end_val2, start_val2, end_val2)
    print(start, end)
else:
    print("Инварианты верны")
```