МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

**Факультет программной инженерии и компьютерной техники**

**Домашнее задание №2**

по дисциплине

“Разработка компиляторов”

**Вариант № 9**

**Студент:**

Румский А.

Группа P3307

**Преподаватель:**

Лаздин Артур Вячеславович

Санкт-Петербург, 2025 год

Оглавление

[Вариант 3](#_Toc200119862)

[Задание 3](#_Toc200119863)

[НКА 4](#_Toc200119864)

[ДКА 4](#_Toc200119865)

[Минимизация ДКА 5](#_Toc200119866)

[Код 7](#_Toc200119867)

# Вариант

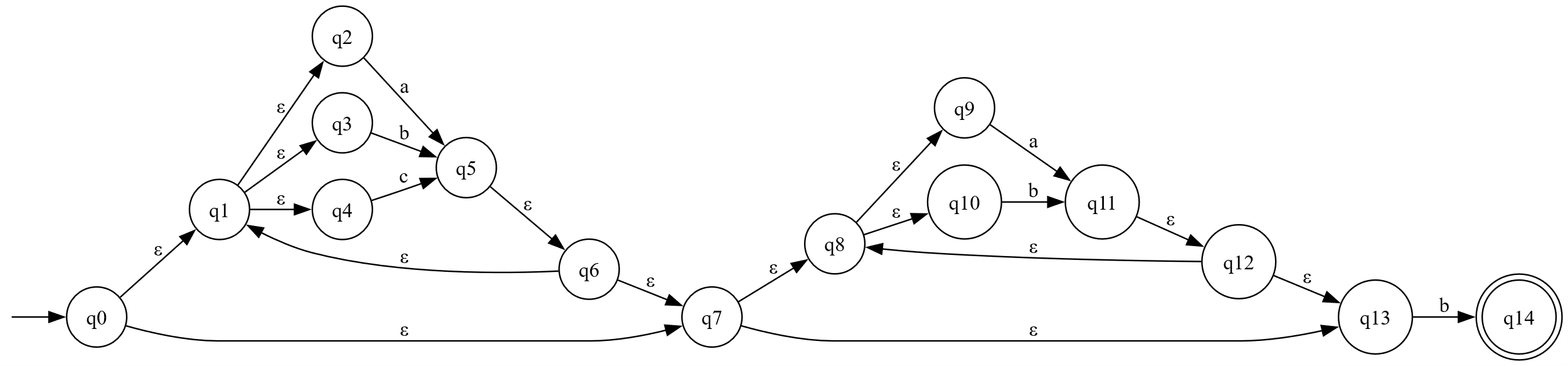
|  |  |  |
| --- | --- | --- |
| 9 | (a|b|c)\*(a|b)\*b | 23 |

Для самопроверки использовался [этот сайт](https://cyberzhg.github.io/toolbox/)

# Задание

По заданному регулярному выражению (см. вариант)  
• Построить недетерминированный КА;  
• По полученному НДА построить ДКА;  
• Минимизировать полученный ДКА;  
• Для минимального ДКА написать программу-распознаватель  
предложений языка, порождаемого регулярным выражением.  
Продемонстрировать работу распознавателя на различных примерах (не менее  
трех правильных) предложений.

# НКА



# ДКА

**Начальное состояние ДКА:**  
ε-closure(q0) = {q0, q1, q2, q3, q4, q6, q7, q8, q9, q10, q12, q13}.  
Назовем это состояние **S0**.

**Построение таблицы переходов ДКА:**

**Для S0 = {q0, q1, q2, q3, q4, q6, q7, q8, q9, q10, q12, q13}** (не конечное):

δ(S0, a): ε-closure(move(S0, a)) = ε-closure({q5, q11}) = {q1..q13}. Назовем **S1**.

δ(S0, b): ε-closure(move(S0, b)) = ε-closure({q5, q11, q14}) = {q1..q14}. Назовем **S2**. **S2** содержит q14, значит это **конечное состояние**.

δ(S0, c): ε-closure(move(S0, c)) = ε-closure({q5}) = {q1..q10, q12, q13}. Назовем **S3**.

**Для S1 = {q1..q13}** (не конечное):

δ(S1, a): ε-closure(move(S1, a)) = ε-closure({q5, q11}) = S1.

δ(S1, b): ε-closure(move(S1, b)) = ε-closure({q5, q11, q14}) = S2.

δ(S1, c): ε-closure(move(S1, c)) = ε-closure({q5}) = S3.

**Для S2 = {q1..q14}** (конечное):

δ(S2, a): ε-closure(move(S2, a)) = ε-closure({q5, q11}) = S1.

δ(S2, b): ε-closure(move(S2, b)) = ε-closure({q5, q11, q14}) = S2.

δ(S2, c): ε-closure(move(S2, c)) = ε-closure({q5}) = S3.

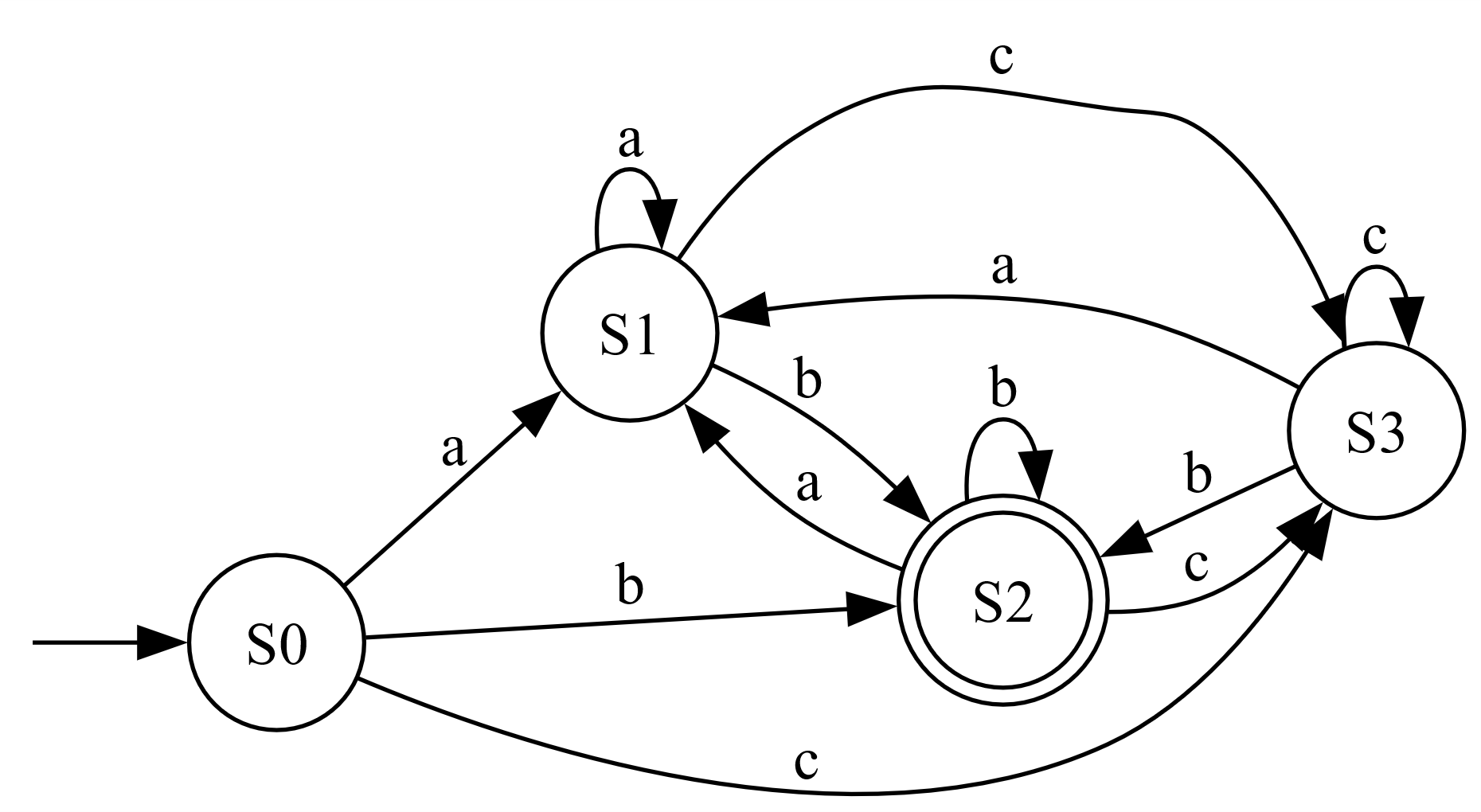
**Для S3 = {q1..q10, q12, q13}** (не конечное):

δ(S3, a): ε-closure(move(S3, a)) = ε-closure({q5, q11}) = S1.

δ(S3, b): ε-closure(move(S3, b)) = ε-closure({q5, q11, q14}) = S2.

δ(S3, c): ε-closure(move(S3, c)) = ε-closure({q5}) = S3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Состояние** | **a** | **b** | **c** |
| S0 (S) | S1 | S2 | S3 |
| S1 | S1 | S2 | S3 |
| S2 (F) | S1 | S2 | S3 |
| S3 | S1 | S2 | S3 |



# Минимизация ДКА

**Начальное разбиение:**

G1 = {S2} (конечные)

G2 = {S0, S1, S3}

**Проверка эквивалентности состояний:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Состояние из G2** | **Переход по a** | **Переход по b** | **Переход по c** |
| S0 | S1 (в G2) | S2 (в G1) | S3 (в G2) |
| S1 | S1 (в G2) | S2 (в G1) | S3 (в G2) |
| S3 | S1 (в G2) | S2 (в G1) | S3 (в G2) |

Состояния неразличимы и их можно объединить в одно состояние.

Объединяем {S0, S1, S3} => **A**.   
**A** становится начальным состоянием.

Состояние {S2} => **B**.   
**B** становится конечным.

**Переходы для A:**

δ(A, a): из S0 по a => S1. S1 теперь является частью A.   
Значит, δ(A, a) = A.

δ(A, b): из S0 по b => S2. S2 теперь является B.   
Значит, δ(A, b) = B.

δ(A, c): из S0 по c => S3. S3 теперь является частью A.   
Значит, δ(A, c) = A.

**Переходы для B:**

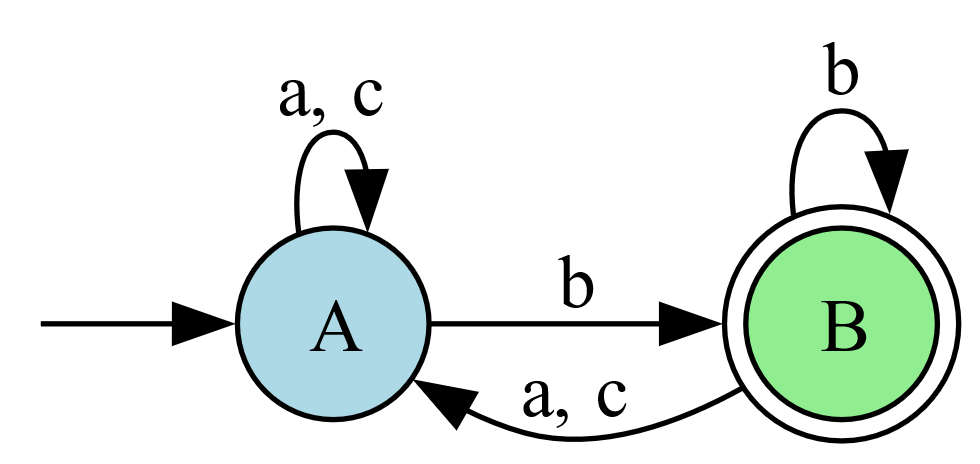
δ(B, a): из S2 по a => S1. S1 теперь является частью A.   
Значит, δ(B, a) = A.

δ(B, b): из S2 по b => S2. S2 теперь является B.   
Значит, δ(B, b) = B.

δ(B, c): из S2 по c => S3. S3 теперь является частью A.   
Значит, δ(B, c) = A.

**Итоговая таблица переходов минимального ДКА:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Состояние** | **a** | **b** | **c** |
| A (S) | A | B | A |
| B (F) | A | B | A |



# Код

def create\_recognizer\_for\_regex():

STATE\_A = "A (не заканчивается на b)"

STATE\_B = "B (заканчивается на b)"

initial\_state = STATE\_A

final\_states = {STATE\_B}

alphabet = {'a', 'b', 'c'}

transitions = {

STATE\_A: {

'a': STATE\_A,

'b': STATE\_B,

'c': STATE\_A

},

STATE\_B: {

'a': STATE\_A,

'b': STATE\_B,

'c': STATE\_A

}

}

def recognizer(input\_string: str) -> bool:

current\_state = initial\_state

print(f"\nПроверка строки: '{input\_string}'")

print(f"Начальное состояние: {current\_state}")

for char in input\_string:

if char not in alphabet:

print(f"Ошибка: символ '{char}' не принадлежит алфавиту {{a, b, c}}.")

return False

previous\_state = current\_state

current\_state = transitions[current\_state][char]

print(f"Символ: '{char}', Переход: {previous\_state} -> {current\_state}")

is\_accepted = current\_state in final\_states

print(f"Конечное состояние: {current\_state}")

print(f"Результат: Строка {'принята' if is\_accepted else 'отклонена'}.")

return is\_accepted

return recognizer

if \_\_name\_\_ == "\_\_main\_\_":

my\_recognizer = create\_recognizer\_for\_regex()

print("--- Тестирование на правильных строках ---")

my\_recognizer("b")

my\_recognizer("ab")

my\_recognizer("cacabb")

my\_recognizer("bb")

print("\n--- Тестирование на неправильных строках ---")

my\_recognizer("a")

my\_recognizer("bca")

my\_recognizer("")

my\_recognizer("d")