Contents

1 Introduction

The main objective of this assignment is to take the Pumping Lemma into practice, and use what we have learned about automata to build a valid C comment lexer.

2 Exercise 1: Pumping Lemma

2.1 Language 1

This language is not regular. We demonstrate the non regularity using contradiction of Pumping Lemma.

Suppose by contradiction that L_1 is regular. Then, by the Pumping Lemma, there exists a pumping length p where if s is **any** string in L_1 of length at least p, it should meet the definition.

Let's choose the string $s = 0^p 1^{5p+1}$. We have $5p < 5p + 1$ thus $s \in L_1$ and $|s| \geq p$.

According to the Pumping Lemma, it is guaranteed that s could be divided into three pieces, $s = xyz$, where $|y| > 0$, $|xy| \leq p$, and $\forall i \geq 0 : xy^i z \in L_1$.

Since the first p characters of s are all 0s, and that we have $|xy| \leq p$, it is for sure that y can only consist of 0s. Let's say $y = 0^k$ where $k \geq 1$ (because $|y| > 0$).

Now let's check the situation where $i = 2$. In this case, y is pumped once to get the string $xyyz = 0^{p+k} 1^{5p+1}$.

For $xyyz$ to be able to be described by L_1, we need to have $5(p + k) < 5p + 1$, which shows that k could only be $k < 0.2$. However, we have already $k \geq 1$ to meet $|y| > 0$. Moreover, $xyyz$ should have been guaranteed by the Pumping Lemma that $xyyz \in L_1$, if L_1 **is** a regular language.

This is a contradiction, and thus our initial assumption that L_1 is a regular language is **False**.

We can thus conclude by contradiction that L_1 is not a regular language.

2.2 Language 2

This language is regular.

In fact, this language could be translated completedly into two cases:

1. i is even, and j mod 3 equals to $0 + 1 = 1$

2. i is odd, and j mod 3 equals to $1 + 1 = 2$

The first case could be translated into the regular language $(00)^*1(111)^*$, while the second case could be translated into the regular language $0(00)^*11(111)^*$.

In conclusion, the regular language should be $(00)^*1(111)^* + 0(00)^*11(111)^*$.

2.3 Language 3

This language is not regular. We demonstrate the non regularity using contradiction of Pumping Lemma.

Suppose by contradiction that this language (let's call it L_3 because it's the third question) is regular. Then, by the Pumping Lemma, there exists a pumping length p where if s is **any** string in L_3 of length at least p, it should meet the definition.

Let's choose the string $s = 0^{p^2}$. We have p^2 be a perfect square of p thus $s \in L_3$ and $|s| = p^2 \geq p$.

According to the Pumping Lemma, it is guaranteed that s could be divided into three pieces, $s = xyz$, where $|y| > 0$, $|xy| \leq p$, and $\forall i \geq 0 : xy^iz \in L_1$.

Since all characters of s are 0s, it is for sure that y can only consist of 0s. Let's say $y = 0^k$ where $k \geq 1$ (because $|y| > 0$) and $k \leq p$ (because $|xy| \leq p$).

Now let's check the situation where $i = 2$. In this case, y is pumped once to get the string $xyyz = 0^{p^2+k}$.

For $xyyz$ to be able to be described by L_3, we need $p^2 + k$ to be a perfect square. However, we already have $0 < k \leq p$ and $p \geq 0$, which means we must have $k < 2p + 1$, and thus p^2+k must fall somewhere in between p^2 and $(p+1)^2$, and thus p^2+k cannot be a perfect square. Moreover, $xyyz$ should have been guaranteed by the Pumping Lemma that $xyyz \in L_3$, if L_3 **is** a regular language.

We can thus conclude by contradiction that L_3 is not a regular language.

3 Exercice 2: Lexers/ Tokenizers/ Scanners

3.1 Sub question (a)

The problem with this regular expression for a skip comment lexer is that it's greedy.

The problem is obvious when we have multiple comments in code. In this case, if the lexer uses $/*(\Sigma|\backslash n)^**/$ as a comment generator, then it will match from the start of the first comment to the end of the last comment, treating everything in between as a single huge chunk of comment. This is not what we want.

For example, imagine the case where we have:

$$/*\text{comment1}*/\text{code}/*\text{comment2}*/$$

Then the lexer will treat it as:

$$\underline{/*}\ \text{comment1}*/\text{code}/*\text{comment2}\ \underline{*/}$$

, leaving all the thing in between (including the code) as comment. This is the problem brought by greedy, so we can conclude that $/*(\Sigma|\backslash n)^**/$ would not make a lexer skip comments correctly.

3.2 Sub question (b)

We propose $/ * * * /$ to be a legal 5-character C comment that the regular expression $/ * (\neg\{*\} \mid *\neg\{/\})^* * /$ failed to match.

In this case, the lexer will take in $/*$ as the beginning of the comment, followed by two consecutive stars $**$ being recognized as the comment body part $*\neg\{/\}$ in middle (because the second $*$ is recognized as $\neg\{/\}$ followed by the first $*$), and leave out a single / that matches nothing. This comment could thus not be closed, while it is indeed a valid comment, with a single $*$ as the body of the comment.

We propose $/ * * * / * /$ to be a ill-formed 7-character C comment that the regular expression $/ * (\neg\{*\} \mid *\neg\{/\})^* * /$ matches erroneously.

In this case, the lexer will take in $/*$ as the beginning of the comment, followed by two consecutive stars $**$ being recognized as the first part of the comment body $*\neg\{/\}$ in middle (because the second $*$ is recognized as $\neg\{/\}$ followed by the first $*$), followed by a / being recognized as the second part of the comment body (because / is recognized as $\neg\{*\}$), and the rest $*/$ will be treated as the ending of the comment. More specifically, this line will be treated as:

$$\underline{/*} \quad \underline{**} \quad \underline{/} \quad \underline{*/}$$

3.3 Sub question (c)

We propose the following NFA that accepts all and only valid traditional C comments, shown in Figure 1.

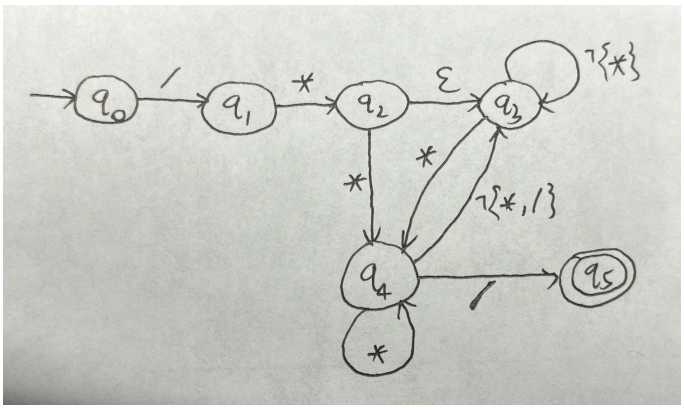

Figure 1: NFA for valid C comments

3.4 Sub question (d)

We try to find the corresponding regular expression using the GNFA transformation, using the following steps.

Step 1 We add extra start state and accept state. The automata is in Figure 2.

Then, we eliminate all the states (except for the start and the accept state) in order.

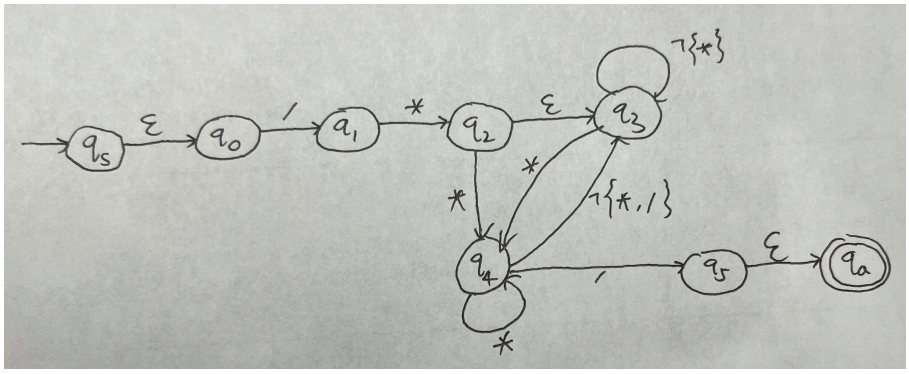

Figure 2: Automata with extra start and accept state

Step 2 We eliminate the state q_0 and q_1. We present the same automata without state q_0 and q_1 as is shown in Figure 3.

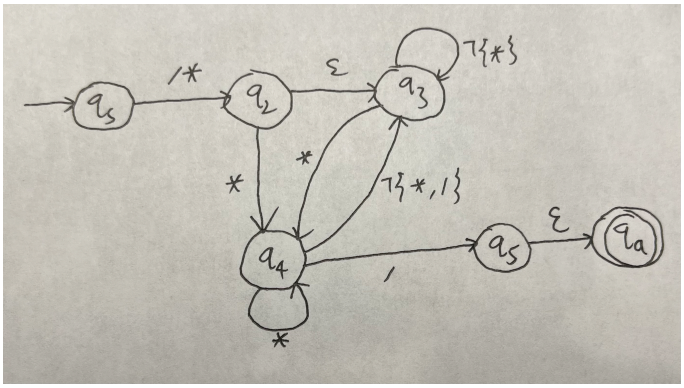

Figure 3: Automata after elimination of q_0 and q_1

Step 3 We eliminate the state q_2. To do so, we use the transition table in Figure 4. With this transition table, we present the same automata without state q_0, q_1, and q_2, as is shown in Figure 5.

Step 4 We eliminate the state q_3. To do so, we use the transition table in Figure 6. With this transition table, we present the same automata without state q_0, q_1, q_2, and q_3, as is shown in Figure 7.

Step 5 We eliminate the state q_4. We present the same automata without state q_0, q_1, q_2, q_3, and q_4, as is shown in Figure 8.

Step 6 We eliminate the state q_5. We present the same automata without state q_0, q_1, q_2, q_3, q_4, and q_5, a.k.a the final automata with only q_s and q_a, as is shown in Figure 9.

Finally, we can present the regular language as:

$$/*(\neg\{*\})^**(*\cup(\neg\{*,/\}(\neg\{*\})^**))^*/$$

4 Conclusion

In this assignment, we learned about Pumping Lemma and valid C comment lexer.

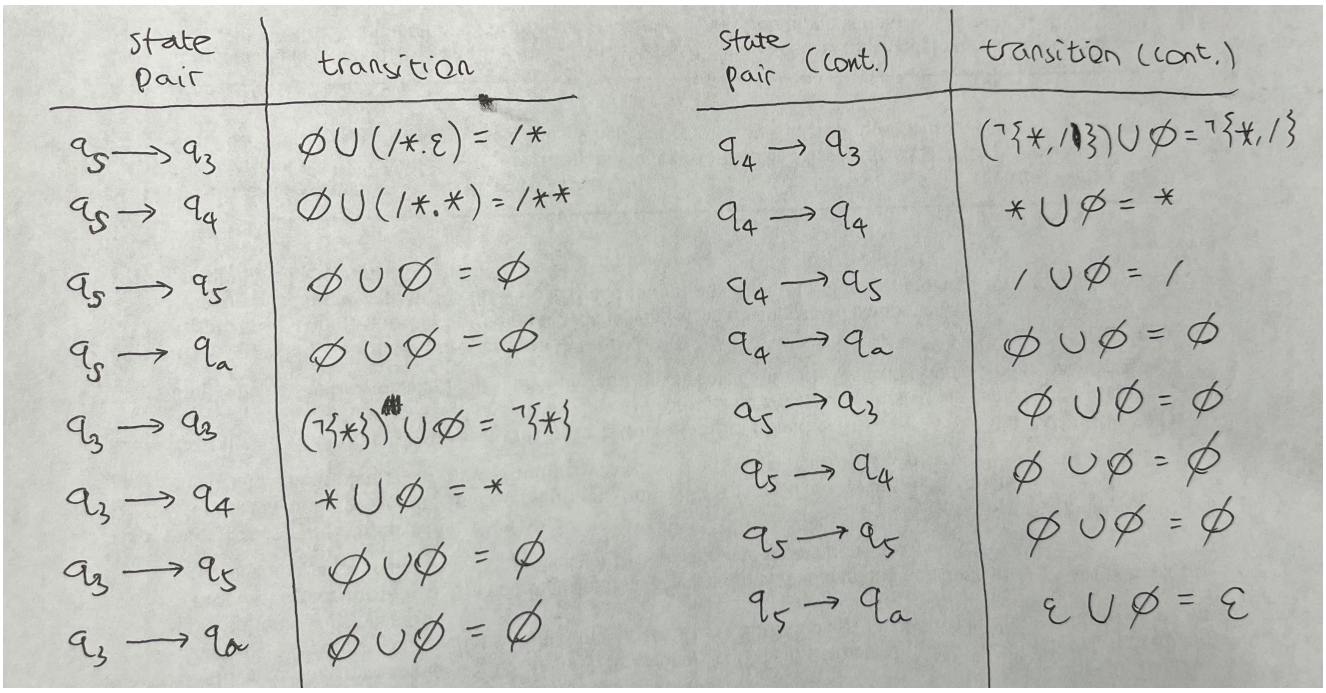

Figure 4: Transition table to eliminate q_2

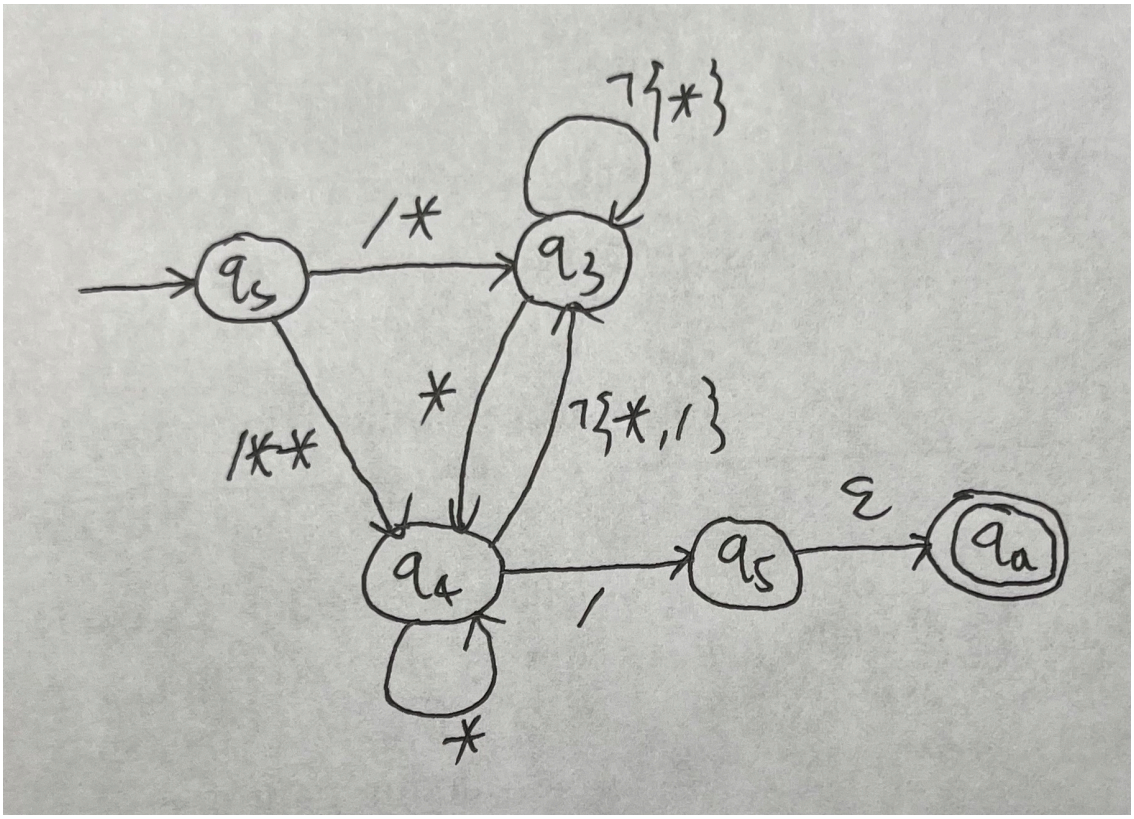

Figure 5: Automata after elimination of q_0, q_1, and q_2

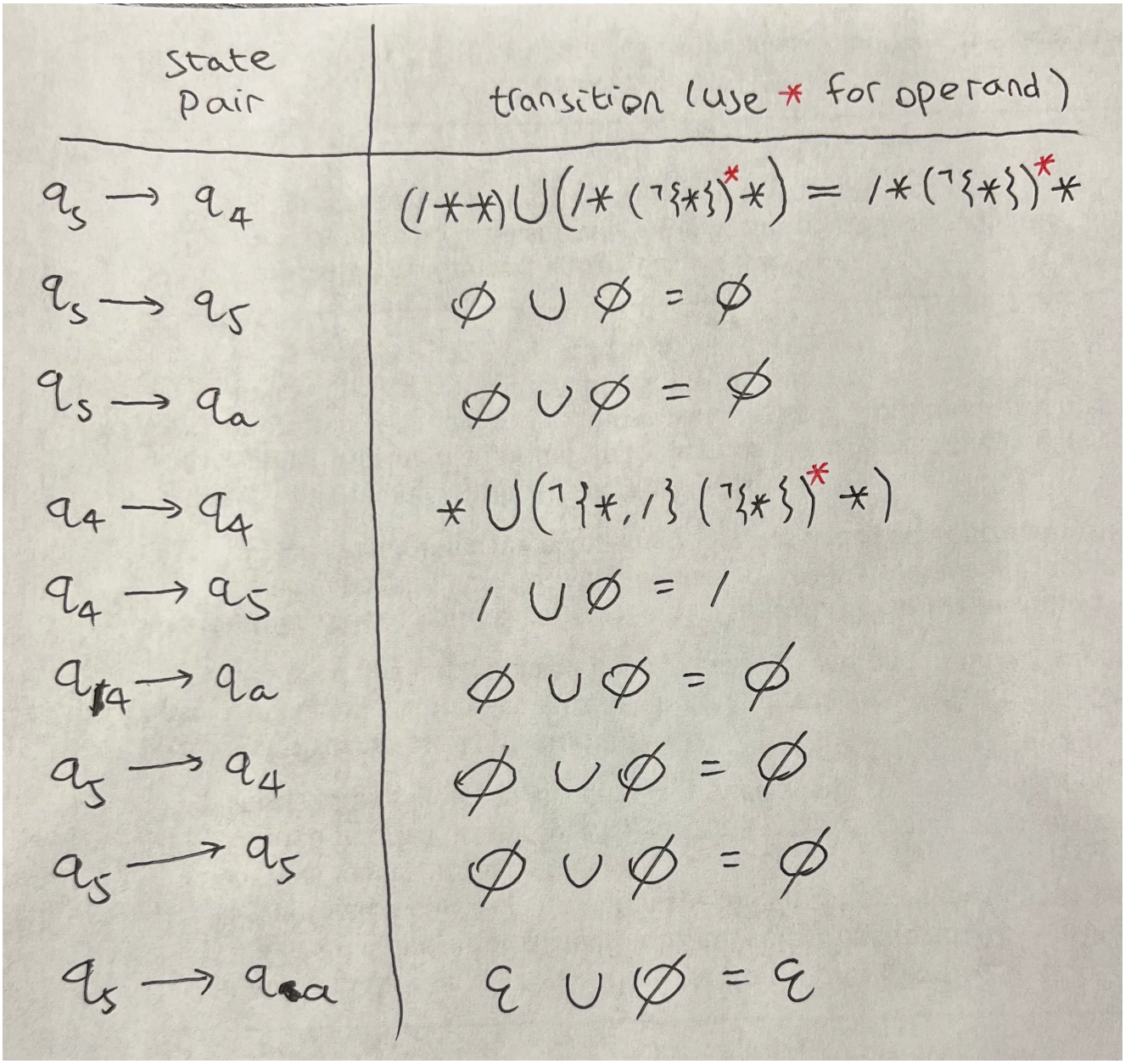

Figure 6: Transition table to eliminate q_3

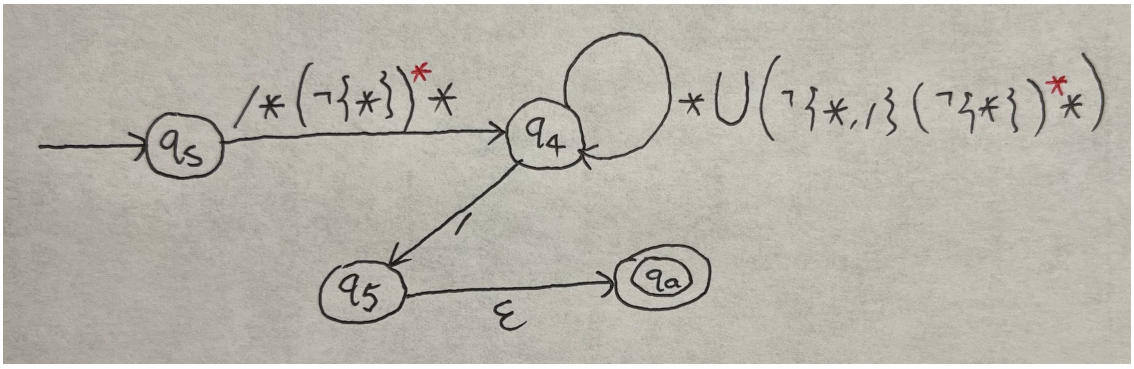

Figure 7: Automata after elimination of q_0, q_1, q_2, and q_3

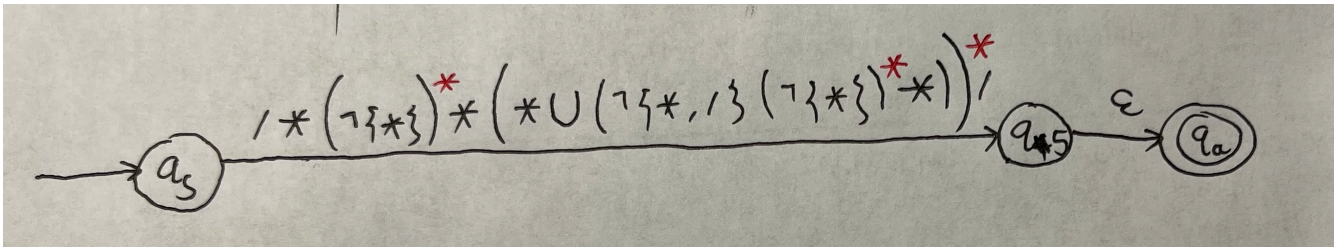

Figure 8: Automata after elimination of q_0, q_1, q_2, q_3, and q_4

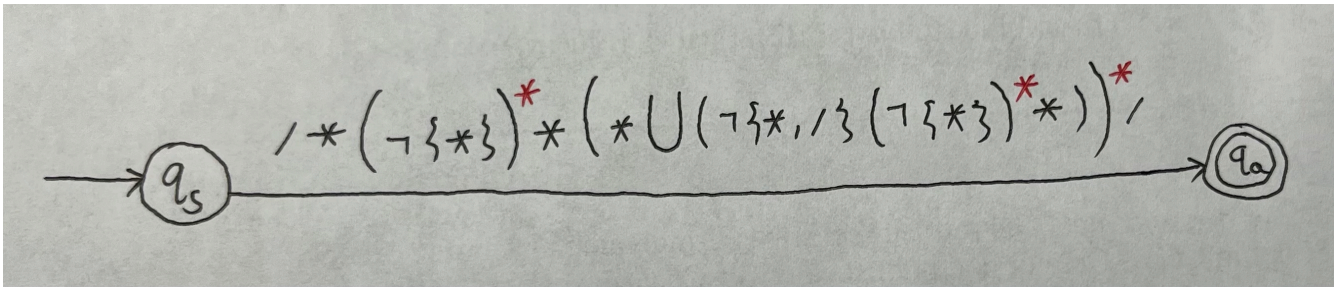

Figure 9: Final automata with only q_s and q_a