Write a CFG for the complement of \{010010001…10^i 1 \mid i \geq 1\}.

The way to approach a problem like this is to consider strings in \(L = \{010010001…10^i 1 \mid i \geq 1\}\), then figure out all the different ways you can alter a string in this set to make it no longer in this set — i.e., how can you introduce an error into such a string.

1. The string might be \(\square\) — it’s not in \(L\).

2. The string might begin with a 1 instead of a 0.

3. The string might begin with a 0, but have the wrong number of zeros — that is, more than one zero.

4. The string might end with a 0 instead of a 1.

5. There may be a 1 somewhere in the string with \(i\) zeros to its left (between it and the previous 1) and \(j\) zeros to its right (between it and the next 1), with \(i + 1 \neq j\) (i.e. it doesn’t have the property that the original sequence in \(L\) has).

6. Case (5) may occur, but the mismatch may be at the beginning of the string; thus, the first one may have \(i\) zeros to its left (between it and the beginning of the string) and \(j\) zeros to its right (between it and the next 1), with \(i + 1 \neq j\). This is a different case because for (5), we’ve assumed there is a 1 preceding the first block of zeros, which is not the case if the first block is at the very beginning.

The trick to this problem is that the moment a single error has been introduced, nothing else matters. As long as the string you
generate differs from all the strings in \( L \) in some property, it’s no longer in \( L \). You needn’t worry about generating the rest of the string to match strings in \( L \).

To explain what I mean, suppose you start with the string 01001000100001000001 and change it to be 0100100000010000100001. The change is in bold and the section with the error is italicized. As long as the italicized section is present, it doesn’t matter what precedes it or follows it, since any string containing this italicized section will never be in \( L \). So we do not need to generate a string starting with 0100 or ending with 000001. We can take the section 1000000100001 and put anything before it or after it, and it will still be in the complement of \( L \). So, instead of thinking of the entire string, think of \((0 + 1)^* \ 1000000100001 \ (0 + 1)^*\).

Let’s define a production that generates strings of the form \( 0^i10^j, i + 1 \neq j \). It turns out you have to break this into two cases, \( i \geq j \) and \( i + 1 < j \). If you try to combine these cases, then you could introduce an extra zero on one side, then cancel it out by introducing an extra zero on the other.

\[
\begin{align*}
C \rightarrow & \ 0C0 \mid C0 \mid 100 \\
D \rightarrow & \ 0D0 \mid 0D \mid 1
\end{align*}
\]

This production will generate a string in the form \( 0^*10^* \). Notice that we can add matching zeros on both sides, which will not change the relation between \( i \) and \( j \). But we can also add extra zeros just to the right-hand side, which will increase \( j \) without increasing \( i \). So if \( j \) is already greater than or equal to \( i \), applying the C0 rule will not change that. It looks like this production generates the case \( 0^i10^j, i \leq j \). But our base case is 100. This is the shortest string in \( 0^*10^* \) with \( i + 1 < j \). So actually, it generates the case \( 0^i10^j, i + 1 < j \).

This production will again generate a string in the form \( 0^*10^* \).
Notice that again we can add matching zeros on both sides, which will not change the relation between $i$ and $j$. But this time, we can add extra zeros just to the left-hand side, which will increase $i$ without increasing $j$. So if $i$ is already greater than or equal to $j$, applying the 0D rule will not change that. It looks like this production generates the case $0^i10^j$, $i \geq j$. Our base case is 1, the shortest string in $0^*10^*$ with $i \geq j$, so the equation on the above line is indeed correct.

In general, we’ll find a mismatch in the form $0^i10^j1$, with $i \geq j$ or $i + 1 < j$. So we can make a production that generates this:

$$B \parallel C_1 \mid D_1$$

It’s very important that we add a one after C and D. We’d like to concatenate an arbitrary string onto the beginning and end (since intuitively, as long as the mismatch is present, we can concatenate any arbitrary string onto the end and preserve the mismatch) of C and D. That gives us all strings over \{0, 1\} with the mismatch within them somewhere. If we didn’t add the one after C and D, then the arbitrary string could add zeros after the string generated by B. This could repair the error and make $i + 1 = j$. We need to ensure no string in $L$ is generated, so we need to make sure the error cannot be repaired. Thus, we add the 1 at the end immediately, before concatenating anything else, to ensure no zeros can be added before the next one.

We will also usually add a one right before the string generated by B, to prevent zeros from being added right at the beginning and repairing the other type of error. But if the mismatched pair is right at the beginning, there shouldn’t be any 1 before it.

$$A \parallel 0A \mid 1A \mid \square$$

This will generate arbitrary strings for us. Now, we’re thinking our strings with mismatches will be produced by:
S ⊦ BA | A1BA

(This is NOT the final answer.)

BA will handle a mismatch right at the beginning, followed by an arbitrary string. A1BA will handle an arbitrary string, then a mismatch with the protective one right before it, followed by another arbitrary string.

We’ve dealt with cases (5) and (6). Finally, we need to deal with cases (1) - (4).

(1) S ⊦ 
(2) S ⊦ 1A
(3) S ⊦ 00A
(4) S ⊦ A0

Our final CFG:

S ⊦ BA | A1BA | | 1A | 00A | A0
A ⊦ 0A | 1A | |
B ⊦ C1 | D1
C ⊦ 0C0 | C0 | 100
D ⊦ 0D0 | 0D | 1

It’s possible that we’ve missed a couple of special cases; this problem has a troublingly large assortment of them. Regardless of that, the important thing for this problem is understanding the main concept: how to get C and D; not obscure corner cases.