Example CFG:

\[ S \rightarrow \epsilon | aB | bA \]
\[ A \rightarrow SS | aS | bAAa \]
\[ B \rightarrow bs | AB \]

1. \( S \) is immediately nullable, so \( \text{NULL} = \{ \epsilon \} \).
2. Rule \( A \rightarrow SS \), \( SS \in \text{NULL}^* \), so \( \text{NULL prep} = \{ S, A \} \). After that we need only consider the rules for \( B \):
   \[ B \rightarrow bs : \text{RHS has a terminal, so never nullable.} \]
   \[ B \rightarrow ABB : \text{rule is self-recursive} \]
   \[ B \rightarrow AB \] is self-recursive too \( ABB \notin \{ S, A \} \), so \( \text{NULL} = \{ S, A \} \).

I: Add rules deleting some occurrence of nullable vars.

\[ S \rightarrow \varepsilon | aB | bA | b \]
\[ A \rightarrow SS | S | aS | a | bAAa | bAa \]
\[ B \rightarrow bs | b | ABB | B \]

II: Delete \( \epsilon \)-rules. New grammar is \( G_1 \).

\[ L(G_1) = L(G) \setminus \{ \epsilon \} \]
Testing some targets - are the variables sound? exact?

\( T_5 = \exists x : \forall y (x \neq y) \land a(x) = b(y) \) ? "equal a, b?"

\( T_A = \exists x : a(x) = b(x) + 1 ? "one more a" \\
T_B = \exists x : b(x) = a(x) + 1 ? "one more b" \\

Is \( S \) sound for \( T_5 \)? We need to suppose A & B are sound for their targets ...

**Rule**

\( S \rightarrow \epsilon \) : We have \( \epsilon \in T_5. \) Since \( a(\epsilon) = b(\epsilon) = 0 \), so this rule is sound (unconditionally).

\( S \rightarrow a \) : Assuming \( B \) is sound, it generates \( 1 \) more b than a. So the total LHS has equal \( a \)s and \( b \)s. Thus this rule is conditionally sound for \( S \).

\( S \rightarrow b \) : Similar, by symmetry, if \( A \) is sound for \( T_A \).

\( A \rightarrow SS \) : alas this busts \( T_A \), and \( T_B \) are voided too. Let's just delete that rule. New grammar \( G' \)

\( S \rightarrow \epsilon | a \) \( BA \) \( B \) \( A \)

\( G' \) : \( A \rightarrow aS | bAa \) \( B \rightarrow bS | ABB \)

Note that in \( G' \), \( A \) is no longer nullable, so \( G' \) is

\( S \rightarrow a B 1 b A \) \( B \rightarrow bS | b1A BB \)

\( G' = A \rightarrow aS | \underline{a} bAa \) \( B \rightarrow bS | b1A BB \)

(When asking about "exact targets" let's use the letter \( E = \exists, \forall, A, B \etc \))
\[ E_S = \exists x : \#a(x) = \#b(x) \]  
\[ E_A = \exists x : \#a(x) = \#b(x) + 1 \]  
\[ E_B = \exists x : \#b(a(x)) = \#b(x) + 1 \]

To prove they are not show that some variable (so \( x \)) actually compiles with a stronger target \( T'A \) such that \( E_A \neq T'A \).

\[ T'_A = \exists x : \#a(x) = \#b(x) + 1 \text{ and: } b \text{ then it's wrong} \]

The string \( x = bbaaab \) belongs to \( E_A \setminus T'_A \).
Thus \( L_A \leq T'_A \setminus \{ e \} \neq E_A \), so \( L_A \neq E_A \) (why, \( L_A \subseteq E_A \)).

Does this knock out \( S \)? i.e. is \( L_S = \{ \} \neq E_S = E_S \setminus \{ e \} \)?
Try \( w = b^4 x = bbaaab \) as a possible counterexample:

\[ S \rightarrow aBbA \]
\[ A \rightarrow aS \rightarrow aA \]
\[ B \rightarrow bSb \rightarrow Bbb \]

\[ bbaab \]

We liberalized the rule \( A \rightarrow bAa \to A \rightarrow BAA \).

Does this fix our immediate problem for \( x = bbaaab \)?

Yes: Parse tree

This gives a BM derivation of \( S \):
\[ S \rightarrow bA \rightarrow bBAA \rightarrow bbAA \rightarrow bbAA \rightarrow bbaA \rightarrow bbaA \rightarrow bbaS \rightarrow bbaaaA \rightarrow bbaaab = w \].

This \( bbaaab \).
Thus we fixed the problem cases \( x = baab, w = bx \).
Did we fix the entire grammar to make \( L_S = E_S \)?
(And \( L_A = E_A \) and \( L_B = E_B \)? Proved via parsing.
Plus induction/recursion.

Consider any \( x \in E_A \). We need to show that \( A \Rightarrow^* X \)
Cases:
1. \( X \) begins with \( a \). Then either \( X = a, \) where \( A \Rightarrow a, \)
or \( X = ay \) where \( y \notin E \) and \( \#a(1) = \#b(1), \) ie \( y \in E_S \).
By induction on \( L_L \), \( S \Rightarrow^* y \), so \( A \Rightarrow AS \Rightarrow^* aY = X \).
2. \( X \) begins with \( b \). Then \( X = bz \) where \( z \) has \( \geq \)
more \( a \)'s than \( b \)'s. Hence \( z \) can be broken as \( z = u \)
while \( u \) and \( v \) each have \( 1 \) more \( a \) than \( b \).
By self-recursion, we get \( A \Rightarrow^* u \) and \( A \Rightarrow^* v \).
Finally we get \( A \Rightarrow BAA \Rightarrow bAA \Rightarrow^* buA \Rightarrow^* buv = X \)

That's the idea. The full proof needs handling \( B \) too
and \( S \) and base cases, for our triple the work.

(Shipped this year)

But the \( Ta \), \( Tb \), is kind of proof will be used
of sand.