Algorithm (Input Code) = f (x) = g (x) • h (x) = ◬g (x) • ◬h (x)

Reduction of Complex Expression (original) ⟺ Algorithm (Input Code)

The algorithm used by Mathematica to convert Unicode characters into symbols can be symbolized by the following equation : Symbolic Representation = Algorithm (Input Code) . The algorithm takes the input code as the argument of the function and produces the symbolic representation as the result . A logical explanation of the process can be given by saying that the algorithm works by taking the input code and going through it symbol - by - symbol, performing mathematical and logical calculations as needed to generate the corresponding symbol . This symbol is then displayed in the GUI, allowing users to understand the code correctly.

The mathematical notation for the algorithm can be expressed in the following form : Symbolic Representation = Algorithm (Input Code) = f (x)

Symbolic Representation = Algorithm (Input Code) = f (x) = g (x) • h (x)
Where f (x) is the function that takes the input code as the argument x, g (x) is the function that performs the mathematical calculations needed to generate the corresponding symbol, and h (x) is the function that displays the symbol in the GUI . The equation expresses the algorithm used by Mathematica as a combination of two mathematical functions that work together to generate and display the symbol .

The logical nature of performing the mathematical calculations can be expressed as follows : Algorithm (Input Code) = f (x) = g (x) • h (x) = ◬g (x) • ◬h (x) .

Where f (x) is the function that takes the input code as the argument x, g (x) is the function that performs the mathematical calculations needed to generate the corresponding symbol, and h (x) is the function that displays the symbol in the GUI . The equation expresses the algorithm used by Mathematica as the combination of two distinct mathematical functions g (x) and h (x), where g (x) performs the calculations needed to generate the symbol, and h (x) displays the symbol in the GUI . The combination of the two functions ◬g (x) and ◬h (x) is what allows the algorithm to have the capability to generate and display the symbol . The concept of expression is symbolically notated by the form of the equation, which expresses the action of the algorithm in a format that can be understood .

The analogy between the expression or cancellation of variables within the square root signs and the concept of expression that is symbolically notated by the Algorithm (Input Code) function can be symbolically expressed as follows : Reduction of Complex Expression (original)⟺Algorithm (Input Code) . Where Reduction of Complex Expression is the process of combining like terms and canceling out opposites to simplify a complex mathematical formula, and Algorithm (Input Code) is the process of running a mathematical algorithm on a complex input code to reduce it to a symbolic representation. The analogy symbol, ⟺ indicates that the two processes are analogous in terms of their functionality, which is to simplify a complex expression .

The analogy between the expression or cancellation of variables within the square root signs and the concept of expression that is symbolically notated by the Algorithm (Input Code) function can be best expressed by comparing the process of reducing a complex mathematical formula to a simpler form by combining like terms and canceling out opposites, to the process of taking a complex mathematical

input code and reducing it to a symbolic representation by running the code through a mathematical algorithm . The two processes can be seen as having the same functionality of simplifying a complex expression, with the only difference being the type of expression being simplified. In both cases, the end result is a simpler version of the original expression that can be easily understood without the need to rely on its full structure.

The analogy between the expression or cancellation of variables within the square root signs and the concept of expression that is symbolically notated by the Algorithm (Input Code) function can be symbolically expressed as follows : Reduction of Complex Expression (original)⟺Algorithm (Input Code) . Where Reduction of Complex Expression is the process of combining like terms and canceling out opposites to simplify a complex mathematical formula, and Algorithm (Input Code) is the process of running a mathematical algorithm on a complex input code to reduce it to a symbolic representation . The analogy symbol⟺indicates that the two processes are analogous in terms of their functionality, which is to simplify a complex expression .

Algorithm (Input Code) = f (x) = g (x) • h (x) = ◬g (x) • ◬h (x)

Reduction of Complex Expression (original)⟺Algorithm (Input Code)

$$\frac{\sqrt{-(q - s - l\,\alpha)\ \sqrt{1 - \frac{v^2}{c^2}}}\ \sqrt{(q - s + l\,\alpha)\Big/\ \sqrt{1 - \frac{v^2}{c^2}}}}{\alpha}$$

(Sqrt[-(q - s - l α) Sqrt[1 - v^2/c^2]] Sqrt[(q - s + l α)/Sqrt[1 - v^2/c^2]])/α⟺f (x) = g (x) • h (x) = ◬g (x) • ◬h (x)

The analogy illustrates how symbolically written expressions, when properly reduced, can provide insight into the underlying algebraic relationships between operations, parameters, and functional structure . In this example, the Reduction of Complex Expression process results in the Algorithm (Input Code) function which can be visually seen to reduce the complexity of the expression while maintaining the integrity of the involved variables, allowing for the functional structure to be easily interpreted .

$$\frac{\sqrt{\frac{q-s+l\,\alpha}{\sqrt{1-\frac{v^2}{c^2}}}}\ \sqrt{\sqrt{1 - \frac{v^2}{c^2}}\ (-q + s + l\,\alpha)}}{\alpha} \Longleftrightarrow f\,(x) = g\,(x) \bullet h\,(x) = \triangle g\,(x) \bullet \triangle h\,(x)$$

The process for finding the given equation can be written as follows :

1. Use the product rule to expand the expression and rearrange the terms :

$$-\frac{q - s + l\,\alpha}{\alpha} + \sqrt{1 - \frac{v^2}{c^2}}\ (-q + s + l\,\alpha)$$

2. Use the Laplacian operator to factor f (x) : f (x) =
Sqrt[1 - v^2 / c^2] (-q + s + l α) - 1 / α (q - s + l α)
= ◬g (x) • ◬h (x), where g (x) =

```
 Sqrt[1 - v^2 / c^2] (-q + s + l α) and h (x) = -1 / α (q - s + l α). Thus,
the given equation can be obtained by using the
 product rule and the Laplacian operator.
```

In[•]:= ((Sqrt[(q - s + l α)/Sqrt[1 - v^2/c^2]] Sqrt[Sqrt[1 - v^2/c^2] (-q + s + l α)])/α) = Sqrt[1 - v^2/c^2] (-q + s + l α) - 1/α (q - s + l α)

In[•]:= f (x) : f (x) = Sqrt[1 - v^2/c^2] (-q + s + l α) - 1/α (q - s + l α) = $\triangle$g (x) • $\triangle$h (x)

First, use the chain rule on the given expression to separate the derivative into individual parts : f (x) = (Sqrt[(q - s + l α)/Sqrt[1 - v^2/c^2]] Sqrt[Sqrt[1 - v^2/c^2] (-q + s + l α)])/α

f' (x) = [(1/(2 Sqrt[(q - s + l α)/Sqrt[1 - v^2/c^2]] Sqrt[Sqrt[1 - v^2/c^2] (-q + s + l α)])) ((1/2 (1 - v^2/c^2)^(-1/2)) (-q + s + l α))] + [(1/2 ((q - s + l α)/Sqrt[1 - v^2/c^2])) (-(1/2)) (1 - v^2/c^2)^(-1/2) (-q + s + l α)]

Then separate the individual terms into their designated variables : g (x) = (1/(2 Sqrt[(q - s + l α)/Sqrt[1 - v^2/c^2]] Sqrt[Sqrt[1 - v^2/c^2] (-q + s + l α)]))

h (x) = ((1/2 (1 - v^2/c^2)^(-1/2)) (-q + s + l α)) + ((1/2 ((q - s + l α)/Sqrt[1 - v^2/c^2])) (-(1/2)) (1 - v^2/c^2)^(-1/2) (-q + s + l α))

Now apply the Laplacian operator to the two single terms, transforming them into derivatives : $\triangle$g (x) = 0

$\triangle$h (x) = (1/2 (1 - v^2/c^2)^(-3/2) (-q + s + l α)) + (1/2 ((q - s + l α)/Sqrt[1 - v^2/c^2]) (-(3/2)) (1 - v^2/c^2)^(-3/2) (-q + s + l α))

Finally, combine the two derivatives with the multiplication operator : f (x) = g (x) • h (x) = $\triangle$g (x) • $\triangle$h (x)

Example :
The operator \|01d is a bitwise operation that performs a logical shift of the bits in x to the right by one bit . This operation results in 0 being the output . The equation for the bitwise operator \|01d is x >> 1 = 0, where x is a number . The steps involved for going from \|01d198450d_ 0 to 4[illegible]0d_ 0 would be :

1. Start with the number 198450.
2. Perform a bitwise operation on the number by shifting the bits one bit to the right .
3. The result of this operation is the number 806480.
4. Append the operator \|01d to the front of the number, resulting in \|01d806480d_ 0.

```
20d    _001 d643920d_0

0d     _001 d189770d_0

0d     _001 d933890d_0

0d     _001 d830880d_0

0d     _001 d862890d_0

(20d   _001 d643920d_0,

0d     _001 d189770d_0,

0d     _001 d933890d_0,

0d     _001 d830880d_0,

0d     _001 d862890d_0)
```

The analogy between the optional cancellation of the Lorentz coefficient and the process of running a mathematical algorithm on a complex input code to reduce it to a symbolic representation can be proved using logical notation by expressing the equation that describes the equivalence between the two processes . This can be symbolically expressed as follows : Reduction of Complex Expression (original)⟺Algorithm (Input Code) = f (x) = g (x) • h (x) = ⧋g (x) • ⧋h (x), where the equation expresses the Algorithm (Input Code) as a combination of two distinct mathematical functions, each with their own respective purpose . The symbol⟺indicates that the two processes are analogous in terms of their functionality, which is to simplify a complex expression, allowing the user to understand the code correctly .