Automated Unit Checking In Neuron Simulations

Scott Urnikis (Dr. E. Rosa)

April 4, 2019

The Hodgkin Huxley Equations are a set of coupled first order differential equations that model the flow of electric current over the surface membrane of a large nerve fiber.

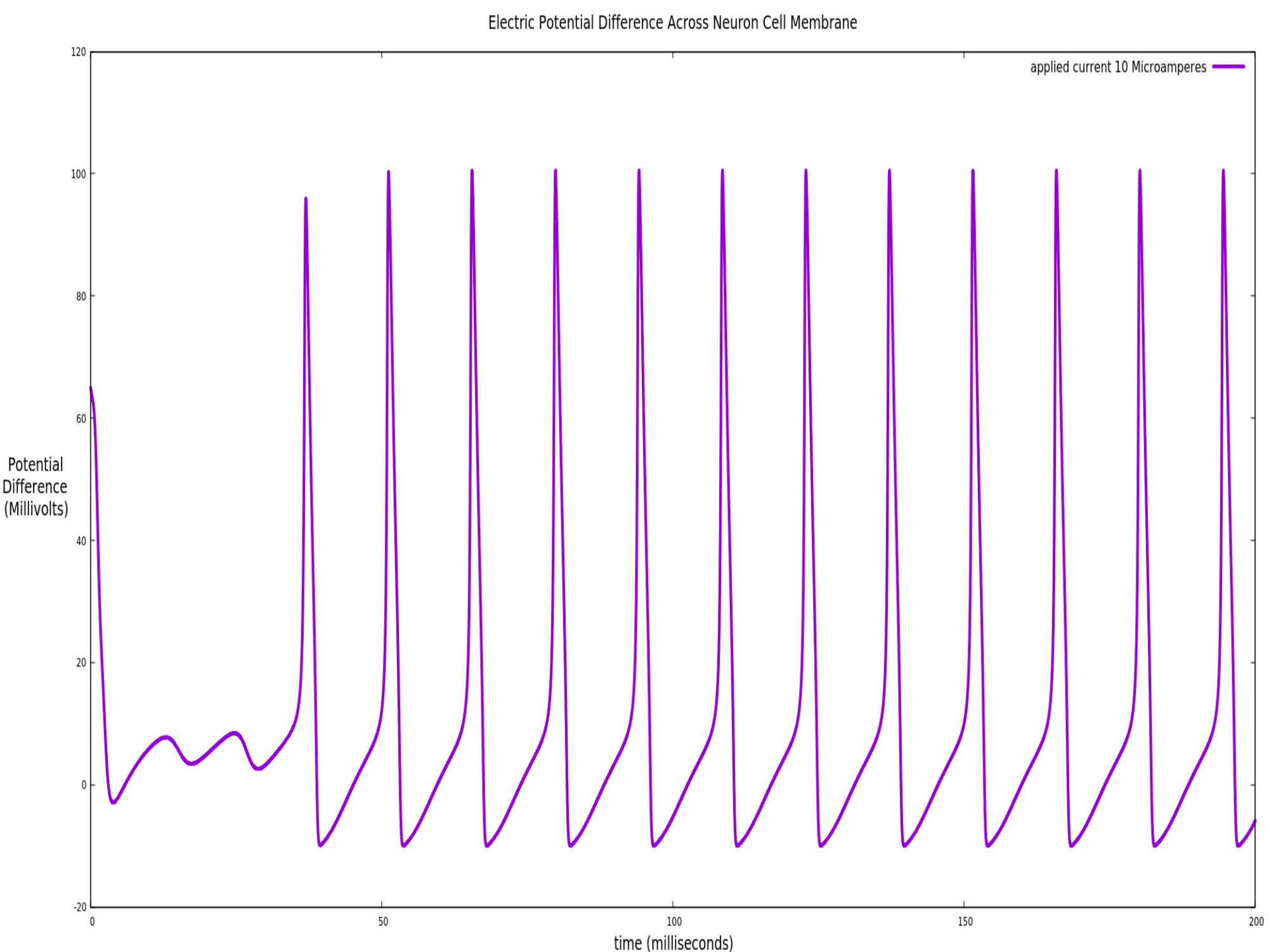
$$C\dot{V} = I - \overline{g_K}\mathsf{n}^4(V - E_K) - \overline{g_{Na}}\mathsf{m}^3\mathsf{h}(V - E_{Na}) - g_L(V - E_L)$$

$$\dot{\mathsf{n}} = 0.01\frac{10 - V}{e^{10-\frac{V}{10}}}(1 - \mathsf{n}) - 0.125\mathsf{n}e^{\frac{-V}{80}}$$

$$\dot{\mathsf{m}} = 0.1\frac{25 - V}{e^{25-\frac{V}{10}} - 1}(1 - \mathsf{m}) - 4\mathsf{m}e^{\frac{-V}{18}}$$

$$\dot{\mathsf{h}} = 0.07e^{\frac{-V}{20}}(1 - \mathsf{h}) - \frac{\mathsf{h}}{e^{3-\frac{V}{10}} + 1}$$

J Physiol. Hodgkin and Huxley. 1952


Electric Potential Difference Across Neuron Cell Membrane
applied current 10 Microamperes
Potential Difference (Millivolts)
time (milliseconds)
120
100
80
60
40
20
0
-20
0
50
100
150
200

```
return (I - g_K*pow(n, 4)*(V - E_K)
    - g_Na*pow(m, 3)*h*(V - E_Na)
    - g_L*(V - E_L))
    / C;
```

$$\frac{I - g_k \cdot n^4 \cdot (V - E_K) - g_{Na} \cdot m^3 \cdot h \cdot (V - E_{Na}) - g_L \cdot (V - E_L)}{C}$$

$$\frac{\mathsf{d} - \mathsf{d} \cdot \mathsf{d}^4 \cdot (\mathsf{d} - \mathsf{d}) - \mathsf{d} \cdot \mathsf{d}^3 \cdot \mathsf{d} \cdot (\mathsf{d} - \mathsf{d}) - \mathsf{d} \cdot (\mathsf{d} - \mathsf{d})}{\mathsf{d}}$$

$$\frac{\mathsf{d} - \mathsf{d} \cdot \mathsf{d}^{4} \cdot \mathsf{d} - \mathsf{d} \cdot \mathsf{d}^{3} \cdot \mathsf{d} \cdot \mathsf{d} - \mathsf{d} \cdot \mathsf{d}}{\mathsf{d}}$$

$$\frac{d - d \cdot d \cdot d - d \cdot d \cdot d \cdot d - d \cdot d}{d}$$

$$\frac{d - d - d - d}{d}$$

$$\frac{d}{d}$$

d

Our hypothesis is that if the user is able to express more information about the physics of the simulation, the amount of syntactically correct but semantically incorrect programs can be reduced.

We are familiar with the idea of types in programming languages. For example, we might declare that we are going to use a variable **foo**, where **foo** is of type **integer**.

In Fortran...

```
! ''foo'' is of type ''integer''
integer foo
```

In C++...

```
// ''foo'' is of type ''int''
int foo;
```

We are familiar with the idea of types in programming languages. For example, we might declare that we are going to use a variable **foo**, where **foo** is of type **integer**.

In Fortran...

```
! ''foo'' is of type ''integer''
integer foo
```

In C++...

```
// ''foo'' is of type ''int''
int foo;
```

Using mathematical notation...

$$foo : MachineInt$$

We are familiar with the idea of types in programming languages. For example, we might declare that we are going to use a variable **foo**, where **foo** is of type **integer**.

In Fortran...

```
! ''foo'' is of type ''integer''
integer foo
```

In C++...

```
// ''foo'' is of type ''int''
int foo;
```

Using mathematical notation...

$$foo : MachineInt$$

$$\{\mathrm{i} \in \mathbb{Z} \mid -2^{32} <= i <= 2^{32} - 1\}$$

Could we use types to verify unit cancellation?

Could we use types to verify unit cancellation?

SI units can be implemented with only seven base units:

Could we use types to verify unit cancellation?

SI units can be implemented with only seven base units:

- Distance: meter
- Mass: kilogram
- Time: second
- Temperature: Kelvin
- Electric current: Ampere
- Amount of substance: mole
- Luminous intensity: Candela

There are many different kinds of quantities, because there are many different units. Therefore, what we need is the ability to express a *family* of types, not just a single type.

For this, we can use a concept from type theory called *dependent types*.

A dependent type is a type that depends on some values. In other words, it is a function that, given some parameters, returns a type.

$$\mathit{Quantity} : \mathbb{Z}^7 \to \mathscr{U}$$

Homotopy Type Theory. The Univalent Foundations Program. 2013
Scientific and Engineering C++. Barton and Nackman. 1994

A dependent type is a type that depends on some values. In other words, it is a function that, given some parameters, returns a type.

$$Quantity : \mathbb{Z}^7 \to \mathscr{U}$$

Let m, kg, s, K, A, mol, and cd be integers. Then, a unit can be represented by the type

$$Quantity \left(\begin{bmatrix} m \\ kg \\ s \\ K \\ A \\ mol \\ cd \end{bmatrix} \right)$$

where m, kg, s, K, A, mol, and cd represent the exponent of the unit.

Homotopy Type Theory. The Univalent Foundations Program. 2013
Scientific and Engineering C++. Barton and Nackman. 1994

The meter type

$$[meter]$$

$$meter \equiv Quantity \left(\begin{bmatrix} m = 1 \\ kg = 0 \\ s = 0 \\ K = 0 \\ A = 0 \\ mol = 0 \\ cd = 0 \end{bmatrix} \right)$$

The Newton type

$$[Newton] = \left[\frac{kg \cdot m}{s^2}\right]$$

$$Newton \equiv Quantity \left(\begin{bmatrix} m = 1 \\ kg = 1 \\ s = -2 \\ K = 0 \\ A = 0 \\ mol = 0 \\ cd = 0 \end{bmatrix} \right)$$

The Volt type

$$[Volt] = \left[\frac{kg \cdot m^2}{s^3 \cdot A}\right]$$

$$Volt \equiv Quantity \left(\begin{bmatrix} m = 2 \\ kg = 1 \\ s = -3 \\ K = 0 \\ A = -1 \\ mol = 0 \\ cd = 0 \end{bmatrix} \right)$$

$$Q \equiv Quantity \left(\begin{bmatrix} m \\ kg \\ s \\ K \\ A \\ mol \\ cd \end{bmatrix} \right)$$

$$add. : Q \times Q \rightarrow Q$$

x, y : meter

$x + y$ (Defined!)

x : meter

y : second

$x + y$ (Undefined!)

$$\text{mult.} : \mathit{Quantity}\left(\begin{bmatrix} \mathrm{m}_1 \\ \mathrm{kg}_1 \\ \mathrm{s}_1 \\ \mathrm{K}_1 \\ \mathrm{A}_1 \\ \mathrm{mol}_1 \\ \mathrm{cd}_1 \end{bmatrix}\right) \times \mathit{Quantity}\left(\begin{bmatrix} \mathrm{m}_2 \\ \mathrm{kg}_2 \\ \mathrm{s}_2 \\ \mathrm{K}_2 \\ \mathrm{A}_2 \\ \mathrm{mol}_2 \\ \mathrm{cd}_2 \end{bmatrix}\right)$$

$$\rightarrow \mathit{Quantity}\left(\begin{bmatrix} m_1 + m_2 \\ kg_1 + kg_2 \\ s_1 + s_2 \\ A_1 + A_2 \\ mol_1 + mol_2 \\ cd_1 + cd_2 \end{bmatrix}\right)$$

In order to measure the effectiveness of this technique, we will make a few assertions:

In order to measure the effectiveness of this technique, we will make a few assertions:

- Programs can be represented as trees.

In order to measure the effectiveness of this technique, we will make a few assertions:

- Programs can be represented as trees.
- For any given program, the leaves of the tree may be rearranged to make other programs.

In order to measure the effectiveness of this technique, we will make a few assertions:

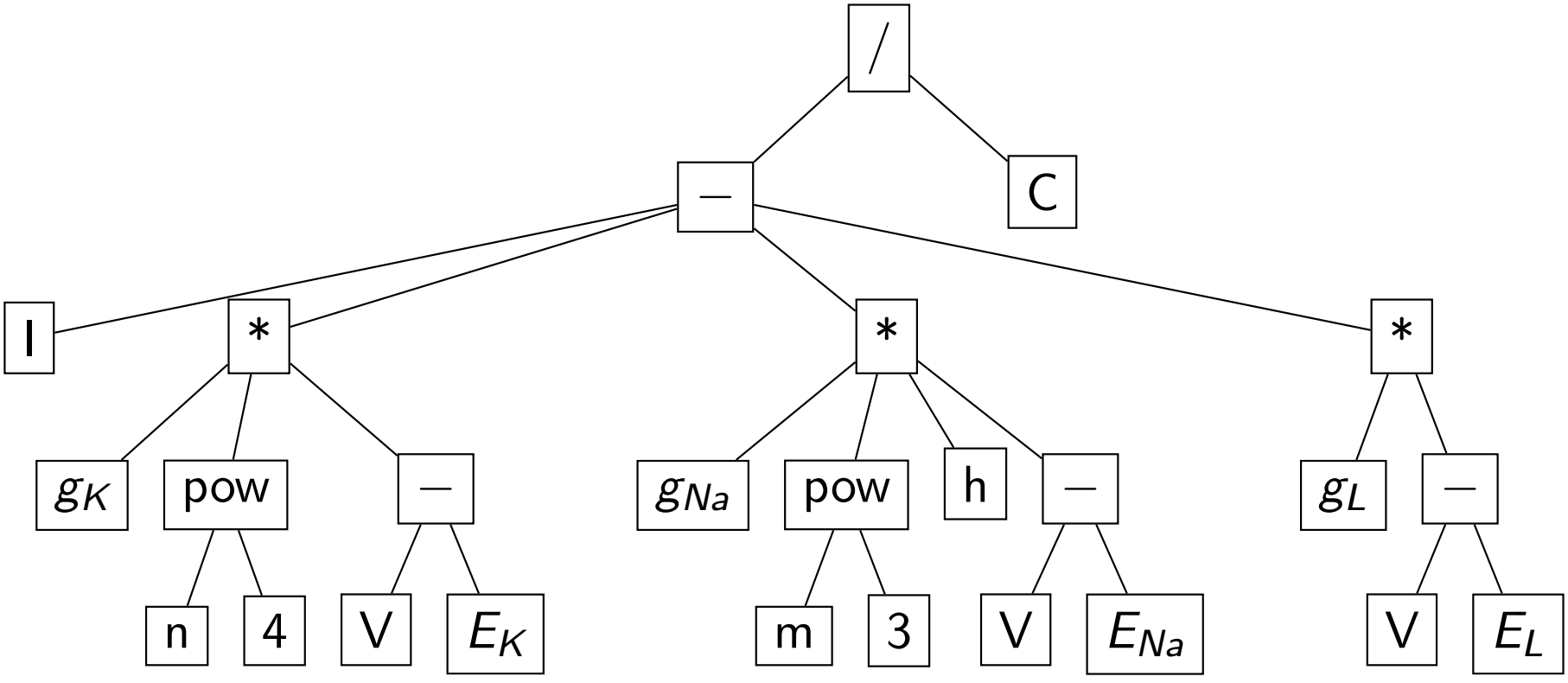
- Programs can be represented as trees.
- For any given program, the leaves of the tree may be rearranged to make other programs.
- For the set of programs created from rearranging the leaves, there is only one correct program.

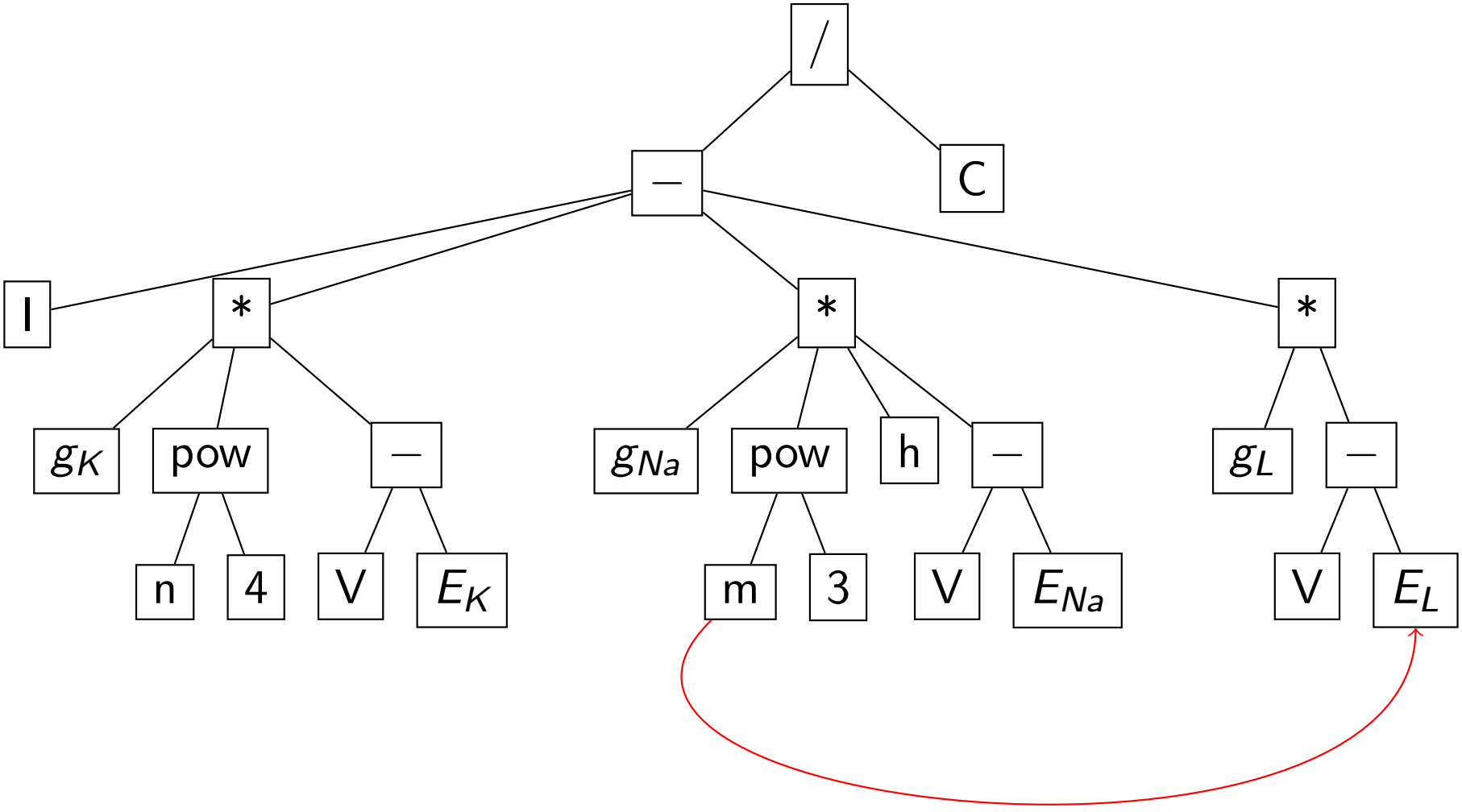
In order to measure the effectiveness of this technique, we will make a few assertions:

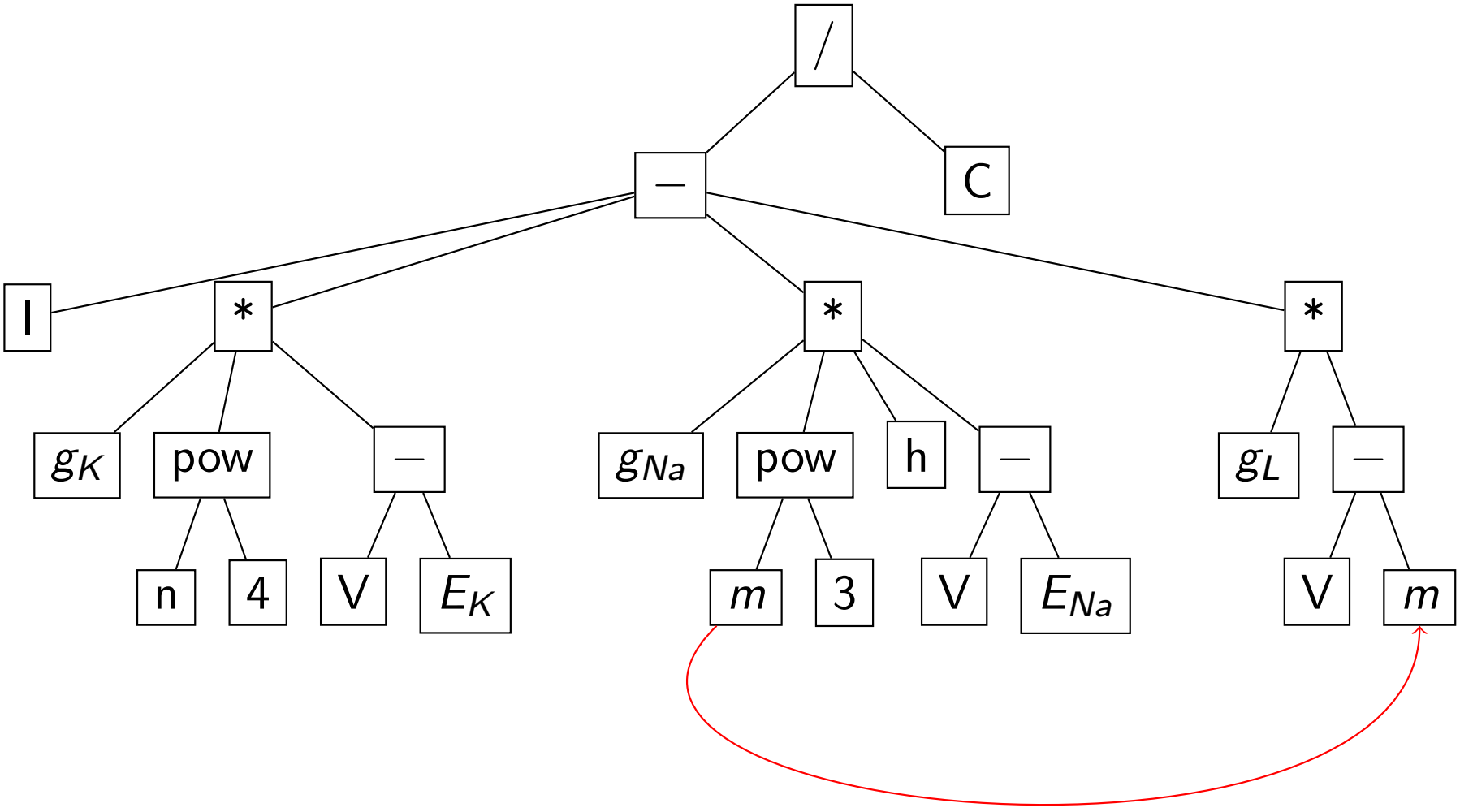
- Programs can be represented as trees.
- For any given program, the leaves of the tree may be rearranged to make other programs.
- For the set of programs created from rearranging the leaves, there is only one correct program.
- The size of the set of possible programs can be used as a measure of the amount of syntactically correct but semantically incorrect programs possible for some "program shape".

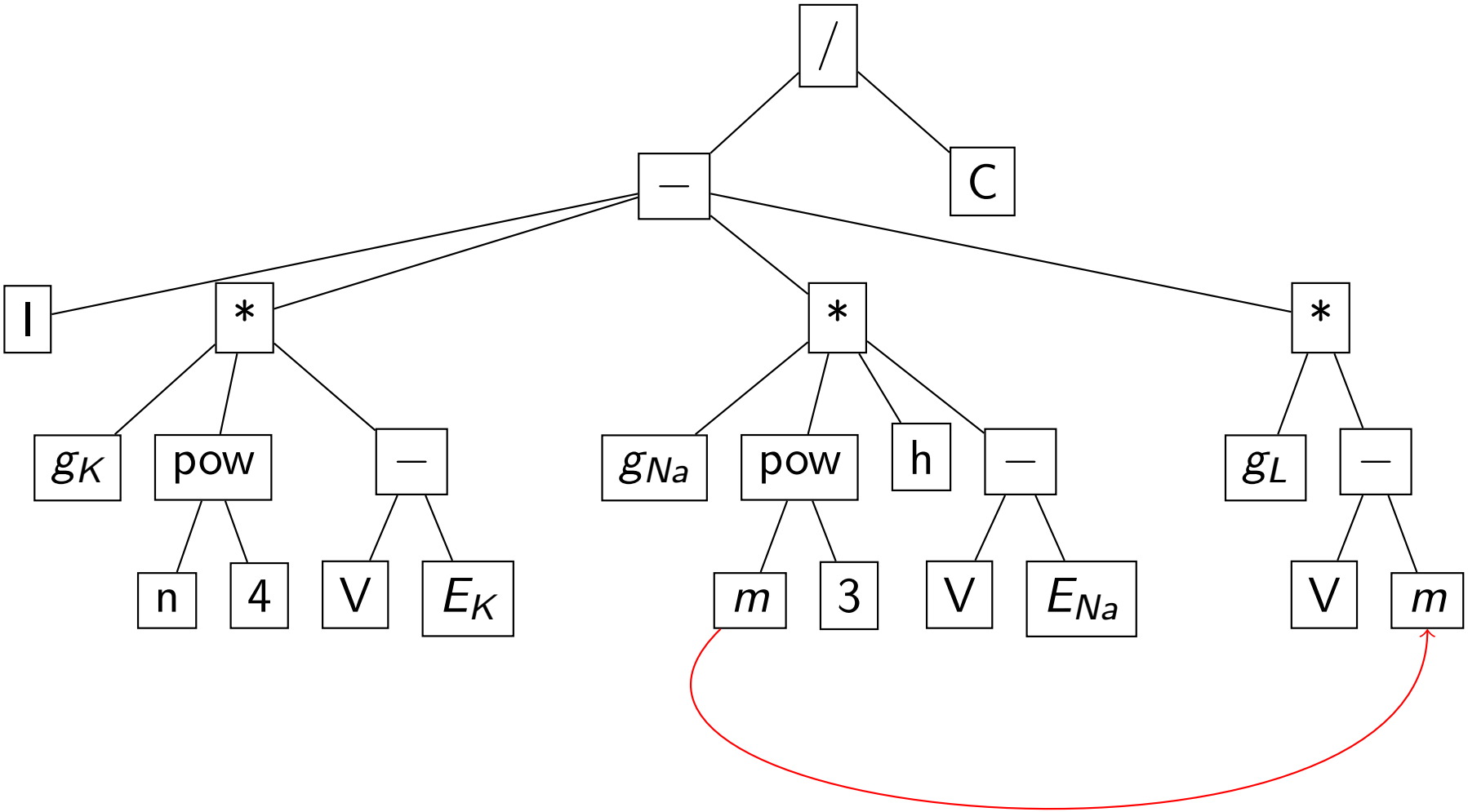
```
double const I     =   10.0  // A
double const C     =   1.0   // F
double const g_Na  =   120.0 // mS
double const g_K   =   36.0  // mS
double const g_L   =   0.3   // mS
double const E_Na  =   120.0 // mV
double const E_K   =  -12.0  // mV
double const E_L   =   10.6  // mV

double
dV_wrt_dt(double V, double n, double m, double h)
{
  return (I - g_K*pow(n, 4)*(V - E_K)
            - g_Na*pow(m, 3)*h*(V - E_Na)
            - g_L*(V - E_L))
            / C;
}
```

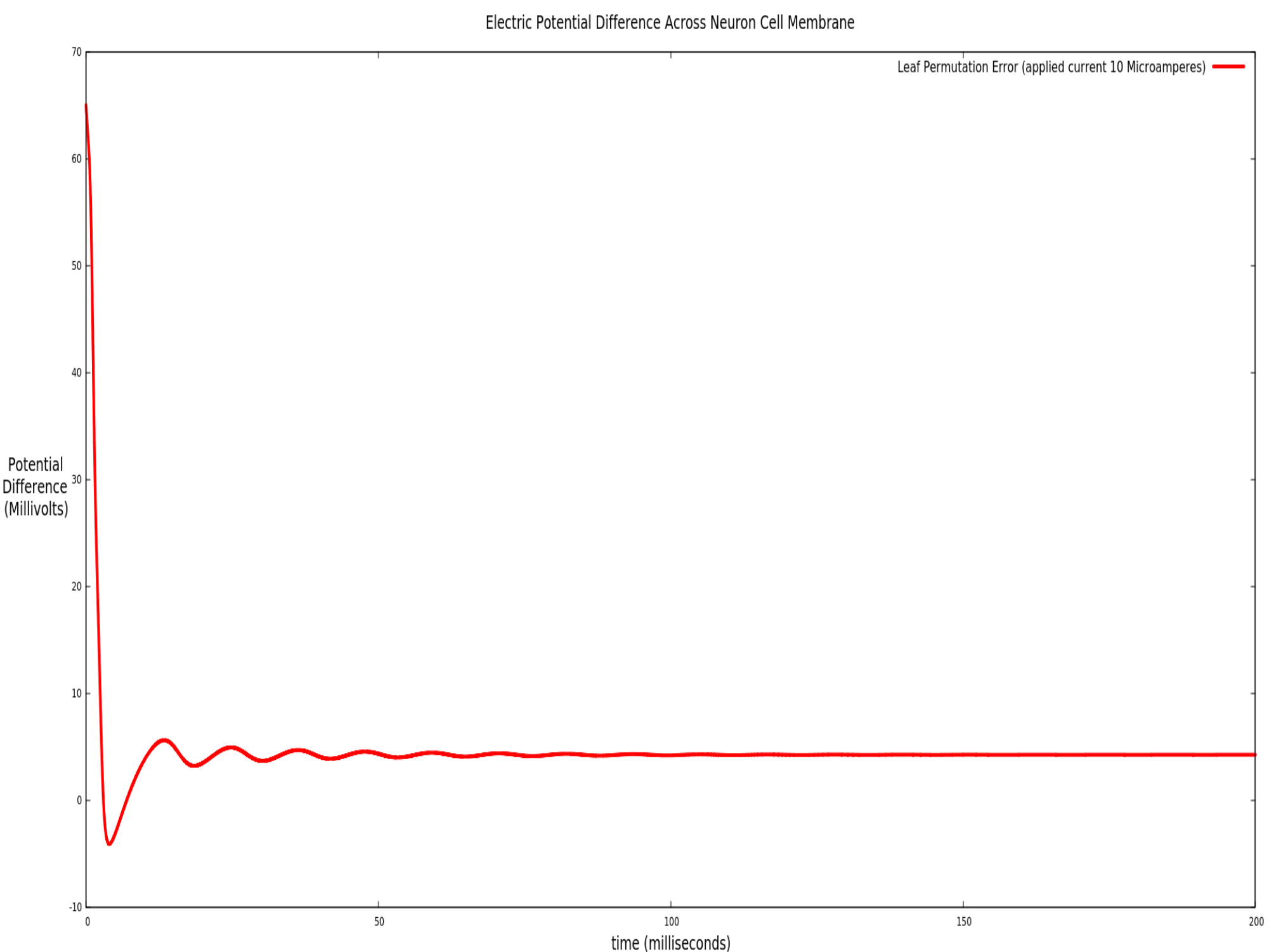

/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
E_L

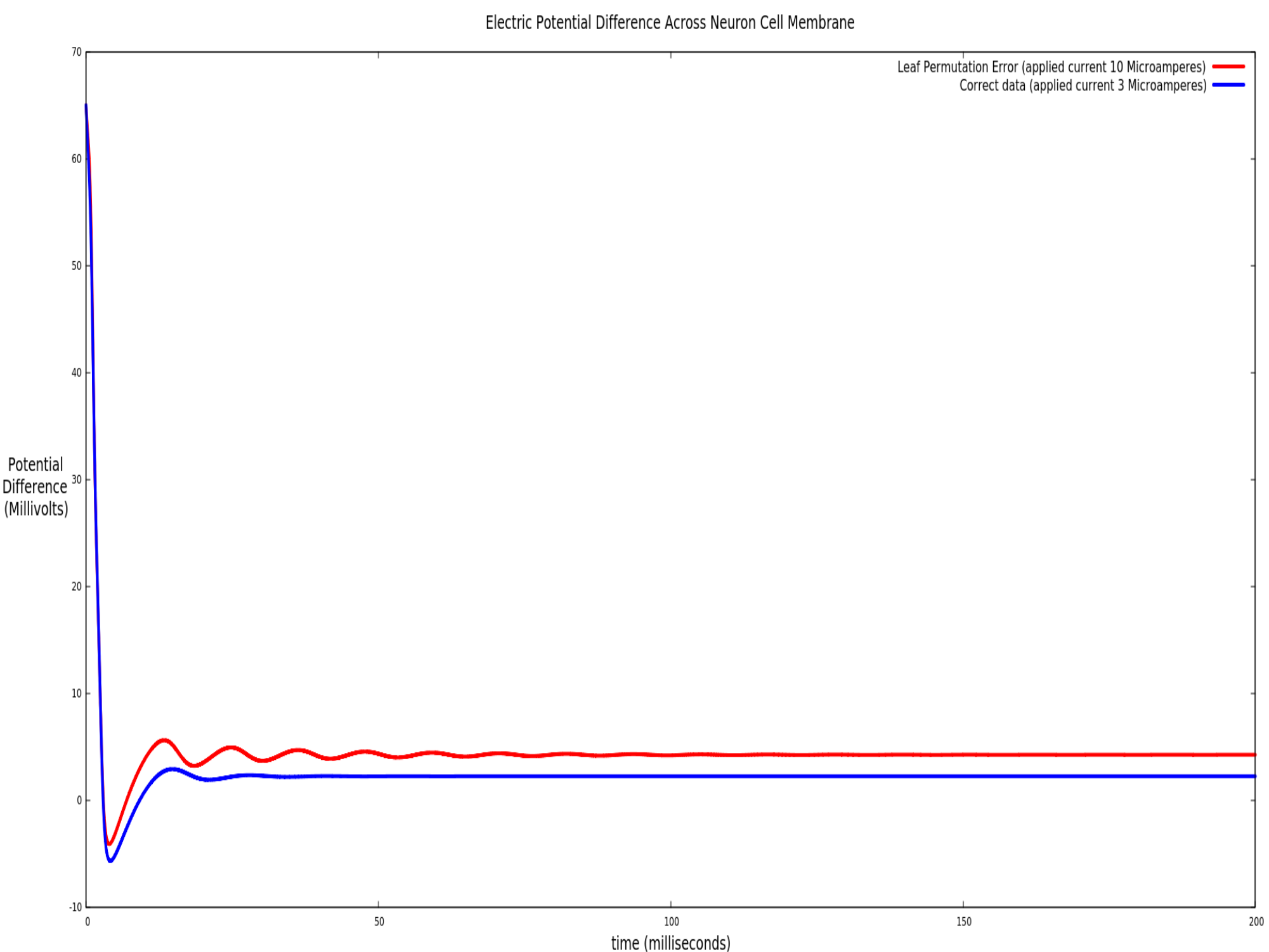

/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
E_L


/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
m



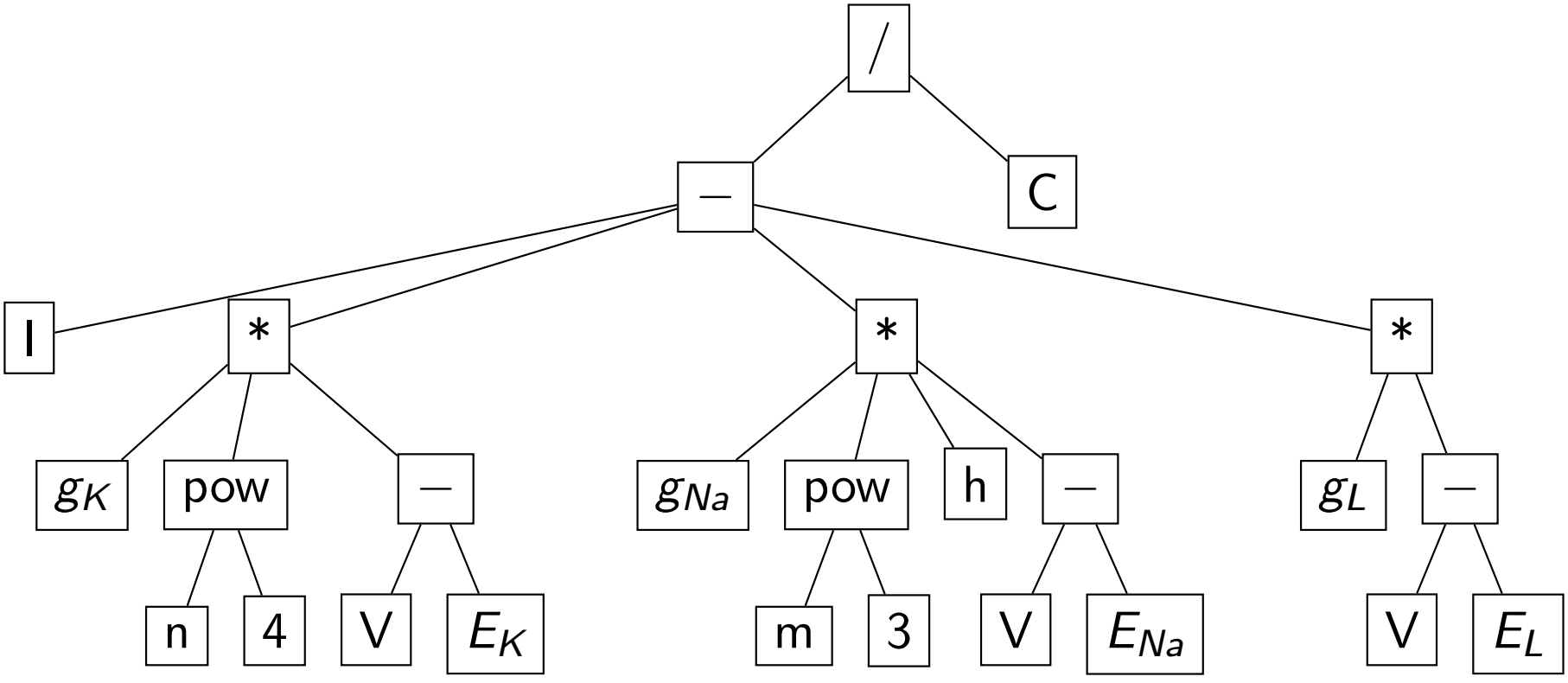
...not an error, program compiles and produces erroneous data.

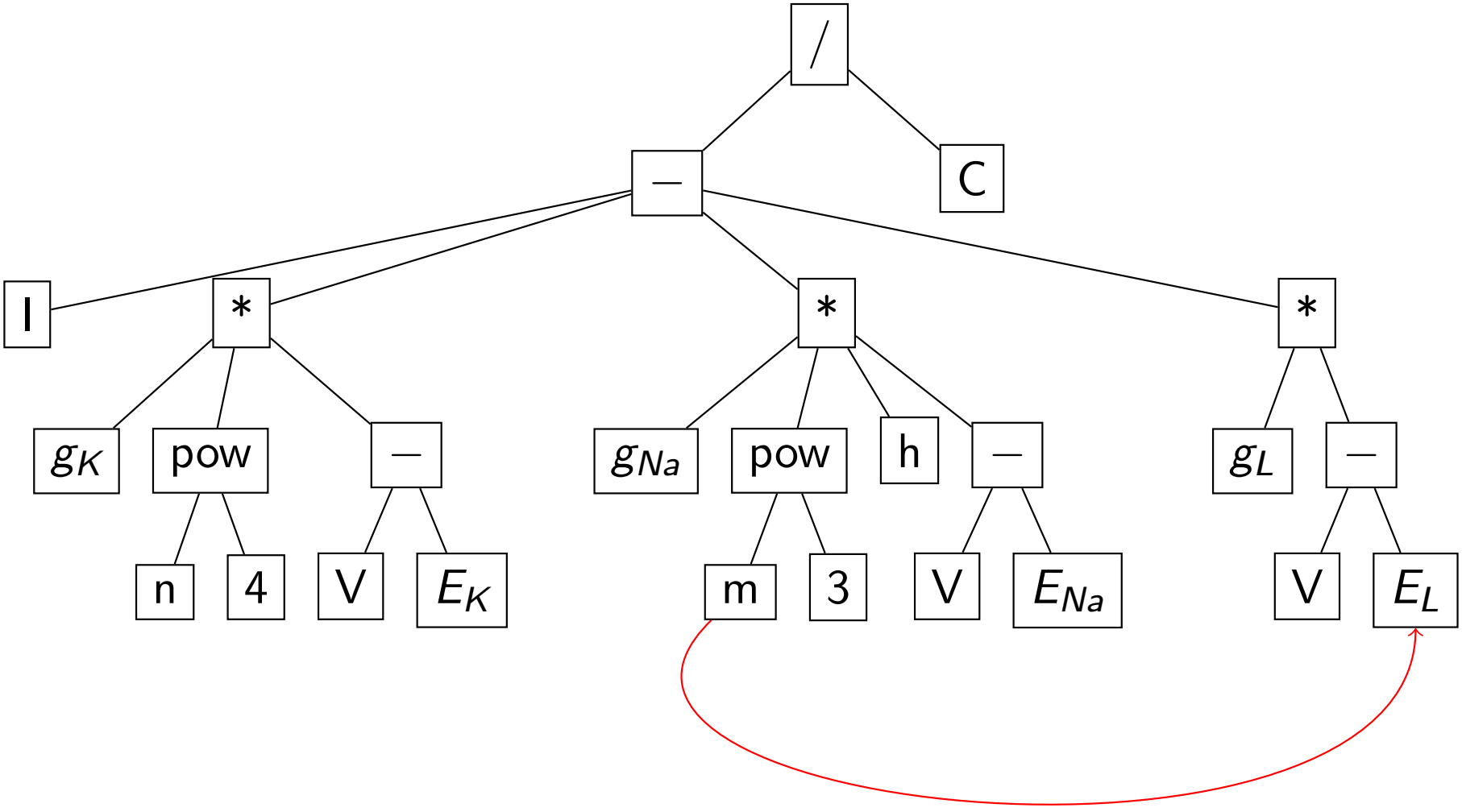

Electric Potential Difference Across Neuron Cell Membrane
Leaf Permutation Error (applied current 10 Microamperes)
Potential Difference (Millivolts)
70
60
50
40
30
20
10
0
-10
0
50
100
150
200
time (milliseconds)

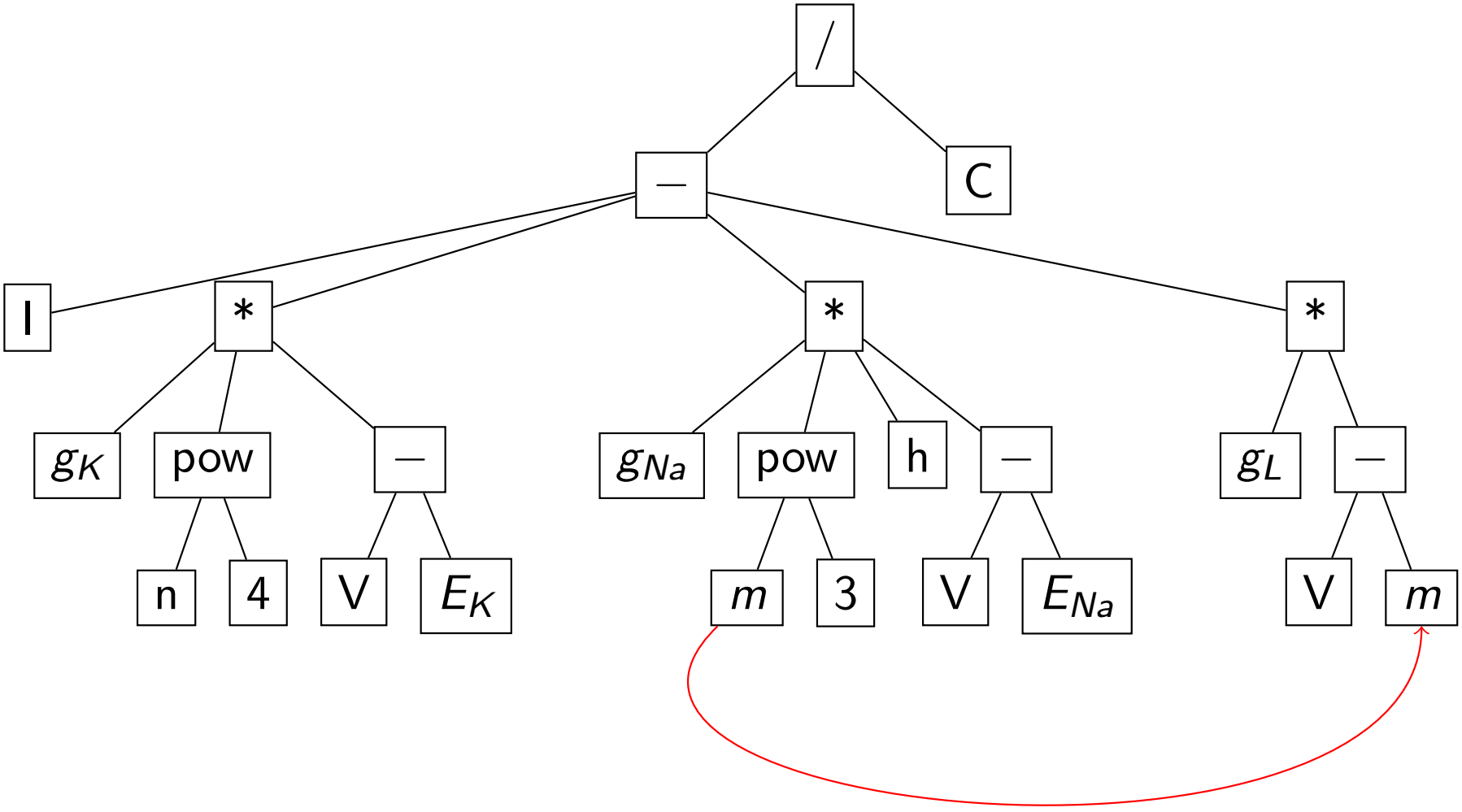

Electric Potential Difference Across Neuron Cell Membrane
Leaf Permutation Error (applied current 10 Microamperes)
Correct data (applied current 3 Microamperes)
Potential Difference (Millivolts)
70
60
50
40
30
20
10
0
-10
0
50
100
150
200
time (milliseconds)

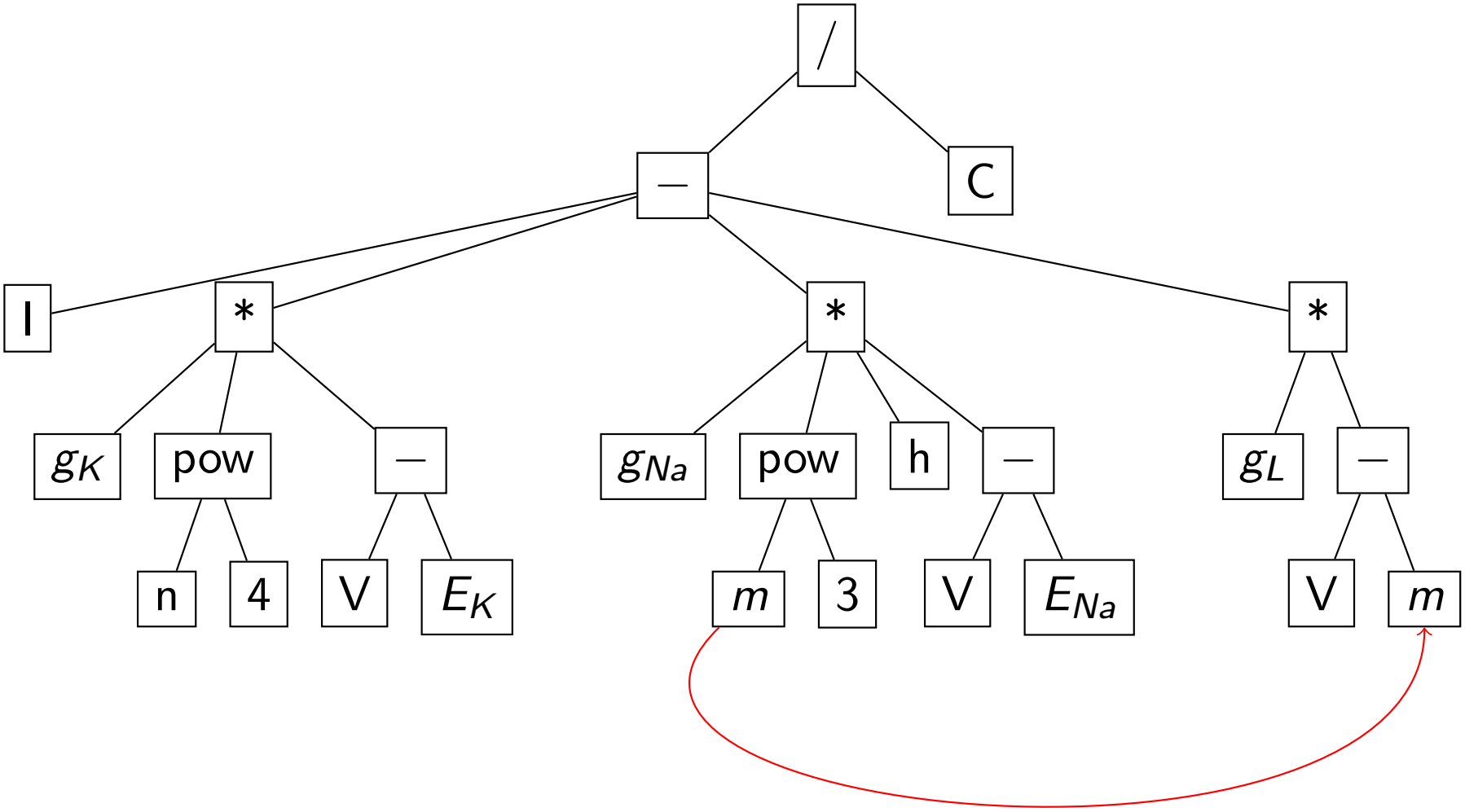
```
Ampere const I      =  10.0;  // micro
Farad const C       =  1.0;   // micro
Siemens const g_Na  =  120.0; // milli
Siemens const g_K   =  36.0;  // milli
Siemens const g_L   =  0.3;   // milli
Volt const E_Na     =  120.0; // milli
Volt const E_K      = -12.0;  // milli
Volt const E_L      =  10.6;  // milli

QuotientType<Volt, second>::type
dV_wrt_dt(Volt V, dimless n, dimless m, dimless h)
{
  return (I - g_K*unit_aux::pow4(n)*(V - E_K)
            - g_Na*unit_aux::pow3(m)*h*(V - E_Na)
            - g_L*(V - E_L))
           / C;
}
```


/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
E_L


/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
E_L


/
−
C
I
*
*
*
g_K
pow
−
g_{Na}
pow
h
−
g_L
−
n
4
V
E_K
m
3
V
E_{Na}
V
m



...produces a compile time error!!

This program cannot be created, so there are no erroneous results!

```
. . . no match for ‘operator-’
(operand types are
‘unit::Volt
{aka unit::Quantity<double, 2, 1, -3, 0, -1, 0, 0>}’
and ‘unit::dimensionless
{aka unit::Quantity<double, 0, 0, 0, 0, 0, 0, 0>}’)

return (I - g_K*pow4(n)*(V - E_K)
          - g_Na*pow3(m)*h*(V - E_Na)
          - g_L*(V - m)) / C;
                  ~~^~~
```

The number of type-checking leaf permutations N can be found via

$$N = \prod_{leaves} n_{leaf}$$

where n_{leaf} is the number of in-scope variables that share the same type as the current leaf.

The number of type-checking leaf permutations N can be found via

$$N = \prod_{leaves} n_{leaf}$$

where n_{leaf} is the number of in-scope variables that share the same type as the current leaf.

The number of leaf permutations for the program without unit analysis:

$$12 \cdot 12 \cdot \dots \cdot 12 = 12^{14} = 1283918464548864$$
$$\approx 1.28 \times 10^{15}$$

The number of type-checking leaf permutations N can be found via

$$N = \prod_{leaves} n_{leaf}$$

where n_{leaf} is the number of in-scope variables that share the same type as the current leaf.

The number of leaf permutations for the program without unit analysis:

$$12 \cdot 12 \cdot \dots \cdot 12 = 12^{14} = 128391846454886 4$$
$$\approx 1.28 \times 10^{15}$$

The number of leaf permutations for the program with unit analysis:

$$1^1 \cdot 3^6 \cdot 4^6 \cdot 1^1 = 2985984$$
$$\approx 2.99 \times 10^6$$