# Towards a Formalization of Claude Shannon's Masters Thesis

andrewtron3000*

March 8, 2018

## 1 Overview

Claude Shannon's seminal MS thesis [3] is considered by many to be the most important masters thesis of the 20th century.

Shannon's thesis laid the foundation for the digital revolution by showing how to reason about electromechanical relay circuits using boolean algebra and propositional logic.

Shannon's thesis contains many theorems and assertions – most of which are not proven. This paper sets out to prove many of those theorems and assertions.

Interested readers can obtain a copy of Shannon's M.S. thesis in the link provided in the reference. Most of the theorems are numbered in the original thesis and this paper refers to those numbers. Occasionally an assertion is not uniquely identifiable in the thesis, and in this case, a page number is used to identify it.

The available copy of the thesis is a scanned PDF of a typewritten manuscript. The quality of the scan is somewhat poor and this was another motivation to subject the contents of the thesis to more rigorous examination. Despite being published in 1936, we find that the typography is sorely lacking: no doubt a harbinger of the degradation of typography during the 20th century [2].

*https://github.com/andrewtron3000/shannon-coq

## 2 Postulates

In order to define the circuit algebra, we first introduce the notion of a circuit, which can be either closed (conducting) or open (non-conducting). This is postulate 4 from the thesis. We specify this postulate as an inductive type in Coq:

```
Inductive circuit : Type :=
| closed : circuit
| open : circuit.
```

Next, we define the notion of "plus" in the circuit algebra. Plus is synonymous with two circuits in series, or the AND operation in boolean algebra. This is defined in postulates 1b, 2a and 3a. We use a Coq definition to formalize this notion.

```
Definition plus (v1 v2 : circuit) : circuit :=
  match v1, v2 with
  | open, open ⇒ open
  | open, closed ⇒ open
  | closed, open ⇒ open
  | closed, closed ⇒ closed
  end.
```

Next, we define the notion of "times" in the circuit algebra. Times is synonymous with two circuits in parallel (or the OR operation in boolean algebra) and it is defined in postulates 1a, 2b, and 3b.

```
Definition times (v1 v2 : circuit) : circuit :=
  match v1, v2 with
  | closed, closed ⇒ closed
  | open, closed ⇒ closed
```

```
  | closed, open ⇒ closed
  | open, open ⇒ open
  end.
```

Now we define some convenient Coq notation for these plus and times functions so that further developments can use the normal symbols for plus and times.

```
Notation "x + y" :=
  (plus x y)
    (at level 50,
     left associativity).
Notation "x * y" :=
  (times x y)
    (at level 40,
     left associativity).
```

Next we will introduce the definition of negation. As one would expect, negation of an open circuit is closed and negation of a closed circuit is open.

```
Definition negation (v1 : circuit) : circuit :=
  match v1 with
  | open ⇒ closed
  | closed ⇒ open
  end.
```

## 3 Proofs of First Theorems

With these postulates defined, we can begin using them to prove theorems 1 through 5.

Throughout this development, with just a few exceptions, our goal is to have all proofs proven with a single tactic application, following the Chlipala discipline [1].

To do this, we start by defining a set of custom Ltac tactics below.

```
Ltac reduce1 X :=
  try destruct X;
  simpl;
  reflexivity.
Ltac reduce2 X Y :=
  try destruct X;
  try destruct Y;
  simpl;
  reflexivity.
Ltac reduce3 X Y Z :=
  try destruct X;
  try destruct Y;
  try destruct Z;
  simpl;
  reflexivity.
```

Now we state Theorem 1a (plus over circuits is commutative) and prove it in a straightforward fashion.

```
Theorem plus_comm : ∀ (x y : circuit),
    x + y = y + x.
Proof.
  intros X Y.
  reduce2 X Y.
Qed.
```

Next we state Theorem 1b (times over circuits is commutative) and prove it.

```
Theorem times_comm : ∀ (x y : circuit),
    x * y = y * x.
Proof.
  intros X Y.
  reduce2 X Y.
Qed.
```

Next we prove Theorem 2a – that plus is associative.

```
Theorem plus_assoc : ∀ (x y z : circuit),
    x + (y + z) = (x + y) + z.
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

Next we prove Theorem 2b – that times is also associative.

```
Theorem times_assoc : ∀ (x y z : circuit),
    x * (y * z) = (x * y) * z.
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

Next, we prove Theorem 3a – that times is distributive.

```
Theorem times_dist : ∀ (x y z : circuit),
    x * (y + z) = (x * y) + (x * z).
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

Next, we prove Theorem 3b, that plus is also distributive.

```
Theorem plus_dist : ∀ (x y z : circuit),
    x + (y * z) = (x + y) * (x + z).
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

Now we get to Theorem 4a which is a theorem about how times works when the first argument is the value open.

```
Theorem open_times_x : ∀ (x : circuit),
    open * x = x.
Proof.
  intros X.
  reduce1 X.
Qed.
```

And we prove Theorem 4b which is a theorem about how plus works when the first argument is the value closed.

```
Theorem closed_plus_x : ∀ (x: circuit),
    closed + x = x.
Proof.
  intros X.
  reduce1 X.
Qed.
```

Next, Theorem 5a asserts a relationship of plus when the first argument is the value open.

```
Theorem open_plus_x : ∀ (x: circuit),
    open + x = open.
Proof.
  intros X.
  reduce1 X.
Qed.
```

And Theorem 5b asserts a relationship of times when the first argument is closed.

```
Theorem closed_times_x : ∀ (x : circuit),
    closed * x = closed.
Proof.
  intros X.
  reduce1 X.
Qed.
```

# 4 Negation Theorems

Theorem 6a asserts the behavior when a circuit and its negative are connected in series. As you might expect, this always results in an open circuit.

```
Theorem plus_neg : ∀ (x : circuit),
    x + (negation x) = open.
Proof.
  intros X.
  reduce1 X.
Qed.
```

And Theorem 6b specifies what happens when you connect a circuit and its negative in parallel. As you would expect, the circuit is always closed in this case.

```
Theorem times_neg : ∀ (x : circuit),
    x * (negation x) = closed.
Proof.
  intros X.
  reduce1 X.
Qed.
```

Theorems 7a and 7b specify what happens when you negate the specific values of open or closed. These are quite simple and are formalized below.

```
Theorem closed_neg :
  negation closed = open.
Proof.
  reduce1 X.
Qed.

Theorem open_neg :
  negation open = closed.
Proof.
  reduce1 X.
Qed.
```

Theorem 8 specifies what happens when you take the negative of the negative of a circuit. As expected one gets the original circuit back.

```
Theorem double_neg : ∀ (x : circuit),
    negation (negation x) = x.
Proof.
  intros X.
  reduce1 X.
Qed.
```

# 5 Equivalence to Calculus of Propositions

Claude then describes how the algebra defined above is equivalent to propositional logic. He does this by showing an equivalence between the algebra above and E.V. Huntington's formulation of symbolic logic. This formulation has 6 postulates and postulates 1, 2, 3, and 4 are clearly met without proof. Postulates 5 and 6 of E.V. Huntington's formulation are proved below.

```
Theorem plus_same : ∀ (x y : circuit),
    x = y →
    x + y = x.
Proof.
  intros X Y.
  intros H.
  rewrite → H.
  reduce1 Y.
Qed.

Theorem dist_neg : ∀ (x y : circuit),
    (x * y) + (x * (negation y)) = x.
Proof.
  intros X Y.
  reduce2 X Y.
Qed.
```

We can, for completeness also prove the definition mentioned in proposition 6.

```
Theorem dist_neg_defn : ∀ (x y : circuit),
    (x * y) = negation ((negation x) +
                        (negation y)).
Proof.
  intros X Y.
  reduce2 X Y.
Qed.
```

# 6 A Proof of De Morgans Law

Once this equivalence between the circuit algebra and propositional logic is shown, it is possible to bring over powerful theorems from propositional logic into our new algebra. We will begin by proving De Morgan's theorem. This is Theorem 9.

While the thesis asserts these theorems for an arbitrary number of variables, we will only illustrate proofs for two and three variables.

```
Theorem demorgan_9a_2 : ∀ (x y: circuit),
    negation (x + y) =
    ( (negation x) *
      (negation y) ).
Proof.
  intros X Y.
  reduce2 X Y.
Qed.

Theorem demorgan_9a_3 : ∀ (x y z: circuit),
    negation (x + y + z) =
    ( (negation x) *
      (negation y) *
      (negation z) ).
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

And we will prove De Morgan's theorem over times over two and three variables. This is Theorem 9b.

```
Theorem demorgan_9b_2 : ∀ (x y: circuit),
    negation (x * y) =
    ( (negation x) +
      (negation y) ).
Proof.
  intros X Y.
  reduce2 X Y.
Qed.

Theorem demorgan_9b_3 : ∀ (x y z: circuit),
    negation (x * y * z) =
```

```
    ( (negation x) +
      (negation y) +
      (negation z) ).
Proof.
  intros X Y Z.
  reduce3 X Y Z.
Qed.
```

## 7 Onward to Taylor Series

Claude then starts the discussion of how to specify arbitrary functions in the circuit algebra. He starts by illustrating the capability to expand an arbitrary function into a Taylor series expansion.

In order to complete these proofs we introduce more Ltac tactic machinery. At this point it will be important for us to be able to leverage many of the above theorems in subsequent proofs. We encapsulate these theorems into a new set of Ltac tactics. The tactics are shown below.

```
Ltac wham :=
  try repeat ( (rewrite → closed_times_x;
                rewrite → closed_neg;
                rewrite → open_times_x) ||
               (rewrite → open_neg;
                rewrite → closed_times_x;
                rewrite → open_times_x) ||
               (rewrite → open_neg) ||
               (rewrite → closed_neg) ).

Ltac open_plus_bam :=
  try ( (rewrite → open_plus_x) ||
        (rewrite plus_comm;
         rewrite open_plus_x) ).

Ltac closed_plus_bam :=
  try ( (rewrite → closed_plus_x) ||
        (rewrite plus_comm;
         rewrite closed_plus_x) ).

Ltac open_times_bam :=
  try ( (rewrite → open_times_x) ||
        (rewrite times_comm;
         rewrite open_times_x) ).

Ltac closed_times_bam :=
  try ( (rewrite → closed_times_x) ||
        (rewrite times_comm;
         rewrite closed_times_x) ).

Ltac bam :=
  try repeat (closed_plus_bam ||
              open_plus_bam ||
              closed_times_bam ||
              open_times_bam).

Ltac wham_bam_1 X :=
  try (destruct X;
       wham; bam;
       reflexivity).

Ltac wham_bam_2 X Y :=
  try (destruct X, Y;
       wham; bam;
       reflexivity).

Ltac wham_bam_3 X Y Z :=
  try (destruct X, Y, Z;
       wham; bam;
       reflexivity).

Ltac wham_bam_4 W X Y Z :=
  try (destruct W, X, Y, Z;
       wham; bam;
       reflexivity).

Ltac wham_bam_5 V W X Y Z :=
  try (destruct V, W, X, Y, Z;
       wham; bam;
       reflexivity).

Ltac wham_bam_6 S V W X Y Z :=
  try (destruct S, V, W, X, Y, Z;
       wham; bam;
       reflexivity).
```

And now we have the appropriate machinery in place to be able to prove the Taylor series expansion on two variables shown in Theorems 10a and 10b, here called taylorA and taylorB respectively.

```
Theorem taylorA :
  ∀ (x y: circuit),
    ∀ (f : circuit → circuit → circuit),
      f x y =
      ( (x * (f open y)) +
        ((negation x) * (f closed y)) ).
Proof.
  intros X Y.
```

```
  intros F.
  wham_bam_2 X Y.
Qed.

Theorem taylorB :
  ∀ (x y: circuit),
  ∀ (f : circuit → circuit → circuit),
    f x y =
    ( ((f closed y) + x) *
      ((f open y) + (negation x)) ).
Proof.
  intros X Y.
  intros F.
  wham_bam_2 X Y.
Qed.
```

We continue with the expansion of the Taylor series to the second variable as described in Theorems 11a and 11b.

```
Theorem taylor11a : ∀ (x y: circuit),
    ∀ (f : circuit → circuit → circuit),
      f x y =
      ( (x * y) *
        (f open open) ) +
      ( (x * (negation y)) *
        (f open closed) ) +
      ( ((negation x) * y) *
        (f closed open) ) +
      ( ((negation x) *
         (negation y)) *
        (f closed closed) ).
Proof.
  intros X Y.
  intros F.
  wham_bam_2 X Y.
Qed.

Theorem taylor11b : ∀ (x y: circuit),
    ∀ (f : circuit → circuit → circuit),
      f x y =
      ( (x + y) +
        (f closed closed) ) *
      ( (x + (negation y)) +
        (f closed open) ) *
      ( ((negation x) + y) +
        (f open closed) ) *
      ( ((negation x) + (negation y)) +
        (f open open) ).
Proof.
  intros X Y.
  intros F.
  wham_bam_2 X Y.
Qed.
```

We skip the proofs of Theorem 12a and 12b as we have shown their validity in the two variable case above. We also leave the proof of Theorem 13 to future work.

At the end of the first paragraph on page 14, the thesis illustrates an example of finding the negative of a particular function using the generalization described in Theorem 13. We prove it here, but do not use the power of Theorem 13. Instead we can use our simple custom tactic with good results.

```
Theorem example1 : ∀ (w x y z: circuit),
    negation ( x +
               ( y *
                 (z + w * (negation x))) ) =
    (negation x) *
    ( (negation y) +
      (negation z) *
      ((negation w) + x) ).
Proof.
  intros W X Y Z.
  wham_bam_4 W X Y Z.
Qed.
```

## 8 Simplification Theorems

Next Claude presents Theorems 14-18, useful for simplifying expressions.

```
Theorem x_plus_x_is_x : ∀ (x: circuit),
    x + x = x.
Proof.
  intros X.
  wham_bam_1 X.
Qed.

Theorem x_times_x_is_x : ∀ (x: circuit),
    x * x = x.
Proof.
  intros X.
```

```
  wham_bam_1 X.
Qed.

Theorem x_plus_xy : ∀ (x y: circuit),
    (x + (x * y)) = x.
Proof.
  intros X Y.
  wham_bam_2 X Y.
Qed.

Theorem x_x_plus_y : ∀ (x y: circuit),
    x * (x + y) = x.
Proof.
  intros X Y.
  wham_bam_2 X Y.
Qed.

Theorem theorem16a : ∀ (x y z: circuit),
    (x * y) + (negation x) * z =
    (x * y) + ((negation x) * z) + (y * z).
Proof.
  intros X Y Z.
  wham_bam_3 X Y Z.
Qed.

Theorem theorem16b : ∀ (x y z: circuit),
    (x + y) * ((negation x) + z) =
    (x + y) * ((negation x) + z) * (y + z).
Proof.
  intros X Y Z.
  wham_bam_3 X Y Z.
Qed.

Theorem theorem17a : ∀ (x: circuit),
    ∀ (f: circuit → circuit),
      x * (f x) = x * (f open).
Proof.
  intros X.
  intros F.
  wham_bam_1 X.
Qed.

Theorem theorem17b : ∀ (x: circuit),
    ∀ (f: circuit → circuit),
      x + (f x) = x + (f closed).
Proof.
  intros X.
  intros F.
  wham_bam_1 X.
Qed.

Theorem theorem18a : ∀ (x: circuit),
    ∀ (f: circuit → circuit),
      (negation x) * (f x) =
      (negation x) * (f closed).
Proof.
  intros X.
  intros F.
  wham_bam_1 X.
Qed.

Theorem theorem18b : ∀ (x: circuit),
    ∀ (f: circuit → circuit),
      (negation x) + (f x) =
      (negation x) + (f open).
Proof.
  intros X.
  intros F.
  wham_bam_1 X.
Qed.
```

## 9 Series Parallel Example

Figure 5 shows an example of an expression that represents a fairly complex series parallel circuit. The figure is first rendered into a hindrance equation. The equation is then manipulated into a simpler form. We prove that the transformation between Figure 5 and Figure 6 is correct.

```
Theorem figure5 : ∀ (s v w x y z: circuit),
    w + ((negation w) * (x + y)) +
    (x + z) * (s + (negation w) + z) *
    ((negation z) + y + (negation s) * v) =
    w + x + y + z * (negation s) * v.
Proof.
  intros S V W X Y Z.
  wham_bam_6 S V W X Y Z.
Qed.
```

## 10 Multi-Terminal Networks

In this section, we discuss Section III of the thesis. Section III begins by illustrating two types of non-serial and non-parallel networks: the delta and wye circuit configurations.

We tackle the equivalence of Figure 8, the delta to wye transformation first. The path from a to b in the delta configuration is r in parallel with the both s and t in series. This should be equivalent to the path from a to b in the wye configuration, where the path is (r in parallel with t) in series with (r in parallel with s). The next proof is a proof of this equivalence. Then we provide proofs of the equivalence of paths from b to c and from c to a.

```
Theorem figure8_a_to_b : ∀ (r s t : circuit),
    r * (s + t) = (r * t) + (r * s).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.

Theorem figure8_b_to_c : ∀ (r s t : circuit),
    s * (t + r) = (r * s) + (s * t).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.

Theorem figure8_c_to_a : ∀ (r s t : circuit),
    t * (r + s) = (s * t) + (r * t).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.
```

Next we tackle Figure 9, the wye to delta transformation. We prove this by proving each path independently as well.

```
Theorem figure9_a_to_b : ∀ (r s t : circuit),
    r + s = (r + s) * ( (t + s) + (r + t) ).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.

Theorem figure9_b_to_c : ∀ (r s t : circuit),
    s + t = (t + s) * ( (r + s) + (r + t) ).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.

Theorem figure9_c_to_a : ∀ (r s t : circuit),
    t + r = (r + t) * ( (r + s) + (t + s) ).
Proof.
  intros R S T.
  wham_bam_3 R S T.
Qed.
```

# 11 More Complex Transformations

Figure 10 illustrates the transformation of a 5 point star to a fully connected graph. We prove the equivalence of the path from a to b in Figure 10 and leave the proof of the other paths to future work.

```
Theorem figure10_a_to_b : ∀ (r s t u v : circuit),
    r + s = (r + s) *
            ( (t + r) + (s + t) ) *
            ( (r + u) + (s + u) ) *
            ( (v + r) + (v + s) ) *
            ( (t + r) + (t + u) + (s + u) ) *
            ( (r + u) + (v + u) + (v + s) ) *
            ( (t + r) + (t + v) + (v + s) ) *
            ( (t + r) + (t + u) + (v + u) +
              (v + s) ).
Proof.
  intros R S T U V.
  wham_bam_5 R S T U V.
Qed.
```

Figure 11 presents a relatively simple non-series and non-parallel network. We first prove that Figure 11 and Figure 12 are equivalent. The thesis mentions that this can be done by using the star to mesh transformations, but we do not need such power. We can use the same tactics we've used in previous proofs.

For convenience we create definitions of the networks in Figure 11 and Figure 12. This can be done by simply creating definitions that cover every possible path from a to b as in Figure 13.

```
Definition figure11 (r s t u v : circuit) :
  circuit :=
  (r + s) *
  (u + v) *
  (r + t + v) *
```

```
  (u + t + s).

Definition figure12 (r s t u v : circuit) :
  circuit :=
  (r + s) *
  ( ((r + t) * u) +
    ((t + s) * v) ).
```

Once we have defined these figures, we can prove their equivalence.

```
Theorem figure_11_12_equiv :
  ∀ (r s t u v : circuit),
    figure11 r s t u v =
    figure12 r s t u v.
Proof.
  intros R S T U V.
  wham_bam_5 R S T U V.
Qed.
```

The thesis also mentions in this section that Figure 11 can be simplified. We prove this assertion here.

```
Theorem figure_11_simpler :
  ∀ (r s t u v : circuit),
    figure11 r s t u v =
    (r * u) + (s * v) +
    (r * t * v) + (s * t * u).
Proof.
  intros R S T U V.
  wham_bam_5 R S T U V.
Qed.
```

## 12 Simultaneous Equations

We leave most of the formalization of simultaneous equations to future work, but prove the implication on page 25. To do this we create several new Ltac tactics.

The following Ltac tactic uses the demorgan_9a_2 theorem on products of negations.

```
Ltac demorgan_2 :=
  match goal with
  | [ ⊢ ( (negation _) *
            (negation _) = _ ) ] ⇒
    rewrite ← demorgan_9a_2
  end.
```

The following tactic applies reflexivity to trivial goals.

```
Ltac explicit_reflexive :=
  try match goal with
      | [ ⊢ (open = open) ] ⇒
        reflexivity
      | [ ⊢ (closed = closed) ] ⇒
        reflexivity
      end.
```

The following tactic leverages potential contradictions in the hypotheses in the context.

```
Ltac contra :=
  match goal with
  | [ Hx : (open * (negation closed) = closed) ⊢
      (closed = open) ] ⇒
    ( rewrite ← closed_neg in Hx;
      simpl in Hx;
      rewrite → Hx;
      reflexivity )
  | [ Hx : (open * (negation closed) = closed) ⊢
      (open = closed) ] ⇒
    ( rewrite ← closed_neg in Hx;
      simpl in Hx;
      rewrite → Hx;
      reflexivity )
  | [ Hx : (open = closed) ⊢
      _ ] ⇒
    ( simpl;
      rewrite → Hx;
      reflexivity )
  | [ Hx : (closed = open) ⊢
      _ ] ⇒
    ( simpl;
      rewrite → Hx;
      reflexivity )
  end.
```

The following nearly trivial tactic just simplifies a hypothesis in the context.

```
Ltac simpl_h :=
  match goal with
  | [ Hx : _ ⊢ _ ] ⇒ simpl in Hx
  end.
```

Now we assemble the above Ltac tactics into more powerful tools.

```
Ltac pow :=
  try repeat ( demorgan_2 ||
               explicit_reflexive ||
               contra ||
               simpl_h ).

Ltac blammo_2 X Y :=
  try repeat (pow;
              destruct X, Y;
              repeat (wham;
                      bam);
              repeat (simpl;
                      reflexivity)).

Ltac blammo_3 X Y Z :=
  try repeat (pow;
              destruct X, Y, Z;
              repeat (wham;
                      bam);
              repeat (simpl;
                      reflexivity)).
```

Now that we have our new Ltac machinery, we can tackle the implication on page 25.

```
Theorem page_25_implication :
  ∀ (a b : circuit),
    a * (negation b) = closed →
    (negation a) * (negation b) = (negation b).
Proof.
  intros A B H.
  blammo_2 A B.
Qed.

Theorem page_25_implication_2 :
  ∀ (a b : circuit),
    a * (negation b) = closed →
    (a * b) = a.
Proof.
  intros A B H.
  blammo_2 A B.
Qed.

Theorem page_25_implication_3 :
  ∀ (a b : circuit),
    a * (negation b) = closed →
    (negation a) + b = open.
Proof.
  intros A B H.
  blammo_2 A B.
Qed.

Theorem page_25_implication_4 :
  ∀ (a b : circuit),
    a * (negation b) = closed →
    (negation a) + (negation b) = (negation a).
Proof.
  intros A B H.
  blammo_2 A B.
Qed.

Theorem page_25_implication_5 :
  ∀ (a b : circuit),
    a * (negation b) = closed →
    (a + b) = b.
Proof.
  intros A B H.
  blammo_2 A B.
Qed.
```

# 13 Matrix Methods and Special Methods

We leave the matrix methods formalization to future work, but prove the implication on page 30.

```
Theorem page_30_implication : ∀ (r s x : circuit),
    (negation x) = (r * (negation x)) + s →
    x = ((negation r) + x) * (negation s).
Proof.
  intros R S X.
  intros H1.
  blammo_3 R S X.
Qed.
```

# 14 Synthesis of Networks

We now move to formalization of synthesis techniques. We first define the disjunct operator on page 32.

```
Definition disjunct (v1 v2 : circuit) :
  circuit :=
  (v1 * (negation v2)) +
  ((negation v1) * v2).
```

We provide a bit of notation that aids our development.

```
Notation "x @ y" :=
  (disjunct x y)
    (at level 50,
     left associativity).
```

We create some new tactics that allow us to automate the use of the disjunct definition.

```
Ltac disjunctor :=
  match goal with
  | [ ⊢ _ @ _ = _ ] ⇒ unfold disjunct
  | [ ⊢ _ * (_ @ _) = _ ] ⇒ unfold disjunct
  | [ ⊢ _ + (_ @ _) = _ ] ⇒ unfold disjunct
  | [ ⊢ negation (_ @ _) = _ ] ⇒ unfold disjunct
  end.

Ltac kapow_1 X :=
  try (disjunctor;
       wham_bam_1 X).

Ltac kapow_2 X Y :=
  try (disjunctor;
       wham_bam_2 X Y).

Ltac kapow_3 X Y Z :=
  try (disjunctor;
       wham_bam_3 X Y Z).
```

## 15 Properties of Disjuncts

Now we can proceed to page 33 and prove that the disjunct operator is commutative, associative and distributive. We also prove the property of negation of a disjunction.

```
Theorem disjunct_comm : ∀ (a b : circuit),
    a @ b = b @ a.
Proof.
  intros A B.
  kapow_2 A B.
Qed.

Theorem disjunct_assoc : ∀ (a b c : circuit),
    (a @ b) @ c = a @ (b @ c).
Proof.
  intros A B C.
  kapow_3 A B C.
Qed.

Theorem disjunct_distrib : ∀ (a b c : circuit),
    a * (b @ c) = (a * b) @ (a * c).
Proof.
  intros A B C.
  kapow_3 A B C.
Qed.

Theorem disjunct_neg : ∀ (a b : circuit),
    negation (a @ b) = a @ (negation b).
Proof.
  intros A B.
  kapow_2 A B.
Qed.

Theorem disjunct_closed : ∀ (a : circuit),
    a @ closed = a.
Proof.
  intros A.
  kapow_1 A.
Qed.

Theorem disjunct_open : ∀ (a : circuit),
    a @ open = negation a.
Proof.
  intros A.
  kapow_1 A.
Qed.
```

## 16 Synthesis of Symmetric Functions

We prove the assertion at the top of page 40 and leave the remainder of the section to future work. We first create two Ltac tactics that will be helpful.

```
Ltac hypothesis_app :=
  match goal with
  | [Hx : (_ = closed) ⊢ _ ] ⇒ rewrite → Hx
  | [Hx : (_ = open) ⊢ _ ] ⇒ rewrite → Hx
  end.

Ltac zap_1 X :=
  try repeat (hypothesis_app;
              wham_bam_1 X).
```

And then we can proceed with the symmetry example on page 40.

```
Theorem symmetry_example : ∀ (x y z : circuit),
    x = closed →
    y = closed →
    x * y + x * z + y * z = closed.
Proof.
  intros X Y Z xc yc.
  zap_1 X.
Qed.
```

We relegate pages 41 to 50 as future work.

## 17 A Selective Circuit

In this section we formalize the example starting on page 51. We verify the reduction to the simplest serial-parallel form.

```
Theorem selective_circuit : ∀ (w x y z : circuit),
    ( w * x * y * z ) +
    ( (negation w) * (negation x) * y * z) +
    ( (negation w) * x * (negation y) * z) +
    ( (negation w) * x * y * (negation z) ) +
    ( w * (negation x) * (negation y) * z ) +
    ( w * (negation x) * y * (negation z) ) +
    ( w * x * (negation y) * (negation z) )
    =
    w * ( x *
          ( (y * z) +
            ( (negation y) *
              (negation z) ) ) +
          (negation x) *
          ( ((negation y) * z) +
            (y * (negation z)) ) ) +
    (negation w) *
    ( ( x * ( ((negation y) * z) +
              (y * (negation z)) ) ) +
      ( (negation x) * y * z ) ).
Proof.
  intros W X Y Z.
  wham_bam_4 W X Y Z.
Qed.
```

## 18 Future Work

There are a significant number of claims and assertions that have been proven in this paper. However, there are still a significant number of claims not explicitly proven which have been relegated to future work.

Additional future work is to convert the postulates into a set of relations. These might allow the more elegant encoding of the non-serial and non-parallel transformations such as wye and delta transformations so that those transformations could be explicitly used in proofs of more complicated hindrance functions. In the current work, we can only prove "slices" of these topologies because of the lack of ability to precisely define the delta and wye transformations.

In the event that a complete formalization of Claude Shannon's thesis were completed, we would have a solid foundation upon which to build electromechanical relay circuits of the future.

## 19 Conclusion

This paper has provided proofs of many of the claims and assertions made in Claude Shannon's masters thesis. In some sense these proofs can serve as an additional reading companion – helping readers stay on the topic of the interesting ideas in the thesis without getting distracted about whether a particular claim or assertion is true.

## References


[1] Adam Chlipala. *Certified Programming With Dependent Types : a Pragmatic Introduction to the Coq Proof Assistant*, chapter 16, pages 363–371. MIT Press, 2013.

[2] Reginald J. Qnuth. A systematic evaluation of the observed degradation of typesetting technology in the 20th century. In *Proceedings of ACH SIGBOVIK*, pages 65–76, 2007.

[3] Claude Elwood Shannon. A symbolic analysis of relay and switching circuits, December 1940. https://dspace.mit.edu/handle/1721.1/11173.