A Sound and Complete Proof Theory for Propositional Logical Contingencies

Alexander Hertel [*] Philipp Hertel [†] Charles Morgan [‡]

Abstract

There are simple, purely syntactic axiomatic proof systems for both the logical truths and the logical falsehoods of propositional logic. However, to date no such system has been developed for the logical contingencies, that is, formulas that are both satisfiable and falsifiable. This paper formalizes and purely syntactic axiomatic proof systems for the logical contingencies and proves its soundness as well as completeness.

1 Motivation

Truth-preserving inference rules are useful in science and mathematics for drawing legitimate conclusions from hypotheses. On the other hand, falsehood-preserving inference rules allow one to derive hypotheses from observations so that the hypotheses logically entail the observations. In this sense, falsehood preserving rules constitute a logic of discovery [3]. However, if the system of falsehood preserving rules is complete, then among the hypotheses generated will be all logical falsehoods.

By contrast, it is the goal of science to produce hypotheses that are neither logically true, nor logically false, but to produce ones that are logically contingent, that is, formulas which are both satisfiable and falsifiable. In science we want our theories to be logically contingent formulas that account for our observations. We may well ask: Is there a 'logic' which captures these formulas? [2]

As shown in the Venn diagram below in Figure 1, the logical contingencies (LC) constitute the intersection between satisfiable and falsifiable formulas. The non-contingent formulas are either logically true (LT), meaning that they are true under all truth assignments, or logically false (LF), meaning that they are false under all truth assignments.

To show that a formula is not LT (i.e. falsifiable), one need simply provide a single row on the truth table which assigns it the value *False*. To show that a formula is not LF (i.e. satisfiable), one only needs to provide a single row on the truth table which assigns it the value *True*. However, to show that a formula is not LC, one must provide the entire truth table. In contrast, showing that a formula is LT or LF requires the entire table, whereas showing one to be LC requires two rows; one row is not sufficient. These observations suggest that in some way the LC formulas are different from the LT or LF formulas. The sets LT and LF are both co$\mathcal{NP}$-Complete, whereas the set of satisfiable formulas, the set of falsifiable formulas, LC are each $\mathcal{NP}$-Complete. LC is interesting because it is an $\mathcal{NP}$-Complete set which does not intersect LT or LF. In addition, there are several other differences between LC and LT/LF. For instance, LC is not closed under uniform substitutions and is closed under taking negations, whereas the others are not.

[*] Supported by NSERC and the University of Toronto Deptartment of Computer Science
[†] Supported by NSERC and the University of Toronto Deptartment of Computer Science
[‡] Supported by SSHRC

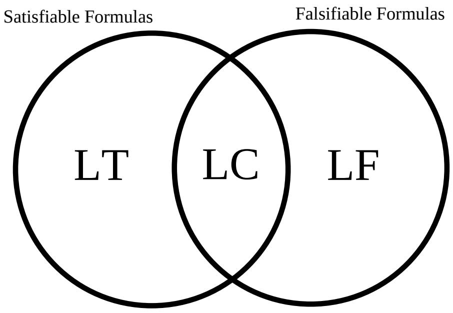

Figure 1:

Both LT as well as LF have simple, syntactically-based axiomatic proof systems that make no use of formal semantics. However, to date no such system seems to have been developed for the logical contingencies. It is possible to formulate a simple proof system for LC that does make use of formal semantics. For example, a proof can just be a pair of truth assignments, one that satisfies the formula and one that falsifies it. In this paper we develop a sound and complete proof theory for logical contingencies which is purely syntactic. We will call this system $\mathcal{LC}$.

Further motivation for this research is described in section 5, 'Future Work'.

2 Proof System

Definition 2.1. *Define $R(A, B, C)$ as the set of formulas which result from replacing as many or as few occurrences of A by B in C as we like. The definition is a simple recursion defined on the complexity of C:*

I. *If C is A, then $R(A, B, C) = \{C, B\}$*

II. *If C is not the same as A, then:*

   a) *If C is a sentence letter, then $R(A, B, C) = \{C\}$*
   b) *If C is of the form $\neg D$, then $R(A, B, \neg D) = \{\neg D' : D' \in R(A, B, D)\}$*
   c) *If C is of the form $D \wedge E$, then $R(A, B, D \wedge E) = \{D' \wedge E' : D' \in R(A, B, D)$ and $E' \in R(A, B, E)\}$*
   d) *If C is of the form $D \vee E$, then $R(A, B, D \vee E) = \{D' \vee E' : D' \in R(A, B, D)$ and $E' \in R(A, B, E)\}$*

The System $\mathcal{LC}$:

$\mathcal{LC}$ is formulated as a set of 18 rules. Rules 1 - 15 transform propositional formulas into any equivalent form [1] using replacement and should be familiar to the reader. Rules 16 - 18 are what set this system apart and make it specific to logical contingencies. They can be thought of as implication rules, and apart from Rule 16 which is just an infinite set of axioms, they produce new logical contingencies from existing ones.

Double Negation Rule:

Rule 1: *If $E' \in R(A, \neg\neg A, E)$ then $\vdash E$ if and only if $\vdash E'$.*

Commutative Rules:

Rule 2: *If* $E' \in R(A \wedge B, B \wedge A, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 3: *If* $E' \in R(A \vee B, B \vee A, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Associative Rules:

Rule 4: *If* $E' \in R((A \wedge B) \wedge C, A \wedge (B \wedge C), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 5: *If* $E' \in R((A \vee B) \vee C, A \vee (B \vee C), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

DeMorgan's Laws:

Rule 6: *If* $E' \in R(\neg(A \wedge B), \neg A \vee \neg B, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 7: *If* $E' \in R(\neg(A \vee B), \neg A \wedge \neg B, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Distributive Rules:

Rule 8: *If* $E' \in R(A \wedge (B \vee C), (A \wedge B) \vee (A \wedge C), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 9: *If* $E' \in R(A \vee (B \wedge C), (A \vee B) \wedge (A \vee C), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Idempotent Rules:

Rule 10: *If* $E' \in R(A, A \wedge A, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 11: *If* $E' \in R(A, A \vee A, E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Identity Rules:

Rule 12: *If* $E' \in R(A, A \wedge (B \vee \neg B), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 13: *If* $E' \in R(A, A \vee (B \wedge \neg B), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Domination Rules:

Rule 14: *If* $E' \in R(A \wedge \neg A, B \wedge (A \wedge \neg A), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Rule 15: *If* $E' \in R(A \vee \neg A, B \vee (A \vee \neg A), E)$ *then* $\vdash E$ *if and only if* $\vdash E'$.

Introduction & Elimination Rules:

Rule 16: *If p is a sentence letter, then $\vdash p$.*

Rule 17: *If $\vdash A$ then $\vdash \neg A$*

Rule 18: *If $\vdash A$ and p is a sentence letter that does not occur in A, then $\vdash (A \wedge p) \vee (B \wedge \neg p)$ and $\vdash (A \wedge \neg p) \vee (B \wedge p)$, where B is any well-formed formula.*

3 Proof of Soundness

Theorem 3.1. *$\mathcal{LC}$ is sound.*

Proof. In order to prove the soundness of $\mathcal{LC}$, we will show that each of its rules preserves logical contingency.

- Rules 1-15 are standard logical equivalences and therefore preserve logical contingency.
- Rule 16 is sound because clearly every sentence letter is logically contingent.
- Rule 17 is sound because if A is a logical contingency, then $\neg A$ is also a logical contingency.
- Rule 18 is sound: Let A be a logical contingency and p be a sentence letter that does not occur in A. Then there exists a truth assignment that assigns *True* to A, and there exists a truth assignment that assigns False to A. Let τ be any arbitrary truth assignment which sets A to *True*. Then we can extend τ to set p to *True*, which sets $(A \wedge p) \vee (B \wedge \neg p)$ to *True*, where B is any well formed formula. On the other hand, let τ be any arbitrary truth assignment which sets A to *False*. Then we can extend τ to set p to *True*, which sets $(A \wedge p) \vee (B \wedge \neg p)$ to *False*. Therefore in either case, τ can be extended to retain the same value that it assigned to A. Since τ is arbitrary, this holds for all truth assignments. Therefore, since A is logically contingent so is $(A \wedge p) \vee (B \wedge \neg p)$. An analogous argument applies to $\vdash (A \wedge \neg p) \vee (B \wedge p)$, so both applications of Rule 18 result in logical contingencies, as required.

Since every rule of $\mathcal{LC}$ preserves logical contingency, the system is sound, as required.

□

4 Proof of Completeness

We begin by providing some definitions and proving three lemmas which will aid us in our proof of completeness.

Definition 4.1. *A* ***term*** *is a conjunction of distinct sentence letters or their negations in which all parentheses are associated to the left. We say that a term is an* ***even term*** *if it contains an even number of unnegated sentence letters and we say that a term is an* ***odd term*** *if it contains an odd number of unnegated sentence letters. It is possible for a term to contain both positive and negative occurrences of the same sentence letters.*

Definition 4.2. *A propositional formula A is in* ***disjunctive normal form (DNF)*** *if A is a disjunction of some number of terms in which all parentheses are associated to the left.*

Definition 4.3. *A propositional formula A over n sentence letters is in* ***perfect DNF*** *if A is in DNF and each of its terms contains exactly one occurrence of each of the n sentence letters.*

Note that all elementary truth functions over n sentence letters can be expressed as perfect DNF formulas, where each term corresponds to one of the rows on the truth table for which the function is True. For each sentence letter p that is made *True* in the row, the term will contain p, and for each sentence letter q that is made *False* in that row, the term will contain $\neg q$. For example the even function on n sentence letters is encoded by the perfect DNF formula which consists of the 2^{n-1} unique terms of n sentence letters that each contain an even number of unnegated sentence letters. The following perfect DNF formula represents the even function over the three sentence letters p, q, and r:

$$\Big(\Big(\Big(((p \wedge q) \wedge \neg r) \vee ((p \wedge \neg q) \wedge r)\Big) \vee ((\neg p \wedge q) \wedge r)\Big) \vee ((\neg p \wedge \neg q) \wedge \neg r)\Big)$$

When we describe a perfect DNF formula as encoding an elementary truth function on n sentence letters, there is an assumed ordering underlying the letters. That is, the order of the sentence letters in each term is assumed to be the same.

Definition 4.4. *Let A be a logical contingency in perfect DNF, then B is a* ***subcontingency*** *of A if and only if B is a logical contingency formed by removing some sentence letter p (and its negation) from every term in A and removing any resulting duplicate terms.*

Lemma 4.5. *If A is a logically contingent elementary truth function over $n \geq 2$ sentence letters expressed in perfect DNF and A contains no subcontingency, then A must have at least 2^{n-1} terms.*

Proof. Let A be any arbitrary logically contingent elementary truth function over $n \geq 2$ sentence letters expressed in perfect DNF such that A contains no subcontingency. Suppose for the sake of contradiction that A has fewer than 2^{n-1} terms. Removing every occurrence of some sentence letter as well as its negation from A certainly cannot increase the number of terms that A contains. The perfect DNF formula representing the tautology on $n-1$ sentence letters has exactly 2^{n-1} terms, one for every row on the truth table. It is therefore impossible to turn A into a tautology by removing every occurrence of some sentence letter as well as its negation. Since the resulting formula is not the tautology on $n-1$ sentence letters or the logically false empty perfect DNF formula (since $n \geq 2$), it is a logical contingency, which means that A contained a subcontingency. This is a contradiction. Therefore A must have at least 2^{n-1} terms. Since A was

arbitrary, we can conclude that every logically contingent elementary truth function over $n \geq 2$ sentence letters with no subcontingency must have at least 2^{n-1} terms, as required. □

Lemma 4.6. *If S is a set containing 2^{n-1} distinct n-bit binary strings such that at least one contains an even number of 1s (i.e. has even parity), and at least one contains an odd number of 1s (i.e. has odd parity), then there are at least two strings in S that differ by exactly one bit.*

Proof. (By induction on n)

Basis: For our basis, $n = 2$, so $|S| = 2$. The only possibilities are $S = \{00, 01\}$, $S = \{00, 10\}$, $S = \{11, 01\}$, and $S = \{11, 10\}$, all of which contain strings which differ by exactly one bit, so our statement holds for the base case.

Induction Hypothesis: Suppose that our statement holds for $n-1$, that is, if $|S| = 2^{n-2}$ and S contains at least one string of each parity, then at least two of the strings in S differ by exactly one bit.

Induction Step: We want to show that our statement holds for n, so $|S| = 2^{n-1}$, and S contains at least one string of each parity. Split S into two sets: take all strings in S starting with a '0', delete this first bit, and put them into set S_0. Similarly, take all strings in S starting with a '1', delete this first bit, and put them into set S_1. Since $|S_0| + |S_1| = |S|$, we know that at least one of $|S_0|$ and $|S_1|$ contains at least 2^{n-2} strings. Let us call this set S_i. Either S_i contains at least one string of each parity or it does not. If it does, then by our induction hypothesis, S_i must contain at least two strings which differ by exactly one bit, so when we add the deleted bits back onto the front of each string in S_i, they still differ by exactly one bit, and these strings came from S, so S contains two strings which differ by exactly one bit, as required. On the other hand, suppose that all of the strings in S_i have the same parity. Since there are only 2^{n-2} strings of length $n-1$ of each parity, we can conclude that $|S_i| = |S_0| = |S_1| = 2^{n-2}$. There are two sub-cases which we must consider: If all of the strings in S_0 and S_1 have the same parity, then $S_0 = S_1$, so when we replace the first bits that we deleted, we will be left with at least two strings which differ by exactly one bit, and our statement holds for n as required. If the strings in S_0 and S_1 have opposite parities, then when we replace the first bits, we will end up with all resulting 2^{n-1} strings having the same parity. However, there are only 2^{n-1} length-n strings of each parity, implying that S does not contain strings of each parity, which violates our definition of S, so we need not consider this case. Therefore, in all relevant cases, our statement holds for n.

Therefore, by induction, all sets containing 2^{n-1} distinct n-bit binary strings where at least one string of each parity is present contain at least two strings which differ by exactly one bit, as required.

□

Lemma 4.7. *If A is a logically contingent elementary truth function over $n \geq 2$ sentence letters expressed in perfect DNF such that A has no subcontingency and A has exactly 2^{n-1} terms, then A must represent the even function or the odd function over n sentence letters.*

Proof. Let A be any arbitrary logically contingent elementary truth function over $n \geq 2$ sentence letters expressed in perfect disjunctive normal form such that A contains no subcontingency and A has exactly 2^{n-1} terms. For the sake of contradiction, suppose that A neither represents the even function nor the odd function over n sentence letters. Because there are only 2^{n-1} even terms and 2^{n-1} odd terms over n sentence letters, A must contain at least one even term, and at least one odd term. Each term has length n. Therefore each term t can be mapped to an n-bit string s such that the i^{th} bit of s is 1 if the i^{th} sentence letter is unnegated, and the i^{th} bit of s is 0 if the i^{th} sentence letter is negated. A therefore encodes a set S of exactly 2^{n-1} n-bit strings. We may therefore apply Lemma 4.6, which allows us to conclude that there are at least two strings in S that differ by exactly one bit, and therefore also conclude that there are at least two terms t_1 and t_2 in A that differ by exactly one literal, call it p. Removing p from A therefore results in a formula which still has exactly 2^{n-1} terms, but t_1 and t_2 are now identical. Removing one of these duplicate terms results in an equivalent perfect DNF formula, call it A', with at most $2^{n-1} - 1$ terms. The tautology on $n-1$ sentence letters is encoded by a perfect DNF formula which consists of 2^{n-1} distinct terms. A' therefore cannot be a tautology. Since $n \geq 2$, we know that A' also cannot be the logically false empty disjunction. Therefore A' is logically contingent. This means that A has a subcontingency, namely A', which is a contradiction. Therefore A must represent the even function or the odd function over n sentence letters, as required. □

Theorem 4.8. *$\mathcal{LC}$ is complete.*

Proof. We will now show that $\mathcal{LC}$ can produce every logically contingent propositional formula and thereby prove the completeness of $\mathcal{LC}$. Given some propositional formula A, rules 1 through 15 can transform A into any equivalent propositional formula. In [1, pages 41-42], the author gives an algorithm which can be used to convert any formula into its equivalent perfect DNF. This process is entirely symmetrical, so performing the steps in reverse allows one to convert any perfect DNF formula into any logically equivalent form. To prove completeness, it therefore suffices to show that for all n, $\mathcal{LC}$ can construct all $2^{2^n} - 2$ logically contingent elementary truth functions over n sentence letters in perfect DNF. We do this by induction on the number n of sentence letters in the formula.

Basis ($n = 1$): Let p be any arbitrary sentence letter. The only DNF formulas representing logically contingent truth functions involving only the sentence letter p are p and $\neg p$. $\mathcal{LC}$ can produce p through the use of rule 16. $\mathcal{LC}$ can produce $\neg p$ from p through the use of rule 17. $\mathcal{LC}$ can therefore produce every logically contingent truth function involving only the sentence letter p. Since p is arbitrary, $\mathcal{LC}$ can produce perfect DNF formulas representing all logically contingent truth functions involving a single sentence letter. As stated above, we can then use rules 1 through 15 to produce any well-formed formula on one sentence letter.

Induction Hypothesis: Assume that $\mathcal{LC}$ can produce every logically contingent propositional formula on $n - 1$ sentence letters.

Induction Step: Let A be any logically contingent elementary truth function over n sentence letters expressed in perfect DNF. Either A has a subcontingency or A does not.

Case 1: Suppose A has a subcontingency B. Let p be a sentence letter which can be removed from A to form the subcontingency. We now have three cases. Either p occurs in A but $\neg p$ does not, or $\neg p$ occurs in A but p does not, or both p and $\neg p$ occur in A. In all three cases, we shall show how to derive A by starting with A and applying the rules of $\mathcal{LC}$ in reverse until we reach a formula which is derivable by the induction hypothesis.

Cases 1a & 1b: Consider the case when p occurs in A but $\neg p$ does not. In this case, every term of A contains p, so working backwards, we can use the commutative, associative, and distributive rules to factor the p out of A to create the formula $B \wedge p$. Again working backwards, we can use rule 13 to form $(B \wedge p) \vee (C \wedge \neg C)$, where C is any formula. We can then use rule 14 and commutativity to produce $(B \wedge p) \vee ((C \wedge \neg C) \wedge \neg p)$. We can then apply rule 18 which leaves us with B, a logical contingency over $n-1$ sentence letters. By the induction hypothesis, we know that $\mathcal{LC}$ can produce B. The same argument holds for $\neg p$. In either case, $\mathcal{LC}$ can be used to derive A by first deriving B and then performing these steps in reverse.

Case 1c: Consider the case when both p and $\neg p$ occur in A. In this case we can work in reverse and use the commutative and associative rules to rearrange all of the terms of A so that the terms containing p are next to each other and the terms containing $\neg p$ are next to each other. We can then use the distributive rules to factor each out of A to form the formula $(C \wedge p) \vee (D \wedge \neg p)$ where the subcontingency B is $C \vee D$. In this case, C and D must both be non-empty because we know that both p and $\neg p$ occur in A. C must also be a logical contingency: it is non-empty so it cannot be logically false, and if it were logically true then $C \vee D$ would also be logically true. By the induction hypothesis $\mathcal{LC}$ can produce C, so then we can apply 18 to produce $(C \wedge p) \vee (D \wedge \neg p)$, which can then be converted to its equivalent form A.

Case 2: Suppose on the other hand that A has no subcontingency. Then by Lemma 4.5, either A has 2^{n-1} terms or A has more than 2^{n-1} terms.

Case 2a: Consider the case in which A has more than 2^{n-1} terms. Since A represents a logically contingent elementary truth function f expressed in perfect DNF, the terms of A represent the rows of the truth table for which f is true. Let B be the perfect DNF formula which represents the rows of the truth table for which f is false. Then clearly A is logically equivalent to $\neg B$, and B consists of all the terms over n sentence letters which do not appear in A. Since A has more than 2^{n-1} terms and there are a total of only 2^n terms over n sentence letters, B must have fewer than 2^{n-1} terms. By Lemma 4.5, B must therefore have a subcontingency. $\mathcal{LC}$ can therefore construct B as described in Case 1. We can then apply rule 17 to form $\neg B$ from B. Rules 1 through 15 then allow us to convert $\neg B$ to its equivalent form A. $\mathcal{LC}$ can therefore produce A.

Case 2b: Consider the case in which A has exactly 2^{n-1} terms. In this case Lemma 4.7 states that A must either represent the even function or the odd function over n sentence letters. Suppose A represents the odd function over n sentence letters. Let p be an unused sentence letter. In order to build the perfect DNF formula representing the odd function on n sentence letters without using $\mathcal{LC}$, we would group all the even terms from the perfect DNF formula representing the tautology on $n-1$ sentence letters and add p to each, and we would group all of the odd terms from the perfect DNF formula and add $\neg p$ to each. The same result can be achieved using only the rules of $\mathcal{LC}$ without the use of any tautology as follows. By our induction hypothesis, $\mathcal{LC}$ can produce the perfect DNF formula representing the even function on $n-1$ sentence letters, call it E. Consider E', the result of negating every occurrence of some sentence letter in E. Clearly, E' (once any double negations have been removed by rule 1) is the perfect DNF formula representing the odd function on $n-1$ sentence letters. Therefore $E \vee E'$ is the tautology on $n-1$ sentence letters and $(E \wedge p) \vee (E' \wedge \neg p)$ is equivalent to A as argued above. The formula $(E \wedge p) \vee (E' \wedge \neg p)$ can then be converted to A using the commutative, distributive, and associative rules. As we have already noted, $\mathcal{LC}$ can produce E. We can therefore apply rule 18 to E to produce $(E \wedge p) \vee (E' \wedge \neg p)$ from E. The same argument allows us to produce the perfect DNF formula representing the even function on n sentence letters from the perfect DNF formula representing the odd function on $n-1$ sentence letters, which is also logically contingent. Therefore, in either sub-case of Case 2, $\mathcal{LC}$ can produce A.

As shown above, $\mathcal{LC}$ can produce A in all cases. Since A represents an arbitrary logically contingent elementary truth function over n sentence letters, $\mathcal{LC}$ can produce all logically contingent elementary truth functions over n sentence letters. We can apply rules 1 through 15 to any of these to produce any propositional formula on n sentence letters, as required. $\square$

5 Future Work

The system $\mathcal{LC}$ is hopefully just the first step into an interesting avenue of research into syntactically-based axiomatic proof systems for logical contingencies.

Just as there are simple, purely syntactic axiomatic proof systems for the logical truths and logical falsehoods in the propositional calculus, so are there simple, purely syntactic axiomatic proof systems for the logical truths of first-order predicate calculus. This shows that the logical truths of first-order logic are semi-decidable. Likewise there are simple, purely syntactic axiomatic proof systems for the logical falsehoods of first-order predicate calculus, similarly showing that they are semi-decidable. However, since first-order predicate calculus is not decidable [5], this means that the class of first-order logical contingencies is not even semi-decidable. That is, there can be no proof system for the logical contingencies of first-order logic.

The first-order logical contingencies therefore appear to be radically different from the logical truths and the logical falsehoods [4]. In addition to its intrinsic interest, at least part of the reason for developing a proof system for the propositional contingencies is to see to what extent such a system may be extended by adding various quantifier rules. Clearly it will not be possible to formulate strong quantifier rules such as existential quantifier introduction or existential quantifier elimination, but it may be possible to formulate weaker quantifier rules. The hope is that we will be able to obtain a tighter characterization of an interesting class of undecidable formulas.

6 Acknowledgements

We would like to thank the anonymous reviewer for his/her valuable remarks and suggestions. We would also like to thank Alasdair Urquhart for his much-appreciated input.

References

[1] S.G. Gindikin. *Algebraic Logic.* Springer, New York, 1985. Pages 41 - 42.

[2] C. Morgan. Hypothesis Generation by Machine. *Artificial Intelligence*, 2:179 – 187, 1971.

[3] C. Morgan. Sentential Calculus For Logical Falsehoods. *Notre Dame Journal of Formal Logic*, 14:347 – 353, 1973.

[4] C. Morgan. Truth, Falsehood, and Contingency in First-Order Predicate Calculus. *Notre Dame Journal of Formal Logic*, 14:536 – 542, 1973.

[5] A.M. Turing. On Computable Numbers with an Application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 42:230 – 65, 1937.