CROTTS, LARRY JOSHUA, M.S. Construction and Evaluation of a Gold Standard Syntax for Formal Logic Formulas and Systems. (2022)
Directed by Dr. Stephen R. Tate. 15 pp.

The abstract page is a required component of the thesis/dissertation. The abstract should be a brief summary of the paper, stating only the problem, procedures used, and the most significant results and conclusions. Explanations and opinions are omitted. Remember to include the necessary information regarding any multimedia components included in the document. The abstract must be approved by your advisor/committee chair.

CONSTRUCTION AND EVALUATION OF A GOLD STANDARD SYNTAX FOR FORMAL LOGIC FORMULAS AND SYSTEMS

by

Larry Joshua Crotts

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2022

Approved by

____

Committee Chair

To my Viola.

APPROVAL PAGE

This thesis written by Larry Joshua Crotts has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair ______________________________

Stephen R. Tate

Committee Members ______________________________

Chun Jiang Zhu

____

Insa R. Lawler

____

Date of Acceptance by Committee

____

Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to extend my gratitude and thanks to the esteemed Dr. Steve Tate for not only overseeing and advising this thesis, but also for being a fantastic mentor and professor throughout my time at UNC Greensboro. I sincerely appreciate Dr. Nancy Green for introducing me to the wonderful world of academic-level computer science research, as well as Dr. Insa Lawler in the Philosophy department for introducing me to the exciting adventure that is formal logic and its pedagogical impact. Their unwavering guidance, mentorship, and insight influenced me to pursue graduate school.

Outside of UNCG, I thank my parents for their love and support throughout my education. I also thank two of my best friends: Audree Logan and Andrew Matzureff, for their support and friendship from high school to now.

It also cannot go without saying that I am forever grateful for the support from my loving fiancée Viola. You never let me down.

Finally, I am deeply indebted to Mr. Tony Smith: my former Advanced Placement[1] Computer Science teacher. Without him, I would not be where I am now. Thank you for seeing (and helping me reach) my potential.

[1] https://ap.collegeboard.org/

PREFACE

The basis for this research stems from my love of teaching. When I took my first introduction to formal logic course as an undergraduate, I was taken aback by its amazing appeal and relation to computer science. From my semesters serving as a tutor/teaching assistant in the Philosophy department at UNC Greensboro, I saw many students that struggled with this material. The problems ranged from its confusing syntax, proof techniques, and esoteric notation. At that time, I thought to myself, "Why not make a tool that helps students understand it better?" Of course, that question had already been answered and deeply investigated across multiple disciplines, but I knew that there had to be more. Once I began my exploration, I quickly realized that online solvers, theorem provers, proof assistants, and similar tools do not have a ubiquitous input format, and testing their algorithms was far more cumbersome than I initially expected. This evolved into the desire for a gold standard syntax for both zeroth and first order logic systems.

Table of Contents

CHAPTER I

INTRODUCTION

I.1 Overview

Formal logic, otherwise known as classical formal logic, is a subset of philosophy that branches into other related disciplines such as computer science, statistics, mathematics, and similar sciences. Logic, however, is taught in non-science fields like communicative studies primarily to reinforce critical thinking and improve deductive skills for argumentation. Per Stanford's Encyclopedia on Classical Logic, logic is a tool used for studying correct reasoning. Its existence spawned questions ranging from its use in mathematics as an aid to disambiguate problems and proofs to considering it as an extension to natural language [**?**]. As Hatcher [**?**] states, due to the increased viewing of rhetoric and opinion versus factual knowledge in modern media across television, social media, and other such mediums, the need for strong logical thinking abilities is crucial for evaluating, analyzing, and debating arguments and claims. Hatcher, likewise, mentions that standard logical deductive forms such as methods of inference and syllogisms serve as critical components for a student's ability to determine the validity of an argument and the relation (or lack thereof) of premises to conclusions. A desire for valid and sound arguments from students constitutes and contributes to a wider adoption of formal logic classes in universities, or at the very least, the pedagogy of invalid arguments with how to refute incorrect and, sometimes egregious, contentions. Formal logic's relation to computer science, in particular, ... **talk about how we can use formal logic for mathematical proofs, Boolean logic for circuitry, set theory, etc.**

I.2 Contribution

I.3 Thesis Content

I.4 Terminology

Before we continue, we will define some terms frequently used in formal logic-related work.

Definition I.1 (Well-Formed Formula)**.**

Definition I.2 (Proposition)**.**

Definition I.3 (Proof)**.**

Definition I.4 (Theorem)**.**

CHAPTER II

RELATED WORK

In this chapter, we will discuss the related work and prior contributions to the discipline of natural deduction pedagogy, as well as efforts to modernize and increase its effectiveness for students with a weaker background in, for example, mathematics. Extending formal logic to a technological education is not a new idea—there exist many online solvers, provers, and programming languages designed to suit the needs of logic students, or those that use formal logic in some manner. We will also mention more powerful theorem provers that are aimed at experts/more experienced users.

II.1 Formal Logic Tutors

II.1.1 Propositional Logic

Propositional logic, also known as zeroth-order logic (or in other disciplines as sentence logic, sentential logic, Boolean logic, combinatorial logic, or propositional calculus), according to Hein [**?**], is a language of propositions that conform to rules. Propositional logic is comparatively simpler than first-order predicate logic described in section II.1.2—it does not use variables, constants, or quantifiers of any kind. Rather, in this language, there are four binary (two-place/two-arity) connectives: logical conjunction, logical disjunction, logical implication, and the biconditional, as well as one unary (one-place/one-arity) operator: logical negation. **Show a table and describe different notation by different authors?**

Because of the reducible nature of propositional logic to simple structures and representations, there exist plentiful online truth table generators that provide detailed and immediate feedback for users while solving problems and well-formed formulas. Further, such generators work well not only for formal logic, but also computer science, mathematics, and electrical engineering, allowing students to enter a Boolean truth value (i.e., true/false) for an operand or proposition and the computer will determine if it is valid or invalid for an arbitrary cell.

II.1.2 First-Order Logic

II.1.3 Problem/Solution Generators

Ahmed et al. wrote......... Hladik... Amendola... LLAT... Graham defines a semantic tableau...

II.2 Automatic Theorem Provers

Coq..., αleanTAP,...

II.3 Boolean Satisfiability Solver Input Formats

CHAPTER III

METHODS

In this chapter, we explain our evaluation method and metrics for assessing three publicly-available natural deduction systems against our prover. Additionally, we construct a formal definition for a standardized and uniform syntax for writing and, more importantly, testing differing logic systems and algorithms.

III.1 Evaluation of Natural Deduction Systems

III.2 Gold Standard for Formal Logic System Syntax

There are several reasons why a standardized grammar does not necessarily already exist for formal logic. Firstly, the symbols used vary widely from one subject to the next e.g., notation used in computer science may ave sublte yet important differences from philosophy-esque logics. Secondly, preexisting sources such as textbooks, websites, professors, and others all use preferential notation, leading to an amalgamation of symbols for students to use and reference which, therefore, leads students and automatic systems astray when (they) expect one syntax yet receuve something completely different. Thirdly, propositional logic learning platforms may or may not include certain operators. For example, because it is trivial to represent the biconditional (if and only if) binary operator as a conjunction of implications, it is certainly possible, albeit rather rare, to omit its symbolic representation from a language. Such omissions cause problems when evaluating formulas either automatically or by hand..... **continue here**

We propose a formal definition that aims to solve most of these problems. One component of this definition allows users to create their own logic language definition as they see fit for their situation. This language is then translatable into a gold standard format, which we will define syntactically and semantically.

III.2.1 Zeroth-Order Logic Well-Formed Formula Representation

Let $M(\mathcal{L}, w)$ be a function that "applies" the zeroth-order logic language $\mathcal{L}$ to the well-formed formula w. Let $\mathcal{L}$ be a pair (f, g) where f is an connective mapping function, and g is an atomic literal mapping function.

The bijective function f maps two sets $f\colon X \to Y$, where X is the set of input connectives defined by $\mathcal{L}$, and $Y \subseteq \{N, C, D, E, I, B, T, F\}$ is the set of output connectives defined by our grammar, where $|X| = |Y|$. N is unary logical negation, C is binary logical conjunction, D is binary logical inclusive disjunction, E is binary logical exclusive disjunction, I is binary logical implication, B is binary logical biconditional (if and only if), T is the truth function, and F is the false function. Note that the arity of any $\phi \in X$ must match the arity of its corresponding output connective in Y.

The bijective function g maps two sets $g\colon A \to B$, where A is the set of atomic literals $\psi \in A$ where ψ is an atomic literal used in w, and B is the set of output atomic formulas a_j where $j \in [1, |A|]$.

We can now define the Polish (Łukasiewicz) notation grammar G used to create a standardized notation for zeroth-order logic. This notation takes inspiration from Scheme-syntax with its parenthesization of connectives and operands. For this, we must extend the definition of typical Extended Backus-Naur Form to account for multiple-arity connectives. Thus, we introduce the notation $<x$—R$>$ to indicate that x is a variable used in the EBNF rule R, and $\{\gamma\}^x$ to denote x applications of γ. In the grammar, α is the arity of a connective.

$\langle atomic \rangle$ ::= ‘a’ (‘1’ | ‘2’ | ...)

$\langle connective \rangle$ ::= ‘N’ | ‘C’ | ‘D’ | ‘E’ | ‘I’ | ‘B’ | ‘T’ | ‘F’

$\langle \alpha$—$wff \rangle$ ::= $\langle atomic \rangle$ | ($\langle connective \rangle$ [‘ ’] $\{\langle wff \rangle\}^{\alpha}$)

Example 1

Let us take a “standard” propositional logic language $\mathcal{L}$ and a formula w. $\mathcal{L}$ consists of two functions f and g where

$$f\colon \{\supset,\, \wedge,\, \vee,\, \leftrightarrow,\, \neg\} \mapsto \{I,\, C,\, D,\, B,\, N\}$$

and

$$g\colon \{A,\, B,\, C\} \mapsto \{a_1,\, a_2,\, a_3\}$$

We will let $w = A \supset (B \leftrightarrow \neg C)$. Thus,

$$M(\mathcal{L},\, w) = (I\ a_1\ (B\ a_2\ (N\ a_3)))$$

While this representation is not as readable as w, it creates a uniform standard for testing zeroth-order logic systems. What’s more is that this application process is reversible; given $M^{-1}(\mathcal{L}',\, w')$ where $\mathcal{L}' = (f^{-1},\, g^{-1})$ and $w' = M(\mathcal{L},\, w)$, we can reproduce w using a simple stack-and-pop parsing evaluation approach (deterministic push-down automaton).

The reason we formalize the language definition is to allow different logic systems with varying syntax—some use lower-case atomic formulas, while others may restrict the alphabet to a subset. This definition allows different connective alphabets to map to the same symbol in the gold standard which provides a seamless translation to and from various host logic languages (i.e., the language of the implementing systems, assuming it does not, by default, use the gold standard internally).

Natural Deduction Extension

It is simple to extend G to support premises and conclusions using the same syntax. We can define a new function $N(\mathcal{L}, P, c)$, where $\mathcal{L}$ is the same definition as before, P is a set of well-formed formula acting as the premises of the proof, and c is the well-formed formula acting as the conclusion of the proof. Our new grammar G' is as follows:

$\langle atomic \rangle$ ::= 'a' ('1' | '2' | ...)

$\langle connective \rangle$ ::= 'N' | 'C' | 'D' | 'E' | 'I' | 'B' | 'T' | 'F'

$\langle \alpha\text{—}wff \rangle$ ::= $\langle atomic \rangle$ | ($\langle connective \rangle$ [' '] $\{\langle wff \rangle\}^{\alpha}$)

$\langle premise \rangle$ ::= ('P' $\langle wff \rangle$)

$\langle conclusion \rangle$ ::= ('H' $\langle wff \rangle$)

$\langle proof \rangle$ ::= ($\langle conclusion \rangle$ $\{\langle premise \rangle\}$)

The preceding grammar states that a premise is preceded by the letter P standing for *premise*, conclusions are preceded by H for *hence*, and a proof is a conclusion followed by zero or more premises (a proof with zero premises is a theorem).

Example 2

Let's create a proof where $P = \{\neg(C \lor D),\ D \leftrightarrow (E \lor F), \neg A \supset (C \lor F)\}$, and $c = A$. We must redefine the function g in $\mathcal{L}$ as follows:

$$g\colon \{A, C, D, E, F\} \mapsto \{a_1,\ a_2,\ a_3,\ a_4,\ a_5\}$$

Thus,

$$\begin{aligned} N(\mathcal{L}, P, c) = ((&H\ a_1) \\ &(P\ (N\ (D\ a_2\ a_3))) \\ &(P\ (B\ a_3\ (D\ a_4\ a_5))) \\ &(P\ (I\ (N\ a_1)\ (D\ a_2\ a_5)))) \end{aligned}$$

III.2.2 First-Order Logic Well-Formed Formula Representation

First-order logic is a superset of zeroth-order logic, meaning we can reuse most of our definitions from the previous section. We will, however, need to slightly redefine $\mathcal{L}$ to allow for mapping predicate definitions, constants, and variables. Further, so as to not confuse the function definitions from zeroth-order logic, we will instead use new letters to represent mapping functions.

Let $\mathcal{L}$ be a quadruple $(p,\ q,\ r,\ s)$ where p is a connective mapping function, q is a predicate mapping function, r is a constant mapping function, and s is a variable mapping function.

The bijective function p is identical to f in the sense that it maps two sets

p: $X \rightarrow Y$, where X is the set of input connectives defined by $\mathcal{L}$, and $Y \subseteq \{N, C, D, E, I, B, T, F, Z, X, V\}$ is the set of output connectives defined by our grammar, where $|X| = |Y|$. N, C, D, E, I, B, T, and F are identical in both syntactic and semantic meaning to zeroth-order logic. Z is the universal quantifier, X is the existential quantifier, and V is the identity operator. Z and X have arities dependent on the formula used, so we cannot restrict it syntactically. Identity V, on the other hand, is a binary operator.

The bijective function q maps two sets q: $A \rightarrow B$, where A is the set of predicate letters $\phi \in A$ where ϕ is a predicate letter used in the wff w, and B is the set of output predicate letters L_i where $i \in [1, |A|]$.

The bijective function r maps two sets r: $C \rightarrow D$, where C is the set of constant letters $\psi \in C$ where ψ is a constant identifier used in w, and D is the set of output constant identifiers c_i where $i \in [1, |C|]$.

Lastly, the bijective function s maps two sets s: $E \rightarrow F$, where E is the set of variable letters $\rho \in E$ where ρ is a variable identifier used in w and F is the set of output variable identifiers v_i where $i \in [1, |E|]$.

Now, similar to zeroth-order logic, we will construct the gold standard Polish notation grammar H for first-order logic. Likewise, we will utilize the previously-defined notation <x—R> to eliminate ambiguity with operator arity. One point to note is that, because identity is a special connective in first-order logic, we restrict its syntactic definition to only constants and variables. Quantifiers also have a restriction in that they must have at least one variable following their declaration, as well as a bound well-formed formula.

$\langle constant \rangle$ ::= ‘`c`’ (‘`1`’ | ‘`2`’ | ...)

$\langle variable \rangle$ ::= 'v' ('1' | '2' | ...)

$\langle literal \rangle$ ::= $\langle constant \rangle$ | $\langle variable \rangle$

$\langle predicate \rangle$ ::= 'L' ('1' | '2' | ...)

$\langle connective \rangle$ ::= 'N' | 'C' | 'D' | 'E' | 'I' | 'B' | 'T' | 'F'

$\langle identity \rangle$ ::= 'V'

$\langle quantifier \rangle$ ::= 'Z' | 'X'

$\langle \alpha$—$wff \rangle$ ::= ($\langle predicate \rangle$ {$\langle literal \rangle$})

| ($\langle connective \rangle$ [' '] $\langle wff \rangle^{\alpha}$)

| ($\langle quantifier \rangle$ $\langle variable \rangle$ {$\langle variable \rangle$} $\langle wff \rangle$)

| ($\langle identity \rangle$ $\langle literal \rangle$ [' '] $\langle literal \rangle$)

Example 3

Let us take a "standard" first-order logic language $\mathcal{L}$ and a formula w. $\mathcal{L}$ consists of the four functions p, q, r, and s where

$$p\colon \{\supset, \wedge, \vee, \leftrightarrow, \neg, \forall, \exists, =\} \mapsto \{I, C, D, B, N, Z, X, V\}$$

and

$$q\colon \{P, Q, R\} \mapsto \{L_1, L_2, L_3\}$$

and

$$r\colon \{c, d\} \mapsto \{c_1, c_2\}$$

and

$$s\colon \{x, y, z\} \mapsto \{v_1, v_2, v_3\}$$

Suppose $w = \forall x \forall y \neg Pxyc \wedge (Qcd \vee \exists z Rz)$. Thus,

$$M(\mathcal{L},\, w) = (C\ (Z\ v_1\ (Z\ v_2\ (N\ (L_1\ v_1\ v_2\ v_3))))\ (D\ (L_2\ c_1\ c_2)\ (Xv_3(L_3v_3))))$$

CHAPTER IV

RESULTS AND DISCUSSION

CHAPTER V

CONCLUSION AND FUTURE DIRECTION

References