Promoting a Common Testbed for Natural Deduction Tutoring Systems

L. JOSHUA CROTTS, Indiana University Bloomington, USA
STEPHEN R. TATE, University of North Carolina Greensboro, USA

This paper makes the case for common testbeds for evaluating educational systems, such as cognitive tutors, and fully develops a language for specifying such testbeds in a particularly challenging case, that of cognitive tutors for logic and natural deduction. As the teaching of natural deduction spans several disciplines, and notation varies widely, it is impossible for a single specification to apply across all systems, so the key contribution of this paper is defining a "super-language" that we call the gold standard which can capture the capabilities of all currently-used systems, as well as other possible future directions. Translators both to and from individual logic languages allow this to serve as both a specification language for developing standard testbeds, and as an intermediate language to enable simpler system-to-system translation. The development described here can be used as an exemplar for developing common testbeds for other educational systems.

CCS Concepts: • **Theory of computation** → **Logic**; • **Software and its engineering** → **Syntax**; • **Applied computing** → *Computer-assisted instruction.*

Additional Key Words and Phrases: logic pedagogy, natural deduction, automated testing

ACM Reference Format:
L. Joshua Crotts and Stephen R. Tate. 2022. Promoting a Common Testbed for Natural Deduction Tutoring Systems. In *6th International Conference on Education and E-Learning (ICEEL), November 23–25, 2022, Tsuru, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

When evaluating diverse systems with a common goal, it is vital to have a shared test set that can be used for comparing the systems. Shared test sets are common in many technical areas, with well-known examples in system benchmarking (the SPEC benchmarks) and software assurance (the Software Assurance Reference Dataset, or SARD, from NIST), and we believe that the field of cognitive tutors needs such standards to enable meaningful comparison of tutoring systems that are developed in education research. The work described here arose from a project to develop objective evaluation metrics for cognitive tutors in the area of natural deduction reasoning and proofs. A common dataset for evaluating dynamic logic tutors would consist of logical propositions and sets of axioms, allowing tutoring systems to be evaluated on effectiveness in settings utilizing either objective system metrics or assessment with students.

In our earlier project, we created an initial dataset along these lines, with 288 natural deduction problems in propositional and first-order logic as a first step in creating a standardized dataset [3]. A particular challenge to logic tutors, however, is that this material is taught in many contexts and many disciplines, including philosophy, mathematics, and computer science, and various communities have developed widely varying notations. This severely complicates our task, as evaluating five systems with 288 problems required manual conversion of each problem to a system-specific notation and language syntax. In this paper we propose a logic language that applies in the most general setting,

Manuscript submitted to ACM

supporting propositional and first-order logic, with automated processes to convert to and from this standard and a wide variety of alternative notations including all five systems we investigated in our other work. Our proposed language is highly flexible, not only supporting obvious differences like different notation for logical connectives and variables, but also supporting custom operator precedence and associativity. We call our flexible language the "gold standard notation," and while it is intended as an intermediate technical representation rather than a directly-usable language for students, it is particularly well-suited for providing a common dataset for evaluating and comparing natural deduction tutoring systems. It can also serve as an intermediate language for converting one representation to another, supporting $n(n-1)/2$ pairwise translators for n representations with only $2n$ translators (to and from the gold standard). This paper is derived from the first author's masters thesis work [2]

2 PRIOR WORK

The International Organization of Standards has a dedicated section to logic symbol syntax in their quantities and units for mathematics standard [5]. Alas, it only addresses a very small subset of widely-used connectives as shown in table 1. Plus, even though many other mathematical symbols are standardized and adopted in practice (e.g., set notation), logic notation never received the same level of attention from its audience.

Table 1. ISO Logic Symbols

Semantic Meaning	Operator
Logical Conjunction	$\wedge$
Logical Disjunction	$\vee$
Logical Negation	$\neg$
Logical Implication	$\Rightarrow$
Logical Equivalence	$\Leftrightarrow$
Universal Quantifier	$\forall$
Existential Quantifier	$\exists$

We could only find one attempt outside the ISO at formally standardizing propositional and first-order logic syntax, and it was for a classroom mathematics setting. In [4], Dougherty attempts to standardize logic symbols for his calculus courses. He noted that several symbols are used, sometimes erroneously, and other times in an understating context, e.g., using an implication when a biconditional is better suited. He proposes that the connectives $\rightarrow$ and $\leftrightarrow$ ought to be used when making a small claim that returns true or false, whereas $\Longrightarrow$ and $\Longleftrightarrow$ are for tautological statements. Dougherty also dislikes implicit precedence, in favor of brackets and parentheses for grouping binary connectives so as to not unnecessarily confuse students. Importantly, the premise of his paper is that standardizing logic syntax helps students clarify their arguments and better illustrate the intended idea behind a proof. The downsides are that students may often worry about what symbol to use when rather than focusing on the concepts. Additionally, standardizing logic syntax usage in one class is helpful for that individual class, but without a universal formalization, the practicality and portability is thereby limited.

Propositional logic and Boolean algebra play a central role in computer science, and some work has been done in creating a common format for specifying formulas in Boolean algebra. Cook proved that the Boolean satisfiability problem SAT is $\mathcal{NP}$-Complete [1], providing the Cook-Levin theorem used in computational complexity (Karp) reduction proofs. Boolean satisfiability answers the question, "Given a Boolean logic formula F, is there an assignment of truth values, i.e., an interpretation that makes F true?" Because this problem is $\mathcal{NP}$-Complete, there exists no known (efficient)

polynomial-time solution. Because of the usefulness of SAT with program verification, graph coloring, constraint satisfaction, artificial intelligence, electronic circuitry correctness verification, and more, the need for heuristically-fast SAT solvers was evident. A lecture by Heule and Martins [8] describes several SAT solvers including DIMACS, CaDiCaL, SAT4J, UBCSAT, and PySAT. Having a plethora of SAT solvers to choose from is not very helpful without meaningful comparisons of the trade-offs between the solvers. To test SAT solvers against one another, SAT Competitions came to light, as did the benchmark submission guidelines detailing the required input format [7]. For fair and consistent evaluation, SAT solvers that participate in this competition must use the standardized DIMACS format. Formulas are entered in conjunctive normal form where numbers represent literals/atoms. Each line designates a clause in the formula. Figure 1 shows an example of the DIMACS format alongside its logic (well-formed formula) representation. As reported by the SAT solver Varisat[1], there are several extensions and variants of DIMACS, but the SAT competitions website[2] strictly states that any deviation from the required input and output formats is unacceptable.

$$(\neg x \vee y \vee \neg z) \wedge (\neg z \vee \neg x)$$

(a) Formula Representation

```
p cnf 3 2
-1 2 -3 0
 -3 -1 0
```

(b) DIMACS Format

Fig. 1. DIMACS Format Example Input

The standard DIMACS format allows for head-to-head comparison of SAT solvers, while nothing similar exists for logic tutors and theorem provers.

3 METHODS

In prior work, we explained a methodology for comparing the efficacy of publicly-available natural deduction tutors and provers [3]. One important point of note was the overhead of manually converting formulas to a system's required schema. With four systems tested each with a different well-formed formula schema, and 288 formulas to test, this required 1152 formula conversions. Furthermore, there's the troublesome issue in that many logic symbols are not available on a standard keyboard layout, requiring either UI buttons or symbol copy-pasting. We aim to propose solutions to two problems: whether it's possible to create an intermediary "gold standard" language that others may use for evaluative testing (similar to DIMACS), and to create a framework that allows for automatic translation between one logic language, the gold standard, and to another logic language.

There are several reasons why a standardized grammar does not necessarily already exist for formal logic. Firstly, symbol usage varies widely from one subject to the next. Case in point, notation used in computer science may contain subtle yet important differences from philosophy-esque logics. Secondly, preexisting sources such as textbooks, websites, professors, and others all use preferential notation (i.e., they use what they think is correct, what they were taught, or what is otherwise preferred in their respective discipline), providing an amalgamation of symbols for students to use and reference which, therefore, leads students and automatic systems astray when expecting one syntax yet receive 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

[1] https://jix.github.io/varisat/manual/0.2.0/formats/dimacs.html
[2] http://www.satcompetition.org/

omissions cause problems when evaluating formulas either automatically or by hand due to the extended requirement of deriving an equivalent format in a language. In terms of computability, this does not pose a significant issue because any connective in zeroth-order logic can be equivalently represented by, for instance, either $\{\wedge, \neg\}$ (NAND) $\{\vee, \neg\}$ (NOR) due to their functional completeness property as proved by Post [10]. While not an issue for automated or algorithmic translation, it is inconvenient to rewrite formulas by hand to fit a restrictive system when, for instance, testing different automatic logic systems.

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 bidirectionally translatable into a gold standard format, which we will define syntactically and semantically.

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). Our language uses prefix notation for operators, which is typically not how logic is written for human work but is beneficial as an internal representation due to its disambiguation of precedence, which we will further discuss at the end of the next section. Some may argue that creating a gold standard from scratch, rather than improving upon and spreading the ISO standard (see table 1), is more trouble than it is worth. As a counterargument, we state that a gold standard allows for more than ISO currently provides via associativity and precedence definitions, as well as the relative ease of converting to and from arbitrary logic languages.

3.1 Zeroth-Order Logic Well-Formed Formula Representation

We will start with a small example and then generalize the approach to achieve a well-defined representation. Suppose we have a set of premises $P = \{(A \leftrightarrow B), \neg(C \wedge D), C, \neg B\}$ with the conclusion $c = (\neg A \wedge D)$. Converting this proof into the three systems we tested is laborious at best and is increasingly tiresome the more systems we wish to evaluate. Table 2 demonstrates the required syntax to parse an equivalent representation of w in three publicly available natural deduction systems aimed towards students: TeachingLogic [6], NaturalDeduction [9], and TAUT-Logic [11] (a further discussion of each system is found in [3]). The need for a uniform standard to rapidly test multiple systems without manual intervention is readily apparent.

Table 2. Required Syntax to Parse Proof (P, c)

Natural Deduction System	Syntax
TeachingLogic	$(p$ <=> $q)$ & -$(r$ & $s)$ & r & -q => $(-p$ & -$s)$
NaturalDeduction	$(A \equiv B), \neg(C \wedge D), C, \neg B \therefore (\neg A \wedge \neg D)$
TAUT-Logic	$(A$ implies $B)$ and $(B$ implies $A)$, not$(C$ and $D)$, C, not $B \therefore$ (not A and not D)

The most obvious differences in the notations are the representations of the connectives (the logical operators) and different standards for literals. Let $M(\mathcal{L}, w)$ be the operation that "applies" the zeroth-order logic language $\mathcal{L}$ to the well-formed formula w. Let $\mathcal{L}$ be a tuple $\langle \delta, \zeta \rangle$ where δ is a connective mapping function, and ζ is an atomic literal mapping function.

The bijective connective-mapping function δ maps two sets $\delta : X \rightarrow Y$, where X is the set of input connectives defined by $\mathcal{L}$, and $Y \subseteq \{N, C, D, E, I, B, T, F\}$ is the set of output connectives defined by our gold standard grammar, where $|X| = |Y|$. Table 3 shows the meaning of each connective in Y. Note that the arity of any connective $\phi \in X$ must match the arity of its corresponding output connective $\psi \in Y$.

Table 3. Connective Symbols for Zeroth-Order Logic Gold Standard

Semantic Meaning	Connective Symbol
Logical Negation	N
Logical Conjunction	C
Logical (Inclusive) Disjunction	D
Logical Exclusive Disjunction	E
Logical Implication	I
Logical Biconditional	B
Logical Tautology	T
Logical Falsehood	F

The surjective function ζ maps two sets $\zeta : A \rightarrow B$, where A is the set of all atomic literals $\phi \in A$ where ϕ is an atomic literal used in w, and B is the set of output atomic formulae a_j where $j \in [1, |A|]$. One property of ζ is that the mapping need not to be linear, i.e., $\phi_1 \in A$ does not necessarily have to map to $a_1 \in B$; as long as the surjective property holds, any mapping is valid. To put it another way, ζ is a non-order-preserving map. Another property is that, because ζ is not necessarily injective, two literals from A may map to the same literal in B. This results in languages that do not disambiguate between casing of atoms to still convert to the gold standard. For instance, a language may say that A and a are the same literal. Surjection allows these to both map to, say, a_1. In following examples, we assume that A is finite, but for languages which allow a countably infinite number of atomic literals, e.g., $A = \{\phi_1, \phi_2, \phi_3, ...\}$, a solution is to map A to a one-to-one set of positive integers (i.e., $\phi_n \in A \mapsto a_m \in B$ where $n, m \in \mathbb{N}^+$).

We can now define the Polish (Łukasiewicz), or prefix notation grammar G used to create a standardized notation for zeroth-order logic. This notation takes inspiration from the syntax of the Scheme programming language 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 $\{\Lambda\}^x$ to denote exactly x applications of Λ. In the grammar, α is the arity of a connective.

⟨*atomic*⟩ ::= 'a' ('1' | '2' | ...)

⟨*connective*⟩ ::= 'N' | 'C' | 'D' | 'E' | 'I' | 'B' | 'T' | 'F'

⟨α—*wff*⟩ ::= ⟨*atomic*⟩ | '('⟨*connective*⟩ [' '] {⟨*wff*⟩}$^{\alpha}$')'

We shall reiterate that our goal is to create a language pipeline that allows for easy conversion between one language $\mathcal{L}$, the unambiguous gold standard $M(\mathcal{L}, w)$, and another arbitrary language of the same class $\mathcal{L}'$. Symbolically, $\mathcal{L} \Leftrightarrow M(\mathcal{L}, w) \Leftrightarrow \mathcal{L}'$.

We will now make note of converting a gold-standard formula w' to a language $\mathcal{L}'$ such that Y', the connective set of w' is disjoint from Y, the connective set of $\mathcal{L}'$, i.e., there exists an operator in the gold standard of which there is no syntactic equivalent in the target language $\mathcal{L}'$. In this situation, $\mathcal{L}'$ must augment its definition with a pattern-matching replacement function $\mathcal{R}$. $\mathcal{R}$ should take a connective as input, and output an equivalent representation that satisfies the grammar of language $\mathcal{L}'$. As an example, suppose $w' = (B\ a_1\ a_2)$. When converting w' to a target language not

identical to the original source language (i.e., $L' \neq L$) that does not support the biconditional connective, we could define $\mathcal{R}(B) = (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$ where $\rightarrow, \wedge \in Y$, and $\phi, \psi \in \mathcal{L}'$. So, when we convert from the gold standard into a target language, any instance of the biconditional B is replaced by $\mathcal{R}$ into a recognizable format. At a minimum, any set of connectives Y for any zeroth-order logic language must be functionally complete to achieve this goal [10]. One note regarding the expansion well-formed formulas via $\mathcal{R}$ is its growth rate consequence. Namely, if the transformed formula is reflective, i.e., $(A \rightarrow B) \wedge (B \rightarrow A)$, the size grows exponentially in the number of replacements.

3.1.1 Zeroth-Order Logic Example 1. Let us take a "standard" propositional logic language $\mathcal{L}$ and a formula w. $\mathcal{L}$ consists of two functions δ and ζ where

$$\delta : \{\supset, \wedge, \vee, \leftrightarrow, \neg\} \mapsto \{I, C, D, B, N\}$$
$$\zeta : \{A, B, C, ..., Z\} \mapsto \{a_1, a_2, a_3, ..., a_{26}\}$$

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

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

This representation reads naturally from left-to-right as follows: "An implication of a_1 and a biconditional of a_2 and negated a_3". We consider all connectives as first-class functions in our definition.

While prefix notation is not as readable as the infix w, it creates a uniform standard for testing zeroth-order logic systems. What is more is that this application process is reversible; given $M^{-1}(\mathcal{L}', w')$ where $\mathcal{L}' = \langle \delta^{-1}, \zeta^{-1} \rangle$ and $w' = M(\mathcal{L}, w)$, we can reproduce w using the inverse functions δ^{-1}, ζ^{-1} and a syntax tree as shown in figure 2.

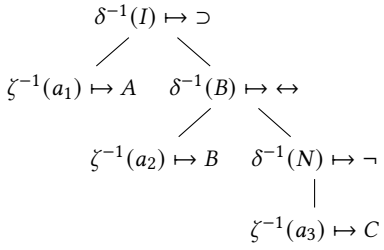

Fig. 2. Syntax tree parsing representation of w' to w

3.1.2 Zeroth-Order Logic Example 2. Quine's syntax in [12] for propositional logic is slightly different from modern variants. Specifically, his use of dots and colons removes superfluous parentheses when defining operator precedence. Largely, we will ignore this notation in favor of his parenthesized form. In addition, Quine uses an empty string ε to represent conjunction (e.g., S_1S_2 represents a conjunction between two well-formed formulas S_1 and S_2). Finally, negations on a single atom p are condensed with a vertical overbar $\overline{p}$. To compensate for the digital representation, we will keep the negation in front of the atom (e.g., $-S_1$ where S_1 is a well-formed formula). Now, we define $\mathcal{L}$ with functions δ and ζ where

$$\delta : \{\supset, \varepsilon, \vee, \equiv, -\} \mapsto \{I, C, D, B, N\}$$
$$\zeta : \{p, q, r, s, ..., z\} \mapsto \{a_1, a_2, a_3, a_4, ..., a_{11}\}$$

We will let $w = -((p \vee q)(-r \vee s)) \supset (-(p \vee q)s)$. Thus,

$$M(\mathcal{L}, w) = (I\ (C\ (N\ (D\ a_1\ a_2))\ (D\ (N\ a_3)\ a_4))\ (C\ (N\ a_1\ a_2)\ a_4))$$

This representation reads as "An implication where the left-hand side is a conjunction between a negated disjunction of a_1 and a_2, and a disjunction of negated a_3 and a_4. The right-hand side of the implication is a conjunction between a negated disjunction of a_1 and a_2, and a_4."

We will now define a precedence mapping function for the incoming formula Γ. By enforcing prefix notation in the gold standard, we no longer have to deal with the inherent complexities of operator precedence present in the commonly-used infix notation.

Let Γ be an injective function that maps the set of connectives δ to $\mathbb{N}$, namely $\delta \mapsto \mathbb{N}$. Γ is designed to give connectives in δ a priority level, where the closer its mapped natural number is to zero, the higher its priority. We define priority as the precedence of a connective. When a system defines Γ, it implies that any ambiguous well-formed formula in its corresponding language is parsable without parenthesization. Γ, as an algorithm, automatically adds parentheses to disambiguate the formula.

3.1.3 Precedence Mapping Example. We will use the same functions δ and ζ from the first propositional logic example in section 3.1.1. We will also define Γ as

$$\Gamma : \{\supset, \wedge, \vee, \leftrightarrow, \neg\} \mapsto \{3, 1, 2, 4, 0\}$$

Now, suppose $w = A \rightarrow \neg B \rightarrow C \wedge \neg A$. The precedence function parenthesizes/disambiguates w to $((A \rightarrow \neg B) \rightarrow (C \wedge \neg A))$, which is then converted into the gold standard as

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

Since we consider Γ to be optional (opting for a default precedence of logical negation, logical conjunction, logical disjunction, logical implication, then logical biconditional), a system without a defined Γ should, optimally, output a warning when it parses an ambiguous well-formed formula. Defining Γ allows for any ambiguous formula to be converted into one that is unambiguous, as we previously stated, and also allows for "custom precedence" levels (i.e., if we want to bind logical disjunction higher than logical conjunction, it is trivial to do so).

Finally, some may question the associativity of connectives. We assume that all operators are left-associative, similar to traditional addition, subtraction, multiplication, and division. For those who wish to not always strictly assume left-associativity for connectives, we will now define the associativity function γ.

Let γ be a surjective function that maps the set of binary connectives $X_B \subseteq X$ to the set $\{L, R\}$, where L and R designate left and right-associativity respectively.

3.1.4 Associativity Mapping Example. We will use the same functions δ and ζ from the previous precedence mapping function example. In addition, we will define γ as

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

$$\zeta : \{A, B, C, ..., Z\} \mapsto \{a_1, a_2, a_3, ..., a_{26}\}$$

$$\gamma : \{\supset, \wedge, \vee, \leftrightarrow\} \mapsto \{L, R\}$$

Where $\forall C \in X_B, \gamma(C) = L$ (i.e., every binary connective is left-associative). Now, suppose $w = (A \rightarrow B \rightarrow C) \wedge (\neg A \wedge \neg B)$. The associativity function disambiguates w to $((A \rightarrow B) \rightarrow C) \wedge (\neg A \wedge \neg B)$. This is converted into the gold standard as

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

3.1.5 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:

⟨*atomic*⟩ ::= 'a' ('1' | '2' | ...)

⟨*connective*⟩ ::= 'N' | 'C' | 'D' | 'E' | 'I' | 'B' | 'T' | 'F'

⟨α—*wff*⟩ ::= ⟨*atomic*⟩ | '('⟨*connective*⟩ [' '] {⟨*wff*⟩}$^{\alpha}$')'

⟨*premise*⟩ ::= '('P' ⟨*wff*⟩')'

⟨*conclusion*⟩ ::= '('H' ⟨*wff*⟩')'

⟨*proof*⟩ ::= '('⟨*conclusion*⟩ {⟨*premise*⟩}')'

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).

3.1.6 Natural Deduction Example. Let us create a proof where $P = \{\neg(C \vee D),\ D \leftrightarrow (E \vee F), \neg A \supset (C \vee F)\}$, and $c = A$. We will, again, use ζ from section 3.1.1. Therefore,

$$\begin{aligned} N(\mathcal{L}, P, c) = ((H\ a_1) \\ &(P\ (N\ (D\ a_3\ a_4))) \\ &(P\ (B\ a_4\ (D\ a_5\ a_6))) \\ &(P\ (I\ (N\ a_1)\ (D\ a_3\ a_6)))) \end{aligned}$$

We read this as "Hence a_1 if P_1 is true and P_2 is true and P_3 is true", where P_1, P_2, and P_3 are the individual premises that comprise the argument. This style largely resembles the way we write Prolog (conditional) rules.

3.2 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 unique to first-order logic semantics.

Let $\mathcal{M}(\mathcal{L}, w)$ be an operation that applies the first-order gold standard to a logic language $\mathcal{L}$ and a well-formed formula w. Let $\mathcal{L}$ be a quadruple $\langle \delta, \varsigma, \chi, \eta \rangle$ where δ is a connective mapping function, ς is a predicate mapping function, χ is a constant mapping function, and η is a variable mapping function. For first-order logic, we slightly modify δ from its zeroth-order definition in the sense that it now maps two sets $\delta : X \rightarrow Y$, 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 as referenced in table 3. Table 4 shows the new operators added by first-order logic. Note that Z

and X have arities dependent on the formula used, so we cannot restrict it syntactically. Identity V, on the other hand, is a special predicate for connecting constants and variables. To simplify its syntactic usage, we will include V in δ definitions for first-order logic.

Table 4. Connective Symbols for First-Order Logic Gold Standard

Semantic Meaning	Connective Symbol
Universal Quantifier	Z
Existential Quantifier	X
Identity	V

The bijective function ς maps two sets $\varsigma : A \rightarrow B$, 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 χ maps two sets $\chi : C \rightarrow D$, 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 η maps two sets $\eta : E \rightarrow F$, 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 $i \in [1, |E|]$.

Like the atomic literal mapping function ζ from zeroth-order logic, ς, χ, and η must define every predicate letter, constant letter, and variable letter respectively supported by their language to represent a valid mapping.

Now, similar to zeroth-order logic, we will construct the gold standard Polish notation grammar $\mathcal{G}$ for first-order logic. Likewise, we will utilize the previously-defined notation <x—R> to eliminate ambiguity with operator arity. Two points to note are that, because identity is a special connective in first-order logic, we restrict its syntactic definition to only constants and variables. Additionally, quantifiers have a restriction in that they must bind one variable following their declaration, as well as a bound well-formed formula. Finally, all predicates must have at least one constant or variable, as a predicate with no terms, i.e., a zero arity predicate is an absurdity in first-order logic.

⟨*constant*⟩ ::= 'c' ('1' | '2' | ...)

⟨*variable*⟩ ::= 'v' ('1' | '2' | ...)

⟨*literal*⟩ ::= ⟨*constant*⟩ | ⟨*variable*⟩

⟨*predicate*⟩ ::= 'L' ('1' | '2' | ...)

⟨*connective*⟩ ::= 'N' | 'C' | 'D' | 'E' | 'I' | 'B' | 'T' | 'F'

⟨*identity*⟩ ::= 'V'

⟨*quantifier*⟩ ::= 'Z' | 'X'

⟨α—*wff*⟩ ::= '('⟨*predicate*⟩ ⟨*literal*⟩ {⟨*literal*⟩}')'
| '('⟨*connective*⟩ [' '] ⟨*wff*⟩$^{\alpha}$')'
| '('⟨*quantifier*⟩ ⟨*variable*⟩ ⟨*wff*⟩')'
| '('⟨*identity*⟩ ⟨*literal*⟩ [' '] ⟨*literal*⟩')'

The above grammar states the following rules: constants use the prefix c with a uniquely identifying successive integer. Variables follow the same rules with the exception that variables are prefixed by v. Literals are either constants or variables. Predicates, likewise, use the same identifying principle except that they use L as a prefix. As aforesaid, the

identity and quantifier operators are special in the wff definition, in that a quantifier binds a variable to a well-formed formula, and an identifier wraps two literals together. In addition to these special cases, predicates bind at least one literal, and a connective contains as many well-formed formula operands as its arity requires.

3.2.1 First-Order Logic Example. We will, once again, use a "standard" first-order logic language $\mathcal{L}$ and a formula w. $\mathcal{L}$ is a quadruple of the four functions δ, ς, χ, and η where

$$\delta: \{\supset, \wedge, \vee, \leftrightarrow, \neg, \forall, \exists, =\} \mapsto \{I, C, D, B, N, Z, X, V\}$$
$$\varsigma: \{P, Q, R, ..., Z\} \mapsto \{L_1, L_2, L_3, ..., L_{11}\}$$
$$\chi: \{a, b, ..., t\} \mapsto \{c_1, c_2, ..., c_{20}\}$$
$$\eta: \{u, v, ..., z\} \mapsto \{v_1, v_2, ..., v_6\}$$

Suppose $w = \forall x \forall y \neg P x y c \wedge (Q c d \vee \exists z R z)$. Thus,

$$\mathcal{M}(\mathcal{L}, w) = (C\ (Z\ v_4\ (Z\ v_5\ (N\ (L_1\ v_4\ v_5\ c_3))))$$
$$(D\ (L_2\ c_3\ c_4)\ (X\ v_6\ (L_3\ v_6))))$$

We read this as "A conjunction of a universal quantifier that binds v_4, a universal quantifier binding v_5, bound to the negation of $L_1\ v_4\ v_5\ c_3$ and a disjunction of the following: $L_2\ c_3\ c_4$ and an existential quantifier which binds v_6, bound to $L_3\ v_6$."

4 DISCUSSION

Our broad idea behind implementing the gold standard as an intermediary logic language is to test real-world natural deduction systems with varying syntax — not semantics; other functionalities are presented to lay the groundwork for future exploration of non-standard representations. The current work allows for fast testability.

4.1 Limitations

While we believe that the gold standard has a lot of potential, its current implementation has a few limitations that we will now list.

Firstly, it is not possible to transform non-infix logic languages into the gold standard and vice-versa. A potential solution may include providing formal inductive definitions of a logic language. We feel, however, that this is largely a theoretical limitation.

It is also not possible to transform the gold standard into non-standard languages with custom precedence or associativity rules (the forward direction, though, is possible as previously discussed). For example, we cannot go from the gold standard to a language $\mathcal{L}'$ such that $\vee_l \geq \wedge_l$ where l is the provided precedence level as a positive integer.

Transforming between a logic language $\mathcal{L}$ and the gold standard removes implicit ambiguity/precedence/associativity as defined by $\mathcal{L}$. This consequently means a formula sent to the gold standard may not always equate the formula received from a gold standard translation. We feel as though an unambiguous formula reduces possible errors and confusion. As such, we prioritize a guaranteed semantic translation over a guaranteed syntactic translation.

Direct translation of natural deduction proofs is currently unsupported because there must exist a mechanism, e.g., a function, to automatically identify premises and the conclusion in the source language (existing solution denotes a

premise set P where every $p \in P$ is a premise, and c is the sole conclusion where P may be the empty set). This means that all premises in P must be individually converted to the gold standard.

5 CONCLUSION AND FUTURE WORK

We have presented a gold standard syntax for both zeroth and first-order logics. In future work we aim to test more natural deduction systems than just those presented in [3]. We also plan to publish a system for performing said transformations to and from the gold standard alongside a database of gold standard formulas for others to use as a testbench. Alongside the system, we wish to publish an ANTLR [3] grammar for the gold standard as presented in this paper that others may contribute to and use in various other programming languages.

REFERENCES

[1] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (Shaker Heights, Ohio, USA) *(STOC '71)*. Association for Computing Machinery, New York, NY, USA, 151–158. https://doi.org/10.1145/800157.805047
[2] Larry Joshua Crotts. 2022. *Construction and Evaluation of a Gold Standard Syntax for Formal Logic Formulas and Systems.* Master's thesis. University of North Carolina Greensboro, 1400 Spring Garden St., Greensboro, NC 27412.
[3] L. Joshua Crotts and Stephen R. Tate. 2022. Comparison of Natural Deduction Theorem Provers used in Electronic Tutoring Systems. (2022). In preparation.
[4] Michael M. Dougherty. 2007. A Standardized Logic Notation for Everyday Classroom Use.
[5] International Organization for Standardization. 2010. Quantities and units - Part 2: Mathematical signs and symbols to be used in the natural sciences and technology.
[6] Grenoble Computer Science Laboratory. 2021. Natural Deduction. http://teachinglogic.liglab.fr/DN/
[7] Marijn Heule, Markus Iser, Matti Jarvisalo, Martin Suda, and Tomáš Balyo. 2011. SAT Competition 2011: Benchmark Submission Guidelines.
[8] Marijn J.H. Heule and Ruben Martins. 2020. SAT and SMT Solvers in Practice.
[9] Jukka Häkkinen. 2017. NaturalDeduction. http://naturaldeduction.org/
[10] Emil Leon Post. 1941. *The Two-Valued Iterative Systems of Mathematical Logic.* London: Oxford University PRess.
[11] Ariel Roffé. 2022. Propositional Logic - Natural Deduction. https://www.taut-logic.com/
[12] William van Orman Quine. 1950. *Methods of Logic.* Harvard University Press.

[3] https://github.com/antlr/antlr4