Outline of a symbolic and PAC-learning approach to instruct a Turing-complete system in natural language

Anonymous submission

Abstract

A machine's capacity of interacting in natural language and processing in its full Turing-complete computing power is our ultimate expectation in natural language processing (NLP). In this paper, we introduce Froglingo, a Turing-complete language system, that is a symbolic approach with a Probably Approximately Correct (PAC) learnability toward NLP, knowledge representation, as well as software development. We discuss a semantic parser that is a side-product of learning from sample utterance, i.e., the parser is mainly comprised of a set of syntactical parsing and semantic mapping templates that are accumulated when the sample utterance are received and processed. Froglingo's PAC learnability guarantees a convergence of such accumulation processes to an ultimate machine that would interact with others in natural language. Froglingo's symbolically rigorous reasoning guarantees noise free, i.e., the machine never say yes when the answer is no. We further propose a user interface for Froglingo that accepts natural language beside its own system language. At an early stage, Froglingo mostly accepts Froglingo expressions to construct data. Later, Froglingo can continue data construction mostly using text.

1. Introduction

Statistical based machine learning approaches toward natural language processing (NLP), particularly large language models (LLMs), have been gratefully and deeply impacting our daily life. Our ultimate expectation on NLP, however, is a machine's real-time capacity of both interacting with surroundings in natural language as if it was a person and reasoning in the full power of Turing machine, e.g., executing Turing-complete expressions for specific software applications. We almost have had the former but not quite yet as statistical machine learning doesn't have a guarantee on a controllable precision, i.e., a learned program (e.g., LLM) doesn't converges to its targeted function for a designed precision. We have the latter on the infrastructure of today's information technologies that are built on the top of programming languages, but not on the learned programs, e.g., LLMs.

Integrating symbolic computing with statistical machine learning, i.e., placing a representation of Turing-complete expressions into the same Eucliean space where a LLM is running, called knowledge (or simply a graph) injection or embedding, has been a challenge practically and theoretically (Apidianaki 2022; Berant, Dagan, and Goldberger 2012; Bordes et al. 2011; Cai, Zheng, and Chang 2017; Nickel, Tresp, and Kriegel 2012; Seshadhri et al. 2020). Particularly, Turing-complete expressions that themselves are not PAC learnable (Gold 1967) are too complex to be injected (bent, precisely converted) to a representation in an Euclidean space with a further dimensionality reduction without losing information.

The Enterprise-Participant (EP) data model is a database language and equivalently a data structure. It represents a class of bounded functions that is both PAC learnable (Xu 2025) and semantically equivalent to Turing machine, i.e., EP can do whatever a Turing machine can do within a given time and space. In other words, although EP cannot finitely express Turing-complete expressions, e.g., $f(x) = x + 1$, as a programming language can, it can record and process a finitely measurable properties of the Turing-complete expressions, i.e., $\{(0, 1), (1, 2), ..., (n, n + 1)\}$ for a $n > 0$ (Xu 2017). EP drives the PAC learnability, or informally say EP databases (expressions) are PAC learnable, because an algorithm exists to take a finite set of randomly selected sample EP expressions and to automatically construct an EP database that supports more meaningful EP expressions than the sample expressions themselves. While statistical machine learning has its unique applications like pattern recognition, the EP driven learnability persists on keeping the realizability assumption, i.e., guarantees a learned EP database converges to its targeted database for a designed precision. EP is a symbolic computing because it is a formal language that rigorously computes bounded functions. Although EP is not as expressive as a Turing-complete programming language, EP can be used to construct certain applications like NLP, because of its PAC learnability, that are challenges to programming languages.

Having an infinite domain and a finite co-domain is a unique characteristics of a bounded function. An directed and cyclical graph expressed in an EP database, for example, is interpreted as a bounded function, where the infinite set of all possible paths is the domain and the finite set of the vertexes is the co-domain. This gives us an opportunity to represent knowledge, including natural language (NL), in an EP database, and further map NL utterance to a machine readable form of knowledge. We can say a set

of Turing-complete expressions exist to represent knowledge. Practically however it is impossible because knowledge is constantly changing and unnecessary to construct a program that represents such a function because knowledge is a collection of syntactical and semantic phenomena including intertwined text, (para)phrases, coreferences, sentences, objects and the relationships among the objects in the real-world. Then we make an assumption:

Assumption 1. *knowledge is isomorphic to a bounded function.*

Human beings roughly view the world with a finite number of entities (in a correspondence to a finite co-domain in EP) but communicate with each other with infinitely possible utterance (in a correspondence to an infinite domain in EP). Given this assumption, we say that we can accumulate knowledge into an EP database which will eventually converge to an ultimate bounded function interpreting the knowledge.

In the assumption, knowledge is referred to everything, including Turing-complete Froglingo expressions that are in the form of EP expressions but evaluated to be a semanticaly bottom value $null$ in EP. The only exception is that knowledge doesn't include the Turing machine with infinite type and space, i.e., knowledge doesn't have non-finitely representable outputs of Turing-machine computing process with the hypothesis of infinite time and space (or an infinite length of utterance is not part of NL).

In this paper, we introduce Froglingo, a Turing-complete language system that extends the type and variable free EP language system with types and variables. Froglingo is implemented with the function of a semantic parser to parses utterance to syntactical structures and maps the syntactical structures to the utterance's meanings. When receiving a sample utterance, Froglingo also receives parsing and mapping instructions from external sources like developers for the given sample. At the earlier stage parsing and mapping instructions are given in Froglingo system language, later more in natural language, and eventually no longer needed to process continuous utterance (Section 8).

With the approach of accumulating templates through processing sample utterance, the parser has a controllable precision that is aimed to be eventually as high as what human being has (Section 5 and 6).

Processing sample utterance is also a process of accumulating knowledge in Froglingo. Accommodating knowledge from various domains together is another critical ability a true NLP system must support all kinds of applications. In Section 7, we discuss such an ability Froglingo has.

As a Turing-equivalent language system, Froglingo is actually a typed system for EP, i.e., a set of parsing and mapping templates actually represents a bounded function, a restricted subset of the bounded function an EP database can inventory without types. Therefore, the parsing and mapping templates are also PAC learnable, a mathematical foundation to support the discussion in Section 5 and 6. A Froglingo expressions can be Turing-complete. We say they can be generated through utterance instructions because they are in the form of EP expressions. All of these are discussed in Appendix B to D that gives proofs to the conclusions excerpted in Section 3, 6, and 7.

In Section 2, we give an high-level overview of the EP data model. See (Xu 2017; Xu, Zhang, and Gao 2010) for more information. In Section 3, we briefly introduce Froglingo and a formal discussion is provided in Appendix B. In Section 4, we use examples to demonstrate how the learnability is applied to natural language processing (NLP). Appendix A also gives related work.

2. The EP data model and transitive relations

Identifiers are the most basic building blocks in EP. Like in programming languages, we can choose alphanumeric tokens as identifiers, such as $abc123$, $\_abc$, and more often we take words from a natural language vocabulary as identifiers, such as $hello$, $John$, $sport$, law, and $person$. The entire set of identifiers, in a correspondence to the set of integers, is denoted as $\mathbf{F}$.

An EP term, or briefly term, is either an identifier or an application, i.e., using $\mathbf{E}$ to denote the entire set of terms, in a correspondence to the set of integers, we have:

$$m \in \mathbf{F} \implies m \in \mathbf{E}$$

$$m, n \in \mathbf{E} \implies (m\ n) \in \mathbf{E}$$

For example, $x\ x, (a\ b\ c)\ (d\ e\ a\ (d\ t\ a))$ are legitimate terms where $x, a, b, c, d, e, t \in \mathbf{F}$. By terms alone, we can represent containment relationships. For example, the hierarchical structure of geographical locations can be expressed: $Massachusetts\ Boston\ Somerville$ and $Manhattan\ (Wall\ Street)$.

The containment relationships among subterms of a given term are tree-structured. The first kind is the relation of a term being contained within one of its leftmost subterms, denoted as {+. For example, we have: $Massachusetts\ Boston\ \overline{Somerville}$ {+ $Massachusetts\ Boston$ {+ $Massachusetts$. The second kind is the relation of a term being contained within one of its rightmost subterms, denoted as {-. For example, we have $Manhattan\ (Wall\ Street)$ {- $(\overline{Wall\ Street})$ {- $Street$. Further the relations {+ and {- are extended to have tree-structured transitive relations {=+ and {=- respectively.

An EP database, or briefly database and denoted as D, is a finite set of terms and a finite set of assignments such that for each assignment $p := q \in D$, where $p, q \in \mathbf{E}$, the constraints are met: 1) p has only one assigner, i.e., $p := q$ and $p := q' \in D \implies q \equiv q'$; 2) a proper subterm of p cannot be an assignee, i.e., $p := q \in D \implies \forall x \in SUB^{+}(p)\ [\forall m \in \mathbf{E}\ [x := m \notin D]]$; and 3) q can not be an assignee, i.e., $p := q \in D \implies \forall a \in \mathbf{E}\ [q := a \notin D]$.

Here are a few sample databases:

- $\{a\ b := b; b\ a := a; b\ c := c\}$, for a directed and cyclical graph with connections from a to b, from b to a, and from b to c.
- $\{College\ John\ major := College\ math;$ $College\ math\ math100\ (College\ John)\ grade := A\}$, for a college administration database.

- $\{Jessie\ (verB\ drove)\ home :=$
  $Jessie\ (verB\ came)\ home\ (preP\ by\ car)\}$, for an English paraphrase presented in EP.

The examples above tell us that we use terms to represent real world entities such as a person $John$ and a college $College$, a sentence structure like the last example above, and relationships among entities such as Jessie's $home$ is related to $Jessie$ by the verbs $verB\ drove$. We further use assignments to relate two sentences together such as the last example database says that the assignee is meant to be the assigner.

A database D has a finite set of EP normal forms, denoted as $NF(D)$. A term $n \in \mathbf{E}$ is an normal form if and only if

1. n is $null$, i.e, $n \equiv null$, where $null$ is a special identifier in $\mathbf{F}$, or
2. n is a term in D and not an assignee, i.e., $n \in D$ and $\forall b \in \mathbf{E}\ [n := b \notin D]$.

Given a database D, we have one-step reduction rules, denoted as $\Rightarrow$:

1. An assignee is reduced to the assigner, i.e., $a := b \in D \implies a \Rightarrow b$
2. An identifier not in the database is reduced to $null$, i.e., $a \in \mathbf{F}, a \notin D \implies a \Rightarrow null$
3. If a and b are normal forms and $a\ b \notin D$, then $a\ b$ is reduced to $null$, i.e., $a, b \in NF(D), a\ b \notin D \implies a\ b \Rightarrow null$
4. $null\ a \implies null$ for any $a \in \mathbf{E}$
5. $a \Rightarrow a', \implies C[a] \Rightarrow C[a']$, where $C[\ ]$ denotes a context, i.e., given any term $e \in \mathbf{E}$, we have $C[e] \in \mathbf{E}$ and e is a subterm in $C[e]$.

Let $a \Rightarrow a_0, a_0 \Rightarrow a_1, \ldots, a_{n-1} \Rightarrow a_n$ for a number $n \in \mathbf{N}$. We say that a is effectively, i.e., in finite steps, reduced to a_n, denoted as $a \rightarrow_D a_n$, and further we say that a and a_n are equal, denoted as $a == a_n$.

A term a has a normal form b if b is in normal form and $a \rightarrow_D b$.

Here are a few sample reductions to their normal forms under the example databases provided at the end of Section 2: $a\ b\ a \rightarrow_D a$, $a\ b\ c \rightarrow_D c$, $c\ b \rightarrow_D null$, $college\ John\ major\ math100\ (college\ John)\ grade \rightarrow_D A$, and $Jessie\ (verB\ drove)\ home \rightarrow_D Jessie\ (verB\ came)\ home\ (preP\ by\ car)$.

Any term $m \in \mathbf{E}$ has one and only one normal form and the reduction system is strongly normalizing, i.e., there is another term $n \in NF(D)$ such that $m \rightarrow_D n$. The set of all the normal forms $NF(D)$ is finite, i.e., $|NF(D)| \leq s$ for a given $s \in \mathbf{N}$.

An EP database is interpreted as a bounded function that is recursive and has a finite co-domain while an infinite domain. Let $f : X \rightarrow Y$ be a function, where X has an arbitrary number of objects and Y a finite number of objects, let $A \subseteq X$ and a be a special object in Y, i.e., $a \in Y$, and further let

$$f(x) = b, \text{ where } b \in Y \text{ and } b \neq a, \quad \text{if } x \in A$$
$$= a \quad \text{if } x \in X \backslash A.$$

Then we say f is bounded, particularly when $A \equiv X$. When A is finite, i.e., $A \subset X$, we say f has a finite support, or simply we say f is finite. A finite function is bounded, but a bounded function may not be finite as it potentially has an infinite domain X. Further, a bounded function is always recursive, i.e., the computation on $f(x)$ terminates and $f(x) \in Y$ for any $x \in X$.

By saying a function being bounded, we mean that under a given EP database D, potentially an arbitrary number of EP expressions are meaningful. For example, the EP database defining a directed cyclic graph earlier has infinite paths, i.e., the terms $a\ b$, $b\ a$, $a\ b\ a$, ..., $a\ b ... a\ b$ will be reduced to non $null$ value. Even a path is not cyclic, we also call it derivable information. For example, we have $a\ b\ c \rightarrow_D c$. These derivable (or later called fresh) information from an EP database is the source of the PAC learnability of a class of bounded functions.

From the equality relation $==$, additional pre-ordering relations are available. One of those relations, denoted as (=+, can be used to express if there is a path or cycle between two vertexes. For example, we have a (=+ $c \rightarrow_D true$ to verify a path from a to c, a (=+ $b\ and\ b$ (=+ $a \rightarrow_D true$ to verify a cycle between a and b, and c (=+ $a \rightarrow_D false$ to verify there is no path from c to a.

Like the rationale data model, EP has a set-oriented operations using the operators $select$ and $there\_is$. For example, the expression $select\ \$x\ where\ \$x$ {=+ $Massachusetts$ would retrieve all the terms, including $Massachusetts$, $Massachusetts\ Boston$, and $Massachusetts\ Boston\ Somerville$, that have $Massachusetts$ as a leftmost subterm. The expression $there\_is\ \$x\ where\ c$ (=+ $\$x$ would inquire if there is a vertex that has a direct connection from c, which would be responded with the answer of $false$ based on the sample graph database provided earlier.

3. Froglingo and more transitive relations

Like in other programming languages, Froglingo has built-in types $integer$, $real$, $string$, $date$, and etc. as well as variables, that makes Froglingo a Turing-equivalent language. For example, the factorial function can be expressed as $fac\ \$n\ :\ [\$n\ isA\ integer\ and\ \$n \geq 0]\ :=\ (\$n \times (fac\ (\$n - 1)))$, where n, an identifier, preceded with \$, is a variable, and isA is a built-in operator for the transitive relation: if $a\ isA\ b$ and $b\ isA\ c$, then $a\ isA\ c$. (Note that the operator isA is different from the tree-structure relation operator {+, and it can only be supported by Froglingo because EP is type free.)

We are more interested in user-defined types because we can infer among types along with the operator isA. A type is in the form of an EP term as well, for example $car, vehicle$, and $person$. A type and an object instance are differentiated by different built-in commands: $schema$ and $create$ respectively. For example, we can have $schema\ vehicle$ and $schema\ person$ to declare types $vehicle$ and $person$, and $create\ joe$ to declare an object instance joe. We can further use isA to associate a type with another type, or a type with an instance. For example: $schema\ car\ isA\ vehicle$ and

$create\ joe\ isA\ person$. This definition in a database would allow us to infer that car and $vehicle$ are paraphrases in the text: "John just bought a new car. The vehicle is powerful".

Inferences among sentences are also supported in Froglingo. For example, we can specify two rules (templates) when $action$ had been declared as a type: $improvement\ isA\ action$, and $person\ (verB\ paint)\ house\ isA\ improvement$. When we have an instance: $joe\ (verB\ painted)\ ((coreF\ his)\ house)$, where the built-in $coreF$ abbreviates "coreference", we can infer the equivalence while $paint$ and $improvement$ are not paraphrases: $joe\ (verB\ improved)\ ((coreF\ his)\ house)$.

Many inferences in the real world are not based on rules from common knowledge but rather on individual instances. Froglingo supports a sequence of terms as an assigner. Froglingo uses assignments with a sequence of terms as an assigner to support such inferences. For example:

$John\ (verB\ took)\ (delimiT\ a)\ vacation\ :=$
$John\ (verB\ visited)\ Jen,$
$John\ (verB\ spent)\ ((delimiT\ a)\ day)$
$((preP\ on)\ beach);$
$John\ (verB\ visited)\ Jen :=$
$John\ (verB\ gave)\ Jen\ ((delimiT\ a)\ gift),$
$John\ (verB\ had)\ dinner\ ((preP\ (together\ with))\ Jen);$

An assignee above is an abstraction and the corresponding assigner is a sequence of sub actions that give the details of the abstraction. When an abstraction is equivalent to its sub actions, we can infer from the abstraction to a specific action, for example, from the assignments above, one can infer "John visited Jen" from "John took a vacation", and inversely we can speculate: "John took a vacation" when one hears "A gentleman gave Jen a gift". These inferences are also called predictions in the PAC learnability.

In summary, we use types to represent real world categories such as $person$. An identifier that is used as a type is no longer an EP term representing an entity instance. An EP term can also represent other part of speeches such as verbs like $verB\ drove$, prepositions like $preP\ at$, adjectives like $adJ\ happy$, and etc.). Variables are placeholders for nouns, numbers, and strings.

The identifiers representing Turing-complete expressions, such as fac for the factorial function, are restrictively not EP terms as well. They were called EP constants, denoted as a set **C** in (Xu 2017). Applying a constant to an arbitrary EP term always yields $null$, e.g., $fac\ 3\ \rightarrow_D\ null$ where $fac\ 3$ is a legitimate EP term in (Xu 2017). In Froglingo, however, this is no longer a case, e.g., $fac\ 3\ \rightarrow_F\ 6$, where $\rightarrow_F$ is denoted a reduction not under EP but under Froglingo. The constant set of **C** in EP is the mathematical ground for Turing-complete expressions to be constructed through learning because these expressions are EP expressions equivalent to $null$ in EP:

Proposition 1. *Froglingo expressions are in the form of EP expressions.*

When not taking a built-in type like $integer$, a variable in Froglingo can only take instances from a finite set of terms defined in Froglingo. Therefore, a database with Froglingo expressions, say a Frogingo Database, doesn't inventory more than what a type and variable free EP database does, when the built-in types with infinite domains are not considered:

Theorem 1. *Froglingo database are PAC learnable provided it doesn't including constants that take built-in types as variables that have infinite domain.*

4. Fresh information from sample utterance

The instance space of the EP-driven learnability presented in (Xu 2025) is the EP terms **E**. This doesn't help us practically because there are not pre-existing EP expressions. However, we can map utterance to EP terms (expressions) first and then apply the converted EP terms to the learnability. In this section, we give a few examples in natural language to demonstrate this idea that the EP-driven learnability can actually be applied to the instance space of natural language.

A primary task of learning from utterance using Froglingo is to implement a parser that can map utterance to Froglingo expressions that represent the syntactical structure of the given utterance. Instead of a formal grammar, we use Froglingo expressions, called rules or templates, serving as a semantic parser for both syntactical parsing and semantic mapping. For the geographical containment relationships described in Section 2, for example, we can have the following template: $location\ (bE\ be)\ part\ (preP\ of)\ \$x\ :\ [\$x\ isA\ location] := there\_is\ \$y\ :\ [\$y\ isA\ location]\ where\ \$y\ \{{=}+\ \$x\ and\ \$y\ \{{-}\ location$, where $location$ is a type for a geographical location, such as $Florida$, $Miami$, and $Florida\ Miami$, $\$x$ and $\$y$ are variables typed as $location$. When a sentence like: "Somerville is part of Boston" is to be parsed, the template above would guide the parser to break the utterance into the EP expression: $Somerville\ (bE\ is)\ part\ (preP\ of)\ Boston$.

The template above guides the syntactical structure $Somerville\ (bE\ is)\ part\ (preP\ of)\ Boston$ to be reduced (mapped) to: $there\_is\ \$y\ :\ [\$y\ isA\ location]\ where\ \$y\ \{{=}+\ Boston\ and\ \$y\ \{{-}\ Somerville$;

Froglingo works in at least two modes as to be further discussion in Section 8: a learning (L) model and an operation (O) mode. When it is in L mode, it creates new data that would meet the $where$ clause, i.e., add $Boston\ Somerville\ isA\ location$ as a new fact into the database, where $Boston\ Somerville\ \{{=}+\ Boston$ [1].

With the same template, the same construction process would be able to receive additional utterance such as "Boston is part of Massachusetts", which causes the database to be updated to have the term $Massachusetts\ Boston\ Somerville\ isA\ location$. Now the database supports "Somerville is part of Massachusetts", which is a new piece of information. Reconstructing the database from the two separate sentences is a learning process enhanced from the learning algorithm on the instance space of EP terms.

[1] When Froglingo is in O mode, it would try to retrieve data $\$y$ from database to see if it meets the condition: $\$y\ \{{=}+\ Boston\ and\ \$y\ \{{-}\ Somerville$.

We also allow text to be part of an assigner. For example: $person\ (verB\ drive)\ home\ :=\ \ Botran\ (person$ "$come\ home\ by\ car$"). This assignment defines a rule that "a person drives home" is equivalent to "a person comes home by car", where we assume a template $person\ (verB\ come)\ home\ (preP\ by)\ car$ has already being constructed in the database. The identifier $Botran$ is a built-in term (operator) that obtains an instance of type $person$, e.g., $Jessie$, that is passed from the assignee at the left hand side, concatenates and adjusts the assigner to a text, e.g., "Jessie comes home by car", and parses the text back to a term in Froglingo, e.g., $Jessie\ (verB\ comes)\ home\ ((preP\ by)\ car)$. As a result, the following conversation would be newly meaningful to the NLP process: "Did Jessie come home by car?", "Yes, Jessie drove home". Given the text "Jessie came home by car" is a sample the NLP process learns, the text "Jessie drove home" is a prediction of the NLP process.

Learning through utterance can also be applied to types. When we define $cat\ isA\ animal$ and $tiger\ isA\ cat$, we will have $tiger\ isA\ animal$.

Breaking complex sentences into simple sentences is another way of learning (note a complex sentence is always broken down to multiple simple sentences in Froglingo). For example, the sentence "Joe with a hat walked on a street" is broken into two EP expressions: $Joe\ (verB\ wore)\ h0001$ and $Joe\ (verB\ walked)\ ((preP\ on)\ s0001$, where $h0001\ isA\ hat$ and $s0001\ isA\ street$, both the sentences "Joe wore a hat" and "Joe walked on a street" will be recognized and are predictions.

5. Parser

A formal grammar is a top-down approach toward parsing, i.e., it always starts with a root non-terminal that is aimed to represent all possible sentences in a language, e.g., the root symbol S in $S \rightarrow NP\ VP$, where S depends on NP and VP to further break down each sentence to a machine readable form. However, this top-down approach is difficult and has not yet produced a grammar to closely represent the grammatical phenomena of natural language (Barton, Berwick, and Ristad 1987; Shieber 1985; Gazdar et al. 1985).

Contemporary semantic parsers map the meanings of utterance to a logic (machine readable) form. Such a logic form is in expressiveness either limited such as SQL (Jiang and Cai 2024) or too powerful to halt in computation such as the lambda calculus (Poon 2013). Although statistical machine learning has advanced parsing technologies that led popular applications in our daily life, such as language translation, question answering, and code generation, it is still a challenge to produce a parser that would be as reliable as a human being. Missing the realizability assumption of the Possibly Approximately Correctly (PAC) learnability is one of the causes, i.e., some sample data cannot be correctly labeled, causing a constructed hypothesis may not converge to its target program.

Collecting parsing templates from sample utterance is a bottom-up approach. A parsing template contains variables and (or) types for entity instances that are represented by phrases or clauses such as Joe and $the\ man\ with\ ahat$. A template may include individual verbs, such as $verB\ walk$, $verB\ drive$, and $bE\ is$. However, a template doesn't include a variable or a type for verbs.

Although strategically different, the bottom-up parsing approach leverages many well-studied parsing techniques from formal grammatical approaches. Adopting the chart-based parser techniques (Allen 1995) can guarantee the time complexity of the parser in Froglingo is not more than $K*n^3$ in the worst case, where n is the length of a given sentence and K is a constant depending on a specific algorithm.

The parser accepts a sentence only if the parsed structure is consistent with the up-to-date database. One of the most important inquiries the parser performs is to find out if there is data (a set of EP terms) in the database that represents an entity or a category that a noun phrase in a given sentence is mapped to. Given a database defines $PNC\ isA\ bank$, where $bank$ has already defined as a type, and given John's bank account is represented as $PNC\ account\ John\ balance\ :=$ 1000, for example, we will be certain that the phrases "John's PNC Bank account" is mapped to $PNC\ account\ John$. Given a database having the set: $\{Family\ John; Family\ House\ (bE\ is)\ (preP\ in\ Florida);$ $Builder\ (verB\ built)\ (Family\ House)\ (preP\ in\ 2025);\}$ where $Family$, $John$, $House$, and $Builder$ are instances defined with types of $organization$, $person$, $house$, and $organization$ respectively, as another example, will certainly support the phrase "John's newly built Florida residence".

6. Ambiguity

As a recall, we approach NLP using Froglingo by processing a collection of sample utterance. For each randomly selected utterance, we give Froglingo expressions as the corresponding parsing rule/templates that maps the utterance to EP expressions as the syntactical structure of the given utterance. We further enhance the Froglingo expressions defined earlier to have a mapping rule that maps the syntactical structure of the utterance to another set of EP expressions as the meaning of the utterance. For each parsing decision, i.e., on which parsing rule to choose or if a new rule is needed for a coming utterance, the system references the database to confirm a utterance makes sense to the up-to-date knowledge stored in the database. Therefore, even if a utterance has a perfect grammatical structure and makes a full sense in the real world, the system may not recognize (read) it, as if a child didn't understand what an adult was talking about. This process is critical to maintain the realizability assumption of the PAC learnability and ensure that the database for knowledge will eventually converge. It is also critical to resolve and many times to avoid the challenges from natural language ambiguity that are important to linguistics and philosophy but unnecessary to most of NLP applications.

Even with the flexibility of the bottom-up approach in parsing, can we still end up with a difficulty not being able to represent certain grammatical structures by collecting parsing rules from sample utterance? The answer is no. First, let's mathematically replace all

variables and types with their instances in a parsing rule, e.g., replace $person$ with $Jessie$, Joe, ..., $Zack$ in $person\ (verB\ drive)\ home$ and we end up with multiple parsing rules $Jessie\ (verB\ drive)\ home$, $Joe\ (verB\ drive)\ home$, ..., $Zack\ (verB\ drive)\ home$. Assuming only a finite number of objects is our concern in representing the world knowledge, we still end up with a finite number of parsing rules if we replace the variables and the types with their instances in all collected parsing rules. Therefore, the rules without types and variables would be still effective and most critically precise in parsing utterance. We say such a parsing is precise because for each utterance: 1) if it is unique in meaning regardless of its context, we give its unique grammatical structure as it is; 2) if it is ambiguous in syntax, such as "Joe saw the girl with a binoculars", we can give its syntactical structure based on what the speaker intended, i.e., referencing the Froglingo database for its true meaning before deciding its syntactical structure; 3) if it is ambiguous in semantics, such as "Joe got what he wanted" that is unique in syntax but could mean many things in meaning, we can still search the database for the context: who "Joe" is, What object Joe wanted and got, and if "got" is a "took", "received", or "purchased"; and 4) if it is ambiguous in pragmatics, such as "put the coffee on the table" while there are two tables next to each other as people often purposely or unconsciously say, we have to raise a question for clarification after a search on the database concludes the ambiguity.

Can we express the utterance that are not context free such as cross-serial dependencies in Swiss German (Shieber 1985)? We don't attempt to develop a parsing rule in Froglingo to represent arbitrary layers of the cross-serial dependencies, in a contrast to a context sensitive grammar. But we can develop a finite number of layers of the dependencies, e.g., the 3rd or 4th, which is an approximation to the context sensitive grammar but should be practically sufficient.

Now, let's come back to the reality: we still need variables and types in parsing rules. Can we adequately manage those variables and types to precisely parse utterance? The answer is yes. If a variable or type originally defined in a rule has unexpected instances, e.g., $person\ (verB\ wear)\ hat$ where we like hat to be red only, we can redefine it as $person\ (verB\ wear)\ \$h:\ [(\$h\ isA\ hat)\ and$ $(\$h\ color == adJ\ red)]$. If there are some extremely difficult utterance to be syntactically broken down, we can always roll back individual instances instead of variables and types, although the effort appears to be tedious.

In summary, ambiguities does not have to hinder the performance of the parsing process in Froglingo as if they didn't bother human beings in their communication. While we cannot describe a precise mapping from all the utterance to parsing templates (unless we had already sampled all necessary utterance for a convergence) to explicitly demonstrate this argument, the following conclusion certainly strengthens it:

Theorem 2. *Provided the assumption that knowledge is isomorphic to a bounded function, NL utterance are efficiently PAC learnable.*

Given the assumption, in other words, accumulating randomly selected sample utterance eventually converges to a machine that communicates with its surroundings in NL as if it was a human being.

7. Ontology

Like ambiguity, ontology is another area Froglingo's bottom up approach has to tackle to accumulate and eventually converge all the phenomena of the real world but not possibly and necessarily resolve many open questions that have been discussed over the entire ontology history.

First of all, we believe the information about entities, i.e., concrete objects, in the real world, such as a person named John, the tree in the front of my house, the water in my cup, etc. that exist in the default dimensions of time and space, is the information ground that other information should eventually land on. In other words, other information are about entities such as a physical object's weight and color, people's sentiment, and relationships among entities, a meeting, sports, law, an entity's sub components, a biological cell, and a natural language. We assume that all of these information must be correlated together in the space of a bounded function that can be mathematically supported in a physical EP database. In reality, we can only inventory pieces of such information that are randomly selected regardless of which piece in what order is entered, where each piece may not immediately but ultimately be able to be related to others along the information collection process in a database. For example, a database can first reflect "John's bank account was reduced from $1000 to $950" before reflecting "John had spent $50 to buy a dinner", which enables the machine to correlate the two events. A database also often needs to inventory partial information, e.g., "John went to New York City with another person", where the phrase "another person" is a partial information, i.e., an instance typed as $person$ without further information.

In Froglingo, we use the transitive relation isA to categorize things in the worlds, where the relation isA forms a Directed Acyclic Graph (DAG), instances are end (leaf) nodes, and types are intermediate or root nodes. For convenience, we may always have only one root node, such as named $thing$. While a child type or instance has more specific information than its parent does (i.e., a child inherits its parent in programming language), a parent (or ancestor) can serve as a coreference to a child and therefore the parent would act on the behalf of the child. For example, we can express: $schema\ animal\ isA\ thing$, $schema\ person\ isA\ animal$, and $create\ John\ isA\ person$ to form a chain: $John\ isA\ person\ isA\ animal\ isA\ thing$. This chain would allow the utterance "the person said ..." (and "the animal said ..." as well though not perfectly fit) in the context that "John said ...", where "the person" actually refers to John. When we define another type $schema\ cat\ isA\ animal$, the utterance "the animal said ..." is invalid because "a cat said ..." would never happen, as the type cat is constructed without the ability of speaking, in the context that "A cat meowed" and "the animal" actually refers to the cat. Note we define $person$'s ability of speaking by defining $schemaperson(verBspeak)$.

A single entity, particularly those entities human beings often interact with, has a lot of different words or phrases used to categorize it. We have to properly arrange them all into a DAG eventually if not necessarily immediately. After the type $person$ is defined, for example, we need to further place words "woman", "female", "mother", and etc. into DAG. We can have the following tentative type structure: $female\ isA\ animal$, $woman\ isA\ person$, $mother\ isA\ female$, $Mary\ isA\ mother$, and $Mary\ isA\ woman$. How do we relate $Mary$ defined earlier to "the lady" in "The New York City welcomed Mary, a lady coming from Florida"? The answer is to add another type: $lady\ isA\ woman$.

DAG along is not sufficient to define all the properties of a type. For example, $mother$ can be enhanced to be related to other entities by the template: $animal\ (bE\ be)\ \$a\ :\ [\$a\ isA\ animal]\ S\ mother\ :=\ person\ (verB\ mothers)\ \$a$, where "S" denotes the mark "'s" in for example "Joe's mother". Statistical data also enhance properties of types. For example, after we have $pal\ isA\ man$, we may have the template: $person\ (bE\ be)\ \$p\ :\ [\$p\ isA\ person]\ S\ pal\ :=\ person\ (verB\ chatted)\ (preP\ with)\ \$p\ (adverB\ often)$, where $adverB\ often$ would trigger the machine to guess who are the pals in "John went to bar with his pals" by calculating statistical data about John's connections with other people. (Note that statistics will play a big role in defining a large number of adjectives and adverbs, which is not further discussed here.)

Actions (i.e., the meaning of a sentences) can be also categorized. We use the built-in relation isA to facilitate the categorization process at the action level. For example, "John plays soccer. He loves sport", where we simply relate the two sentences by defining types: $schema\ sport\ isA\ thing$ and $schema\ soccer\ isA\ sport$. We also use action nouns to explicitly categorize a sentence using isA, such as improvement categorizing painting a house as discussed in Section 3. Not every action can be and needs to be categorized. "John run yesterday" which could be meant for sport or something else such as for escaping from being arrested. But an action may be eventually categorized by providing additional context, such as "John run in New York Marathon yesterday".

The bottom line is that we have the following conclusion:

Theorem 3. *Provided that knowledge is isomorphic to a bounded function, all the objects in knowledge can be inventoried in an EP database and the relationships among the objects can be calculated.*

where "objects" refer to particulars, universals, concrete objects, abstract objects, and etc. in ontological theories.

8. Natural language along system language

Like in statistical machine learning, Froglingo consists of two major components: 1) a learning model - A learning algorithm encoded in Froglingo expressions and 2) a predictor - generated EP assignments in a database (hypothesis) from positive utterance samples. A learning model consists of a parser and a set of templates - Frogling expressions representing the core of the learning algorithm. The learning model uses the parser to parse utterance and references the templates to generate the predictor. Given any positive sample utterance, developers can start to write templates against sample utterance. A predictor can be used anytime as soon as it is generated. The more sample utterance are provided, the richer model and predictor will be. When the system is accumulated with certain amount of templates, additional templates may no longer needed even more sample utterance is needed to further enrich the learning model and the predictor. At this point, we would say that the NLP system "fully understands" a natural language, and that individuals without a software development experience would be able to operate in both L and O modes, driving knowledge accumulation and giving instructions in natural language for tasks that are required to be reliably executed.

In Appendix E, we focus on L mode using 4 sentences, including the first 3 from the famous folktale "Jack and Bean Stalk", to demonstrate how developers (users) interact with the Froglingo-based NLP system using both NL and Froglingo expressions to construct a database representing knowledge. The fourth example show how the factorial function can be described in NL and converted to an EP expression that is convertible to $null$ in EP but a Turing-complete expression in Froglingo.

9. Conclusion

Provided that knowledge is isomorphic to a bounded function, we showed 1) a parser exists to parse utterance as accurate as human being does, and 2) the world knowledge can be inventoried in an EP database and the relationships among the objects in knowledge can be calculated. We provided many examples, observations, and critical strategies that support the assumption leading to the two conclusions above. We finally proposed an interface for the Froglingo NLP that accepts NL beside of system language during the NL learning process.

Instead of the word "sentence", we used the word "utterance" in this paper in terms of NLP. We meant the text that can be processed includes not only grammatically correct sentences, but also various utterance including grammatically incorrect sentences and slangs. EP can model any language structures without a formal grammar. In addition, many similarity operations are supported by certain EP operators (Xu, Zhang, and Gao 2010).

Unlike a statistical machine learning approach, the symbolic approach to the bottom-up parsing rules needs manual development, at least supervising if a statistical machine learning can help automatically collect parsing and mapping rules.

EP data cannot directly be applied to the modern statistical machine learning technologies, though a mapping between the two may be developed. The latter uses similarities on geometric measures such as distance or cosine functions among objects that are embedded to a Euclidean space to represent their relationships. The former uses transitive relations among the symbolic objects to calculate their relationships.

References

Allen, J. 1995. *Natural Language Understanding, second edition*. The Benjamin/Cummings Publishing Company, Inc.

Apidianaki, M. 2022. *From word types to tokens and back: a Survey of approaches to word meaning representation and interpretation*. Computational Linguistics (2023) 49 (2): 465–523.

Barton, G.; Berwick, R.; and Ristad, E. 1987. *Computational Complexity and Natural Language*. The MIT Press Classics.

Berant, J.; Dagan, I.; and Goldberger, J. 2012. *Learning entailment relations by global graph structure optimization*. publisher of Computational Linguistics, 38(1):73-111.

Bordes, A.; Weston, J.; Collobert, R.; and Bengio, Y. 2011. *Learning structured embeddings of knowledge bases*. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. *Language Models are Few-Shot Learners*. NeurIPS 2020.

Cai, H.; Zheng, V.; and Chang, K. 2017. *A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications*. IEEE Transaction on Knowledge and Data Engineering, Sept. 2017.

Gazdar; Klein; Pullum; and Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge.

Gold, E. 1967. *Language Identification in the Limit*. Information and Controls, 10, 447-474 (1967).

Jiang, P.; and Cai, X. 2024. *A Survey of Semantic Parsing Techniques*. Symmetry 2024, 16, 1201. https://doi.org/10.3390/sym16091201.

Jiang, X.; Dong, Y.; Wang, L.; Fang, Z.; Shang, Q.; Li, G.; Jin, Z.; and Jiao, W. 2023. *Self-planning Code Generation with Large Language Models*. ACMTrans. Softw. Eng. Methodol., Vol. 1, No. 1, Article . Publication date: October 2023.

Kearns, M.; and Vazirani, U. 1994. *An introduction to computational learning theory*. The MIT Press.

Kleinberg, J.; and Mullainathan, S. 2024. *Language generation in the limit*. NeurIPS 2024, arXiv:2404.06757.

Li, J.; Raman, V.; and Tewari, A. 2025. *Generation through the lens of learning theory*. 38th Annual Conference on Learning Theory (COLT 2025).

Li, Q. 2025. *Position: Transformers Have the Potential to Achieve AGI*.

Littlestone, N. 1987. *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*. Machine Learning, 2:285–318, 1987.

Merrill, W.; and Sabharwal, A. 2024. *The expressive power of transformers with chain of thought*. International Conference on Learning Representations.

Nickel, M.; Tresp, V.; and Kriegel, H. 2012. *Factorizing YAGO – Scalable Machine Learning for Linked Data*. WWW2012 – Session: Creating and Using Links between Data Objects. April 2012, Lyon, France.

Poon, H. 2013. *Grounded Unsupervised Semantic Parsing*. Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages 933–943, Sofia, Bulgaria, August 4-9 2013.

Seshadhri, C.; Sharma, A.; Stolman, A.; and Goel, A. 2020. *The impossibility of low rank representation for triangle-rich complex network*. The Proceedings of National Academy of Sciences, March 2020.

Shieber, S. 1985. *Evidence against the context-freeness of natural langauge*. Linguistics and philosophy 8 (1985) 333-343, Reidel Publishing Company.

Xu, K. 2017. *A class of bounded functions, a database language and an extended lambda calculus*. publisher of Theoretical Computer Science, Vol. 691, August 2017, Page 81 - 106.

Xu, K. 2024. *Outline of a PAC Learnable Class of Bounded Functions Including Graphs*. The 7th International Conference on Machine Learning and Intelligent Systems (MLIS 2014).

Xu, K. 2025. *Classes of bounded functions that are semantically equivallent to Turing machine are PAC learnable*. To present in the 38th Annual Conference on Learning Theory (COLT 2025) - Theory of AI for Scientific Computing Workshop, June 30–July 4, 2025 in Lyon, France. Preprint: DOI: 10.13140/RG.2.2.28499.49443.

Xu, K.; Zhang, J.; and Gao, S. 2010. *Higher-level functions and their ordering-relations*. The Fifth International Conference on Digital Information Management, 2010.