Parsing for a machine-readable form that is semantically equivalent to Turing machine and PAC learnable

Anonymous ACL submission

Abstract

Developing a parser as reliable as a human being is a key to map natural language (utterance) to a comprehensive logic form. In this paper, we introduce the Enterprise-Participant (EP) data model and propose a semantic parser that maps utterance to an EP database. Because EP, a recursive language, is semantically equivalent to Turing machine, i.e., an EP database is mathematically capable of inventorying all the properties of a partial recursive function with the hypothesis of infinite space and time, we assume and expect the meaning of natural language can be adequately fit (or precisely approximated) into an EP database having finite objects with infinite properties including self-applicable functions. Instead of using a formal grammar, we accumulate parsing rules from sample sentences, i.e., given a randomly selected sentence, we add the corresponding syntactical structure and meaning into an EP database. Because an EP database is PAC learnable, the accumulation process converges, when sample sentences amount to a foreseeable size, to the ultimate machine readable form of the entire natural language. As a side effect, the collection of the parsing rules will be converged to map arbitrary utterance to their syntactical structures as part of the ultimate machine readable form in an EP database.

1 Introduction

If a parser can break down all possible natural language (utterance) into a machine readable form, either for syntax (grammatical structure) or for semantics (meanings), as accurate as human being does, a computer will be able to be instructed in natural language, as if in programs coded by human beings, to perform many intelligent tasks that human being does, including reasoning, decision making, and running machine operations, utterance generation, language translation, and certainly information management as well.

Formal grammars are aimed to map the syntactical structure of all possible sentences (utterance) into a machine readable form. However, no known grammar has been developed yet to closely represent a natural language (Barton et al., 1987; Shieber, 1985; Gazdar et al., 1985). Contemporary semantic parsers map the meanings of natural language (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.

The Enterprise-Particpant (EP) data model is a type and variable free language system and equivalently a data structure with which an EP database can be constructed. An EP database can be syntactically converted from a finite approximation to the lambda calculus and is interpreted as a bounded function, i.e., recursive with infinite domain while guaranteed with a finite co-domain (Xu, 2017). As a result, the union of all (infinite) EP databases and equivalently the class of all bounded functions are semantically equivalent to the lambda calculus. In other words, we can use EP to accumulate information, formally the properties of partially recursive functions, as much as the time and space were allowed. EP is more expressive than the contemporary data structures, including relational data (vectors), tree structures, and network structures (graphs) because all of them can be expressed in EP. For example, a directed cyclic graph with edges: v_1

to v_2 , v_2 to v_1, and v_2 to v_3 is constructed with a database: $D = \{v_1\ v_2 := v_2; v_2\ v_1 := v_1; v_2\ v_3 := v_3\}$. EP with the constructed database supports the following queries simulating one's walks along the cyclic graph, i.e., reduces the left expressions to the right expressions: $v_1\ v_2\ v_3 \rightarrow_D v_3, v_1\ v_2\ v_1 \rightarrow_D v_1, v_2\ v_1\ v_2\ ...\ v_1 \rightarrow_D v_1, ...$ (Note that a sequence of nodes that do not form a path would be reduced to a special value $null$, e.g., $v_3\ v_2 \rightarrow_D null$.)

Further, EP has a built-in transitive relational operator $(= +$ to express if two nodes in a graph has a path or a cycle. For example: $v_3\ (= +\ v_1 \rightarrow_D true$, i.e., a path from v_1 to v_3; $v_1\ (= +\ v_3 \rightarrow_D false$, i.e., from v_3 to v_1 is not a path; and $v_2\ (= +\ v_1\ and\ v_1\ (= +\ v_2 \rightarrow_D true$, i.e., a cycle between v_1 and v_2.

Another unique feature of EP is that EP databases are (PAC) learnable, i.e., a class of bounded functions represented in a class of EP databases is PAC learnable (Xu, 2025). Taking a class of graphs as a proper subset of the bounded functions expressible in EP, for example, there is an algorithm that can construct the example database D described earlier by taking following paths as positive samples: $v_2\ v_1$ and $v_1\ v_2\ v_3$.

Because a bounded function represents a arbitrary proper subset of the properties of partial recursive functions, such a learnability, i.e., gaining an arbitrary number of meaningful expressions by databasing a finite set of sample expressions without explicit programming, gives us an opportunity to represent the world knowledge to EP expressions in an EP database, and further map natural language utterance to the machine readable form of knowledge in the EP database. We assume there is a partial recursive function that precisely represents and computes the knowledge. We know that it is impossible (because knowledge is constantly changing) and practically unnecessary to construct a program that precisely 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 knowledge. But we can further assume that knowledge can be reasonably represented as a bounded function because 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, as is all a computer can do, we say that we can accumulate knowledge into an EP database which will eventually converge to the ultimate bounded function representing the knowledge.

We further argue that learning and collecting knowledge in a machine representable form is a paraphrase of learning natural language (NL) and mapping NL to a machine representable form. If we know how to do the latter, we can collect knowledge in NL. If we know how to do the former, since NL is part of knowledge, we become capable of doing the latter by default. This assumption actually has become evident from the practice of statistical machine learning on NLP.

In this paper, we present a symbolic learning approach in EP, assisted by Froglingo - a Turing complete language extended with variables and types on the top of EP, to tackle knowledge representation and a mapping from utterance to EP expressions representing knowledge. It is done through a collection of sample sentences. For each randomly selected sentence, we give Froglingo expressions as the corresponding parsing rule that maps the sentence (and possibly others) to EP expressions as its syntactical structure. We further enhance the Froglingo expressions defined earlier to have a mapping rule that maps the syntactical structure of the sentence (and probably others) to another set of EP expressions as the meaning of the sentence (and probably others). For each parsing decision, i.e., on which parsing rule to choose or if a new rule is needed for a coming sentence, the system references the database to confirm a sentence makes sense to the up-to-date knowledge stored in the database. Therefore, even if a sentence has a perfect grammatical structure, 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.

A contemporary programming language is a top-down approach in the sense that we know exactly what a function we need to construct and we construct exactly the same function, e.g., $f(x) = x+1$. EP is a bottom-up approach in the sense that we don't know what the ultimate function in construction would be ended up with but we add one piece of its properties at a time, e.g., $D = \{f\ 0 := 1; f\ 1 := 2; ...; f\ 100000 := 100001\}$. This difference gives an intuitive view on why the parsing approach in EP is different from the contemporary

symbolic solutions.

In Section 2, 3, and 4, we review the notions of EP database, reduction , and transitive relations. In Section 5, we introduce Froglingo and additional transitive relations. By examples, in Section 6, we demonstrate how the learnability is applied to natural language processing (NLP). Through the discussion, we observe that Froglingo serves as a tool to construct parsing and mapping rules as part of a learning algorithm in processing NL. In Section 7, we show that the collection of parsing rules from sample text converges to a target parser needed to parse arbitrary utterance provided the utterance is representable, precisely can be approximated, in EP.

2 EP databases

The Enterprise-Participant (EP) data model is a language system and equivalently a data structure with which an EP database can be constructed. The idea behind EP is that we treat all objects to be represented as functions. Given a function f that produces a value m when it is applied to an argument n, denoted as $f\ (n) = m$, let's think of an exercise in which we inventory the properties of f in a database. We can rewrite $f\ (n) = m$ as $f\ n := m$, reading it as: applying f to n is assigned a value m. The set $\{f\ n := m\}$, called a database, is an approximation of f. When we apply f to an additional argument n', we would obtain a better approximation $\{f\ n := m, f\ n' := m'\}$ where $f\ (n') = m'$. In addition, m could be another function such that $m\ (p) = q$ for a given input p. So we can exhibit more properties of f with the accumulated approximation $\{f\ n := m, f\ n' := m', m\ p := q\}$ or equivalently $\{f\ n\ p := q, f\ n' := m'\}$. From the database $\{f\ n := m, f\ n' := m', m\ p := q\}$, we can derive: $(f\ (n))\ (p) = q$.

Definition 2.1 The EP data model is described as a language system $(\mathbf{F}, \cdot, (,), \mathbf{E}, :=, D)$ where

1. $\mathbf{F}$ is a set of identifiers (function names)

2. $\cdot$ is a binary operation that produces a set $\mathbf{E}$ such that
   $m \in \mathbf{F} \implies m \in \mathbf{E}$
   $m, n \in \mathbf{E} \implies (m \cdot n) \in \mathbf{E}$

   Here we simply write $(m \cdot n)$ as $(m\ n)$ and further $m\ n$ when $(m \cdot n)$ is implied [1], where m, n, and $m\ n$ are called a function, an argument, and the corresponding application. For a $x \in \mathbf{E}$, we call x a term. Given an application term $m\ n$, m and n are called proper subterms of $m\ n$, and $m\ n$ is also called a subterm of itself.

3. $:=$ is the Cartesian product $\mathbf{E} \times \mathbf{E}$, i.e., $:= \equiv \mathbf{E} \times \mathbf{E}$. When a pair $(p, q) \in :=$, we denote it as $p := q$, which is called an assignment, where p and q are the assignee and assigner respectively.

4. D, called a database, is a finite set of terms and a finite set of assignments, i.e., $D \subset (\mathbf{E} \cup :=)$, such that for each assignment $p := q \in D$, where $p, q \in \mathbf{E}$, the following constraints are met:

   (a) p has only one assigner, i.e., $p := q$ and $p := q' \in D \implies q \equiv q'$

   (b) A proper subterm of p cannot be an assignee, i.e., $p := q \in D \implies \forall x \in SUB^{+}(p)\ [\forall m \in \mathbf{E}\ [x := m \notin D]]$

   (c) q can not be an assignee, i.e., $p := q \in D \implies \forall a \in \mathbf{E}\ [q := a \notin D]$

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 commonly we take words from a natural language vocabulary as identifiers, such as $hello$, $John$, $sport$, law, and $person$.

A term is either an identifier $x \in \mathbf{F}$ or an application $x\ y \in \mathbf{E}$ where $x \in \mathbf{E}, y \in \mathbf{E}$, such as $x\ x, (a\ b\ c)\ (d\ e\ a\ (d\ t\ a))$ are legitimate terms where $x, a, b, c, d, e, t \in \mathbf{F}$.

Given a term, e.g., $m_0\ m_1 \ldots m_i$ for an $i \in N$, we call all the leftmost subterms of the term, i.e., $m_0, m_0\ m_1, \ldots, m_0\ m_1 \ldots m_i$ a leftmost subterm, denoted as lms. Given a term t, we use $|t|$ to denote the size of the term, e.g., $|m_0\ m_1\ \ldots\ m_i|$ = $i + 1$, and $LMS(t)$ to denote the set of all lmss in t. (Then we have $t \in LMS(t)$. If $m\ n \in LMS(t)$, so is m.) We further use $LMS^{+}(t)$ to denotes all the proper lmss in t, i.e., $LMS^{+}(t) = LMS(t) \setminus t$. We further use $SUB(p)$ to denote all the subterms of a term p, i.e., given $p \equiv m\ n$, then $m, n, m\ n \in SUB(p)$. We use

[1] When a combination of two terms $m\ n$ is given without surrounding parentheses, we consider the term is parsed by the preference of left association and therefore $(m\ n)$ is implied. For example, $a\ b\ c$ always implies $((a\ b)\ c)$. If one needs to express $(a\ (b\ c))$, then it has to be written explicitly like $a\ (b\ c)$.

$SUB^+(p)$ to denote all the proper subterms of p, i.e., $SUB^+(p) = SUB(p)\backslash\{p\}$.

In addition to the example database discussed in Section 1, we list a few more sample databases here:

- $\{x\ x := x\}$, for a graph with a single vertex x, counting a vertex having a directed loopback edge
- $\{a\ b\ c := d; d\ (e\ f) := a\ b;\}$, for a random database.
- $\{college\ John\ major := college\ math;$ $college\ math\ math100\ (college\ John)\ grade := A\}$, for a college administration database.

3 EP database reductions and bounded functions

We are ready to introduce a reduction system over $\mathbf{E}$. First, we identify a special identifier $null \in \mathbf{F}$ that has the default reduction rule: $null\ m \rightarrow_D null$ for any $m \in \mathbf{E}$. Because we explicitly single out the special identifier $null$ from $\mathbf{F}$, we further restrict that a lms of an assignee cannot be $null$ in a database. Before providing the full set of reductions rules, let's first define the notation of EP normal form.

Definition 3.1 Given a database D, a term $n \in \mathbf{E}$ is an EP normal form (or normal form in brief) if and only if

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

We use $NF(D)$ to denote the set of all the normal forms under a database D.

Most terms in $\mathbf{E}$ are not normal forms given a database D, For example, $x\ x$ and $x\ x\ x$ are not normal forms in the sample database $\{x\ x := x\}$. We now define a set of rules to reduce an arbitrary term to a normal form.

Definition 3.2 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. $a \Rightarrow a', b \Rightarrow b' \implies a\ b \Rightarrow a'\ b'$

Definition 3.3 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 $a == a_n$.

Definition 3.4 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: $x\ x\ \ldots\ x \rightarrow_D x$, $a\ b\ c \rightarrow_D d$, $a\ b\ c\ (e\ f) \rightarrow_D a\ b$, $d\ (e\ f)\ c \rightarrow_D d$, and $college\ John\ major\ math100\ (college\ John)\ grade \rightarrow_D A$.

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 $m \rightarrow_D n$ (Theorem 4.5 in (Xu, 2017)).

The set of all the normal forms $NF(D)$ is finite, i.e., $|NF(D)| \leq s$ for a given $s \in \mathbf{N}$, e.g., s could be the number of partial computation steps from which D is transformed (Lemma 4.6 in (Xu, 2017)).

There exists a function $Y(D) : \mathbf{E} \rightarrow_D NF(D)$, where $Y(D) = \{(m, n) \mid m \in \mathbf{E}, n \in NF(D),$ and $m \rightarrow_D n\}$, and $Y(D)$ is bounded because it has an potentially infinite number of terms m and a finite numbrer of terms n, where $n \neq null$, such that $m \rightarrow_D n$. (Such a function is said to have an potentially infinite domain while only having a finite co-domain (Theorem 4.7 and Theorem 4.8 in (Xu, 2017)).) A function $f : X \rightarrow Y$ has a finite support if and only if X is an arbitrary set of objects and Y is a finite set of objects, and there exists a finite set $A \subset X$ and a unique member $a \in Y$ such that

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

A function $f : X \rightarrow Y$ is bounded if and only if X is an arbitrary set of objects and Y is a finite set of objects. (Such a bounded function is always recursive, i.e., the computation on $f(x)$ terminates and $f(x) \in Y$ for any $x \in X$.) In this article, we simply call a function finite if it has a finite support. A finite function is bounded, but a bounded function may not be finite.

By saying a function being bounded, we mean that under a given database D, an infinite number of terms $m \in \mathbf{E}$ are meaningful, i.e., reducible to a

finite set of normal forms $NF(D)$. The ability of mapping infinite objects to finite objects is both the symptom and the pre-condition of the learnability, i.e., one object in the co-domain is represented by multiple objects in the domain, or saying differently one object in the co-domain is derivable from others in the domain.

Even if $Y(D)$ is finite for a given D, $Y(D)$ may provide derivable information beyond what are defined in D, i.e., we say it PAC learnable. For example, the database $D = \{a\ b := c; c\ d := e\}$ allows the reduction (derivation): $a\ b\ d \rightarrow_D e$, which is not defined in D. Lastly, an EP database may not have any derivable information, e.g., $D = \{a\ b := c; e\ e := f\}$. Such databases without derivable information have nothing to do with learnability.

4 Transitive relations in EP

The transitive relations and the corresponding built-in operators come from the structure available among identifiers, terms, and assignments, as well as from the reduction rules. They play important rules in defining syntactical parsing rules and semantic mapping rules. In this section, we quickly review the material presented in (Xu et al., 2010).

4.1 Tree-structured relations

A term alone without an assignment is allowed to be in a database. When a term is in a database, its subterms are considered in the database as well. In terms, we can represent containment relationships. For example, the hierarchical structure of a geographical location can be expressed as: $Massachusetts\ Boston\ Somverville$, where we can infer $Somerville$ is part of $Massachusetts$ because $Somverville$ is part of $Boston$ and $Boston$ is part of $Massachusetts$.

The containment relationships among subterms of a given term are true-structured, denoted as $\{+$, by taking a most left subterm of the given term as the root of a tree, e.g., $Massachusetts\ Boston\ \{+\ Massachusetts$. When we take a right most subterm of a term as a root, we have another tree-structured relation, denoted as $\{-$, e.g., $Massachusetts\ Boston\ \{-\ Boston$. The tree-structured relations can be extended to transitive relations, denoted as $\{= +$ and $\{= -$, e.g., $Massachusetts\ Boston\ somerville$ $\{= +\ Massachusetts$ and $Massachusetts\ Boston\ somerville$ $\{= -\ Somerville$ respectively.

These tree-structured and transitive relations and the corresponding operators are summarized as following:

- Given an application $m\ n$ in a database D, the operators $\{+$ and $\{-$ in the following expressions are defined such that the expressions are evaluated to be true: $m\ n\ \{+\ m$, $m\ n\ \{-\ n$
- Given a term m in a database D, let l, s, r are a left-most subterm, a subterm, and a right-most subterm accordingly, then the operators $\{= +$, $\{= -$, and $\{=$ in the following expressions are defined such that the expressions are evaluated to be true: $m\ \{= +\ l$, $m\ \{= -\ r$, and $m\ \{=\ s$.

4.2 Pre-ordering relations

The normal form, resulting from an application, doesn't have to depend on the function and the argument of the application because it exists independently. However, the normal form is derivable from the application; and therefore it is derivable from the function, from the argument, and from the sub-terms of the function and the argument. This leads to the development of the pre-ordering relations.

- Let $m,\ n,\ q \in \mathbf{E}$ and D a database, if $m\ n\ ==\ q$, then the operators $(+$ and $(-$ in the following expressions are defined such that the expressions are evaluated to be true: $q\ (+\ m$ and $q\ (-\ n$.
- Let $m, q, l, s, r \in \mathbf{E}$, D a database, $m\ ==\ q$, l is a left-most subterm of m, s is a subterm of m, and r is a right-most subterm of m. Then the operators $(=\ +$, $(=\ -$, and $(=$ in the following expressions are defined such that the expressions are evaluated to be true: $q\ (= +\ l$, $q\ (= -\ r$, and $q\ (=\ s$.

The operator $(=\ +$ defined above is used to express paths and cycles in a graph as demonstrated in Section 1.

5 Froglingo and more equivalent relations

In Section 3, we have introduced the equivalence relation ==, which is a transitive relation and can be used to express paraphrases in natural language, such as $New\ York\ City := NYC$. In this section, we introduce types and variables in Froglingo that

raise different forms of assignments (called rules or templates) and consequently additional equivalence 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 $fac\ \$n\ :\ [\$n\ isA\ integer]\ :=\ (\$n\ \times\ (fac\ (\$n\ -\ 1)))$, 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 $\{+$, 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 a 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 occur only for the specific circumstances.

6 Fresh information from sample text

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 text 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 text using Froglingo is to implement a parser that can map text to Froglingo expressions that represent the syntactical structure of the given text. Instead of a formal grammar, we use Froglingo expressions, called rules or templates, serving as a semantic parser for both text 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 text 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:
$there\_is\ \$y\ :\ [\$y\ isA\ location]$

$where\ \$y\ \{=\ +\ Boston\ and\ \$y\ \{-\ Somerville$; where $there\_is$ is a built-in operator [2] to find out if there is a location $\$y$ that meets the $where$ clause.

Froglingo works in at least two modes: 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 $Boston\ Somerville\ \{=\ +\ Boston$ [3].

With the same template, the same construction process would be able to receive additional text 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 (Xu, 2025).

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), introduced in (Xu, 2022), 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 text can also be applied to types. When we have already defined $thing$ as a type and $word$ as the type for an identifier from **E**, regardless of already in the database or not, we can further define: $word\ (bE\ be)\ thing\ :=\ word\ isA\ thing$. The template above allows the prediction: "Tiger is an animal" after learning "Animal is a thing", "Cat is an animal", and "Tiger is a cat".

Breaking complex sentences into simple sentences is another symptom of learning, reflecting the Step 2 and 3 of the learning algorithm given in Section 6.1. For example, the database has templates: $person\ (verB\ wear)\ hat$ and $person\ (verB\ walk)\ ((preP\ on)\ street)$. When the texts "Joe wears a hat" and "Joe walks on a street" have already be learned and constructed with the facts in the database with $Joe\ (verB\ wore)\ \$x\ :\ [\$x\ isA\ hat]$ and $Joe\ (verB\ walked)\ ((preP\ on)\ (\$x\ :\ [\$x\ isA\ street])$, the text "Joe with a hat walked on a street" will be parsed and mapped to the two templates above when an additional template for the prepositional phrase "with a hat" that modifies a person. Recognizing the last text is a prediction.

7 The parser

The difference between EP's bottom-up approach and programming language's top-down approach is similarly applied between a formal grammar and a collection of parsing rules in Froglingo expressions discussed in Section 6. A formal grammar always starts with a root non-terminal, serving as a variable in programming language, 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 et al., 1987).

Collecting parsing rules from sample text is a button-up approach. A parsing rule can include variables for phrases acting as nouns, such as the types $location$ and $person$. Additionally there are needs to have built-in operators introducing various clauses including infinitive and gerund phrases as well as noun clauses (Xu, 2022). A rule may include individual verbs, such as $verB\ walk$, $verB\ drive$, and $bE\ is$. However, a rule doesn't include a variable or a type representing a category of verbs and there is not a single rule (variable) to

[2] This operator is similar to the *select* operation in the SQL language for the relational data model and is applicable to all binary operators including arithmetical, boolean, and transitive ones introduced in Section 4 and 5. See more in (Xu et al., 2010)

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

represent all possible sentences.

Even with the flexibility of the bottom-up approach, can we still end up with a difficulty not being able to represent certain grammatical structures by collecting parsing rules from sample text? 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 text. We say such a parsing is precise because for each sentence: 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.

8 Related work

Data mining technologies acquire machine readable knowledge from text. Statistical based NLPs acquire vector-based (machine readable) knowledge from text, which is not human-readable and may not be quite accurate. In this paper, we propose a symbolic learning approach in Froglingo to more accurately acquire knowledge from text and represent it in both machine- and human-readable Froglingo expressions.

Statistical based NLPs learn from various text indiscriminately available in the society including harmful materials. Therefore, it is hard to control to not generate unpredictable and harmful materials. A symbolic approach, on the other hand, has a full control on what should be generated.

9 Conclusion

The main contribution of this paper is: the bottom-up approach of finitely collecting parsing rules through sample text converges to a parser that can be as reliable as a human being.

There is an open question: Is a collection of parsing rules, with variable and types, PAC learnable while we only concluded that the parser will work correctly in parsing utterance? So far, we know that an EP data model is PAC learnable and the bottom-up parsing rules correctly map utterance to EP terms. When a finite set of sample utterance make an effect of more meaningful new utterances through learning, does the corresponding finite set of parsing rules make an effect of more meaningful new parsing rules? The answer should be yes because the new meaningful utterances must be parsed by the given finite parsing rules. This conclusion is consistent with the notation "undetermined", denoted as the symbol "*" designated as a variable, as part of boolean functions that were concluded PAC learnable in (Valiant, 1984). A more thorough work is needed to confirm the conclusion.

10 Limitations

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 rules.

The symbolic approach to the bottom-up parsing rules cannot directly be applied to the 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 function 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

R. Agrawal and A. Borgida. 1989. Efficient management of transitive relationships in large data and knowledge bases. *ACM SIGMOD Record, Volume 18, Issue 2*.

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

M. Apidianaki. 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*.

H. P. Barendregt. 1984. The lambda calculus - its syntax and semantics. *North-Holland*.

H.P. Barendregt and J.W. Klop. 2009. Applications of infinitary lambda calculus. *Information and Computation, Volume 207, Issue 5, May 2009, Page 559-582*.

G.E. Barton, R. Berwick, and E.S. Ristad. 1987. Computational complexity and natural language. *The MIT Press Classics*.

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

R. Bhattacharjee and S. Dasgupta. 2019. What relations are reliably embeddable in euclidean space? *Journal of Machine Learning Research 1 (2019) 1-48*.

C. Bizer, T. Heath, and T. Berners-Lee. 2009. Linked data-the story so far. *International Journal on Semantic Web and Information Systems, 5(3):1–22*.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. 1989. Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery. Vol. 36. No. 4. October 1989, pp. 929-965*.

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

S. Brass and M. Wenzel. 2019. Performance analysis and comparison of deductive systems and sql databases. *CEUR-WS.org Vol-2368 paper3.pdf*.

N. Brukhim, D. Carmon, I. Dinur, S. Moran, and A. Yehudayof. 2022. A characterization of multiclass learnability. *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*.

H. Cai, V.W. Zheng, and K. Chang. 2017. A comprehensive survey of graph embedding: Problems, techniques and applications. *IEEE Transaction on Knowledge and Data Engineering, Sept. 2017*.

S. B. Cooper. 2004. Computability theory. *Chapman Hall CRC Mathematics*.

A. Daniely and S. Shalev-Shwartz. 2014. Optimal learners for multiclass problems. *Proceedings of The 27th Conference on Learning Theory, PMLR 35:287-316, 20144*.

J. Firth. 1957. A synopsis of linguistic theory 1930-1955. *In Studies in Linguistic Analysis, Philological Society, Oxford. Reprinted in Palmer, F. (ed. 1968) Selected papers of J. R. Firth, Longman, Harlow*.

J. Flach and L. Lamb. 2023. A neural lambda calculus: Neurosymbolic ai meets the foundations of computing and functional programming. *arXiv:2304.09276*.

A. Garcez and L. Lamb. 2023. Neurosysmbolic ai: The 3rd wave. *Artificial Intelligence Review, Volume 56, Issue 11*.

Gazdar, Klein, Pullum, and Sag. 1985. Generalized phrase structure grammar. *Harvard University Press, Cambridge*.

E.M. Gold. 1967. Language identification in the limit. *Information and Controls, 10, 447-474 (1967)*.

H. Halpin, P. Hayes, J. McCusker, D. Mcguinness, and H. Thompson. 2010. When owl: sameas isn't the same: An analysis of identity in linked data. *The Semantic Web–ISWC 2010, page 305–320*.

Z. Harris. 1954. Distributional structure. word, 10(2-3): 146-162. *Republished in 2015: ISSN: 0043-7956 (Print) 2373-5112 (Online) Journal homepage: www.tandfonline.com/journals/rwrd20, https://doi.org/10.1080/00437956.1954.11659520*.

P. Jappy and R. Nock. 1998. Pac learning conceptual graphs. *International Conference on Conceptual Structures, 10 August 1998*.

P. Jiang and X. Cai. 2024. A survey of semantic parsing techniques. *Symmetry 2024, 16, 1201. https://doi.org/10.3390/sym16091201*.

V. Kanade. 2023. Computational learning theory - lecture notes. *Computational Learning Theory (ox.ac.uk)*.

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

S.C. Kleene. 1952. Introduction to meta-mathematics. *Ishi Press International, ISBN 0-923891-57-9*.

D. Lin. 1998a. Automatic retrieval and clustering of similar words. *In Proceedings of COLING-ACL, page 768-774, Montreal*.

D. Lin. 1998b. Dependency-based evaluation of minipar. *In Proceedings of the Workshop on Evaluation of Parsing Systems at LREC, Page 317-329, Granada*.

D. Lin and P Pantel. 2001. Discovery of inference rules for question answering. *Natural Language Engineering, 7(4): 343-360*.

M. Liu. 1999. Deductive database languages: Problems and solutions. *ACM Computing Surveys, Vol. 31, No, 1, March 1999*.

S. A. Ludwig, C. Thompson, and K. Amundson. 2009. Performance analysis of a deductive database with a semantic web reasoning engine: Conceptbase and racer. *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*.

M. Minsky. 1991. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. *in Artificial Intelligence at MIT., Expanding Frontiers, Patrick H. Winston (Ed.), Vol 1, MIT Press, 1990. Reprinted in AI Magazine, 1991*.

B. K. Natarajan. 1989. On learning sets and functions. *Machine learning 4, 67-97, 1989*.

M. Nickel and D. Kiela. 2017. Poincaré embeddings for learning hierarchical representations. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems Pages 6341 - 635*.

M. Nickel, V. Tresp, and H.P. Kriegel. 2012. Factorizing YAGO – scalable machine learning for linked data. *WWW2012 – Session: Creating and Using Links between Data Objects. April 2012, Lyon, France*.

J. R. Pierce, J. B. Carroll, E. P. Hamp, Hays D. G, Hockett C, F, A. G. Oettinger, and A. Perlis. 1966. Language and machines, computers in translation and linguistics. *A Report by the Automatic Language Processing Advisory Committee, Division of Behavioral Science, National Academy of Sciences, National Research Council, National Academy of Science, National Research Councile, Publication 1416, 1966*.

H. Poon. 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.*

D. Rohatgi, T. Marwah, Z.C. Lipton, J.Lu, A.Moitra, and A.Risteski. 2024. Towards characterizing the value of edge embeddings in graph neural networks. *Mathematics of Modern Machine Learning (M3L), arXiv:2410.09867v1*.

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

S. Shalev-Shwartz and S. Ben-David. 2014. Understanding machine learning: From theory to algorithms. *Cambridge University Press, 2014*.

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

R. Socher, A. Perelygin, J.Y. Wu, J. chuang, C.D. Manning, A.Y.Ng, and C. Potts. 2013. Recursive deep models for semantics compositionality over a sentiment treebank. *Proceedings of the 2013 Conference on Empirical Methods in Natural Langauge Processing, pages 1631 - 1642*.

L.G. Valiant. 1984. A theory of the learnable. *Communication of the ACM, November 1984, Volume 27, Number 11*.

L.G. Valiant and J.C. Shepherdson. 1984b. Deductive learning, philosophical transactions of the royal society of london. series a. *Mathematical and Physical Sciences. Vol. 312, No. 1522, Mathematical Logic and Programming Languages [Displayed chronologically; published out of order] (Oct. 1, 1984), pp. 441-446 (6 pages)*.

J. Weizenbaum. 1966. Computational linguistics, ELIZA - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*.

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

K. Xu. 2022. A deductive system based on froglingo for natural language processing. *DOI: http://dx.doi.org/10.13140/RG.2.2.13859.32807*.

K. Xu. 2024. Outline of a PAC learnable class of bounded functions including graphs. *The 7th International Conference on Machine Learning and Intelligent Systems (MLIS 2014)*.

K. Xu. 2025. Classes of bounded functions that approximate the lambda calculus are PAC learnable. *Preprint: DOI: 10.13140/RG.2.2.28499.49443*.

K. Xu and B. Bhargava. 1996. An introduction to the Enterprise-Participant data model. *the Seventh International Workshop on Database and Expert Systems Applications, September, 1996, Zurich, Switzerland, page 410 - 417*.

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

A Supplementary data

A separate file is attached in the Data Section, where additional examples are given to demonstrate a user interface that accepts both Froglingo expressions and natural language utterance. 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 gives the processing of constructing the factorial function using NL as an instruction. While natural language understanding and code generation are not new, presenting a symbolic approach here is aimed to demonstrate a sense of feasibility in implementing such a system with a precision we expect.

The file contains a table capturing the discussion. It is in PDF format wrapped in ZIP in order to be able to upload.