Weak hnf but not hnf λ-terms and their finitely measurable properties

Abstract. A closed lambda term M with head normal form (hnf) is said to have infinitely measurable properties, i.e., for all closed lambda terms N, there are an infinite set of non-beta-convertible terms Q such that $M\ N$ can be beta-reduced to Q. Precisely, a hnf represents a partial recursive function that has an infinite co-domain and doesn't map certain inputs to meaningful outputs. In this paper, we show that a closed lambda term M with weak head normal form (whnf) but not hnf has finitely measurable properties, i.e., for all closed lambda terms N, there are a finite set of non-beta-convertible terms Q such that $M\ N$ can be beta-reduced to Q with a size not larger than M. (Precisely, a whnf but not hnf represents a bounded and recursive function that has a finite co-domain and effectively maps an arbitrary input to a output within the finite co-domain.)

Keywords: lambda calculus · head normal form · weak head normal form · computability · bounded function

1 Introduction

We use $\mathbf{H}$ to denote all closed hnfs and $\mathbf{H}^\lambda$ for the closed λ-terms having a hnf, and $\mathbf{W}\backslash\mathbf{H}$ for all closed whnfs without a hnf, and $\mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$ for those closed λ-terms having a whnf but without a hnf ([1] and [4]). We also use $\mathbf{Z}$ for the set of zero-terms.

A closed term $M \in \mathbf{H}$ is in the form of $\lambda x_1...x_n.x_i t_0...t_k$ where $n > 0, k \geq 0$, $1 \leq i \leq n$, and $FV(t_l) \subseteq \{x_j : \ 1 \leq j \leq n\}$ for all l between 0 and k. When M is applied to an arbitrary number of second terms $N \in \Lambda^0$, one term at a time, there are an arbitrary number of third terms Q such that the third terms have hnf and are not β-convertible, i.e., $|\{Q : \ M\ N \twoheadrightarrow_\beta Q, N \in \Lambda^0, Q \in \mathbf{H}\}| = \infty$. We say M is active because of the head variable x_i that can be freely substituted by a second term in an application. Formally, we say $M \in \mathbf{H}$ represents a partial recursive function.

When a term $M \in \mathbf{Z}$, M always keeps in the form of an application. When M is applied to a second term $N \in \Lambda^0$ and $M\ N \twoheadrightarrow_\beta Q$, the size of Q is never be less than the sum of the terms M and N, i.e., $|Q| \geq |M| + |N|$. We say M is inactive. Formally, we say M is "undefined". Here the size of a term M refers to the number of symbols appearing in M.

A closed $M \in \mathbf{W}\backslash\mathbf{H}$ is in the form of $\lambda x_1...x_n.(\lambda y.s)ut_0...t_k$ where $n > 0, k \geq 0$, and $\{FV(t_i) : \ 0 \leq i \leq k\} \cup FV(s) \cup FV(u) \subseteq \{x_j : \ 1 \leq j \leq n\} \cup \{y\}$ for all j between 1 and n. We know such a term has its activeness between those of the inactive zero terms and the active hnf terms. In this paper, we quantify this common sense by showing a term $M \in \mathbf{W}\backslash\mathbf{H}$ is one of the following:

1. M is meaningful, i.e., we say M identifies a finite number of $\mathbf{W}\backslash\mathbf{H}$ terms. When M is applied to an arbitrary number of second terms $N \in \Lambda^0$, one term at a time, there are only a finite number of third terms such that each of the third terms is a $\mathbf{W}\backslash\mathbf{H}$ term and has a size less or equal to the size of M, i.e., $|\{Q : \ M\ N \twoheadrightarrow_\beta Q, N \in \Lambda^0, Q \in \mathbf{W}\backslash\mathbf{H}, |Q| \leq |M|\}$ is finite.
2. M is meaningless like a $\mathbf{Z}$ term, i.e., when a term $M \in \mathbf{W}\backslash\mathbf{H}$ is applied to a second term $N \in \Lambda^0$ and $M\ N \twoheadrightarrow_\beta Q$ where $Q \in \mathbf{W}\backslash\mathbf{H}$, the size of Q is always larger than the size of term M, i.e., $|Q| > |M|$. When M is meaningless, we also say that M doesn't identify any $\mathbf{W}\backslash\mathbf{H}$ terms.

In Section 2, we develop the conclusion only in the context of the standard lambda calculus as the conclusion is directly related to the lambda calculus itself. For the readers who are interested in the applications of the conclusion, we relate the main conclusion of this paper to an extended lambda calculus and the notion of bounded functions developed in [9]. For those readers who are interested in the theory of Possibly Approximately Correct (PAC) learnability, we further related it to the work in [8]. The related work is provided in Section 3.

2 A $\mathbf{W}\backslash\mathbf{H}$ term identifies only a finite set of $\mathbf{W}\backslash\mathbf{H}$ terms

We start from the basic properties of the set $\mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$:

Proposition 1. *$\forall M \in \mathbf{W}\backslash\mathbf{H}$, $\forall \overline{N} \in \overline{\Lambda^0}$, and $M\ \overline{N} \twoheadrightarrow_\beta Q$, then Q cannot have a hnf, i.e., $Q \in (\mathbf{W}^\lambda\backslash\mathbf{H}^\lambda) \cup \mathbf{Z}^\lambda$.*

Proof. A term M is defined to have a hnf, i.e., $M \in \mathbf{H}^\lambda$, if there is a sequence of terms $M_0, ..., M_i \in \Lambda^0$ such that $M\ M_0\ ...\ M_i \twoheadrightarrow_\beta \mathbf{I}$, where $i \geq 0$ and $\mathbf{I} \equiv \lambda x.x$. To differentiate itself from a term with a hnf, a term having a whnf but not a hnf, i.e., $M \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$, must have: $\forall x_i \in \Lambda^0$, $M\ x_0\ ...\ x_i \twoheadrightarrow_\beta Q$, where $Q \not\equiv \mathbf{I}$ and $i \geq 0$. (Otherwise, there would be a $M\ x_0\ ...\ x_i \rightarrow_\beta \mathbf{I}$ for a $i \in \mathcal{N}$, which is a contradiction.)

We further determine: $Q \in (\mathbf{W}^\lambda\backslash\mathbf{H}^\lambda) \cup \mathbf{Z}^\lambda$. (Otherwise, if $Q \in \mathbf{H}^\lambda\backslash\mathbf{I}$, then we would have $M\ N\ \overline{P} \rightarrow_\beta Q\overline{P} \rightarrow_\beta \mathbf{I}$, for a $\overline{P} \in \overline{\Lambda^0}$ and $|\overline{P}| \geq 0$, which is another contradiction.) □

For any $M \in \Lambda^0$, if we know M can be reduced to $N \in \Lambda^0$, there is a standard reduction to reduce M to N (Theorem 11.4.7 of [2]). This theorem says that not all term M can be effectively reduced to N when only the leftmost reduction strategy is used if we know M can be reduced to N. For example, $\Omega(\mathbf{II})$ would never be reduced to $\Omega\mathbf{I}$ if only the leftmost reduction strategy is used, where the zero term $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$. However, if we know M has a normal form, then only the leftmost strategy is sufficient to effectively reduce M to its normal form (Theorem 13.2.2 of [2]). Similarly, we show that if we know that M is reducible to N, where $N \in \mathbf{W}$, then we can effectively reduce M to N using the leftmost reduction strategy only. The strategy of proving the conclusion is not different from the proof for Theorem 13.2.2 provided in [2]: a non leftmost reduction doesn't help to contract the leftmost redex of a $\mathbf{W}^\lambda$ term that must be contracted in order to be reduced to an abstraction.

Proposition 2. *If $M \in \mathbf{W}^\lambda$, $M \twoheadrightarrow_\beta Q$, where $Q \in \mathbf{W}$ and $\twoheadrightarrow_\beta$ is a multi-step β-reduction (in the standard reduction strategy), then the leftmost reduction strategy is sufficient to reduce M to Q, i.e., $M \twoheadrightarrow_l Q$.*

Proof. We prove it by induction on the size of the term M:
Case 1: $M \in \mathbf{W}$. It is true by itself.
Case 2: $M \equiv (\lambda x.M_0)\ M_1$.

1. The leftmost reduction strategy always makes progress in reducing M to Q: $(\lambda x.M_0)\ M_1 \rightarrow_l M_0[M_1/x]$.
   (a) If $M_0[M_1/x] \in \mathbf{W}$. Therefore we have $M \twoheadrightarrow_l Q$.
   (b) If $M_0[M_1/x] \equiv M_{01}\ M_{11}...M_{n1}$. We conclude $M \twoheadrightarrow_l Q$ by induction.
2. A non leftmost reduction strategy doesn't make any progress in reducing M to Q: $(\lambda x.M_0)\ M_1 \rightarrow_{not\ l} (\lambda x.M_0')\ M_1'$, where M_0' and M_1' denote alternated terms of M_0 and M_1 after a non leftmost reduction. Because a non leftmost reduction cannot get rid of the leftmost redex though the components M_0 and M_1 may be altered. To reduce $(\lambda x.M_0')\ M_1'$ to an abstraction, a leftmost reduction must be used. This tells us that if we use a non leftmost reduction, it may be able to reduce M to Q eventually, but it would take more steps before a leftmost reduction must be applied. Therefore, we conclude that a non leftmost reduction is not necessary to reduce an application to an abstraction.
3. Because only a leftmost reduction makes progress and a non leftmost reduction is not essential, and because there is an effective standard reduction strategy to reduce M to Q, we conclude that the leftmost reduction strategy is sufficient to effecctively reduce M to Q.

Case 3 $M \equiv (\lambda x.M_0)\ M_1...M_n$. We assume that $(\lambda x.M_0)\ M_1...M_{n-1}$ has been reduced to $Q' \in \mathbf{W}$. According to Case 2, we prove that $(\lambda x.M_0)\ M_1...M_n \twoheadrightarrow_l Q'\ M_n \twoheadrightarrow_l Q$ by induction. □

Proposition 3. *If $M \in \mathbf{W}^\lambda \backslash \mathbf{H}^\lambda$, $M \twoheadrightarrow_\beta Q$, where $Q \in \mathbf{W}\backslash\mathbf{H}$ and $\twoheadrightarrow_\beta$ is a multi-step β-reduction (in the standard reduction strategy), then the leftmost reduction strategy is sufficient to reduce M to Q, i.e., $M \twoheadrightarrow_l Q$.*

Proof. This is true by following Proposition 2: if $M \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$, i.e., $M \in \mathbf{W}^\lambda$, $M \twoheadrightarrow_l Q$, where $Q \in \mathbf{W}$. We further determine $Q \in \mathbf{W}\backslash\mathbf{H}$ according to the definition of terms having whnf but not hnf. □

Proposition 4. *If $M \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$ and $\forall N \in \Lambda^0$, $M\ N \twoheadrightarrow_\beta Q$, where $Q \in \mathbf{W}$, then*

1. *$Q \in \mathbf{W}\backslash\mathbf{H}$.*
2. *the leftmost reduction strategy is sufficient to reduce $M\ N$ to Q, i.e., $M\ N \twoheadrightarrow_l Q$.*

Proof. 1. It follows Proposition 1.
2. According to Proposition 1, we have $M\ N \twoheadrightarrow_\beta Q'$, where $Q' \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$. Q' cannot be a zero term as Q is not a zero term. Because $Q' \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$, so is $M\ N \in \mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$. Then according to Proposition 3, we have $M\ N \twoheadrightarrow_l Q$. □

Proposition 5. *$\forall M \in \mathbf{W}\backslash\mathbf{H}$, $\forall N \in \Lambda^0$, $M\ N \twoheadrightarrow_l Q$, where $Q \in (\mathbf{W}\backslash\mathbf{H})$, the head redex of each reduction step from $M\ N$ to Q doesn't have N being contracted, i.e., for all $P \equiv R_0R_1...R_n$, where $n > 0$, such that $M\ N \twoheadrightarrow_l P \twoheadrightarrow_l Q$, we have: $N \not\equiv R_0$.*

Proof. If N is contracted, i.e., $\exists P \equiv R_0R_1...R_n$, where $n > 0$ such that $M\ N \twoheadrightarrow_l P \twoheadrightarrow_l Q$ and $N \equiv R_0$, we will be able to choose N to be a hnf, i.e., $N \in \mathbf{H}$, and further choose $R_1, ..., R_n$ for a n such that $M\ N \twoheadrightarrow_l NR_1...R_n \twoheadrightarrow_l \mathbf{I}$. Choosing $N \in \mathbf{H}$ is possible because N is a random term in Λ^0. According to the definition of hnf, such a reduction is possible, a contradiction to Proposition 1. □

When N is not contracted and Q is an abstraction, N must either appear in Q as a substitution instance or never be a substitution instance in Q:

Notation 1 $\forall M \in \mathbf{W}\backslash\mathbf{H}$, $\forall N \in \Lambda^0$, i.e., $M \equiv \lambda x_1...x_n.(\lambda y.P_0)P_1...P_k$, when only the leftmost reduction strategy is used to make the reduction: $(\lambda x_1...x_n.(\lambda y.P_0)P_1...P_k)$ $N \twoheadrightarrow_l Q$, where $Q \in (\mathbf{W}\backslash\mathbf{H})$ and N is never contracted, then we use $[N] \in Q$ to denote that N is a substitution instance remaining in Q if it is, and $[N] \notin Q$ to denote N is not a substitution instance remaining in Q if it isn't.

Sometimes, we simply say N disappears from Q when $[N] \notin Q$. A reduction like $(\lambda xy.y)NR \rightarrow_l R$ is a typical example of making N to disappear.

Note, we avoided to say N is a subterm of Q when $[N] \in Q$ or N is not a subterm in Q when $[N] \notin Q$ because Q may include N as a subterm which is not from a substitution but from M.

Definition 1. *Let $M \in \mathbf{W}\backslash\mathbf{H}$, $\forall N \in \Lambda^0$, and let $\mathcal{I}(M) \equiv \{Q :\ M\ N \twoheadrightarrow_l Q, Q \in (\mathbf{W}\backslash\mathbf{H}), [N] \notin Q\}$, then we say M identifies the set $\mathcal{I}(M)$.*

We are almost ready to show that $\mathcal{I}(M)$ is finite as long as the size $|M|$ is finite. Before doing so, we give examples on what various $\mathcal{I}(M)$ sets look like. We start from the $\mathbf{W}\backslash\mathbf{H}$ term $NULL \equiv \lambda z.(\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx))$ introduced in [9], where $NULL\ N \twoheadrightarrow_l NULL$ for all $N \in \Lambda^0$. Therefore we have $\mathcal{I}(NULL) = \{NULL\}$, a singleton set.

Theorem 5.9 in [9] also shows an element in an Enterprise-Participant (EP) database can be expressed by a $\mathbf{W}\backslash\mathbf{H}$ term. For example, we may have a database $D = \{a\ b := b; b\ a := a; b\ c := c; \}$ that defines a cyclical directed graph with connections from a to b, from b to a, and from b to c, where we have EP reductions $a\ b\ c \twoheadrightarrow_D c, a\ b\ ...\ b \twoheadrightarrow_D b$, and $a\ d \twoheadrightarrow_D d$ for having paths $a\ b\ c$ and $a\ b\ ...\ b$ but not having a path $a\ d$ respectively. When we use $\lambda(a)$, $\lambda(b)$, and $\lambda(c)$ for the corresponding $\mathbf{W}\backslash\mathbf{H}$ terms of a, b, and c respectively, we have $\lambda(a)\ \lambda(b)\ \lambda(c) \twoheadrightarrow_l \lambda(c)$, $\lambda(a)\ \lambda(b)\ ...\ \lambda(b) \twoheadrightarrow_l \lambda(b)$, and $\lambda(a)\ \lambda(d) \twoheadrightarrow_l NULL$, where $NULL$ is in a correspondence of a special EP-term $null$ [1] [2]. Therefore, we have: $\mathcal{I}(\lambda(a)) = \{\lambda(b), NULL\}$, $\mathcal{I}(\lambda(b)) = \{\lambda(a), \lambda(c), NULL\}$, and for the rest of the identifiers: $\mathcal{I}(\lambda(c)) = \mathcal{I}(\lambda(d)) = \ ... \ = \{NULL\}$, where any

[1] Essentially, the lambda term representing a conditional statement like "$if...else\ if...then\ NULL$" [2] plays an important role in defining a $\mathbf{W}\backslash\mathbf{H}$ term that identifies a finite number of $\mathbf{W}\backslash\mathbf{H}$ terms

[2] When we say N disappears from Q after a reduction $M\ N \twoheadrightarrow_l Q$, where $M, Q \in \mathbf{W}\backslash\mathbf{H}$, we will say that N doesn't have an impact to the fact that M identifies a finite set $\mathcal{I}(M)$ and

identifiers like d not in D and D-reduced to $null$ have $NULL$ as the corresponding $\mathbf{W}\backslash\mathbf{H}$ term. Note that a term like $\lambda(a)$ could be better denoted as $\lambda(D, a)$ because it is determined by a specific database instance of D in addition to a being defined in D.

When $M\ N \twoheadrightarrow_l Q$, where $M, Q \in \mathbf{W}\backslash\mathbf{H}$, N may not disappear from Q. An example is developed by slightly modifying the $NULL$ λ-term, and denoted as $NULL^0$:
$NULL^0$
$\equiv (\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))$
$=_\beta \lambda y_1.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))y_1)$
$=_\beta \lambda y_1.(\lambda y_2.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))y_2)y_1)$
...
$=_\beta \lambda y_1.(\lambda y_2.(...(\lambda y_n.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))y_n)...)y_2)y_1)$, for any $n \geq 1$.

When applying $NULL^0$ to other terms:
$NULL^0 N_1 N_2 ... N_n$
$\equiv (\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))N_1 N_2 ... N_n$
$=_\beta \lambda y_1.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))N_1)N_2 ... N_n$
$=_\beta \lambda y_1.(\lambda y_2.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))N_2)N_1)N_3 ... N_n$
...
$=_\beta \lambda y_1.(\lambda y_2.(...(\lambda y_n.((\lambda x.\lambda y.(xxy))(\lambda x.\lambda y.(xxy))N_n)...)N_2)N_1)$, for any $n \geq 1$.

According to Definition 1, we have $\mathcal{I}(NULL^0) = \emptyset$.

Lemma 1. *Let $M \in (\mathbf{W}\backslash\mathbf{H})$ and $\mathcal{I}(M)$ be the set identified by M:*
1. for each $Q \in \mathcal{I}(M)$, $|Q| \leq |M|$
2. If $\mathcal{I}(M) = \emptyset$, then for each $N \in \Lambda_0$ and $M\ N \twoheadrightarrow_l Q$, $|Q| > |M|$.

Proof. A term $M \in \mathbf{W}\backslash\mathbf{H}$ is in the form of $\lambda x_1 ... x_n.(\lambda y.s)ut_0 ... t_k$ where $n > 0, k \geq 0$, and $FV(s) \cup FV(u) \cup \{FV(t_i) : 0 \leq i \leq k\} \subseteq \{x_j : 1 \leq j \leq n\} \cup \{y\}$. When a $N \in \Lambda^0$ is applied to M, we have: $M\ N \equiv (\lambda x_1 ... x_n.(\lambda y.s)ut_1 ... t_k)\ N \rightarrow_l Q \equiv \lambda x_2 ... x_n.(\lambda y.s[N/x_1]u[N/x_1]t_1[N/x_1] ... t_k[N/x_1]$.

1. When N disappears from Q, we have: $s[N/x_1] \equiv s$, $t_i[N/x_1] \equiv |t_i|$, and therefore $|s[N/x_1]| = |s|$, $|t_i[N/x_1]| = |t_i|$ for all $0 \leq i \leq k$.
   (a) When $n > 1$, we have $|\lambda x_2 ... x_n.| = |(\lambda x_1 ... x_n.| - 1$. it is clear that $|Q| < |M|$.
   (b) When $n = 1$, $Q \equiv (\lambda y.s[N/x_1]u[N/x_1]t_1[N/x_1] ... t_k[N/x_1] \equiv (\lambda y.s)ut_1 ... t_k$, where $(\lambda y.s)u$ becomes a new head redex, which is a $\mathbf{W}^\lambda\backslash\mathbf{H}^\lambda$ term. After the contraction, the resulting $s[u/y]$ must be another abstraction $\lambda y'.C[\ ]$, where $C[\ ]$ is a context containing another head redex, y' has a chance to become a variable that is never a symbol in $C[\ ]$, i.e., $\{y'\} \cap FV(C[\ ] = \emptyset$. This means that $\lambda y'$ arises to be additional symbol after the elimination of λx_1 from its contraction. Therefore, when $M\ N \rightarrow_l (\lambda y.s)ut_1 ... t_k \rightarrow_l \lambda y'.C[\ ]t_1 ... t_k$, we have: $|\lambda y'.C[\ ]t_1 ... t_k| = |M|$. For example $NULL \equiv \lambda y.(\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx)) =_\beta (\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx)) \rightarrow_l NULL$,

each $Q \in \mathcal{I}(M)$ is independent from N. However N plays a role of which Q is ended with when $M\ N$ is reduced. Therefore, N should be embedded as a subterm in M, for a role to be syntactically compared with other using the λ-term of comparing syntactical structures of two λ-terms that is defined in [3].

and we have $NULL\ N \equiv (\lambda y.(\lambda x.\lambda y.(xx))(\lambda x.\lambda y.(xx)))\ N \twoheadrightarrow_l NULL$.

We don't need to worry about the size of the term $(\lambda y.s)$ vs. the size of $s[u/y]$: because $(\lambda y.s)$ is a $\mathbf{W}\backslash\mathbf{H}$ term, we have $|s[u/y]| \leq |(\lambda y.s)|$ by induction.

2. When N doesn't disappear from Q, we have at least one subterm P, either s or a t_i for $0 \leq i \leq k$, such that N appears in P, i.e., $[N] \in P$. Because the size of x_1 is 1 and the size of N is at least 2, the substitution $|[N/x_1]| > |x_1|$ and therefore, $|Q| > |M|$.□

Theorem 1. *Let $M \in \mathbf{W}\backslash\mathbf{H}$, $|M|$ is finite, and M identifies a set of terms, i.e., $\mathcal{I}(M) \equiv \{Q :\ M\ N \twoheadrightarrow_l Q, Q \in (\mathbf{W}\backslash\mathbf{H}), [N] \notin Q\}$, then $\mathcal{I}(M)$ is finite.*

Proof. Given a $\mathbf{W}\backslash\mathbf{H}$ term $M \equiv (\lambda x_1...x_n.(\lambda y.s)ut_1...t_k)$ and let $Q \in \mathcal{I}(M)$. For any $N \in \Lambda^0$,
1) if we have:
$M\ N$
$\equiv (\lambda x_1...x_n.(\lambda y.s)ut_1...t_k)\ N$
$\rightarrow_l Q$
$\equiv \lambda x_2...x_n.(\lambda y.s[N/x_1)u[N/x_1]t_1[N/x_1]...t_k[N/x_1]$
$\equiv \lambda x_2...x_n.(\lambda y.s)ut_1...t_k$, (because N disappears from Q),
the last expression $Q \equiv \lambda x_2...x_n.(\lambda y.s)ut_1...t_k$ clearly indicates that Q is one of the symbol permutations of the symbols in M, independently from N. Because $|M|$ is finite, there are only a finite number of permutations that potentially are terms $Q \in \mathcal{I}(M)$. Therefore $\mathcal{I}(M)$ must be finite. □

3 Related work

In [9], an approximation $[\Lambda^0]_s$ to the lambda calculus was developed, where $s \in \mathcal{N}$ denotes a number of steps a partial computation takes to enumerate the closed λ-terms. Such a finite set $[\Lambda^0]_s$ is syntactically converted to a database D in the Enterprise-Participant (EP) data model [10]. Independently from approximations to the lambda calculus, we can construct an EP for certain applications such as a graph as an example given in Section 2.

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

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

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, we require a bounded function be 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 that is defined by a finite number of EP expressions for a directed cyclic graph, as given in Section 2, supports infinite queries for infinite possible paths in the graph.

With the setting of the partial computations for approximations to the lambda calculus, where we can infinitely increase computation steps, it was concluded that EP is semantically equivalent to the lambda calculus because we have a sequence of databases: $D_0, D_1, ...$ and the union of all the databases essentially contains all the properties of the lambda calculus. Consequently, we say the sequence of the bounded functions in the correspondence to $D_0, D_1, ...$ is semantically equivalent to Turing machine as well.

An approximation to the lambda calculus, i.e., an EP database converted from the approximation, has been found to have a space in the lambda calculus itself: an EP database D is expressed in a finite set of weak head normal forms (whnf) without a head normal form (hnf). Given a database with a finite set of elements, denoted as $NF(D)$, there is a corresponding set of λ-terms, denoted as $\lambda(NF(D)) \subset \mathbf{W}\backslash\mathbf{H}$ that interprets the database [3], i.e.,

$$\forall m \in NF(D), \forall n \in NF(D), \exists q \in NF(D), m\ n =_D q \implies M\ N =_\beta Q \quad (1)$$

where $M, N, Q \in \lambda(NF(D)) \subset \mathbf{W}\backslash\mathbf{H}$ in a correspondence to the given $m, n, q \in NF(D)$ respectively, and $=_D$ denotes an equality relation in EP.

The bounded function defined by a database is recursive in the sense that for any $m, n \in NF(D)$, $m\ n$ can be effectively reduced to $q \in NF(D)$ in EP, i.e., $m\ n \twoheadrightarrow_D q$. More generally, for all $m \in NF(D)$ and all $n \in \mathbf{E}$ [4], $m\ n \twoheadrightarrow_D q$, where $q \in NF(D)$.

The corresponding recursiveness in $\lambda(NF(D))$ is in the sense of the following:

1. We know what each element in $\lambda(NF(D))$ looks like because we converted the elements in NF(D) to the corresponding elements in $\lambda(NF(D))$. Therefore for a reduction $M\ N \twoheadrightarrow_\beta Q$, where $M, N, Q \in \lambda(NF(D))$, M is defined to produce Q when it is applied to N.
2. For each $M \in \lambda(NF(D))$, any λ term $M' \in \Lambda^0$ that is β-convertible to M can be effectively reduced to M in the extended lambda calculus introduced in Section 5 of [9], i.e., $M' \twoheadrightarrow_{\beta,NULL,\lambda(D)} M$
3. For each $M \in \lambda(NF(D))$ and for any $N' \in \Lambda^0$ that is not β-convertible to a term $N \in \lambda(NF(D))$, applying M to N' can be effectively reduced to $NULL$, i.e., $MN' \twoheadrightarrow_{\beta,NULL,\lambda(D)} NULL$ in the extended lambda calculus defined in Section 5 of [9].

In Section 2 of this paper, we showed that a $\mathbf{W}\backslash\mathbf{H}$ term identifies a finite set of other $\mathbf{W}\backslash\mathbf{H}$ terms, i.e., let $M \in \mathbf{W}\backslash\mathbf{H}$, $\forall N \in \Lambda^0$, we have $\mathcal{I}(M) \equiv \{Q :\ M\ N \twoheadrightarrow_l Q, Q \in (\mathbf{W}\backslash\mathbf{H}), [N] \notin Q\}$, and $0 \leq |\mathcal{I}(M)| \leq \infty$. Theorem 1 essentially says that a $\mathbf{W}\backslash\mathbf{H}$ terms represent a bounded function and the entirety of its properties can be inventoried in a finite EP database while the properties of a partial recursive function can only be approximated in a finite EP database. To this end, we conclude:

[3] See Theorem 5.6 of [9] for the full description. The description provided here is simplified but sufficient.

[4] $\mathbf{E}$ denotes the set of all EP terms in a correspondence to all the closed λ terms. See more in [9].

Theorem 2. *A function is bounded if and only if it can be represented in a* $\mathbf{W}\backslash\mathbf{H}$ *term.*

What is the sense of recursiveness of $\mathbf{W}\backslash\mathbf{H}$ terms when we may not know the corresponding EP databases? Given a $M \in \mathbf{W}^{\lambda}\backslash\mathbf{H}^{\lambda}$, we randomly choose a $N \in \Lambda^0$ and make a leftmost multi-step reduction $M\ N \twoheadrightarrow_l Q$. According to Proposition 4, we will effectively obtain $Q \in \mathbf{W}\backslash\mathbf{H}$. Further,

1. if we find that N is a substitution occurrence in the resulting Q after the contraction, then we conclude M is not meaningful, or $\mathcal{I}(M) = \emptyset$.
2. if N disappears from Q, then we conclude that M identifies Q. Further by the standard reduction strategy, we can effectively confirm that Q can be β-convertible to a term formed from a permutation of the symbols in M, where the size of a formed term is always not larger than the size of M itself.
3. By remembering all the permutations of the symbols in M, where the size of each permutation is not more than the size of M, we can estimate what are the elements in $\mathcal{I}(M)$.

In [8], it is concluded that a class of bounded functions is Probably Approximately Correct (PAC) learnable. In practice, it means that an EP database can be automatically constructed purely based on sample EP expresses and the constructed database can produce fresh data that are not entered through samples. According to Theorem 2, we have to say that $\mathbf{W}\backslash\mathbf{H}$ terms can be automatically constructed too. Instead of taking $\mathbf{W}\backslash\mathbf{H}$ terms as samples, e.g., $(M_0...N_n, Q)$ when $M_0...N_n \twoheadrightarrow_\beta Q$ for a $n > 0$, we take EP expressions, e.g., $(m_0...n_n, q)$ when $m_0...n_n \twoheadrightarrow_D q$ for a $n > 0$, as samples for a learning algorithm to construct an EP database first and then convert the EP database to the corresponding $\mathbf{W}\backslash\mathbf{H}$ terms using the converted algorithm given in Definition 5.4 of [9].

Given a Turing-complete system, we can analyze each known sub languages and their corresponding functions to find out if any of those is PAC learnable. Some systems, such as Kleene's systems of equations [7] that represents the class of n-ary number-theoretic partial recursive functions, don't have any known subclass PAC learnable [5] [5]. But it doesn't say there doesn't exists another formal system with a sublanguage that is both PAC learnable and semantically equivalent to Turing machine. The EP data model and the $\mathbf{W}\backslash\mathbf{H}$ λ-terms are such languages. Given all the known sublanguages $\mathbf{H}^{\lambda}$, $\mathbf{W}^{\lambda}\backslash\mathbf{H}^{\lambda}$, and $\mathbf{Z}$ that are available in the lambda calculus, we may conclude as a thesis: A class of functions is PAC learnable if and only if a function is bounded.

Acknowledgment Thank Prof. Anselm Blumer for his extensive discussion with the author regarding the mathematical upper bound of Probably Approximately Correct (PAC) learnables in terms of computability.

References

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

[5] A PAC learnable sublanguage is not considered in this work if it is not related to Turing machine. A class of PAC learnable conjunctions of boolean literals [6] is an example.

2. H. P. Barendregt. “The Lambda Calculus - its Syntax and Semantics”. North-Holland, 1984.
3. H.P. Barendregt, Discriminating Coded Lambda Terms, Logic, Meaning and Computation, Springer, 2001.
4. A. Bucciarellia, A. Carrarob, G. Favroa, and A. Salibrab. Graph easy sets of mute lambda terms, Theoretical Computer Science, 629(2016) 51 - 63, 2016.
5. E.M. Gold. Language Identification in the Limit, Information and Controls, 10, 447-474, 1967.
6. M.J. Kearns and U.V. Vazirani. An introduction to computational learning theory, The MIT Press, 1994.
7. S.C. Kleene. Introduction to Meta-Mathematics, Ishi Press International, ISBN 0-923891-57-9, 1952
8. K. Xu. Classes of bounded functions that are semantically equivalent to Turing machine are PAC learnable, the 38th Annual Conference on Learning Theory (COLT 2025) - Theory of AI for Scientific Computing Workshop, June 30–July 4, 2025 in Lyon, France.
9. K. Xu. 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.
10. K. Xu, B. Bhargava. 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.