A new characterization of provably recursive functions of first-order arithmetic is described. Its main feature is using only basic terms, i.e., terms consisting of 0, the successor S and variables in the quantifier rules, namely, universal elimination and existential introduction.

1 Introduction

This paper presents a new characterization of provably recursive functions of first-order arithmetic. We consider functions defined by sets of equations. The equations can be arbitrary, not necessarily defining primitive recursive, or even total, functions. The main result states that a function is provably recursive iff its totality is provable (using natural deduction) from the defining set of equations, with one restriction: only terms consisting of 0, the successor S and variables can be used in the inference rules dealing with quantifiers, namely universal elimination and existential introduction. We call such terms basic.

Provably recursive functions is a classic topic in proof theory [1]. Let $T(e,\vec{x},y)$ be an arithmetic formula expressing that a deterministic Turing machine with a code $e$ terminates on inputs $\vec{x}$ producing a computation trace with code $y$. A function $f$ is a provably recursive function of an arithmetic theory $T$ if

$$T \vdash \forall \vec{x} \exists y T(e,\vec{x},y)$$

for the code $e$ of some Turing machine that computes $f$. In other words, $f$ is provably recursive if the termination of its algorithm is provable in $T$.

The class of provably recursive functions of $T$ can serve as a measure of $T$’s strength. For example, almost all usual functions on natural numbers are provably recursive in Peano Arithmetic (PA). In contrast, when induction is limited to $\Sigma_1$-formulas, the set of provably recursive functions coincides with the set of primitive recursive functions [1]. Studying provably recursive functions is also useful because a function that is computable but not provably recursive in $T$ gives rise to a true formula (1) that is independent of $T$.

In [3], Leivant proposed a characterization of provably recursive function of PA using a formalism for reasoning about inductively generated data called intrinsic theories. The intrinsic theory of natural numbers has a unary data-predicate $N$, which is supposed to mean that its argument is a natural number. Unlike PA, intrinsic theories don’t use functional symbols other than the constructors (0 and S in the case of natural numbers). Thus, provably recursive functions can be characterized using only constructors and the data-predicate. Our result goes in the same direction by additionally replacing the data-predicate with restrictions on quantifier rules.

A deduction system with such restrictions can be considered as a way of reasoning about non-denoting terms. A set of equations $P$ can define non-total functions over natural numbers, and a deduction system with regular quantifier rules has quantified variables ranging over all, not necessarily denoting, terms. For example, a formula $\forall x \exists y f(x) = y$ is trivially provable in a regular system regardless of the
The structure of the paper is the following. In the next section, relevant definitions are given. Sect. 3 shows that provably recursive functions of PA are provably total when quantifier rules are restricted to basic terms, and Sect. 4 proves the converse.

2 Definitions

Let \( P \) be a set of first-order equations. Let \( \mathcal{L} \) be the language of \( P \) plus a constant 0 and a unary functional symbol \( S \) (if they are not already used in \( P \)). The theory \( A[P] \) is a first-order theory with equality in the language \( \mathcal{L} \). The axioms of \( A[P] \) are the universal closures of the equations in \( P \), denoted by \( \forall P \), the separation axioms \( \forall x S(x) \neq 0 \) and \( \forall x, y S(x) = S(y) \rightarrow x = y \), and induction

\[
A[0] \rightarrow \forall x (A[x] \rightarrow A[S(x)]) \rightarrow \forall x A[x]
\]

for all formulas \( A \) in \( \mathcal{L} \). The inference rules are the usual rules of classical natural deduction (see, e.g., [4]) plus the rules of equality:

\[
\begin{align*}
A[t] & \quad t = s \\
A[s] & \quad t = t
\end{align*}
\]

for all formulas \( A \) and terms \( t, s \) in \( \mathcal{L} \) (\( A[s] \) is obtained from \( A[t] \) by replacing some occurrences of \( t \) by \( s \)). The natural deduction rules dealing with quantifiers are shown in Fig. 1. It is easy to see that the rules of equality make it a congruence.

For example, let AM be the usual axioms for addition and multiplication and let PR be the set of standard defining equations for all primitive recursive functions. Then \( A[AM] \) is Peano Arithmetic and \( A[PR] \) is Peano Arithmetic with all primitive recursive functional symbols.

A program is a pair \( (P, f) \) consisting of a set of equations \( P \) and a functional symbol \( f \) occurring in \( P \). (When \( f \) is clear from the context or is irrelevant, we will write \( P \) instead of \( (P, f) \).)
We use programs to define functions using an analog of Herbrand-Gödel computability (see [2, 3]). Given a program $P$, we write $P \vdash E$ if $E$ is an equation derivable from $P$ in equational logic. The rules of equational logic are the following:

1. $P \vdash E$ for every $E \in P$;
2. $P \vdash t = t$ for every term $t$;
3. if $P \vdash E[x]$, then $P \vdash E[t]$ for every term $t$ and a variable $x$;
4. if $P \vdash s[t] = r[t]$ and $P \vdash t = t'$, then $P \vdash s[t'] = r[t']$.

The relation computed by $(P, f)$ is \{$(\bar{n}, m) \mid P \vdash f(\bar{n}) = m$\} (as usual, $\bar{n}$ is a numeral for a number $n$, consisting of $n$ occurrences of $S$ applied to 0). This relation does not have to be a function. Let us call $P$ coherent if $P \not\vdash \bar{m} = \bar{n}$ for two distinct numerals $\bar{m}$ and $\bar{n}$. It is easy to see that the relation computed by a coherent program is a partial function.

However, even for a coherent program $P$ the theory $A[P]$ can be inconsistent because of the separation axioms. This is the case, for example, for $P = \{f(g(0)) = S(g(0)), f(x) = g(0)\}$. Call a program $P$ strongly coherent if $A[P]$ is consistent. It is clear that if a program is strongly coherent, then it is coherent.

Later it will be important that a program containing a functional symbol $f$ corresponding to a primitive recursive function $f$ also contains all defining equations for $f$. Programs that satisfy this property are called full.

A term is called basic if it consists of $0$, $S$ and variables only. A term is called primitive recursive if it is in the language of PR. We write $T \vdash \Gamma \Rightarrow A$ (respectively, $T \not\vdash \Gamma \Rightarrow A$) if there is a classical natural deduction derivation of $A$ from open assumptions $\Gamma$ in $T$ where the eigenterms of the rules of universal elimination and existential introduction (i.e., terms $t$ in the rules $(\forall E)$ and $(\exists I)$ in Fig. 1) are basic (respectively, primitive recursive). If $\Gamma$ is empty, we write $T \vdash A$ or $T \not\vdash A$.

A function $f$ is called provable with basic terms if $f$ is computed by a strongly coherent full program $(P, f)$ and $A[P] \vdash \forall \bar{x} \exists y f(\bar{x}) = y$, and similarly for a function provable with primitive recursive terms.

3 Provable recursive functions are provable with basic terms

In this section, we prove one direction of the main result.

**Lemma 1.**

1. $A[PR] \vdash \forall \bar{x} \exists y f(\bar{x}) = y$ for every functional symbol $f$ from PR.
2. $A[PR] \vdash \forall \bar{x} \exists y t[\bar{x}] = y$ for every primitive recursive term $t[\bar{x}]$.
3. If $A[PR] \vdash A$, then $A[PR] \vdash A$ for every formula $A$.

**Proof.** 1. By induction on the definition of the primitive recursive function $f$ corresponding to the functional symbol $f$. If it is one of the base functions, i.e., zero, addition of one or a projection, then the claim is obvious. Suppose that $f$ is defined by composition, e.g., $f(x) = h(g(x))$. By induction hypothesis, we know that

$$A[PR] \vdash \forall x \exists y g(x) = y$$

and

$$A[PR] \vdash \forall y \exists z h(y) = z$$

(2)

Given $x$, we can use $y$ such that $g(x) = y$ to perform universal elimination on (2) and then use equality rules to derive $\exists z h(g(x)) = z$ and $\exists z f(x) = z$. 


Suppose \( f(\bar{x}, y) \) is defined by primitive recurrence on \( y \). Then it is easy to prove \( \forall y \exists z f(\bar{x}, y) = z \) by induction on \( y \).

2. By induction on \( t \), using point 1 in the induction step.

3. By induction on the derivation, using point 2 for \((\forall E)\) and \((\exists I)\). \[\square\]

**Theorem 2.** All provably recursive functions of \( A[PR] \) are provable with basic terms.

**Proof.** Suppose that \( f(\bar{x}) \) is provably recursive, i.e., \( A[PR] \vdash \forall \bar{x} \exists y T(e, \bar{x}, y) \) for some Turing machine with code \( e \) that computes \( f \). It is well-known that \( T \) is a primitive recursive relation, so we can assume that \( T(e, \bar{x}, y) \) has the form \( g(\bar{x}, y) = 0 \) where \( g \) is the functional symbol for some primitive recursive function \( g \). Let \( h(y) \) be the primitive recursive function that extracts the final result from a computation trace with code \( y \). Since the machine computing \( f \) is deterministic, for each \( \bar{x} \) we have \( g(\bar{x}, y) = 0 \) for exactly one \( y \).

By Lemma 1.3, \( A[PR] \vdash \forall \bar{x} \exists y g(\bar{x}, y) = 0 \). Also, by Lemma 1.1, \( A[PR] \vdash \forall y \exists z h(y) = z \). Let \( P \) be the minimal full program containing equalities from \( PR \) for all primitive recursive functional symbols used in these derivations, plus the following equalities.

\[
f(\bar{x}) = h(k(g(\bar{x}, y), \bar{x}, y)) \\
k(0, \bar{x}, y) = y
\]

The following is an outline of a derivation of \( \forall \bar{x} \exists z f(\bar{x}) = z \) in \( A[P] \). Given some \( \bar{x} \), let \( y \) be such that \( g(\bar{x}, y) = 0 \) and let \( z \) be such that \( h(y) = z \). Then \( k(g(\bar{x}, y), \bar{x}, y) = y \), so \( f(\bar{x}) = h(y) = z \).

It is left to show that \( P \) is strongly coherent and computes \( f \). If \( f \) is interpreted by \( f \) and \( k \) is interpreted by the total function

\[
k(z, \bar{x}, u) = \begin{cases} u & \text{if } z = 0, \\ y & \text{such that } g(\bar{x}, y) = 0 \quad \text{otherwise} \end{cases}
\]

then \( \mathbb{N} \models P \); therefore, \( A[P] \) is consistent. Further, for every \( \bar{m}, n \), if \( f(\bar{m}) = n \) then \( P \vdash f(\bar{m}) = n \). On the other hand, if \( f(\bar{m}) \neq n \), then \( P \nvdash f(\bar{m}) = n \) because \( f \) is total and \( P \) is strongly coherent. \[\square\]

### 4 Functions that are provable with basic terms are provably recursive

To remind, under the assumption \( A[P] \nvdash \forall \bar{x} \exists y f(\bar{x}) = y \) we have to prove that \( f \) is provably recursive according to the definition of Sect. 1, not that \( A[P] \vdash \forall \bar{x} \exists y f(\bar{x}) = y \), which is trivial. We will prove this statement indirectly, using intrinsic theories [3].

The intrinsic theory of natural numbers, \( IT(\mathbb{N}) \), is a first-order theory with equality whose vocabulary has functional symbols \( 0 \), \( S \) and a unary predicate symbol \( \mathbb{N} \). The additional inference rules are:

\[
\begin{align*}
N(0) & \quad N(t) & \quad A[0] & \quad \forall x (A[x] \rightarrow A[Sx]) \\
N(Sr) & \quad A[t]
\end{align*}
\]

The variant of intrinsic theory that we are using, called discrete intrinsic theory and denoted by \( \overline{IT}(\mathbb{N}) \) in [3], also includes the separation axioms. Note that \( \overline{IT}(\mathbb{N}) \) uses regular first-order quantifier rules.

A function \( f \) is called provable in \( \overline{IT}(\mathbb{N}) \) if it is computed by a strongly coherent program \((P, f)\) and \( \overline{IT}(\mathbb{N}), \forall P \vdash \forall \bar{x} (N(\bar{x}) \rightarrow N(f(\bar{x}))) \).

The following theorem is proved in [3].

**Theorem 3.** A function is provably recursive in \( A[PR] \) iff it is provable in \( \overline{IT}(\mathbb{N}) \).
Thus, it is enough to show that functions provable with basic terms are provable in $\mathcal{IT}(\mathbb{N})$. In fact, we can show that functions provable with primitive recursive terms are provable in $\mathcal{IT}(\mathbb{N})$.

Let us introduce some notation. If $A$ is a formula, then $A^N$ denotes $A$ with all quantifiers relativized to $\mathbb{N}$, i.e., having all subformulas of the form $\forall x B$ replaced by $\forall x (N(x) \rightarrow B)$ and all subformulas of the form $\exists x B$ replaced by $\exists x (N(x) \land B)$. If $\Gamma$ is a set of formulas, then $\Gamma^N = \{ A^N \mid A \in \Gamma \}$. If $\bar{x} = x_1, \ldots, x_n$, then $N(\bar{x})$ denotes $N(x_1) \land \ldots \land N(x_n)$.

**Lemma 4.** Let $P$ be a full program and let $t[\bar{x}]$ be a primitive recursive term in the language of $P$. Then $\mathcal{IT}(\mathbb{N}), \forall P \vdash N(\bar{x}) \Rightarrow N(t[\bar{x}])$.

**Proof.** The proof is similar to Lemma 1. For example, to show that a function $f(\bar{x}, y)$ defined by primitive recurrence on $y$ is provable, one needs to use induction on the formula $N(y) \land N(f(\bar{x}, y))$. The fullness of $P$ is necessary to ensure that the induction hypothesis is true of all subterms of $t$. $\square$

**Lemma 5.** Suppose that $P$ is a full program and $\Gamma \cup \{ A \}$ is a set of formulas whose free variables are among $\bar{x}$. If $A[P] \models \Gamma \Rightarrow A$ and all primitive recursive functional symbols in the derivation occur in $P$, then $\mathcal{IT}(\mathbb{N}), \forall P \vdash N(\bar{x}), \Gamma^N \Rightarrow A^N$.

**Proof.** The proof is by induction on the derivation. If $A$ is an axiom of $\mathcal{A}[P]$ other than induction, then $\mathcal{IT}(\mathbb{N}), \forall P \vdash A$ and $A \vdash A^N$. The only other cases that need attention are those dealing with quantifiers and induction.

If $A[t]$ is derived from $\forall y A[y]$, then by induction hypothesis, $\forall y (N(y) \to A^N[y])$ is derivable. Since $t$ is a primitive recursive term in the language of $P$, $N(t)$ is derivable by Lemma 4, so $A^N[t]$ is derivable as well. The case of ($\exists I$) is similar. The cases of ($\forall I$) and ($\exists E$) are also straightforward.

The relativized version of the induction axiom is

$$B^N[0] \to \forall y (N(y) \to B^N[y] \to B^N[Sy]) \to \forall y (N(y) \to B^N[y])$$

It is proved by induction in $\mathcal{IT}(\mathbb{N})$ for the formula $N(y) \land B^N[y]$. $\square$

**Theorem 6.** All functions provable with primitive recursive terms are provably recursive.

**Proof.** Let $f$ be computed by a strongly coherent full program $(P, f)$ and let $A[P] \models \forall \bar{x} \exists y f(\bar{x}) = y$. Then by Lemma 5, $\mathcal{IT}(\mathbb{N}), \forall P \vdash \forall \bar{x} (N(\bar{x}) \to \exists y N(y) \land f(\bar{x}) = y)$. This implies that $\mathcal{IT}(\mathbb{N}), \forall P \vdash \forall \bar{x} (N(\bar{x}) \to N(f(\bar{x})))$, so by Theorem 3, $f$ is provably recursive. $\square$

**Acknowledgments**

I am grateful to Daniel Leivant, Lev Beklemishev and Tatiana Yavorskaya for constructive discussion.

**References**

[1] Samuel R. Buss (1998): *First-Order Proof Theory of Arithmetic*. In: *Handbook of Proof Theory*, chapter II, Studies in Logic and the Foundations of Mathematics 137, Elsevier, pp. 79–147.

[2] Stephen Kleene (1952): *Introduction to Metamathematics*. Wolters-Noordhof, Groningen.

[3] Daniel Leivant (2002): *Intrinsic reasoning about functional programs I: first order theories*. Annals of Pure and Applied Logic 114(1–3), pp. 117–153, doi:10.1016/S0168-0072(01)00078-1.

[4] Anne Sjerp Troelstra & Helmut Schwichtenberg (2000): *Basic proof theory (2nd edition)*. Cambridge University Press, New York, NY, USA.