ALGEBRAS OF FEASIBLE FUNCTIONS

Yuri Gurevich*

The University of Michigan
Computer and Communication Sciences
221 Angell Hall
Ann Arbor, Michigan 48109

ABSTRACT

What happens if we interpret the syntax of primitive recursive functions in finite domains rather than in the (Platonic) realm of all natural numbers? The answer is somewhat surprising: primitive recursiveness coincides with LOGSPACE computability. Analogously, recursiveness coincides with PTIME computability on finite domains (cf. [Sa]). Inductive definitions for some other complexity classes are discussed too.

INTRODUCTION

Given a certain complexity level (LOGSPACE, PTIME, etc.) we are interested in an inductive definition of functions computed within the complexity level. The problem can be seen as an algebraic one: make the functions in question into an algebra i.e. the closure of some initial functions under certain operations. An inductive definition of PTIME computable functions appeared already in [Co]. Later Lind found a similar inductive definition for LOGSPACE computable functions [Li]. In both cases the main operation is a bounded primitive recursion and both authors consider functions from natural numbers to natural numbers.

In contrast, we view computable functions as database queries rather than pure arithmetical objects. For example, the function "the diameter of the connected component of a vertex x in a finite graph G" can be coded into a pure arithmetical object. We prefer, however, to view it as a global function (or a function schema) that becomes an ordinary function in every finite graph G. The global functions are most common in computer science. Any database-dependent function is a global function. A precise definition of global functions is given in §1.

In §2 we adapt the definition of primitive recursiveness to global functions and prove that the primitive recursive global functions are exactly the LOGSPACE computable ones. In §3 we define recursive global functions. Different equivalent definitions of recursive functions on natural numbers may lead to different classes of global functions. We adjust the classical Herbrand-Gödel-Kleene definition to capture all passible recursions over global functions. Then we prove that the recursive global functions are exactly the PTIME computable ones. B.A. Trakhtenbrot has pointed out that his former student Sazonov had already published a version of this theorem [Sa]. Finally, we give in §3 a couple of algebras of PTIME computable functions. To show that our approach can handle some other complexity classes, we mention additional results in §4.

The problem of inductive definition can also be seen as a logical one and a special case of the problem for constructing a language that captures exactly the properties of finite structures (graphs, databases) computable within the given complexity level. Languages that capture PTIME and LOGSPACE in the case of ordered structures are known [Im1, Va, Im2]. They are extensions of first-order logic by certain generalized quantifiers: the least fixed point operator and the deterministic transitive closure respectively. There is a certain tradition in logic to treat functions as orphans (allowing only composition usually) and give all the attention to predicates (by means of boolean operations, quantifiers, generalized quantifiers). Of course, functions can be coded as predicates. It is not too effective however: an ℓ-ary function is coded by $(\ell+1)$-ary predicate. On the other hand, predicates are most efficiently coded by their characteristic functions. We feel that it is worth while to explore functional, rather than predicate, logic as appropriate to computer science where one is interested not only in expressing but also in computing. One advantage of our functional logic (and particularly of our functional logic for PTIME versus first-order logic with the least fixed point operator) is that it is able to express computations in a way that very much preserves bounds on the resources in question.

Gilles Brassard asked about a logic for $NP \cap \text{co-}NP$. Existence of such a logic seems to be quite unlikely if $P \neq NP \cap \text{co-}NP \neq NP$. Under a very liberal definition of a logic, existence of a logic for $NP \cap \text{co-}NP$ implies existence of a complete set for $NP \cap \text{co-}NP$ under many-one PTIME reductions [Gu]. Such a complete set does not exist relative to a certain oracle [Si]. (By the way, there is no complete set A for $NP \cap \text{co-}NP$ under Turing PTIME reductions in the presence of the same oracle.

*Supported in part by NSF grant MCS83-01022.

Otherwise the set $\{(M,x,1^{\ell}):$ M is a PTIME bounded query machine, $x\in\{0,1\}^*$ and M^A accepts x within ℓ steps$\}$ would be a complete set for $NP\cap$co-NP under many-one PTIME reductions. This answers a question in [Si].)

Let us note that the structures under consideration in this paper are essentially linearly ordered. On the one hand it is a natural requirement: computers work with presentations of structures rather than with structures themselves. On the other hand one may naturally be interested in those properties of ordered structures that do not depend on order. We conjecture that if $P\neq NP$ then there is no reasonable logic to capture exactly those properties of ordered structures that are PTIME computable and do not depend on order.

We believe that algebras of feasible functions may be useful for creating new query languages, for creating special purpose programming languages and for proofs by induction. Deep problems like $P=?NP$ cannot be solved simply by translating them into algebra or logic. However, such translations can shed some new light on these problems.

The paper has gained from useful comments of my Michigan colleagues Peter Hinman, Bill Rounds and especially Andreas Blass.

§1. Global predicates and functions

Let U be a nonempty set, ℓ be a natural number and r be a positive natural number. An ℓ-ary predicate on U can be identified with a total function from U^{ℓ} to {True, False}. A partial function f from U^{ℓ} to U^{r} will be called a function on U of arity ℓ and co-arity r. It can be seen as a vector function whose components are functions of co-arity 1.

A vocabulary is a finite list of predicate and function symbols with specifications of the arity of each symbol and the co-arity of each function symbol. A structure S of a vocabulary σ is a nonempty set U (the universe) together with interpretations of all symbols in σ on U. E.g. if σ consists of one binary predicate than σ-structures are directed graphs. The same name is often used for a structure and its universe. The cardinality $|S|$ of a structure S is the cardinality of its universe. We will view structures as inputs for algorithms.

Proviso. The cardinality of any structure under consideration is finite. The universe of any structure S under consideration consists of numbers $0,1,\ldots,|S|-1$.

A global ℓ-ary predicate P of vocabulary σ assigns to each σ-structure S an ordinary ℓ-ary predicate P^S on S. A global function f of vocabulary σ, arity ℓ and co-arity r assigns to each σ-structure S a function f^S on S of arity ℓ and co-arity r. The superscript S will be usually omitted.) Symbols of a vocabulary σ name basic σ-predicates and σ-functions. The difference between basic and nonbasic σ-predicates and σ-functions is intentional: the first provide parts of the input whereas the second provide objects to be computed.

Pseudo-Claim. Let σ be a vocabulary, ℓ be a natural number and Σ be a finite alphabet. If f assigns to each σ-structure S a partial function f^S from S^{ℓ} to Σ then f is a global function. In particular, every global σ-predicate is a global σ-function.

A more general and better definition of global functions would allow additional sorts (types) of variables e.g. the type boolean. For the sake of simplicity we stick for the time being to our definition and choose the following way to make the Pseudo-Claim true: disregard structures of cardinality 1 and code the letters of Σ by tuples of zeroes and ones of the same length r. The given f becomes a σ-function of co-arity r.

We say that a Turing machine (or some other computational device) M computes an ℓ-ary global function f of a vocabulary σ if it satisfies the following condition. Let S be an arbitrary σ-structure and $\bar{a}\in S^{\ell}$. If f^S is defined at $\bar{a}$ then M with input $(S,\bar{a})$ outputs $f(\bar{a})$ and enters a special halting state OK, otherwise it does not enter the state OK. The size of the input is supposed to be at least $|S|$. A global function f is LOGSPACE (PTIME, etc.) computable if there is a LOGSPACE (PTIME, etc.) bounded Turing machine that computes f.

§2. Primitive recursive equals LOGSPACE

We adapt the usual definition of primitive recursiveness to global functions and prove

Theorem 1. A global function is primitive recursive iff it is LOGSPACE computable.

First we describe initial primitive recursive global functions of the empty vocabulary.

Constant functions. For every $\ell\geq 0$ we have an ℓ-ary constant function with value 0 and an ℓ-ary constant function with value End. In particular, 0 and End are individual constants. End is interpreted as $|S|-1$ in each structure S. Hence a function, indentically equal to End, is constant on each structure but its value depends on $|S|$.

Successor functions. For every $\ell \geq 1$ we have an ℓ-ary successor function. If $\bar{x}$ is a vector $(x_1,\ldots,x_\ell)$ of elements of a structure S and some x_i is different from End in S then the successor of $\bar{x}$ is the lexicographically next ℓ-tuple of elements of S. If every x_i is equal to End in S then the successor of $\bar{x}$ is not defined. We will denote the successor of $\bar{x}$ by $\bar{x}+1$. This is especially natural if we view $\bar{x}$ as an ℓ-digit base $|S|$ number $\Sigma x_i |S|^{\ell-i}$. For example, consider the case when $|S|=10$ and $\ell=3$. Then $(2,3,4)+1=(2,3,5)$, $(2,3,9)+1=(2,4,0)$, $(2,9,9)+1=(3,0,0)$, and $(9,9,9)+1$ is not defined.

Projection functions. For every $\ell \geq 1$ and every sequence $1 \leq i_1 < i_2 < \ldots < i_r \leq \ell$ a separate projection function takes $(x_1,\ldots,x_\ell)$ to $(x_{i_1},\ldots,x_{i_r})$.

Second, for every vocabulary σ the basic σ-functions and the characteristic functions of the basic σ-predicates are initial primitive recursive σ-functions too. Third, we describe two operations on global functions.

Composition $f(\bar{x}) := g(h_1(\bar{x}),\ldots,h_m(\bar{x}))$. The values $h_1(\bar{x}),\ldots,h_m(\bar{x})$ are concatenated of course into one vector because the domain of every g^S consists of vectors rather than tuples of vectors.

Primitive recursion.

$$(1)\quad \begin{cases} f(\bar{x},\bar{0}) := g(\bar{x}) \\ f(\bar{x},\bar{t}+1) := h(\bar{x},\bar{t},f(\bar{x},\bar{t})) \end{cases}$$

where $\bar{0}$ is the vector of zeroes of the dimension of $\bar{t}$.

A global function of a vocabulary σ will be called primitive recursive (PR) if it belongs to the closure of the initial primitive recursive σ-functions under composition and primitive recursion.

We may turn our attention to proving Theorem 1 now. The only if implication is easy. We sketch a proof of the if implication. In virtue of the Pseudo-Claim in §1 one may speak about primitive recursive global predicates. We begin with a few easy lemmas.

Lemma 1. Any boolean combination of PR global predicates is PR.

Lemma 2. Suppose that a global function f is defined as follows:

If $P(\bar{x})$ then $f(\bar{x})=g(\bar{x})$ else $f(\bar{x})=h(\bar{x})$. If the global predicate P and the global functions g,h are PR then f is PR.

Lemma 3. A concatenation $(f_1(\bar{x}_1),\ldots,f_m(\bar{x}_m))$ of PR global functions $f_1,\ldots,f_m$ is a PR global function.

Lemma 4. Suppose that global functions $f_1,\ldots,f_m$ are defined by a simultaneous primitive recursion

$$f_i(\bar{x},\bar{0}) := g_i(\bar{x},\bar{0}),$$

$$f_i(\bar{x},\bar{t}+1) := h_i(\bar{x},\bar{t},f_1(\bar{x},\bar{t}),\ldots,f_m(\bar{x},\bar{t}))$$

If the global functions g_i,h_i are PR then so are $f_1,\ldots,f_m$.

Now, let f be a LOGSPACE computable global function of some vocabulary σ. We will show that f is PR. Since f is LOGSPACE computable there is a two-way multihead finite automaton M that computes f. We assume, of course, that inputs $(S,\bar{a})$ are presented in some standard way. We may suppose for simplicity that each basic ℓ-ary predicate P^S is presented on a separate input tape of length $|S|^\ell$ where for each $\bar{x}\in S^\ell$, the truth value of $P^S(\bar{x})$ is coded in cell number $\Sigma x_i |S|^{\ell-i}$. We may suppose also that the components $f_1^S,\ldots,f_r^S$ of a basic function f^S of co-arity r are presented on separate tapes as their respective graph predicates, and that the components of $\bar{a}$ are presented in unary notation on separate tapes. Then every input tape can be described by a global function that is easily definable by cases. A more sophisticated presentation of inputs may require a more sophisticated definition of the global functions describing input tapes.

Let $H_1,\ldots,H_m$ be the heads on input tapes of M. For every $i=1,\ldots,m$ let $Sym_i(\bar{x}_i)$ be the content of cell $\bar{x}_i$ of the tape for H_i. There is a positive integer k such that if f^S is defined then M finds itself in the halting state OK at the moment $|S|^k-1$. Let $\bar{t}$ range over S^k,

$State(\bar{t})$ be the state of M at moment $\bar{t}$. $Head_i(\bar{t})$ be the position of head H_i at moment $\bar{t}$. The global functions $State$ and $Head_i$ are defined by an easy simultaneous induction which uses the compositions $Sym_1(Head_1(\bar{t})),\ldots,Sym_m(Head_m(\bar{t}))$.

Under natural assumptions about the output mechanism of M we have

$$f = \begin{cases} Output(End^k) & \text{if } State(End^k)=OK, \\ End+1 & \text{otherwise,} \end{cases}$$

where $Output(\bar{t})$ is defined by an easy induction from $State(\bar{t})$. Here End^k is the vector of k Ends, and f is not defined if $State(End^k)\neq OK$. □

Remark 1. It is easy to change the syntax of primitive recursive global functions in order to express only total LOGSPACE computable global functions. Set $End+1=0$, $(End,End)+1=(0,0)$, etc. and allow a new operation If ... then ... else ... in order to avoid the consequence $f(\bar{x},\overline{End}+1)=h(\bar{x},\overline{End},f(\bar{x},\overline{End}))$ of the schema (1).

§3. Recursive equals PTIME

One goal of this section is a thesis that a global function is recursive iff it is PTIME computable. The only if implication may be controversial because we have to resolve the question what are arbitrary recursive global function. First we justify the only if implication informally.

Suppose that a global function f is defined by a certain recursion from some global functions that are already known to be PTIME computable. The arguments of f take only polynomially many values. The recursive definition gives a computation where each round provides one or more new values of f in polynomially many steps. Hence all values of f can be computed in a polynomial number of steps.

The characterization of recursive functions on natural numbers by means of primitive recursion and the minimization operator is well-known. Note that the minimization operator turns out to be bounded (by End) in the case of global functions and does not lead out of the class of primitive recursive global functions. If the minimization operator can replace the recursive schema (2) below then PTIME $\subseteq$ LOGSPACE.

Remark 2. For a moment let A be the algebra that is obtained by omitting the constant functions with value End in the definition of the algebra of primitive recursive global functions. It is easy to check that End is not expressible in A but End is expressible in A with the minimization operator and the minimization operator is expressible in the algebra of primitive recursive global functions.

An analogous question, what are general recursive functions in the case of the infinite domain of natural numbers, was very satisfactorily resolved by Herbrand, Gödel and Kleene [Kℓ] (without invoking the Church-Turing thesis). We adapt their ideas to define recursive global functions.

Consider a system E of equations $t_1=s_1,\ldots,t_k=s_k$ where the terms t_i,s_i are composed from the following symbols: 0, End, the signs of successor functions (one successor function for each arity $\ell\geq 1$), individual variables and function symbols of specified arities and co-arities. Schema (1) above gives examples of equations of the sort under consideration. We suppose that the leftmost symbols of the left-side terms t_i are some function symbols $f,f_1,\ldots,f_p$. Let $\ell=arity(f)$ and $r=co\text{-}arity(f)$. Let $g_1,\ldots,g_q$ be the remaining function symbols in E and let σ be the vocabulary $\{g_1,\ldots,g_q\}$.

Given a σ-structure S interprete End and the successor functions as above in §2. View the terms $0,0',\ldots,End$ as names for the corresponding elements of S. In that sense elements of S may appear in equations of E.

The equation system E *recursively defines* f *in* S if for every $\bar{a}\in S^\ell$ there is at most one $\bar{b}\in S^r$ such that the equation $f(\bar{a})=\bar{b}$ is derivable from E and from valid in S equalities $g(\bar{c})=\bar{d}$ by means of

(i) substitution of elements of S for variables, and

(ii) replacement of a term $t(\bar{c})$ without variables by a vector $\bar{d}$ of elements of S provided the equation $t(\bar{c})=\bar{d}$ has already been proved.

Remark 3. This is a least fixed point type definition. A posteriori only very special equation system suffice.

The system E *recursively defines* f if it recursively defines f in every σ-structure. Finally, a global function f is *recursive* if a certain equation system E recursively defines f.

Theorem 2. A global function is recursive iff it is PTIME computable.

Proof. The only if implication is now an easy exercise.

The if implication of Theorem 2 is not controversial and is proved in the same way the if implication of Theorem 1 was proved, but this time

the syntax is used to describe computations of a given Turing machine with read-and-write tapes. This results in a somewhat more complicated simultaneous recursion for the state function, the head position functions and the tape content functions. All the recursions used can easily be merged into one recursion that is readily expressible by equations. We skip details.□

Analyzing the proof of Theorem 2 we arrive at

Theorem 3. For every vocabulary σ, the PTIME computable global σ-functions constitute the closure of the initial primitive recursive σ-functions under composition, primitive recursion and either one of the two following recursions (2) and (3):

$$(2)\quad \begin{cases} f(\bar{x},\bar{0}) := g(\bar{x}), \\ f(\bar{x},\bar{t}+1) := h(\bar{x}, f(\bar{x},\bar{t}), f(pf(\bar{x},\bar{t}),\bar{t})), \end{cases}$$

$$(3)\quad \begin{cases} f(\bar{0}) := g,\ F(\bar{x},\bar{0}) := G(\bar{x}), \\ f(\bar{t}+1) := h(f(\bar{t}),\ F(pf(\bar{t}),\bar{t})), \\ F(\bar{x},\bar{t}+1) := \begin{cases} F(\bar{x},\bar{t}) & \text{if } \bar{x} \neq pf(\bar{t}), \\ H(f(\bar{t}),\ F(\bar{x},\bar{t})) & \text{otherwise,} \end{cases} \end{cases}$$

where in both cases p is a projection.

Theorem 3 gives two algebras of PTIME computable functions. The schema (2) looks simpler than (3). Moreover, an easy trick allows to replace the term $p(f(\bar{x},\bar{t}))$ in (2) by a primitive recursive $q(\bar{t})$. We are concerned however with preserving time bounds. A single recursion of type (2) or (3) with some preceding and succeeding trivialities is used to describe computations of a given multi-head multi-tape Turing machine. The straight-forward evaluation of a recursion of type (2) requires that for each $\bar{t}+1$ the values of $f(\bar{x},\bar{t})$ are updated for each $\bar{x}$. This inflates the time bound. A recursion of type (3) requires minimal updating. Also, it can be restricted without loss of generality to extremely simple functions h and H. As a result, the time bound is very much preserved. It would be nice to have a more elegant recursion than (3) serving the same purpose. Finally, let us mention an important feature of primitive recursion versus recursions (2) and (3): the parameters are not altered.

Remark 4. The change, proposed in Remark 1 above, adjusts Theorems 2 and 3 to the case of total global functions.

§4. A combined restriction on time and space

We say that a function is PTIME-LOG^k-SPACE if it can be computed on a Turing machine under a simultaneous restriction of polynomial time and $\log^k$ space. We say that a function is PTIME-PLOGSPACE if it is PTIME-LOG^k-SPACE for some k. This section gives inductive definitions of all these classes of functions.

We generalize the notion of global functions by introducing a new sort of individual variables, called log-restricted variables. Log-restricted variables range over natural numbers less than or equal to $\log_2|S|$ in each structure S. Defining a global function $f(x_1,\ldots,x_\ell)=(y_1,\ldots,y_r)$ we have to specify now which variables x_i, y_i are log-restricted. Fix a vocabulary σ.

Theorem 4. A global σ-function f is PTIME-PLOGSPACE iff it can be obtained from initial primitive recursive σ-functions by composition, primitive recursion and recursion (3) where all parameters are log-restricted.

Theorem 5. A σ-function f is PTIME-LOG^k-SPACE iff it can be obtained from initial primitive recursive σ-functions by composition, primitive recursion and recursion (3) where $\text{Dimension}(\bar{x}) \leq k$ and all components of $\bar{x}$ are log-restricted.

We could avoid introducing a new sort of individual variables because the following global function is primitive recursive: the length of the binary notation for $x+1$. Recursion (3) can be replaced by recursion (2) in Theorems 4 and 5.

REFERENCES

[Co] A. Cobham, The intrinsic computational difficulty of functions, Logic, Method. and Phil. of Sci., 1964 Internat. Congress, 24-30, North-Holland.

[Gu] Y. Gurevich, Logic tailored for computational complexity, to appear.

[Im1] N. Immerman, Relational queries computable in polynomial time, STOC 1982, 147-152.

[Im2] N. Immerman, Languages which capture complexity classes, STOC 1983, 347-354.

[Kℓ] S.C. Kleene, Introduction to Metamathematics, Van Nostrand, Princeton, NJ, 1952.

[Li] J.C. Lind, Computing in logarithmic space, Manuscript, M.I.T., 1974, 66 pages.

[Sa] V.Y. Sazonov, Polynomial computability and recursivity in finite domains, Elektronische Informationverarbeitung und Kybernetik, 16 (1980), 319-323.

[Si] M. Sipser, On relativization and the existence of complete sets. ICALP 1982, 523-531.

[Va] M. Vardi, Complexity of relational query languages, STOC 1982, 137-146.