THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

Algebras of Feasible Functions

Yuri Gurevich

CRL-TR-21-83

MAY 1983

Computing Research Laboratory
Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

---

[1]All correspondence should be sent to Professor Yuri Gurevich. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

Algebras of feasible functions

Yuri Gurevich, University of Michigan

Introduction. For different complexity levels (PTIME, LOGSPACE, etc.) and an arbitrary set σ of functions we give an inductive definition of the class of functions computable from σ within the complexity level. Our inductive definition for the class of PTIME computable functions is different from and more robust than the inductive definition of Cobham [Co] for the same class. As far as we know, it is the first time that inductive definitions for the other complexity classes have been given.

The idea is to view computable functions as database queries rather than pure arithmetical objects. For example, the function

$f^G(x)$:=the diameter of the connected component of vertex x of graph G

can be coded into a pure arithmetical function. We prefer, however, to view it as a kind of global function (or a function schema) that becomes an ordinary function in each graph. Global functions are defined precisely in §1. The usual definition of primitive recursive functions, adapted to global functions, surprisingly gives exactly LOGSPACE computable global functions, see §2. Recursive global functions appear to be exactly PTIME computable global functions, see §3. To show that our approach can handle some other complexity classes, we mention some more results in §4.

This work is related in spirit to [Im2], and its subject could be called functional (rather than predicate) logic. One advantage of our approach is that computations in context can be expressed in our logic in a way that preserves bounds on the resources in question.

§1. Global predicates and functions

Consider the statement "Vertices x and y of the graph are connected". For each graph this statement becomes a binary predicate. We call such a statement a global predicate (or predicate schema). Analogously, a term (xy)z gives a ternary function for each set with a binary operation on it. It is an example of a global function (or function schema). From this point of view first-order or second-order formulas are global predicates. A relational query is a global predicate or a global function. We will define global predicates and functions formally.

A vocabulary σ is here a finite list of predicate and function symbols with specifications of the arity of each predicate symbol and the arity 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. An ℓ-ary predicate symbol P is interpreted as a predicate $P^S \subseteq U^\ell$. A function symbol f of arity ℓ and co-arity r is interpreted as a function $f^S: U^\ell \to U^r$. (The co-arity r is always positive.) If $\ell=0, r=1$ then the function symbol is interpreted as an individual constant. The same name is used often for a structure and its universe. The cardinality $|S|$ of a structure S is the cardinality of its universe. Here is a typical example: σ consists of one binary predicate symbol, σ-structures are (directed) graphs. We will view structures as inputs for algorithms.

Proviso 1. The term "structure" will refer to structures S of finite cardinality with Universe $(S)=\{0,1,\ldots,|S|-1\}$.

A global ℓ-ary predicate P of vocabulary σ (in short, an ℓ-ary σ-predicate P) assigns to each σ-structure S an ordinary ℓ-ary predicate $P^S \subseteq S^\ell$. A global function f of vocabulary σ (in short, a σ-function f) of arity ℓ and co-arity r assigns to each σ-structure S a function $f^S: S^\ell \to S^r$. (The superscript S will usually be omitted.)

Symbols of a vocabulary σ name so-called basic σ-predicates and σ-functions. The difference between the basic and the other σ-predicates and σ-functions is intentional. The first provide parts of inputs, the second provide objects to be computed.

Pseudo-Claim. Let σ be a vocabulary, ℓ be a natural number and Σ be an alphabet. Let f assign to each σ-structure S a function $f^S: S^\ell \to \Sigma$. Then f is a global function.

Instead of relaxing the definition of global functions we choose (at this stage) the following way to make the Pseudo-Claim true. First, we disregard structures of cardinality 1.

Proviso 2. Only structures of cardinality at least 2 are considered.

Second, we code the letters of Σ by tuples of zeroes and ones of the same length r. The given f becomes a σ-function of arity ℓ and co-arity r.

We use the term "vector" for tuples of elements (of a structure). With each vector $\bar{x}=(x_1,\ldots,x_m)$ of elements of a structure S we associate a number, $S\text{-value}(\bar{x}) := \sum x_i |S|^{m-i}$. To compute the S-value of $\bar{x}$ just view the vector $\bar{x}$ as an $|S|$-adic number. The Pseudo-Claim allows us to see a function $g: \Sigma_0^* \to \Sigma_1^*$ with $|g(u)| \leq |u|^m$ for some m, as a global function. Namely, let's disregard the words in Σ_0^* of length ≤ 1. Then a word $w \in \Sigma_0^*$ can be seen as a structure of cardinality $|w|$ with a unary basic function taking values in Σ_0. Pad $g(w)$ to a word $g_1(w)$ of length $|w|^m$ using a new symbol (the blank). Define $g^w(x_1,\ldots,x_m)$ to be the kth letter of the $g_1(w)$ where k is the w-value of $(x_1,\ldots,x_m)$.

We will say that a computational device computes an ℓ-ary σ-function f if, given a σ-structure S and a vector $\bar{x} \in S^\ell$, the device computes $f^S(\bar{x})$. The size of the input $(S,\bar{x})$ is supposed to be at least $|S|$.

§2. LOGSPACE computable functions

We adapt the usual definition of primitive recursive functions to the case of global functions and show that for each vocabulary σ, LOGSPACE computable σ-functions are exactly primitive recursive σ-functions. First, we describe initial primitive recursive functions of the empty vocabulary.

Individual constants 0 and End, and constant functions Zero(x)=0, END(x)=End. The individual constant End is interpreted as $|S|-1$ in each structure S. Thus, the function END is constant on each structure S but its value depends on $|S|$.

Successor functions ${}^{m}Sc(x_1,\ldots,x_m)$ for $m\geq 1$. If S is a structure, $\bar{x}\in S^m$ and S-value$(\bar{x})<|S|^m-1$ then ${}^{m}Sc(\bar{x})$ is the vector $\bar{y}\in S^m$ with S-value$(\bar{y})$=S-value$(\bar{x})+1$. In this case we will usually write $\bar{x}+1$ instead of ${}^{m}Sc(\bar{x})$. If, on the other hand, $\bar{x}=End^m$ then ${}^{m}Sc(\bar{x})=0^m$.

Projection functions ${}^{m}_{s}P(x_1,\ldots,x_m)=(x_{i_1},\ldots,x_{i_\ell})$ where $1\leq i_1<i_2<\ldots<i_\ell\leq m$ and s is the sequence $i_1,\ldots,i_\ell$.

Second, for every nonempty vocabulary σ the basic functions $f\in\sigma$ and the characteristic functions of the basic predicates $P\in\sigma$ are initial primitive recursive σ-functions. Third, we describe two operations on global functions.

Composition $f(\bar{x}):=g(h_1(\bar{x}),\ldots,h_m(\bar{x}))$. The notation is not perfect here. The values $h_1(\bar{x}),\ldots,h_m(\bar{x})$ should be concatenated into one vector because the domain of every f^S consists of vectors rather than tuples of vectors.

Primitive recursion:

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

Here $\bar{t}+1$ is the successor of $\bar{t}$ and $\bar{t}+1\neq\bar{0}$.

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

Theorem 1. Let σ be an arbitrary vocabulary. A global σ-function is LOGSPACE computable iff it is primitive recursive.

The if direction is easy. The only if direction is proved in the rest of the section. We begin with a few easy lemmas.

Lemma 1. Let $f(\bar{x},y)$ be a global function of some vocabulary σ_1 and let σ_2 be the extension of σ_1 by a new individual constant c. If the σ_2-function $g(\bar{x}):=f(\bar{x},c)$ is PR then f is a PR σ_1-function.

A global σ-predicate will be called PR if its characteristic function is so.

Lemma 2. (i) Any boolean combination of PR σ-predicate.

(ii) Suppose that a σ-function f is defined by cases

$$f(\bar{x}):=\begin{cases} g_1(\bar{x}) & \text{if } P_1(\bar{x}) \\ \dots & \\ g_m(\bar{x}) & \text{if } P_m(\bar{x}). \end{cases}$$

If the σ-functions $g_1,\dots,g_m$ and the σ-predicates $P_1,\dots,P_m$ are PR then f is so.

Lemma 3. Suppose that a global σ-function f is a concatenation of global σ-functions $f_1,\dots,f_m$:

$$f(\bar{x}):=(f_1(\bar{x}_1),\dots,f_m(\bar{x}_m))$$

where each $\bar{x}_i$ is a projection of $\bar{x}$. If $f_1,\dots,f_n$ are PR then f is so, and vice versa.

Lemma 4. Suppose that σ-functions f_1,f_2 are defined by a simultaneous primitive recursion:

$$f_i(\bar{x},\bar{0}):=g_i(\bar{x}),$$
$$f_i(\bar{x},\bar{t}+1):=h_i(\bar{x},\bar{t},f_1(\bar{x},\bar{t}),f_2(\bar{x},\bar{t})).$$

If the global σ-functions g_i,h_i are PR then f_1,f_2 are so.

Now, let f be a LOGSPACE computable σ-function. We will show that f is PR. Due to Lemmas 1 and 3 we can suppose that the arity of f is zero and the co-arity of f is one.

Since f is LOGSPACE computable there is a two-way multihead finite automaton M that computes f. We can suppose the following about inputs S. Each basic predicate P^S is presented on a separate input tape of length $|S|^{\ell}$ where ℓ is the arity of P. The cells of the tape are numbered by (the S-values of) vectors of dimension ℓ. A cell $\bar{x}$ codes the truth-value of $P(\bar{x})$. The components $f_1^S,\ldots,f_r^S$ of a basic function f^S of co-arity r are presented on separate input tapes as their respective graph predicates. The components of the output vector are printed in the unary notation on separate output tapes.

Let $H_1,\ldots,H_m$ be the heads of M. Let $Sym_i(\bar{x}_i)$ be the content of cell $\bar{x}_i$ of the tape for H_i. The functions Sym_i are easily definable by cases. For example, if H_i works on the input tape for some basic predicate P then there are 6 cases determined by the truth-value of $P(\bar{x}_i)$ and by whether cell $\bar{x}$ is leftmost, rightmost or neither. Hence the functions Sym_i are PR. Hence the concatenation $Sym(\bar{x})$ of $Sym_1(\bar{x}),\ldots,Sym_m(\bar{x})$ is PR.

The steps of the computation of f can be numbered by vectors $\bar{t}$ of some dimension k. Let $State(\bar{t})$ be the state of M at moment $\bar{t}$. Let $Head_i(E)$ be the position (cell) of H_i at moment $\bar{t}$, and let $Head(\bar{t})$ be the concatenation of $Head_1(\bar{t}),\ldots,Head_m(\bar{t})$. It is easy to define by cases PR functions α,β such that

$$State(\bar{t}+1)=\alpha[State(\bar{t}),\ Sym(Head(\bar{t}))],$$

$$Head(\bar{t}+1)=\beta[State(\bar{t}),\ Head(\bar{t})].$$

The zero-ary functions $State(0^k)$, $Head(0^k)$ are obviously PR. By Lemma 4, State and Head are PR.

Let $Print(\bar{t})$ express that M prints on the output tape at moment $\bar{t}$. This σ-predicate is easily definable by cases and therefore is PR. Finally, we define a PR σ-function

$$Out(0^k):=0,$$

$$Out(\bar{t}+1):=\begin{cases} Out(\bar{t})+1 & \text{if } Print(\bar{t}),\\ Out(\bar{t}) & \text{otherwise,}\end{cases}$$

and note that $f(\bar{t})=Out(End^k)$. □

§3. PTIME computable functions

We justify the thesis that for each vocabulary σ, PTIME computable σ-functions are exactly recursive σ-functions. Fix σ. PTIME computable σ-functions are closed under many kinds of recursion. For example, let

$$(2)\quad \begin{cases} f(\bar{x},\bar{0}) := g(\bar{x}), \\ f(\bar{x},\bar{t}+1) := h(\bar{x},\bar{t},f(\bar{y}_1,\bar{t}),\ldots,f(\bar{y}_m,\bar{t})) \text{ where } \bar{y}_i := \alpha_i(f(\bar{x},\bar{t})). \end{cases}$$

If the σ-functions g,h and α_i are PTIME computable then f is so: for each $\bar{t}$ compute $f(\bar{x},\bar{t})$ for all $\bar{x}$. Recall that $\bar{x}$ takes only polynomially many values.

Moreover, suppose that a σ-function f is defined by some recursion from PTIME computable σ-functions. The arguments of f take only polynomially many values. It is natural to suppose that the recursive definition gives a computation where each round provides a new value of f in a polynomial number of steps. Hence all values of f can be computed in a polynomial number of steps. In this sense PTIME computable functions are closed under any recursion. (The phenomenon is related to the fact that σ-predicates are closed under the least fixed point operator.)

In order to define recursive σ-functions formally we are looking for a particular recursion schema. An essential difference between recursions (1) and (2) is that the parameters are not altered in (1). A disadvantage of recursion (2) is that for each $\bar{t}+1$ we have to update values $f(\bar{x},\bar{t})$ for each $\bar{x}$. If (2) reflects some computation with $\bar{t}$ coding the steps and if we use (2) to compute f then we end up with a much worse time bound than that for the original computation. The following schema gives minimal updating.

$$(3)\quad \begin{cases} f(\bar{0}) := g,\ F(\bar{x},\bar{0}) := G(\bar{x}), \\ f(\bar{t}+1) := h(f(\bar{t}),\ F(f'(\bar{t}),\bar{t})), \\ F(\bar{x},\bar{t}+1) := \begin{cases} F(\bar{x},\bar{t}) & \text{if } \bar{x}\neq f'(\bar{t}), \\ H(f(\bar{t}),\ F(\bar{x},t)) & \text{otherwise.} \end{cases} \end{cases}$$

Here $f'(\bar{t})$ is the projection of $f(\bar{t})$ on the first Dimension$(\bar{x})$ components.

We will say that a global σ-function is recursive if it belongs to the closure of the initial primitive recursive σ-functions under composition, primitive recursion (1) and recursion (3).

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

The proof is similar to that of Theorem 1. We sketch here only a part that is essentially new. Fix a Turing machine M that computes f. The time (computational steps) can be represented by k-tuple $\bar{t}$ of individual variables. Let $H_0,H_1,\ldots,H_m$ be the heads of M where H_0 works on the working tape and $H_1,\ldots,H_m$ work on input tapes. For $i=0,1,\ldots,m$ let $\mathrm{Head}_i(\bar{t})$ be the position of H_i at time $\bar{t}$, let $\mathrm{State}(\bar{t})$ be the state of M at time $\bar{t}$, and let $\mathrm{Head}(\bar{t})$ be the concatenation of

$$\mathrm{Head}_0(\bar{t}),\ \mathrm{Head}_1(\bar{t}),\ldots,\ \mathrm{Head}_m(\bar{t}),\ \mathrm{State}(\bar{t}).$$

Let $\mathrm{Sym}_0(\bar{x},\bar{t})$ be the symbol in cell $\bar{x}$ of the working tape at time $\bar{t}$. For $i=1,\ldots,m$ let $\mathrm{Sym}_i(\bar{x}_i)$ be the symbol in cell $\bar{x}_i$ of the input tape for H_i. As in the proof of Theorem 1 the functions $\mathrm{Sym}_1,\ldots,\mathrm{Sym}_m$ are PR.

Now use the recursion schema (3) with $f(\bar{t})=\mathrm{Head}(\bar{t})$, $f'(\bar{t})=\mathrm{Head}_0(t)$ and $F(\bar{x},\bar{t})=\mathrm{Sym}_0(\bar{x},\bar{t})$. The corresponding functions h and H are primitive recursive. They use $\mathrm{Sym}_i(\mathrm{Head}_i)$ for $i=1,\ldots,m$ and are easily definable by cases.

§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.

PTIME-PLOGSPACE computable σ-functions are closed under a wide variety of recursions with log-restricted parameters. Consider, for example, the recursion schema (2) and suppose that the components of $\bar{x}$ are log-restricted. If σ-functions g,h,α_i are PTIME-PLOGSPACE computable then f is so.

Theorem 3. 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 4. 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 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 $x+1$.

Concluding remarks. This paper is a continuation of [Gu] where we argue for altering classical logic in order to make it more appropriate to Computer Science. We believe that algebras of feasible functions may be useful for creating new query languages, for creating special purpose programming languages and for proving (or disproving) properties by induction. We do not believe that deep problems such as P=?NP can be solved simply by translating them into algebra or logic. However, such translations can shed some new light on these problems.

Acknowledgements. Papers [Im1], [Va] and discussions with Neil Immerman were useful in my work on the preceeding paper [Gu]. This paper has gained from useful comments of my Michigan colleagues Andreas Blass, Peter Hinman, Jane Kister and Bill Rounds.

References

[Co] A. Cobham, The intrinsic computational difficulty of functions, Proc. 1964 Internat. Congress for Logic, Method. and Phil. of Sciences, North-Holland.

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

[Im1] N. Immerman, Relational queries computable in polynomial time, 14th ACM Sympos. on Theory of Computing, San Francisco, May 1982, 147-152.

[Im2] N. Immerman, Languages which capture complexity classes, 15th ACM Sympos. on Theory of Computing, Boston, April 1983, 347-354.

[Va] M. Vardi, Complexity of relational query languages, 14th ACM Sympos. on Theory of Computing, San-Francisco, May 1982, 137-146.