UNIVERSITY OF OSLO
Department of informatics

Some General Criteria on Equations to Guarantee Poly-time Functions

Vuokko-Helena Caseiro

Research report 224

ISBN 82-7368-138-6
ISSN 0806-3036

November 1996

Some General Criteria on Equations to Guarantee Poly-time Functions

Vuokko-Helena Caseiro
University of Oslo, Department of informatics
P. O. Box 1080 Blindern, N-0316 Oslo, Norway
Tel: +4722852405 Fax: +4722852401 E-mail: vuokko@ifi.uio.no

November 1996

Contents

1 Introduction

We consider equations defining functions on data structures built from constructors, e.g. sorting lists constructed from nullary nil and binary cons

$$\begin{array}{lcl}
\mathsf{quicksort}\,\mathsf{nil} & = & \mathsf{nil} \\
\mathsf{quicksort}\,(\mathsf{cons}\,x\,y) & = & \mathsf{qs}\,y\,x\,\mathsf{nil}\,\mathsf{nil} \\
\mathsf{qs}\,\mathsf{nil}\,z\,l\,r & = & \mathsf{append}\,(\mathsf{quicksort}\,l)\,(\mathsf{cons}\,z\,(\mathsf{quicksort}\,r)) \\
\mathsf{qs}\,(\mathsf{cons}\,x\,y)\,z\,l\,r & = & \mathsf{if\text{-}then\text{-}else}\,(\mathsf{less}\,x\,z)\,(\mathsf{qs}\,y\,z\,(\mathsf{cons}\,x\,l)\,r))\,(\mathsf{qs}\,y\,z\,l\,(\mathsf{cons}\,x\,r)) \\
\mathsf{append}\,\mathsf{nil}\,z & = & z \\
\mathsf{append}\,(\mathsf{cons}\,x\,y)\,z & = & \mathsf{cons}\,x\,(\mathsf{append}\,y\,z) \\
\mathsf{if\text{-}then\text{-}else}\,\mathsf{true}\,x\,y & = & x \\
\mathsf{if\text{-}then\text{-}else}\,\mathsf{false}\,x\,y & = & y
\end{array}$$

or exponentiation on unary numbers built from nullary 0 and unary succ

$$\begin{array}{lcllcl}
\mathsf{exp}\,(\mathsf{succ}\,x) & = & \mathsf{double}\,(\mathsf{exp}\,x) & \mathsf{exp}\,0 & = & \mathsf{succ}\,0 \\
\mathsf{double}\,(\mathsf{succ}\,x) & = & \mathsf{succ}\,(\mathsf{succ}\,(\mathsf{double}\,x)) & \mathsf{double}\,0 & = & 0
\end{array}$$

We know that sorting may be done in polynomial time (poly-time), whereas exponentiation cannot. Is there some way of detecting this by a syntactical consideration of the equations?

In [1], Bellantoni and Cook gave a recursion-theoretic characterization of all poly-time functions on binary numbers. In [3], other data structures are dealt with, but the resulting classes are larger than poly-time[1]. So results are lacking.

In this report we prepare the grounds for ongoing work on syntactical criteria to guarantee that defined functions are poly-time. We want to be very liberal w.r.t. the shape of equations so that natural definitions like quicksort may be included. We start by giving a first simple system, *poly-basic*$_e$, consisting of functions characterized by having polynomial bounds (in the length of the input) 1) on the number of calls to mutually recursive functions and 2) on the length of intermediate results. All *poly-basic*$_e$ functions may be computed eagerly in poly-time.

Then we observe a problem limiting the class of *poly-basic*$_e$ functions - the problem is that in (1) we consider more calls to mutually recursive functions

[1] since the length of a constructor term t is taken as the height of t as a tree

than what is needed (e.g. quicksort isn't *poly-basic*$_e$ since if we compute eagerly we get an exponential number of calls to qs in the fourth equation). So we need to formalize what are the *needed* calls to mutually recursive functions.

We assume that the programmer, when defining a function, additionally can give suggestions about which combinations of arguments may be needed on one call to this function. E.g. for if-then-else, either the first and second, or the first and the third arguments are needed. These combinations may be used to delimit which parts of right-hand sides might be needed, and in particular to delimit which calls to mutually recursive functions might be needed.

We then define a larger class *poly-basic*$_l$ of functions characterized by having polynomial bounds 1) on the number of *needed* calls to mutually recursive functions and 2) on the length of *needed* intermediate results. We show that all functions in *poly-basic*$_l$ are poly-time by computing lazily and observing that then only "needed" things are computed and only once.

2 Preliminaries: Function Definitions

Given three disjoint sets, of variables, of constructors with arity, and of functions with arity greater than zero, respectively, we define *terms* in the usual way: A variable is a term, and if $t_1, \ldots, t_n$ are terms and h is a constructor or a function, then $(h\, t_1 \ldots t_n)$ is a term. Furthermore $(h\, t_1 \ldots t_n)$ is called an *application* with h as *head* and the t_i's as *arguments* of h. s is a *subterm* of t if s is t, or if t is an application $(h\, t_1 \ldots t_m)$ and s is a subterm of some t_i. A *constructor term* is a term built from constructors only. A *ground term* is a term built from constructors and functions only.

Define a *canonical equation system* to be a set of equations such that each function f is defined by

$$(f\ (c\, y_1 \ldots y_m)\, x_2 \ldots x_n) = r$$

where $n \geq 1, m \geq 0$ and there's one equation for each constructor c, where $y_1, \ldots y_m, x_2, \ldots, x_n$ all are different variables and r is a term with variables among $y_1, \ldots y_m, x_2, \ldots, x_n$. We consider only finite systems. All our equations will be in this form. For the rest of this section we will assume that a canonical equation system is given.

If a function g occurs in the right-hand side (rhs) of an equation for f then f *calls* g. If there is a sequence $f_1, f_2, \ldots, f_n$, where $n \geq 1$, of different functions such that f_1 calls f_2, $\ldots$,f_n calls f_1 then each f_i is *recursive*, and every two functions from the sequence are *mutually recursive*.

We set up a directed graph, *the dependency graph* where the nodes are the functions $\{f, g, \ldots\}$, and there's an arc $f \to g$ iff f calls g. Define the binary relation $\stackrel{+}{\to}$ to be the transitive closure of $\to$. Define $f \equiv g$ iff $f \stackrel{+}{\to} g$ and $g \stackrel{+}{\to} f$. Note that $f \stackrel{+}{\to} f$ iff (the function) f is recursive, and $f \equiv g$ iff (the functions) f and g are mutually recursive.

Given a node f, define the *M-set*[2] for f to be $M_f = \{g \mid f \equiv g\} \cup \{f\}$. Define a binary relation $\triangleright$ (written infix) on M-sets S, T by $(S \triangleright T)$ iff $S \neq T$ and there is a node (function) s in S and a node (function) t in T such that $s \to t$. So $\triangleright$ is antisymmetric and by definition it is irreflexive.

The system is *terminating* if when we turn the equations into rewrite rules by orienting them from lhs to rhs, then for any term t, any rewriting sequence of t is finite. The (unique) normal form of a term t is denoted $t!$.

If the system is terminating, then for a given n-ary f in an M-set M and ground terms $X_1, \ldots, X_n$ (henceforth, $\overline{X}$), define the *complete tree of recursive calls* $T_{f\overline{X}}$ *for* f *on* $\overline{X}$: The smallest tree such that

- The root is $(f\, X_1! \ldots X_n!)$, and
- for each node $(g\, Y_1 \ldots Y_m)$ in $T_{f\overline{X}}$, let r be the rhs of the equation such that $(g\, Y_1 \ldots Y_m)$ matches the lhs and let σ be the matching substitution, then for every subterm $(h\, z_1 \ldots z_k)$ of r such that $h \in M$, $(g\, \overline{Y})$ has a child $(h\, (z_1\sigma)! \ldots (z_k\sigma)!)$.

3 The First System: *poly-basic*$_e$

In this section we give two general criteria on equations, $PC1_e$[3] and $PC2_e$, strong enough to ensure that the functions may be computed in poly-time by an eager strategy. $PC1_e$ is a straightforward formalization of the wish that during computation, the number of calls to mutually recursive functions be bound polynomially. $PC2_e$ restricts the length of subterms of rhs. Note that $PC1_e$ guarantees termination.

3.1 Definition and Result

For a constructor term t, *the length of* t, written $|t|$, is the number of constructors in t. If the system is terminating then for a ground term t, $|t|$ is the length of t in normal form.

[2] "M-set" means "set of mutually recursive functions"

[3] "PC" for "polynomially correct", "e" for "eager", see Theorem 1

Whenever we say *polynomial*, we mean a unary, monotone polynomial function p on nonnegative integers, i.e. $p(x) = a_0 + a_1 x + \cdots + a_k x^k$, where $a_i \geq 0$, $k \geq 0$.

Definition 1 (*poly-basic*$_e$) A canonical equation system such that $PC1_e$ and $PC2_e$ hold, is called *poly-basic*$_e$.

PC1$_e$ For every M-set M there is a polynomial P_M such that for any n-ary function $f \in M$, for any ground terms $X_1, \ldots, X_n$, let $T_{f\,\overline{X}}$ be the complete tree of recursive calls for f on $\overline{X}$: The number of nodes in $T_{f\,\overline{X}}$ is bound by $P_M(\sum_{i=1}^{n} |X_i|)$.

PC2$_e$ For every M-set M there is a polynomial Q_M and an integer constant C_M such that for any ground term $(f\,X_1 \ldots X_n)$ where $f \in M$, let r be the rhs of the equation such that $(f\,\overline{X})$ matches the lhs, let σ be the matching substitution, then for any subterm $(g\,t_1 \ldots t_m)$ of r

1. if $g \in M$ then $\sum_{i=1}^{m} |t_i\sigma| \leq \sum_{i=1}^{n} |X_i| + C_M$.
2. if $g \notin M$: $\sum_{i=1}^{m} |t_i\sigma| \leq Q_M(\sum_{i=1}^{n} |X_i|)$.

Theorem 1 (the poly-time theorem for *poly-basic*$_e$) *Let a* poly-basic$_e$ *system be given. For every M-set M there is a polynomial R_M such that given any n-ary function $f \in M$ and given any constructor terms $X_1, \ldots, X_n$: $(f\,\overline{X})$ can be computed in time bound by $R_M(\sum_{i=1}^{n} |X_i|)$ by using an eager rewriting strategy.*

The proof follows easily from that for the *poly-basic*$_l$ system - see that.

3.2 Discussion, Examples

The definition of *poly-basic*$_e$ is rather loose and liberal; we haven't imposed any syntactical restrictions on the form of the canonical equations, and any data structure may be used. We think that quite some equations are *poly-basic*$_e$.

On the other hand, in [1], the Bellantoni-Cook (BC) system is on a strict, primitive recursive form. An advantage with this is that it is easy to see whether given equations satisfy the criteria. And an disadvantage: That not so many equations *do* satisfy the criteria.

As for the definitional strengths of the two systems, consider only those *poly-basic*$_e$ subsystems where definitions of recursive functions are in a general primitive recursive form, i.e. $f\,(c\,\overline{y})\,\overline{x} = h\,\overline{y}\,\overline{x}\,(f\,y_1\,\overline{x}) \ldots (f\,y_m\,\overline{x})$. Note that then $PC1_e$ and $PC2.1_e$ are trivially satisfied, so only $PC2.2_e$ remains

as an requirement. Then we see that the BC system is such a "primitive recursive" subsystem of *poly-basic*$_e$[4] (also $PC2.2_e$ is satisfied). And so since in [1], it is shown that the BC functions characterize the poly-time functions on binary numbers, also in *poly-basic*$_e$ any poly-time function on binary numbers is definable.

In [2] we study polynomially bounded output lengths in order to give suggestions for replacing $PC2.2_l$ by (weaker) syntactical criteria. We use the safe-normal idea from [1] and combine it with linearity requirements on variables and the use of *units* (to be introduced for the very first time in Section 4).

3.2.1 Insert sort is *poly-basic*$_e$

Consider the M-sets $\{\mathsf{insertsort}\}, \{\mathsf{put}\}, \{\mathsf{if\text{-}then\text{-}else}\}$. To satisfy $PC1_e$ use the identity polynomial; to satisfy $PC2.1_e$ use the constant zero; to satisfy $PC2.2_e$, use $Q_{\{\mathsf{insertsort}\}}(x) = Q_{\{\mathsf{if\text{-}then\text{-}else}\}}(x) = x$, $Q_{\{\mathsf{put}\}}(x) = 2x + 5$.

$$
\begin{array}{lcl}
\mathsf{insertsort}\,\mathsf{nil} & = & \mathsf{nil} \\
\mathsf{insertsort}\,(\mathsf{cons}\,x\,y) & = & \mathsf{put}\,(\mathsf{insertsort}\,y)\,x \\
\mathsf{put}\,\mathsf{nil}\,u & = & \mathsf{cons}\,u\,\mathsf{nil} \\
\mathsf{put}\,(\mathsf{cons}\,w\,v)\,u & = & \mathsf{if\text{-}then\text{-}else}\,(\mathsf{less}\,u\,w)\,(\mathsf{cons}\,u\,(\mathsf{cons}\,w\,v))\,(\mathsf{cons}\,w\,(\mathsf{put}\,v\,u))
\end{array}
$$

3.2.2 Exponentiation on Unary Numbers (not *poly-basic*$_e$)

First consider exp in the Introduction - only $PC2.2_e$ is false. Then consider exp' - both $PC1_e$ and $PC2.2_e$ fail to hold.

$$
\begin{array}{lcllcl}
\mathsf{exp}'\,(\mathsf{succ}\,x) & = & \mathsf{add}\ (\mathsf{exp}'\,x)\,(\mathsf{exp}'\,x), & \mathsf{exp}'\,0 & = & \mathsf{succ}\,0 \\
\mathsf{add}\,0\,y & = & y, & \mathsf{add}\,(\mathsf{succ}\,x)\,y & = & \mathsf{succ}\,(\mathsf{add}\,x\,y)
\end{array}
$$

Finally, consider exp'' where only $PC2.1_e$ fails to hold.

$$\mathsf{exp}''\,(\mathsf{succ}\,x)\,y = \mathsf{exp}''\,x\,(\mathsf{double}\,y) \qquad \mathsf{exp}''\,0\,y = y$$

3.2.3 Quick sort is not *poly-basic*$_e$ only because of $PC1_e$

$\mathsf{quicksort}$ is a poly-time function but unfortunately it isn't *poly-basic*$_e$. Any complete tree of recursive calls to the mutually recursive functions $\mathsf{quicksort}$ and qs has exponential size, so $PC1_e$ doesn't hold. Note that it's qs calling itself doubly which is the problem, not qs calling $\mathsf{quicksort}$ doubly.

[4] Composition, i.e. "given g, h, define f by $f\,\overline{x} = h\,(g\,\overline{x})$", is split into many equations in the *canonical* system. Nullary functions are actually lost.

3.3 A Weak Point in *poly-basic*$_e$

The example of quicksort reveals a weak point in *poly-basic*$_e$. $PC1_e$ requires the complete tree of recursive calls to be polynomial-sized. But intuitively we don't need the whole tree. Only one of the alternative recursive calls to qs is actually needed. Both the calls to quicksort are needed but that's ok - because they are on different parts of the input.

So we should instead require that a smaller tree is polynomial-sized. But which parts of the complete tree can be cut away? In the next section we generalize this problem before giving an answer.

4 Needed Parts: Fit Units and Fit rhs Trees

Our goal is to formalize the concepts of "needed input" and "needed parts of rhs", in particular we want to prove Lemma 1 and give Definition 2.

Assume that the programmer has defined an n-ary function f. Then we ask the programmer additionally to suggest some *f-units*. An f-unit is a subset of $\{1, \ldots, n\}$ - with the intuition that each f-unit should indicate an argument combination of f that may be needed to compute one call $(f\,\overline{X})$ for ground terms $\overline{X}$. E.g. for if-then-else (see Introduction) the programmer might suggest the units $\{1, 2\}$ and $\{1, 3\}$ since only arguments 1 and 2 *or* 1 and 3 will ever be needed to compute if-then-else. Formally, $\{1, 2\}$ and $\{1, 3\}$ will be an acceptable collection of if-then-else-units since each unit contains the position 1, and since each of the two rhs for if-then-else is uniquely "covered" by one if-then-else-unit (x is covered by 2, y is covered by 3). Such formally acceptable units will be called *fit units*. Alternatively, $\{1, 2, 3\}$ alone would be ok, but less useful since it doesn't narrow the set of arguments.

Indeed, given a set of units they may be used to delimit which parts of a rhs might be needed: We draw each rhs as a tree in the usual way but in each function node choosing a unit U (among the given ones) and only including the children corresponding to U. Lemma 1 below states that when the given units are fit units, then for each function call $(f\,\overline{X})$ there is a unique such tree τ such that only things occurring in the "instantiated" τ are needed in order to compute $(f\,\overline{X})$.

Note 1 From now on, argument positions, rhs subterms, trees of recursive calls ..., will be "modulo what may be needed". If things seem complicated, try to think of the special case when we call a function f on ground input $X_1, \ldots, X_n$ and *everything* is needed. Then a fit f-unit is just the whole set $\{1, \ldots, n\}$ (the trivial unit), a fit tree is just a subterm of the rhs (thought of as a tree), a fit based tree of recursive calls the complete tree of recursive

calls.

In the following definitions we make precise the concepts of need.

4.1 Definitions

Let a canonical system with equation set E be given. For an n-ary function f, define an *f-unit* to be a (possibly empty) set of f-positions, i.e. a subset of $\{1, \ldots, n\}$.

Given an equation $e : (f\,(c\,y_1 \ldots y_m)\,x_2 \ldots x_n) = r$ in E, and an f-unit u, we define the variable set from e corresponding to u:

$$W_u^e = \{y_j \mid 1 \in u \text{ and } 1 \leq j \leq m\} \cup \{x_i \mid i \in u \text{ and } 2 \leq i \leq n\}$$

E.g. let e_1 and e_2 be the equations for append (in the Introduction), then $W_{\{1\}}^{e_1} = \emptyset, W_{\{1\}}^{e_2} = \{x, y\}$.

In the following, let $\mathcal{F}$ be the family consisting of the sets $S_f^{\mathcal{F}}$, where for each function f in the canonical system, $S_f^{\mathcal{F}}$ is a nonempty set of f-units.

For every equation $l = r$ in E: For any particular occurrence of a subterm t of r, define the set $\tau_{\mathcal{F}}(t)$ of *rhs trees for t* by induction on t: $\tau_{\mathcal{F}}(t)$ is the smallest set such that

- If t is a variable then the tree consisting of the single node (labeled) "t" is in $\tau_{\mathcal{F}}(t)$.
- If $t = (c\,t_1 \ldots t_n)$ where c is a constructor, then for any choice of rhs trees $\tau_1, \ldots, \tau_n$ from $\tau_{\mathcal{F}}(t_1), \ldots, \tau_{\mathcal{F}}(t_n)$ respectively, the tree with root c and immediate subtrees $\tau_1, \ldots, \tau_n$ is in $\tau_{\mathcal{F}}(t)$.
- If $t = (g\,t_1 \ldots t_n)$ where g is a function, then for any choice of nonempty g-unit $u = \{i_1, \ldots, i_p\}$ from $S_g^{\mathcal{F}}$ and for any choice of rhs trees $\tau_{i_1}, \ldots, \tau_{i_p}$ from $\tau_{\mathcal{F}}(t_{i_1}), \ldots, \tau_{\mathcal{F}}(t_{i_p})$ respectively, the tree with root g and immediate subtrees $\tau_{i_1}, \ldots, \tau_{i_p}$ is in $\tau_{\mathcal{F}}(t)$.

For a function f, the set of f-units $S_f^{\mathcal{F}}$ is said to be *adequate* if for every equation e for f with rhs r and for every rhs tree $\tau_r \in \tau_{\mathcal{F}}(r)$, when we let V be the set of variables occurring in τ_r then there is an f-unit $u \in S_f^{\mathcal{F}}$ such that $V \subseteq W_u^e$. We say that u *covers* τ_r. If for every r and τ_r, when $V \neq \emptyset$ there's exactly one f-unit in $S_f^{\mathcal{F}}$ covering τ_r, then $S_f^{\mathcal{F}}$ is said to be *uniquely covering.*

If for every function f, $S_f^{\mathcal{F}}$ is adequate and uniquely covering and moreoever contains the position 1, then $\mathcal{F}$ is a *fit family*, each $S_f^{\mathcal{F}}$ is a *set of fit units*, the units are *fit units*, and the rhs trees are *fit trees.*

Given an equation $f\,(c\,y_1 \ldots y_m)\,x_2 \ldots x_n = r$ in E, particular subterms s, t of r such that t is a subterm of s, a rhs tree $\tau_s \in \tau_{\mathcal{F}}(s)$. Define t *is in* τ_s if either t and s are the same occurrence of a subterm of r, or if for some constructor or function k, $s = (k\,s_1 \ldots s_n)$ and for some immediate subtree τ_{s_i} of τ_s, t is in τ_{s_i}.

Example 1 Example: f receives a list of binary strings and outputs the list where those elements that didn't contain a string with an "s_1" in it, have been removed.

$$\begin{array}{lcllcl} \mathsf{f}\,\mathsf{nil} & = & \mathsf{nil}, & \mathsf{f}\,(\mathsf{cons}\,x\,l) & = & \mathsf{g}\,x\,(\mathsf{cons}\,x\,(\mathsf{f}\,l))\,(\mathsf{f}\,l) \\ \mathsf{g}\,\epsilon\,a\,b & = & b, & \mathsf{g}\,(\mathsf{s}_0\,x)\,a\,b = \mathsf{g}\,x\,a\,b, & \mathsf{g}\,(\mathsf{s}_1\,x)\,a\,b = a & \end{array}$$

Let $\mathcal{F}$ consist of $S^{\mathcal{F}}_{\mathsf{g}} = \{\{1,2\},\{1,3\}\}$ and $S^{\mathcal{F}}_{\mathsf{f}} = \{\{1\}\}$, then $\mathcal{F}$ is a fit family.

Note 2 The unit u consisting of all positions is called *the trivial unit* and $\{u\}$ is always a fit unit. But finding *small* fit units is a nontrivial task. Future work should be to make "completion algorithms" for handling some cases automatically.

4.2 Fit Trees Indicate What Is Needed.

Lemma 1 *Let the following be given: A terminating canonical system, a fit family $\mathcal{F}$, an n-ary function f in the system, ground input $X_1, \ldots, X_n$ to f, and let r be the rhs of the equation such that $(f\,\overline{X})$ matches the lhs. Then there's a unique fit tree $\tau_r \in \tau_{\mathcal{F}}(r)$ such that to compute $(f\,\overline{X})$*

- *from the constructors and functions in r we may only need those that occur in τ_r, and*
- *from the input $X_1, \ldots, X_n$ we may only need X_i such that $i \in U$, where $U \in S^{\mathcal{F}}_f$ is the fit unit covering τ_r.*

So Lemma 1 states that to compute a function f on input $\overline{X}$ there's a unique fit tree τ_r that contains all that's needed (and perhaps more). We will call τ_r *the needed fit tree* and U *the needed fit unit* (w.r.t. f and $\overline{X}$). Note that the lemma only asserts the existence of τ_r, not how to find it, but to us that won't matter.

Now we are able to define a smaller version of the tree of recursive calls by only including those recursive calls that are in needed fit trees:

Definition 2 (fit based tree of recursive calls) Given a terminating canonical system and a fit family $\mathcal{F}$. Let M be an M-set in the system, let $f \in M$

with ground input $X_1, \ldots, X_n$. The *fit based tree of recursive calls* $T_{f\,\overline{X}}$ *for* f *on* $\overline{X}$ is the smallest tree such that

- the root is $(f\,X_1! \ldots X_n!)$, and
- for each node $(g\,Y_1 \ldots Y_m)$ in $T_{f\,\overline{X}}$, let r be the rhs of the equation such that $(g\,\overline{Y})$ matches the lhs, let σ be the matching substitution, let the needed fit tree w.r.t. $(g\,\overline{Y})$ be τ_r, then for every subterm $t = (h\,z_1 \ldots z_k)$ of r such that t is in τ_r[5] and $h \in M$, $(g\,\overline{Y})$ has a child $(h\,(z_1\sigma)! \ldots (z_k\sigma)!)$.

In the quick sort example, we might choose trivial units for all functions except $\{1,2\}, \{1,3\}$ for if-then-else, then in any fit based tree of recursive calls for quicksort and qs, every qs-node has at most one child qs. If we didn't delimit by using fit units, every qs-node would have had two children.

5 The Second System: *poly-basic*$_l$

We modify *poly-basic*$_e$ to talk of only formally *needed* entities and show that the functions definable are computable in poly-time by a lazy strategy.

5.1 The Definition

Definition 3 (*poly-basic*$_l$) A canonical system with a fit family $\mathcal{F}$ and such that $PC1_l$ and $PC2_l$ hold, is called *poly-basic*$_l$.

PC1$_l$ For every M-set M there is a polynomial P_M such that for any n-ary function $f \in M$, for any ground terms $X_1, \ldots, X_n$, let $U \in S_f^{\mathcal{F}}$ be the needed fit f-unit, let $T_{f\,\overline{X}}$ be the fit based tree of recursive calls for f on $\overline{X}$: The number of nodes in $T_{f\,\overline{X}}$ is bounded by $P_M(\sum_{i\in U} |X_i|)$.

PC2$_l$ For every M-set M there is a polynomial Q_M and an integer constant C_M such that for any ground term $(f\,X_1 \ldots X_n)$ where $f \in M$, let $U \in S_f^{\mathcal{F}}$ be the needed f-unit, let r be the rhs of the equation such that $(f\,\overline{X})$ matches the lhs, let σ be the matching substitution, let τ_r be the needed fit tree, then for any subterm $t = (g\,t_1 \ldots t_m)$ of r such that t is in τ_r and the needed g-unit for $t\sigma$ is $V \in S_g^{\mathcal{F}}$:

1. if $g \in M$ then $\sum_{i\in V} |t_i\sigma| \leq \sum_{i\in U} |X_i| + C_M$.
2. if $g \notin M$ then $\sum_{i\in V} |t_i\sigma| \leq Q_M(\sum_{i\in U} |X_i|)$.

[5] This is the only change w.r.t. the definition of the *complete* tree of recursive calls.

quicksort (choosing trivial units except for if-then-else: $\{1,2\},\{1,3\}$) is *poly-basic*$_l$ - we may e.g. use $P_M(x) = x^2$ for the M-set $M = \{\mathsf{quicksort}, \mathsf{qs}\}$. So by Theorem 2 below, quicksort is poly-time.

Note that by choosing trivial units, any *poly-basic*$_e$ system is a *poly-basic*$_l$ system.

5.2 The Poly-time Theorem

Theorem 2 (the poly-time theorem) *Let a* poly-basic$_l$ *system be given. For every M-set M there is a polynomial R_M such that for any n-ary function $f \in M$, any constructor terms $X_1, \ldots, X_n$, let $U \in S_f^{\mathcal{F}}$ be the needed f-unit for f on $\overline{X}$, then $(f\,\overline{X})$ can be computed in time bounded by $R_M(\sum_{i\in U} |X_i|)$ by using a lazy computation strategy.*

Corollary 2 *Moreover (as an addition to Theorem 2): Any n-ary function f that has the trivial unit $\{1,\ldots,n\}$, may be computed eagerly.*

Essentially, our lazy computation strategy *COMP* works by finding the leftmost-outermost redex $(f\,X_1 \ldots X_n)$, then puts X_1 in normal form X_1', then rewrites $(f\,X_1' \ldots X_n)$ one step, then repeats. *COMP* uses sharing, and it marks subterms that have been computed to normal form.

To go in more detail: In *COMP* a ground term $(k\,t_1 \ldots t_n)$, where k is a function or a constructor, is represented by a record with $n+2$ fields, $\langle k; p_1; \ldots; p_n; O\rangle$, where each p_i is a pointer to a record representing t_i and O is either $\circ$ (meaning "unmarked") or $\bullet$ (meaning "marked"). If p is a pointer, then by $p\uparrow$ we mean the record to which p is pointing.

COMP(p) takes as input a pointer p pointing to the unmarked representation of a ground term t and the result is p pointing to the representation of the normal form of t, all records are marked.

algorithm *COMP*(p) ! note: $p\uparrow$ is unmarked
Let $p\uparrow$ be $\langle k; p_1; \ldots, ; p_n; \circ\rangle$.
if k is a constructor **then**
  for every p_i **do** **if** $p_i\uparrow$ is unmarked **then** *COMP*(p_i) **endif** **endfor**
  Mark $p\uparrow$ with $\bullet$. ! note: this is the only way of marking records
else ! note: k is a function
  if $p_1\uparrow$ is unmarked **then** *COMP*(p_1) **endif** ! note: $p_1\uparrow$ is marked
  Let $p_1\uparrow$ be $\langle c; r_1; \ldots; r_m; \bullet\rangle$.
  Use c to decide which equation to apply to $p\uparrow$.
  Replace $p\uparrow$ by a fresh and unmarked representation of the rhs where
  for the variables we use $r_1\uparrow, \ldots, r_m\uparrow, p_2\uparrow, \ldots, p_n\uparrow$.
  COMP(p).

endif
endalgorithm

Proof idea for Theorem 2: We want to compute $(f\,\overline{X})$ and the needed unit is U. We can show that

- The total number of needed fit trees is bound by a polynomial in $\sum_{i\in U}|X_i|$ (since the system is *poly-basic*$_l$).
- If we call *COMP* with a pointer to the unmarked representation of $(f\,\overline{X})$, then in every call *COMP*(p), $p\uparrow$ is an unmarked record with first field containing a k such that k is in a needed fit tree.
- The time it takes to rewrite in one step is bound by some constant.

6 Conclusion

We have studied a class of equations defining functions on constructor data structures and tried to single out some characteristics of those equations that define poly-time functions.

Our main result is the system *poly-basic*$_l$, obtained by, intuitively, putting polynomial bounds on the "length" (*PC1*$_l$) and "width" (*PC2*$_l$) of "needed things" (needed input, needed parts of rhs). To formalize the needed things, we invented the fit units, and we think that this formalization of need is interesting in itself. We showed that by a lazy computation strategy *COMP*, all *poly-basic*$_l$ definable functions are computable in poly-time. Intuitively, in *COMP* we are all the time moving within the group of needed fit trees of which each is guaranteed to be at least as great as what is "really" needed; and the size of this group is polynomially bounded. Moreover, we showed how one may permit a mixed lazy/eager evaluation strategy - those functions that may need *all* their arguments (formally: They have the trivial unit) may just as well be evaluated eagerly.

We think that *poly-basic*$_l$ is very liberal. Many natural examples of equations fit in as they are, just how many depends heavily on how close the formalism of fit units gets to expressing what is "really" needed; this is still to be investigated. We know that any poly-time function on binary numbers is definable in *poly-basic*$_l$ (since the Bellantoni and Cook system is *poly-basic*$_l$ (even *poly-basic*$_e$)). And because *poly-basic*$_l$ is so liberal, in ongoing work we are using it as a handy framework for developing syntactically defined subsystems.

References

[1] S. Bellantoni, S. Cook: *A new recursion-theoretic characterization of the polytime functions*, 24th Annual ACM STOC (1992), 283-293

[2] V.H. Caseiro: *Criticality conditions on equations to ensure poly-time functions*, Research report no 225, ISBN 82-7368-139-4, ISSN 0806-3036, University of Oslo, Department of Informatics.

[3] D. Leivant: *Stratified functional programs and computational complexity*, 20th ACM Symposium on Principles of Programming Languages, 1993

7 Appendix: Proofs

7.1 Proof of Lemma 1

Let $f \in M$ with input $X_1, \ldots, X_n$ be given with appropriate substitution σ and with appropriate equation e with rhs r. We proceed by induction on the longest call sequence of functions starting with f on $X_1, \ldots, X_n$. Basis and step are proved in the same way.

We will show that: For every subterm t of r, there's a unique fit tree $\tau_t \in \tau(t)$ such that to compute $t\sigma$, from the constructors and functions in t we only need those that occur in τ_t, and from $X_1, \ldots, X_n$ we only need X_i such that there is a variable $v \in W^e_{\{i\}}$ such that $v \in \tau_t$.

We proceed by induction on t.

- If t is a variable, let τ_t consist of the single node labeled t.
- If t is $(c\, t_1 \ldots t_m)$ where c is a constructor, then by induction hypothesis there are $\tau_{t_1}, \ldots, \tau_{t_m}$ as wished, so let τ_t be the tree with root c and immediate subtrees $\tau_{t_1}, \ldots, \tau_{t_m}$.
- If t is $(g\, t_1 \ldots t_m)$ where g is a function, then to compute $t\sigma$, by induction hypothesis (since g's call sequence is shorter) the lemma holds so there's a unique fit unit U_g such that from the input $t_1\sigma, \ldots, t_m\sigma$ we only need $t_j\sigma$ such that $j \in U_g$. For each $t_j, j \in U_g$, by induction hypothesis there is a τ_{t_j}, so that to compute $t_j\sigma$, from the constructors and functions in t_j we only need those that occur in τ_{t_j}, and from $X_1, \ldots, X_n$ we only need X_i such that there is a variable $v \in W^e_{\{i\}}$ such that v occurs in τ_{t_j}. So let τ_t be the tree with root g and as immediate subtrees the τ_{t_j}'s. Then τ_t is a fit tree as wished.

Then to compute $(f\, X_1 \ldots X_n)$ we need X_1 and the variables $X_i, i \geq 2$ such that some corresponding variable occurs in the uniquely chosen fit tree τ_r.

By definition of fit units, we know that there's a unique fit unit covering τ_r, and by definition every fit unit contains 1.

7.2 Proof of Theorem 2

Choose some $f \in M$ and $X_1, \ldots, X_n$. Define the tree $\mathcal{T}$ of needed fit trees, where each node is an (extended) fit tree as follows:

- The root of $\mathcal{T}$ is the needed fit tree τ on $(f\, X_1! \ldots X_n!)$, but such that each function node g in τ also contains as information the actual input to g, put in normal form .
- For every node τ_1 in $\mathcal{T}$, for every function node h in τ_1 with actual input $Y_1, \ldots Y_m$, τ_1 has as a child the needed fit tree τ_2 for $(h\, Y_1 \ldots Y_m)$ and such that each function node g in τ_2 also contains as information the actual input to g, put in normal form.

Call $COMP(q)$ with q pointing to the unmarked representation of $(f\, X_1 \ldots X_n)$. By Lemma 3, during this computation we only call $COMP(p)$ with $p\uparrow$ unmarked and $p\uparrow$'s first field containing functions and constructors in trees in $\mathcal{T}$. If $p\uparrow$ is a constructor record then $COMP$ marks it, and if $p\uparrow$ is a function record then $COMP$ throws it away. So we arrive maximally once at each function/constructor in a fit tree τ in T.

Observe that the work associated with one function or constructor node in a fit tree τ in $\mathcal{T}$ takes constant time - in particular there's a constant K such that for any g, any arguments $Y_1, \ldots, Y_k$ the time it takes to rewrite $(g\, Y_1 \ldots Y_k)$ in one step is bound by K.

Finally we show that the size of $\mathcal{T}$ is bound by a polynomial in $\sum_{i \in U} |X_i|$: To each M-set N below or equal to M, assign *levels*: M has level zero and for $N \neq M$, N has level i if in the dependency graph there is a path of length i from M to N. For each level i, let the M-sets on level i be $N_1, \ldots, N_k$, define polynomials $q_i(x) = Q_{N_1}(x) + \cdots + Q_{N_k}$, $p_i(x) = P_{N_1}(x) + \cdots + P_{N_k}$ (where Q_{N_i}, P_{N_i} are from $PC1_l$, $PC2_l$). Let ϕ be the maximal number of functions in any rhs, let C be the sum of all the constants C_M (from $PC2_l$), define polynomials:

$$\begin{array}{lcllcl} P_0(x) & = & P_M(x) & P_{i+1}(x) & = & P_i(x)\, \phi\, p_i(Q_{i+1}(x)) \\ Q_0(x) & = & x & Q_{i+1}(x) & = & q_i(Q_i(x) + CP_i(x)) \end{array}$$

Then by Lemma 4, the number of nodes in $\mathcal{T}$ is bound by a polynomial in $\sum_{i \in U} |X_i|$.

Lemma 3 *Call* COMP *with a pointer to the unmarked representation of* $(f\, X_1 \ldots X_n)$. *For every call* COMP(p) *in this computation, let* $p\uparrow$ *be a record* $(k, p_1, \ldots, p_n, \circ)$. *We have that*

original *If k is among constructors and functions in $f, X_1, \ldots, X_n$ then k is either f or a constructor in an X_i such that $i \in U$.*

introduced *If k has been introduced in a rewriting step of some $(g, q_1, \ldots, q_m, \circ)$: Let τ be the needed fit tree on the term represented by $(g, q_1, \ldots, q_m)$. Then k is in τ and τ is in the tree $\mathcal{T}$.*

Proof of Lemma 3 By induction on the number of *COMP*-calls before our call.
Basis: For the very first *COMP*-call the lemma obviously holds.
Step: We consider each of the possible recursive calls to *COMP* in the algorithm. Assume that $p\uparrow$ is now (at the entrance) a record $\langle k_p; p_1; \ldots; p_n, \circ\rangle$.

We show that the lemma holds for *COMP*(p_1): (similar for $p_2, \ldots, p_n$ when k_p is a constructor). We may assume that $p_1\uparrow$ is now (before the recursive call *COMP*(p_1)) an unmarked record with first field k_{p_1}.

By induction hypothesis the lemma holds for *COMP*(p) - consider the "original" and "introduced"-cases separately:

If k_p is original, then the first field of *COMP*$(p_1)\uparrow$ is original too.

If k_p has been introduced by rewriting some record $\langle g_1; q_1; \ldots; q_m; \circ\rangle$, then k_p is in a needed fit tree τ_{g_1} and τ_{g_1} is in $\mathcal{T}$. When p's record was made, p_1 was erected as part of that record and p_1 must have been set to point to the unmarked record with first field k_{p_1} which it is still pointing to.

We will now pass upwards through the $\mathcal{T}$-branch with τ_{g_1} in it in order to find the needed fit tree in which k_{p_1} is a node.

Start by considering τ_{g_1}. Is the first child of k_p a constructor or a function? If yes, then this child is k_{p_1} so k_{p_1} is in τ_{g_1} in $\mathcal{T}$ - ok. If no, then this child is a variable v. Let U_{g_1} be the g_1-unit covering τ_{g_1}. Then in particular v is covered by a position $i_{g_1} \in U_{g_1}$. We must have that $i_{g_1} \neq 1$ since if $i_{g_1} = 1$ then $p_1\uparrow$ would be marked. By induction hypothesis for g_1, if g_1 isn't the very first f, then g_1 must have been introduced by rewriting a record $\langle g_2; r_1; \ldots; r_k; \circ\rangle$ and g_1 is in the needed fit tree τ_{g_2} and τ_{g_2} is in $\mathcal{T}$.

Then consider τ_{g_2}. Is the i_{g_1}'th child of g_1 a constructor or a function? If yes, then this child is k_{p_1} and ok since we have chosen U_{g_1}. If no, then this child is a variable v. Let U_{g_2} be the g_2-unit covering τ_{g_2}. Then in particular v is covered by a position $i_{g_2} \in U_{g_2}$. We must have that $i_{g_2} \neq 1$ since if $i_{g_2} = 1$ then $p_1\uparrow$ would be marked.

Continue, if necessary, until we have considered $\tau_{g_n}, n \geq 1$ where g_n is the very first f. Then k_{p_1} is an outermost constructor in an X_i such that $i \in U, i \neq 1$.

We show that the lemma holds for the second call *COMP*(p) when k_p is a function: Let l_p be the first field of p's new record. If k_p is the original f then l_p is the root in the root of $\mathcal{T}$. If k_p has been introduced then l_p is the root of a fit tree that is the child of the tree in which k_p is a node. ◯

Lemma 4 (layer size) *Consider* $(f\, X_1, \ldots, X_n)$ *and let* $L = \sum_{i \in U} |X_i|$. *Partition the fit trees in* $\mathcal{T}$ *into disjoint* layers*: For every branch* G *in* $\mathcal{T}$*: Divide into layers as to which* M*-set each fit tree is for. Number the layers successively such that the fit trees for* M *are numbered zero. Define a function* g *to be an* outsider to layer i *if* g *is not in any fit tree in layer* i *and* g*'s rhs fit tree is in layer* i*. Then*

- *The length of all the needed input to any outsider to layer* i *is bound by* $Q_i(L)$.
- *The number of fit trees in layer* i *is bound by* $P_i(L)$.

Proof of Lemma 4 Observe that if a fit tree τ for an M-function is on layer i, then M has level i. By induction on layers.

Layer 0: The first point is obvious (g is the first call to f). For the second point: By $PC1_l$ there are at most $P_M(L)$ calls to M-functions, so ok.

Layer $i+1$: For the first point follow the ancestor-branch in layer i and use the induction hypotheses for layer i, the first point in $PC2_l$ repeatedly and the second point once, and use that all the fit trees along the branch are in an M-set N such that N is on level i.

For the second point: The number of outsider to layer $i+1$ is bound by the total number of function nodes in layer i, that is (by the second induction hypothesis) $P_i(L)\phi$. For every outsider g to layer $i+1$ the number of fit trees below it and in layer $i+1$ is bound by (by $PC1_l$ and the first point) $P_{M_g}(Q_{i+1}(L))$ and M_g is on level $i+1$. ◯

7.2.1 Proof of Theorem 1 and Corollary 2

Proof of Theorem 1: Note that by using trivial units, *poly-basic*$_e$ functions are *poly-basic*$_l$ functions. So *poly-basic*$_e$ functions are poly-time. *poly-basic*$_e$ functions may be computed eagerly in poly-time: Change the given lazy algorithm *COMP*: On call *COMP*(p), where $p\uparrow$ is a function record $\langle g; p_1; \ldots; p_n, \circ\rangle$: Call *COMP*$(p_i)$ for each unmarked p_i (instead of just for p_1). The proof is as before with the only difference that in Lemma 3 we must also prove the lemma for $p_2, \ldots, p_n$ - that's ok since units are trivial.

Proof of Corollary 2: Again change the given lazy algorithm *COMP*: On call *COMP*(p), where $p\uparrow$ is a function record $\langle g; p_1; \ldots; p_n, \circ\rangle$ *such that*

$S_g^{\mathcal{F}} = \{\{1, \ldots, n\}\}$: Call $COMP(p_i)$ for each unmarked p_i (instead of just for p_1).