A syntactical analysis of non-size-increasing polynomial time computation

Klaus Aehlig Helmut Schwichtenberg

Mathematisches Institut
Ludwig-Maximilians-Universität München
E-mail: {aehlig|schwicht}@rz.mathematik.uni-muenchen.de

Abstract

A purely syntactical proof is given that all functions definable in a certain affine linear typed λ-calculus with iteration in all types are polynomial time computable. The proof also gives explicit polynomial bounds that can easily be calculated.

1 Summary

In [6] Hofmann presented a linear type system for non-size-increasing polynomial time computation allowing unrestricted recursion for inductive datatypes. The proof that all definable functions of type $\mathbf{N} \multimap \mathbf{N}$ are polynomial time computable essentially used semantic concepts, such as the set-theoretic interpretation of terms.

We present a different proof of the same result for a slightly modified version of the term system, which uses syntactical arguments only. However, this paper is more than a new proof of an already known result, as the method choosen has several benefits:

- A reduction relation is defined on the term system such that the term system is closed under reduction. Therefore calculations can be done *within* the system. This is in contrast to [6], where no reduction rules are given; instead, the result is obtained via a set theoretic interpretation.
- It is not only shown that every function is polytime, but explicit polynomial bounds are given that can be determined easily for every given term. An example (for insertion sort) is given in 5.10.
- The semantical size measure "minimal upper bound of growth" (the size function in [6, §3.2]) for terms of higher type is replaced by the much simpler, computable, syntactical measure: "number of free variables".
- The role of the $\diamond$-type becomes more transparent, for the following reasons. First, all definable functions obviously are non-size-increasing with respect to our size measure, since there will be no reduction rule introducing new variables. Secondly, the facts that
  - there are no closed terms of type $\diamond$,
  - the successor functions have type $\diamond \multimap \mathbf{N} \multimap \mathbf{N}$,
  - we have a linear typing discipline

  together ensure that ground terms that are large in the usual size measure are also large in this new size measure.
- It seems that the simplicity of the present approach makes it easier to design arithmetical systems where the proofs can be normalised in polynomial time.

2 Introduction and related work

Recent research has provided many characterizations of `Ptime` (the class of all functions computable in polynomial time) by means of appropriate restrictions of the terms in Gödel's T [1, 2, 5]. Consider for example the following functions on unary coded natural numbers[1], which lead to exponential growth:

$$\begin{aligned} \mathtt{double}(0) &= 0 \\ \mathtt{double}(Sx) &= SS\mathtt{double}(x) \end{aligned}$$

$$\begin{aligned} \mathtt{exp}(0) &= S0 \\ \mathtt{exp}(Sx) &= \mathtt{double}(\mathtt{exp}(x)) \end{aligned}$$

[1] Of course, no-one actually uses unary coded numbers when dealing with questions of complexity. However, examples with unary coded numbers are very general ones, as many data structures can be seen as extensions of unary coded numbers: binary numerals are unary ones, if we don't distinguish the two successor functions; lists can be seen as unary coded numbers if we don't care about their elements.

In other words, as unary numbers can be embedded into many commonly used data structures, counterexamples with unary numbers also work with almost any of the interesting data structures.

Approaches based on “predicative recursion” [1, 8] argue that this is due to the way `double` is called: the previous value of an outer recursion (i.e. $\texttt{exp}(x)$) is used as recursive argument of an inner recursion (i.e. is used as argument for `double`).

However, although all *functions* computable in polynomial time can be defined in such a restricted term system, many natural *algorithms* are excluded, particularly if they involve nested recursions. Standard examples are sorting algorithms like insertion sort, which has the same recursive structure as `exp`:

$$\begin{aligned}
\texttt{insert}(a, []) &= [a] \\
\texttt{insert}(a, [b \mid \ell]) &= [b' \mid \texttt{insert}(a', \ell)] \\
& \quad a', b' \text{ a permutation of } a, b \\
& \quad \text{with } b' \leq a'
\end{aligned}$$

$$\begin{aligned}
\texttt{sort}([]) &= [] \\
\texttt{sort}([a \mid \ell]) &= \texttt{insert}(a, \texttt{sort}(\ell))
\end{aligned}$$

Caseiro [3] observed this fact and studied many related examples; her work resulted in some (partially semantic) criteria for algorithms which ensure that they run in polynomial time. In the example of insertion sort the reason why this recursion does not lead to exponential growth is that `insert` does not increase the size of its input. Hofmann [6] took up this line of research and formulated a new term system with a special type $\Diamond$ of “coins”, which only allows to define non-size-increasing `Ptime` functions, but can also accomodate nested recursions. The basic idea is that whenever a function increases the size of its argument (like a binary successor function), then one has to “pay” for it by inputting a coin (i.e. a term of type $\Diamond$).

The present note can be seen as a straightening of Hofmann's [6]. We present a new proof of his main result, which apart from being simpler provides more insight into what actually is going on, and in particular provides an explicit construction of the polynomials. We have listed what we view as advantages of our proof in the summary above. It is our hope that this simplification will make further progress easier.

3 Types and terms

Types are built from the base types $\mathbf{N}$ (to be interpreted as binary coded natural numbers), bool, and a special type $\Diamond$. This type may be interpreted as a pointer to free memory, as in [4]. From a more technical point of view, lemma 5.3 will show that there are no closed terms of this type. Therefore this type can be used to ensure that terms contain free variables. For example the type $\Diamond \multimap \mathbf{N} \multimap \mathbf{N}$ of the successor functions together with the linear typing discipline will make sure that the usual length measure for natural numbers and the more technical measure “number of free variables” will be the same. As the term system is based on the intuition that length is expressed by the number of free variables, the interesting terms are not only closed terms, but more importantly “almost closed” terms, i.e. terms with free variables of type $\Diamond$ only.

Definition 3.1 (Finite linear types). The set of linear types is defined inductively as

$$\tau, \rho ::= \Diamond \mid \mathbf{N} \mid \mathrm{bool} \mid \tau \multimap \rho \mid \tau \otimes \rho \mid \tau \times \rho \mid \mathbf{L}(\tau)$$

$\mathbf{L}(\tau)$ denotes the list type. $\tau \otimes \rho$ and $\tau \times \rho$ can both be interpreted as ordered pairs; however, the meaning of linearity for those types is different: from a tensor-product $\tau \otimes \rho$ both components can be used once, whereas in the case of an ordinary pair, the pair itself can be used only once, i.e. one has to choose which component of the pair is to be used. As can be seen from 4.3, $\otimes$ corresponds to $\multimap$, whereas $\times$ corresponds to bool.

Terms are built from variables $x, y, z, \ldots$ and constructor symbols c. Each variable has a type and it is assumed that there are infinitely many variables of each type. The notation x^τ should express that the variable x has type τ.

Definition 3.2 (Terms). The set of terms is inductively defined by

$$t, s ::= x^\tau \mid c \mid \lambda x^\tau\, t \mid \langle t, s \rangle \mid ts \mid \{t\} \mid \{t, s\}$$

These terms should be seen as our “raw syntax”; only correctly typed terms (in the sense of definition 3.4) represent meaningful functions.

The idea for the $\{\cdot\}$ and $\{\cdot, \cdot\}$ term constructs is taken from [7]. Terms of this form appear only as arguments in an $\mathbf{N}$ or $\mathbf{L}(\tau)$ elimination. This explicit marking of the step terms in iteration constructs is not only technically convenient but also allows one to see the time complexity of a term at first glance: the constant coefficient of the polynomial defining the run-time is simply the number of symbols outside any braces, the coefficient for X is number of symbols within one pair of braces and so on. See definition 5.5 for details.

The notations $t\vec{t}$ and $\lambda \vec{x}^{\vec{\tau}}\, t$ are defined as usual, so $\lambda x^\tau, y^\rho\, t$ is an abbreviation for $\lambda x^\tau (\lambda y^\rho\, t)$. Terms that only differ in the naming of bound variables are identified.

Definition 3.3 (Constructor symbols). The constructor symbols and their types are

$\mathbf{t}$	bool
$\mathbf{f}$	bool
$\mathbf{1}$	$\mathbf{N}$
$\mathbf{S}_0$	$\Diamond \multimap \mathbf{N} \multimap \mathbf{N}$
$\mathbf{S}_1$	$\Diamond \multimap \mathbf{N} \multimap \mathbf{N}$
$\mathbf{nil}_\tau$	$\mathbf{L}(\tau)$
$\mathbf{cons}_\tau$	$\Diamond \multimap \tau \multimap \mathbf{L}(\tau) \multimap \mathbf{L}(\tau)$
$\otimes_{\tau,\rho}$	$\tau \multimap \rho \multimap \tau \otimes \rho$

$\mathbf{1}$ is to be interpreted as the natural number one and the successor functions $\mathbf{S}_i$ as the usual binary successor functions, i.e. $x \mapsto 2 \cdot x + i$.

A *context* is a set of variables. For two contexts Γ_1 and Γ_2 the notation Γ_1, Γ_2 stands for the union, expressing that Γ_1 and Γ_2 are disjoint. For contexts consisting of one element we also write x^τ instead of $\{x^\tau\}$.

The next definition states which terms of our raw syntax are correctly typed. $\Gamma \vdash t^\tau$ is to be read as "t is a typed term of type τ with free variables in Γ".

The typing system is based on elimination rules. Therefore for every type different to $\Diamond$ there is a rule describing how to use terms of this type; in all these rules the elimination is written as (formal) application. For the right hand side of such an application a notation is choosen that expresses best the computational behaviour of the terms used for the elimination. This avoids duplication of syntax, e.g. with the pair $\langle t, s\rangle$ we have a notation expressing that exactly one of the terms t and s will be needed. Using this syntax also for "if ... then ... else ... " avoids introducing another notation with exactly the same computational meaning.

"Data types", i.e. types where it is possible to retrieve all the stored information by a single use of an object of this type, are introduced by constants. Only "abstractions" have special introduction rules. (In the system we have two forms of abstraction: the λ-abstraction and the pair.) This way of introduction allows the relatively simple definition 4.4 of the term closure of reductions expressing that we may reduce within data, but not under abstractions.

Definition 3.4 **($\Gamma \vdash t^\tau$).** The relation $\Gamma \vdash t^\tau$ is inductively defined as follows:

$$\frac{}{\Gamma, x^\tau \vdash x^\tau} \qquad (Var)$$

$$\frac{c \text{ of type } \tau}{\Gamma \vdash c^\tau} \qquad (const)$$

$$\frac{\Gamma \cup \{x^\tau\} \vdash t^\rho}{\Gamma \vdash (\lambda x^\tau\, t)^{\tau \multimap \rho}} \qquad (\multimap^+)$$

$$\frac{\Gamma_1 \vdash t^{\tau \multimap \rho} \qquad \Gamma_2 \vdash s^\tau}{\Gamma_1, \Gamma_2 \vdash (ts)^\rho} \qquad (\multimap^-)$$

$$\frac{\Gamma \vdash t^\tau \qquad \Gamma \vdash s^\rho}{\Gamma \vdash \langle t, s\rangle^{\tau \times \rho}} \qquad (\times^+)$$

$$\frac{\Gamma \vdash t^{\tau \times \rho}}{\Gamma \vdash (t\mathbf{t})^\tau} \qquad (\times_1^-)$$

$$\frac{\Gamma \vdash t^{\tau \times \rho}}{\Gamma \vdash (t\mathbf{f})^\rho} \qquad (\times_0^-)$$

$$\frac{\Gamma_1 \vdash t^{\mathrm{bool}} \qquad \Gamma_2 \vdash s^\tau \qquad \Gamma_2 \vdash r^\tau}{\Gamma_1, \Gamma_2 \vdash (t\, \langle s, r\rangle)^\tau} \qquad (\mathrm{bool}^-)$$

$$\frac{\Gamma_1 \vdash t^{\tau \otimes \rho} \qquad \Gamma_2, x^\tau, y^\rho \vdash s^\sigma}{\Gamma_1, \Gamma_2 \vdash (t(\lambda x^\tau, y^\rho\, s))^\sigma} \qquad (\otimes^-)$$

$$\frac{\Gamma \vdash t^{\mathbf{N}} \qquad \emptyset \vdash {s_i}^{\Diamond \multimap \tau \multimap \tau}}{\Gamma \vdash (t\, \{s_0, s_1\})^{\tau \multimap \tau}} \qquad (\mathbf{N}^-)$$

$$\frac{\Gamma \vdash t^{\mathbf{L}(\tau)} \qquad \emptyset \vdash s^{\Diamond \multimap \tau \multimap \rho \multimap \rho}}{\Gamma \vdash (t\, \{s\})^{\rho \multimap \rho}} \qquad (\mathbf{L}(\tau)^-)$$

It is crucial that we require the step terms in $(\mathbf{N}^-)$ and $(\mathbf{L}(\tau)^-)$ to be closed. Otherwise it would not be possible to define reduction rules in such a way that the system is closed under reduction, since free variables in the step terms would be duplicated otherwise.

Immediately obvious from the definition is

Proposition 3.5 (Weakening). $\Gamma \vdash t^\tau;\ \Gamma \subset \Gamma' \Longrightarrow \Gamma' \vdash t^\tau$.

The notation t^τ should express that there is a Γ such that $\Gamma \vdash t^\tau$. The smallest such Γ is called the set of free variables. This is a meaningful definition; more precisely, by induction on the definition of $\Gamma \vdash t^\tau$, one easily verifies

Lemma 3.6. *For each term t there is at most one type τ such that t^τ. In this case there is a smallest Γ with $\Gamma \vdash t^\tau$.*

The fact that all eliminations are written as applications ensures that all typed terms have a uniform appearance, as can easily be verified by induction on the definition of the $\cdot \vdash \cdot$ relation.

Lemma 3.7 (Head form). *Assume $\Gamma \vdash t^\tau$. Then t is of the form $x\vec{t}$, $c\vec{t}$, $(\lambda x^\rho\, s)\vec{t}$ or $\langle s, r\rangle\, \vec{t}$.*

4 Reductions

We now define reduction rules on the terms. They are all correct with respect to the set-theoretic semantics. In order to be able to control the effects of recursion we allow reduction only if the argument is already calculated, i.e. if the argument is a numeral.

Definition 4.1 (Numerals, lists). Terms of the form $\mathbf{S}_{i_1} d_1^{\diamond}(\ldots(\mathbf{S}_{i_n} d_n^{\diamond}\mathbf{1}))$ are called *numerals*. *Lists* are terms of the form $\mathbf{cons}_\tau d_1^{\diamond} a_1^{\tau}(\ldots(\mathbf{cons}_\tau d_n^{\diamond} a_n^{\tau}\mathbf{nil}_\tau))$. (Here the $\vec{d}$ denote arbitrary terms of type $\diamond$.)

Remark 4.2. It should be noted that we could have also required the $\vec{d}$ to be variables and get the same results. However, this definition allows more reductions (see definition 4.3) and therefore is slightly more flexible when dealing with terms that are not almost closed, i.e. contain variables of types different to $\diamond$.

Definition 4.3 (Conversions). $\mapsto$ is defined as:

$$\begin{array}{l}
(\lambda x^\tau\, t)s \mapsto t[s/x] \\
\langle t, s\rangle\, \mathbf{t} \mapsto t \\
\langle t, s\rangle\, \mathbf{f} \mapsto s \\
\mathbf{t}\, \langle t, s\rangle \mapsto t \\
\mathbf{f}\, \langle t, s\rangle \mapsto s \\
\otimes_{\tau,\rho} ts(\lambda x^\tau, y^\rho\, r) \mapsto r[t, s/x, y] \\
\mathbf{1}\, \{t_0, t_1\}\, s \mapsto s \\
\mathbf{S}_i d^\diamond \nu\, \{t_0, t_1\}\, s \mapsto t_i d^\diamond(\nu\, \{t_0, t_1\}\, s) \quad \nu \text{ a numeral} \\
\mathbf{nil}_\tau\, \{t\}\, s \mapsto s \\
\mathbf{cons}_\tau d^\diamond a\ell\, \{t\}\, s \mapsto t d^\diamond a(\ell\, \{t\}\, s) \quad \ell \text{ a list}
\end{array}$$

Definition 4.4 ($t \to t'$). The term closure $t \to t'$ is inductively defined as follows:

$$\frac{t \mapsto t'}{t \to t'} \qquad \frac{t \to t'}{ts \to t's} \qquad \frac{s \to s'}{ts \to ts'}$$

Next we have to show that the term system is closed under reduction, i.e. that the reduct of a typed term is also typed with at most the same free variables.

Lemma 4.5. $\Gamma_1 \cup \{x^\rho\} \vdash t^\tau$*;* $\Gamma_2 \vdash s^\rho$*;* Γ_1, Γ_2 *disjoint* $\Longrightarrow$ $\Gamma_1, \Gamma_2 \vdash t[s/x^\rho]^\tau$

Proof. Induction on t exploiting the fact that whenever $t_1 t_2$ is a typed term, then either t_2 is closed or there are disjoint Γ_{11}, Γ_{12} with $\Gamma_{11} \vdash {t_1}^{\tau'}$ and $\Gamma_{12} \vdash {t_2}^{\tau''}$. ∎

Lemma 4.6 (Closure under reduction). $\Gamma \vdash t^\tau$*;* $t \to t'$ $\Longrightarrow \Gamma \vdash {t'}^\tau$

Proof. Induction on t shows that only conversions have to be considered. The only non-trivial case is handled in Lemma 4.5. ∎

The next step is to show is that we have sufficiently many reduction rules, i.e. that normal forms have the expected form. As the size of a term is represented by its number of free variables, we also have to consider non-closed terms. As, however, $\diamond$ is the only base type without closed terms, we can restrict ourselves to "almost closed" terms, i.e. to terms with free variables of type $\diamond$ only.

Proposition 4.7. *Every normal, almost closed term of type* $\mathbf{N}$ *is a numeral. Every normal, almost closed term of type* $\mathbf{L}(\tau)$ *is a list. Every normal, almost closed term of type* bool *is* $\mathbf{t}$ *or* $\mathbf{f}$*. Every normal, almost closed term of type* $\diamond$ *is a variable.*

Proof. Induction on t and case distinction, according to Lemma 3.7. ∎

5 Length of reduction chains

We use three different measures for terms: the number of free variables, which corresponds to the "size function" in [6]; the length, which is the number of symbols of a term that can be seen (using the interpretation that you can only see one component of an ordinary pair) and the polynomial which is the upper bound for the complexity of a function.

Definition 5.1 (Length). The length $|t|$ of a term t is defined by induction on t as follows:

$$\begin{array}{l}
|c| = |x| = 1 \\
|ts| = |t| + |s| \\
|\lambda x^\tau\, s| = |s| + 1 \\
|\langle t, s\rangle| = \max\{|t|, |s|\} + 1 \\
|\{t\}| = |\{t_1, t_2\}| = 0
\end{array}$$

The notation $|\vec{t}|$ is used for $\sum_{i=1}^{n} |t_i|$.

Lemma 5.2. $\Gamma_1 \vdash t^\tau$*,* $\Gamma_2 \vdash s^\rho$*;* Γ_1, Γ_2 *disjoint* $\Longrightarrow$ $|t[s/x]| \leq |t| + |s|$

In particular, $|(\lambda x^\tau\, t)s| > |t[s/x]|$.

Proof. Induction on t, using the fact that t is typed. ∎

Lemma 5.3. $\Gamma \vdash t^\diamond \Longrightarrow \Gamma \neq \emptyset$

Proof. Induction on $|t|$. Case distinction according to lemma 3.7. For $\otimes_{\tau,\rho}\vec{t}$ and $(\lambda x\, s)\vec{t}$ use lemma 5.2. For $\mathbf{S}_0\vec{t}$, $\mathbf{S}_1\vec{t}$ and $\mathbf{cons}_\tau\vec{t}$ use induction hypothesis for t_1. For $\mathbf{1}\vec{t}$ and $\mathbf{nil}_\tau\vec{t}$ use induction hypothesis for $t_2 \ldots t_n$. The remaining cases are trivial. ∎

The following corollary is formulated for first-order functions only. However, if one accepts the nubmer of free variables as a size measure for definable functions for higher types as well, then the corollary trivially remains valid for *every* definable function.

Corollary 5.4 (Non-size-increasing property). *Every first-order function definable by a closed term has the property that the output is not longer than the input.*

Proof. Lemma 5.3 shows that the usual length for ground types and the number of free variables are the same (due to the typing $\diamond \multimap \mathbf{N} \multimap \mathbf{N}$ of the successor functions). The number of free variables trivially does not increase when reducing the term to normal form. ◻

Let the set $\mathbb{N}[X]$ of polynomials with natural number as coefficients be equipped with the partial ordering $\preccurlyeq$ defined by $\sum a_i X^i \preccurlyeq \sum b_i X^i \iff \forall i. a_i \leq b_i$. Then there exist finite suprema with respect to $\preccurlyeq$; these will be denoted by $\sup_{\preccurlyeq}\{\ldots\}$. Obviously: $f, g \in \mathbb{N}[X]$; $f \preccurlyeq g$; $n \in \mathbb{N} \Longrightarrow f(n) \leq g(n)$

Definition 5.5 (Polynomial $\vartheta(\cdot)$ of a term). For each (not neccessarily typed) term t we define a polynomial $\vartheta(t) \in \mathbb{N}[X]$ by induction on t.

$$
\begin{array}{l}
\vartheta(x) = \vartheta(c) = 0 \\
\vartheta(ts) = \vartheta(t) + \vartheta(s) \\
\vartheta(\lambda x^\tau\, t) = \vartheta(t) \\
\vartheta(\langle t, s\rangle) = \sup_{\preccurlyeq}\{t, s\} \\
\vartheta(\{t\}) = X \cdot \vartheta(t) + X \cdot |t| \\
\vartheta(\{t, s\}) = X \cdot \sup_{\preccurlyeq}\{\vartheta(t), \vartheta(s)\} + X \cdot \max\{|t|, |s|\}
\end{array}
$$

Moreover $\vartheta(\vec{t})$ is short for $\sum_{i=1}^{n} \vartheta(t_i)$.

Again by trivial induction on typed terms, we get

Lemma 5.6. $\Gamma_1 \vdash t^\tau, \Gamma_2 \vdash s^\rho$; Γ_1, Γ_2 *disjoint* $\Longrightarrow \vartheta(t[s/x]) \preccurlyeq \vartheta(t) + \vartheta(s)$

We are now able to prove our main theorem.

Theorem 5.7. *Assume* $\Gamma \vdash t^\tau$ *and* $N \geq |\mathrm{FV}(t)|$ *for some* $N \in \mathbb{N}$. *Then there are* $t = t_0 \to t_1 \to \ldots \to t_k$ *with* t_k *normal and*

$$\vartheta(t_k)(N) + |t_k| + k \leq \vartheta(t)(N) + |t|$$

Proof. Induction on $\vartheta(t)(N) + |t|$ and case distinction according to Lemma 3.7. The only difficult cases are those that are instances of $t = r^{\mathbf{N}}\{h_0, h_1\}\, s\vec{t}$; the case $r^{\mathbf{L}(\tau)}\{h\}\, s\vec{t}$ can be handled similarly and for $(\lambda x\, s)\vec{t}$, Lemmata 5.2 and 5.6 can be used.

Normalize r to r' in k_0 steps using the induction hypothesis. We have

$$\vartheta(r')(N) + |r'| + k_0 \leq \vartheta(r)(N) + |r|$$

The only interesting (and in the case of t being almost closed the only) case is that when r' is a numeral. Consider $r' = \mathbf{S}_{i_1} r_1^\diamond(\ldots(\mathbf{S}_{i_n} r_n^\diamond \mathbf{1}))$. Using lemma 5.3 and the fact that the term is linear typed we get $n \leq N$. In $n+1$ steps form $t' = h_{i_1} r_1^\diamond(\ldots(h_{i_n} r_n^\diamond s))\vec{t}$. Using the induction hypothesis, normalize t' to t'' in k_1 steps; we have

$$\vartheta(t'')(N) + |t''| + k_1 \leq \vartheta(t')(N) + |t'|.$$

Because of the relations $\vartheta(t') \preccurlyeq n \cdot \sup_{\preccurlyeq}\{\vartheta(h_0), \vartheta(h_1)\} + \vartheta(\vec{r}) + \vartheta(s) + \vartheta(\vec{t})$ and $|t'| \leq n \cdot \max\{|h_0|, |h_1|\} + |\vec{r}| + |s| + |\vec{t}|$ we finally get

$$
\begin{array}{cl}
 & \vartheta(t'')(N) + |t''| + k_1 + n + 1 + k_0 \\
\leq & \vartheta(t')(N) + |t'| + n + 1 + k_0 \\
\leq & n \cdot \sup_{\preccurlyeq}\{\vartheta(h_0), \vartheta(h_1)\}(N) + \vartheta(\vec{r})(N) \\
 & +\vartheta(s)(N) + \vartheta(\vec{t})(N) \\
 & +n \cdot \max\{|h_0|, |h_1|\} + |\vec{r}| \\
 & + |s| + |\vec{t}| + n + 1 + k_0 \\
= & \underbrace{|\vec{r}| + n + 1}_{|r'|} + k_0 + \underbrace{\vartheta(\vec{r})}_{\vartheta(r')}(N) \\
 & + |s| + |\vec{t}| + \vartheta(s)(N) + \vartheta(\vec{t})(N) + \\
 & n \cdot \sup_{\preccurlyeq}\{\vartheta(h_0), \vartheta(h_1)\}(N) \\
 & +n \cdot \max\{|h_0|, |h_1|\} \\
\leq & \vartheta(r)(N) + \underbrace{|r| + |s| + |\vec{t}|}_{|t|} + \vartheta(s)(N) + \vartheta(\vec{t})(N) \\
 & +n \cdot \sup_{\preccurlyeq}\{\vartheta(h_0), \vartheta(h_1)\}(N) \\
 & +n \cdot \max\{|h_0|, |h_1|\} \\
\leq & \vartheta(t)(N) + |t| \quad \text{by definition of } \vartheta(t)
\end{array}
$$

◻

Remark 5.8. The special reduction strategy choosen in the proof can be discribed as follows: "In (the only non-trivial) case of an application st first normalize s to s', then form the "canonical reduct" for $s't$, i.e. completely unfold an iteration, and finally normalize the canonical reduct." As this strategy depends only on the term structure, the decision which redex to choose next can be made in sufficiently short time.

Corollary 5.9 (Hofmann, 1999). $\emptyset \vdash t^{\vec{\mathbf{N}} \multimap \mathbf{N}}$. *Then* t *denotes a function computable in polynomial many steps.*

Proof. Let $\vec{\nu}$ be almost closed normal numerals. Then $\vartheta(t\vec{\nu}) = \vartheta(t)$, as $\vartheta(\vec{\nu}) = 0$. Therefore an upper bound for the number of steps is $\vartheta(t)(|\mathrm{FV}(\vec{\nu})|) + |t| + |\vec{\nu}|$, which is polynomial in the length of the input $\vec{\nu}$. ◻

Example 5.10 (Insertion sort). Let τ be a type equipped with a linear ordering. Assume that this ordering is represented by the term $\lhd_\tau$ of type $\tau \multimap \tau \multimap \mathrm{bool} \otimes (\tau \otimes \tau)$, i.e. $\lhd_\tau ts \to^* \otimes_{\mathrm{bool},\tau\otimes\tau}\mathbf{t}(\otimes_{\tau,\tau}ts)$, if s is "smaller" than t, and $\lhd_\tau ts \to^* \otimes_{\mathrm{bool},\tau\otimes\tau}\mathbf{f}(\otimes_{\tau,\tau}ts)$ otherwise.

Using this function, we can define a sort function of type $\tau \multimap \tau \multimap \tau \otimes \tau$ for two elements, i.e. a function taking two arguments and returning them in the correct order:

$$\lhd'_\tau = \lambda p_1^\tau, p_2^\tau \; \lhd_\tau \; p_1 p_2 (\lambda y^{\mathrm{bool}}, p^{\tau\otimes\tau}\; p(\lambda p_1^\tau, p_2^\tau \\ y \, \langle \otimes_{\tau,\tau} p_1 p_2, \otimes_{\tau,\tau} p_2 p_1 \rangle)$$

Now we can immediately define a function of type $\mathbf{L}(\tau) \multimap \diamond \multimap \tau \multimap \mathbf{L}(\tau)$ inserting an element at the correct position in a given sorted list. At each step we compare the given element with the first element of the list, take the "smaller" element at the beginning of the list and insert the "larger" one in the rest of the list.

$$\begin{aligned}\mathtt{insert} = \lambda l \quad & l\{\lambda x_1^\diamond, y_1^\tau, p^{\diamond\multimap\tau\multimap\mathbf{L}(\tau)}, x_2^\diamond, y_2^\tau \\ & \lhd'_\tau y_1 y_2 (\lambda z_1^\tau, z_2^\tau \; \mathbf{cons}_\tau x_1 z_1 (p x_2 z_2))\} \\ & (\lambda x^\diamond, y^\tau \; \mathbf{cons}_\tau x y \mathbf{nil}_\tau)\end{aligned}$$

Then insertion-sort is defined as usual:

$$\mathtt{sort} = \lambda l^\tau \; l \left\{ \lambda x^\diamond, y^\tau, l^{\mathbf{L}(\tau)} \; \mathtt{insert} l x y \right\} \\ \mathbf{nil}_\tau$$

Counting braces, we get $\vartheta\,(\mathtt{insertionsort}) = X \cdot \vartheta\,(\mathtt{insert}) + O(X) = X^2 \cdot \vartheta\,(\lhd'_\tau) + O(X^2) = X^2 \cdot \vartheta\,(\lhd_\tau) + O(X^2)$, as $\vartheta\,(\lhd'_\tau) = \vartheta\,(\lhd_\tau)$. This reflects the well known fact that insertion sort is quadratic in the number of comparison operations.

Remark 5.11. By simple modifications of (the proof of) [6, §4.3] we may conclude that many "natural orderings", e.g. the normal ordering on binary coded natural numbers, *can* be defined in the given term system.

However, it should be noted that it is *not necessary* that every interesting ordering is definable in the given system. It would also be possible to just add a new symbol $\lhd_\tau$ with the conversion rules $\lhd_\tau ts \mapsto \otimes_{\mathrm{bool},\tau\otimes\tau}\mathbf{t}(\otimes_{\tau,\tau}ts)$, if s is "smaller" than t, and $\lhd_\tau ts \mapsto \otimes_{\mathrm{bool},\tau\otimes\tau}\mathbf{f}(\otimes_{\tau,\tau}ts)$ otherwise. With the definition $|\lhd_\tau| = 4$ the above theory remains valid and shows that there are at most X^2 of the newly introduced conversions in the normalizing sequence. Therefore this theory can be used to calculate the number of calls to a "subroutine", even if the subroutine itself is not definable in the given system, or not even polynomial time computable.

6 Conclusions and further work

For terms in Hofmann's term calculus from [6] we have presented an explicit method to calculate a polynomial bounding the size of the value as well as the number of reduction steps. The construction is quite direct, i.e. one can see immediately how the polynomial is generated. A possible benefit of such a perspicuous approach is that generalizations should be more manageable, e.g. – via the Curry-Howard correspondence – to systems of proof terms for natural deduction calculi involving induction on notation. This is one line of research we would like to follow up.

7 Appendix: Extending the system to full `Ptime`

Once one has understood that the non-size-increasing property is just a question of free variables, it is obvious how to extend the system in such a way that it characterises `Ptime`: just add a type ι of binary words that can be formed without free variables (and therefore without elimination rules). A similar idea was used by Leivant and Marion. In [9] they use "tiers" in the form of different forms of data representation. Their "Church-like abstraction terms" correspond to our type $\mathbf{N}$, and their "words" to the new type ι. The idea for the proof of theorem 7.4 is due to [6, §4.3].

Definition 7.1 (Extended term system). The term system is extended by

- adding a new ground type ι
- adding the following constants with their respective types:

$$\begin{array}{ll} \circ & \iota \\ \mathfrak{s}_0 & \iota \multimap \iota \\ \mathfrak{s}_1 & \iota \multimap \iota \\ \mathbf{p} & \iota \multimap \iota \\ \mathbf{iszero} & \iota \multimap (\mathrm{bool} \otimes \iota) \\ \mathbf{head} & \iota \multimap (\mathrm{bool} \otimes \iota) \end{array}$$

- Extending $\mapsto$ by

$$\begin{array}{l} \mathbf{p}\circ \mapsto \circ \\ \mathbf{p}(\mathfrak{s}_i t^\iota) \mapsto t^\iota \\ \mathbf{iszero}\circ \mapsto \otimes_{\mathrm{bool},\iota}\mathbf{t}\circ \\ \mathbf{iszero}(\mathfrak{s}_i t^\iota) \mapsto \otimes_{\mathrm{bool},\iota}\mathbf{f}(\mathfrak{s}_i t^\iota) \\ \mathbf{head}\circ \mapsto \otimes_{\mathrm{bool},\iota}\mathbf{f}\circ \\ \mathbf{head}(\mathfrak{s}_0 t^\iota) \mapsto \otimes_{\mathrm{bool},\iota}\mathbf{f}(\mathfrak{s}_0 t^\iota) \\ \mathbf{head}(\mathfrak{s}_1 t^\iota) \mapsto \otimes_{\mathrm{bool},\iota}\mathbf{t}(\mathfrak{s}_1 t^\iota) \end{array}$$

The definitions of the relations $\Gamma \vdash t^\tau$ and $t \to t'$ remain unchanged. With almost identical proofs it can be shown that proposition 3.5, and Lemmata 3.6, 3.7, 4.5 and 4.6 remain vaild.

We call terms of the form $\mathfrak{s}_{i_1}(\ldots(\mathfrak{s}_{i_n}\circ))$ *short numerals*. Then proposition 4.7 remains valid (with the proof extended in the obvious way); moreover, every normal, almost closed term of type ι is a short numeral (and therefore closed).

The definition of the term length $|t|$ had the property that for all reductions different to induction the length decreased. To make this property remain valid we define

$$|\mathbf{iszero}| = |\mathbf{head}| = 3$$

and keep the rest of definition 5.1. (In particular, every constant different to $\mathbf{iszero}$ and $\mathbf{head}$ still has length 1.) Then Lemmata 5.2 and 5.3 remain valid; for the latter ($\Gamma \vdash t^{\diamond} \implies \Gamma \neq \emptyset$) the following cases have to be added to the proof:

- Cases $\circ\vec{t}$, $\mathfrak{s}_i\vec{t}$ and $\mathbf{p}\vec{t}$: terms of this form *never* have type $\diamond$, as there are *no* elimination rules for type ι.
- Cases $\mathbf{iszero}\vec{t}$ and $\mathbf{head}\vec{t}$: as t has type $\diamond$ it follows that $\vec{t} = t1, \ldots, t_n$ for some $n \geq 2$. Therefore the induction hypothesis can be applied to the *shorter* term $\otimes_{\mathrm{bool},\iota}\mathbf{t} \circ t_2 \ldots t_n$.

The definition of $\vartheta\,(t)$ remains unchanged. Then (with identical proof) lemma 5.6 remains valid and also the main theorem 5.7. For the proof of the latter we notice that the additional redexes due to the new constants $\mathbf{p}$ and $\mathbf{head}$ can be reduced at any time, as the reduct has a shorter length. In particular, the extended system still consists of `Ptime` functions only.

Lemma 7.2. *If $\mathfrak{f}\colon \tau \otimes \mathbf{L}(\rho) \multimap \tau \otimes \mathbf{L}(\rho)$ is definable by a closed term, then so is a function $\mathfrak{f}^{\sharp}: \tau \otimes \mathbf{L}(\rho) \multimap \tau \otimes \mathbf{L}(\rho)$ with*

$$\mathfrak{f}^{\sharp}(a \otimes \ell) = \mathfrak{f}^{n}(a \otimes \ell),$$

where n is the (semantical) length of ℓ.

Proof. First, by $(\mathbf{L}(\rho)^{-})$ define a function $\mathfrak{g}\colon \mathbf{L}(\rho) \multimap \mathbf{L}(\rho) \multimap \tau \multimap \tau \otimes \mathbf{L}(\rho)$ with

$$\mathfrak{g}(\mathbf{nil}_{\rho})(\ell')(b) = b \otimes \ell'$$
$$\mathfrak{g}(\mathbf{cons}_{\rho}d^{\diamond}a\ell)\ell' b = \mathfrak{f}(\mathfrak{g}(\ell)(\mathtt{app}d^{\diamond}(\ell')(\mathbf{cons}_{\rho}a\mathbf{nil}_{\rho})))$$

with $\mathtt{app}$ being the append function for lists of type $\diamond \multimap \mathbf{L}(\rho) \multimap \mathbf{L}(\rho)$. (This function obviously can be defined in the given system.)

By induction on ℓ we note that $\mathfrak{g}\ell\ell' b = \mathfrak{f}^{n}(b \otimes (\mathtt{app}\ell'\ell))$ where n is the length of ℓ. So we can define $\mathfrak{f}^{\sharp}(a \otimes \ell) = \mathfrak{g}\ell\mathbf{nil}_{\rho}a$. ∎

Proposition 7.3. *If $\mathfrak{f}\colon \tau \multimap \tau$ is definable by a closed term and $k \in \mathbb{N}$, then $\mathfrak{g}: \tau \otimes \mathbf{L}(\rho) \multimap \tau \otimes \mathbf{L}(\rho)$ is also definable by a closed term with $\mathfrak{g}(a \otimes \ell) = \mathfrak{f}^{n^k}(a) \otimes \ell$, where n is the length of k.*

Proof. First define $\tilde{\mathfrak{f}}\colon \tau \otimes \mathbf{L}(\rho) \multimap \tau \otimes \mathbf{L}(\rho)$ with $\tilde{\mathfrak{f}}(a \otimes \ell) = \mathfrak{f}(a) \otimes \ell$. By induction on k, one verifies that

$$\tilde{\mathfrak{f}}^{\overbrace{\sharp\cdots\sharp}^{k}}(a \otimes \ell) = \mathfrak{f}^{n^k}(a) \otimes \ell,$$

where n is the length of ℓ. ∎

Theorem 7.4 (completeness for `Ptime`). *Every `Ptime` function can be denoted by a closed term of type $\mathbf{N} \multimap \iota$.*

Proof. Let $\mathfrak{f}$ be a `Ptime` function. Then there is a Turing machine computing $\mathfrak{f}$ in polynomial many steps. We may assume that this machine works over the alphabet $\{0, 1\}$ and has N states $\{S_0, \ldots, S_{N-1}\}$. Let n be the least natural number with $2^n \geq N$.

A configuration of the machine with the symbols $i_0 \ldots i_k$ before and including the head and the symbols $j_0 \ldots j_{k'}$ followed by the non-visited positions after the head and the machine in state S_m is coded by

$$\otimes(\mathfrak{s}_{i_k}(\ldots(\mathfrak{s}_{i_0}\circ)))$$
$$(\otimes(\mathfrak{s}_{j_0}(\ldots(\mathfrak{s}_{i_{k'}}\circ)))$$
$$(\otimes t_1(\otimes\ldots(\otimes t_{n-1}t_n))))$$

with each of the $\vec{t}$ being $\mathbf{t}$ or $\mathbf{f}$ such that this is the binary coding of m. So configurations are coded by *closed* terms of type $\tau = \iota \otimes \iota \otimes \mathrm{bool} \otimes \ldots \otimes \mathrm{bool}$.

By $(\mathbf{N}^{-})$ with result type $\iota \otimes \mathbf{N}$ a term can be defined returning both, its input and a copy of the input as short numeral. This is possible as the successor of type ι can be built without requiring free variables. Using this term, we can define a closed term of type $\mathbf{N} \multimap \mathbf{N} \otimes \tau$ returning its input and creating the initial configuration of the Turing machine.

The one-step transition function can also be defined by a closed term: via $(\otimes^{-})$ the coding of the state is split into its components; using the *constants* $\mathbf{iszero}$ and $\mathbf{head}$ followed by more applications of $(\otimes^{-})$, we get additional variables of type bool indicating whether the symbol currently read is at the beginning or the "end" of the tape. Complete case distinctions via (bool^{-}) reduce the problem to the construction of the new state from the two parts of the tape. This can be done easily using the constants $\mathfrak{s}_0$, $\mathfrak{s}_1$, $\mathbf{p}$, $\otimes$, $\mathbf{t}$ and $\mathbf{f}$.

Iterating the one-step transition function via proposition 7.3 completes the proof. ∎

References

[1] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational complexity*, 2:97–110, 1992.

[2] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. To appear: Annals of Pure and Applied Logic.

[3] V.-H. Caseiro. *Equations for Defining Poly-time Functions*. PhD thesis, University of Oslo, Feb. 1997. ftp.ifi.uio.no/pub/vuokko/0adm.ps.

[4] M. Hofmann. In-place update with linear types or how to compile functional programms into malloc()-free C. Preprint, www.dcs.ed.ac.uk/ ˜mxh/malloc.ps.gz.

[5] M. Hofmann. Typed lambda calculi for polynomial-time computation. Habilitation thesis, TU Darmstadt, Germany. See www.dcs.ed.ac.uk/home/mxh/habil.ps.gz, 1998.

[6] M. Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proceedings 14'th Symposium on Logic in Computer Science (Lics'99)*, pages 464–473, 1999.

[7] F. Joachimski and R. Matthes. Short Proofs of Normalization for the simply-typed λ-calculus, permutative conversions and Gödel's T. Submitted to the Archive for Mathematical Logic, www.tcs.informatik.uni-muenchen.de/ ˜matthes/papers/sn.html, 1998.

[8] D. Leivant. Ramified recurrence and computational complexity I: Word recurrence and poly–time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, Boston, 1995.

[9] D. Leivant and J.-Y. Marion. Lambda calculus characterization of poly–time. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications*, pages 274–288. Springer Lecture Notes in Computer Science Vol. 664, 1993.