CHAPTER 3

COMPLEXITY MEASURES ON RECURSIVE PROGRAMS

Suppose $\Pi$ is a class of algorithms (or programs) which compute partial functions $f : A^n \rightarrow A$ on some set $A$. In the most general terms, a complexity measure for $\Pi$ associates with each $n$-ary $\alpha \in \Pi$ an $n$-ary partial function

$$\texttt{t}_\alpha : A^n \rightarrow \mathbb{N}$$

which intuitively assigns to each $\vec{x}$ such that $\sigma(\vec{x}) \downarrow$ a cost of some sort of the computation of $\sigma(\vec{x})$ by $\alpha$.

We introduce here five natural complexity measures on the recursive programs of a $\Phi$-structure $A$, directly from the programs, i.e., without reference to the recursive machine. These somewhat abstract, “implementation-independent” (logical) definitions help clarify some complexity questions, and they are also useful in the derivation of specific upper and lower bounds.

3A. The basic complexity measures on recursive programs

3A.1. The tree-depth complexity $D(M)$. Fix a $\Phi$-algebra $A$ and a $\Phi$-program $E$. In the notation of (22), (23), we can associate with each closed, convergent $\text{sig}(E)[A]$-term $M$ a computation tree which represents an abstract, parallel computation of $M$. The nodes of this tree are labelled with convergent $\text{sig}(E)[A]$-terms, and the root is $M$.

These trees are illustrated in Figure 3. We leave for the problems the precise definitions, since they are not needed for most proofs—although they illuminate them, and they help explain the meaning of the results. Our main interest is in the function

$$D_A(M) = D(M) = \text{the depth of the computation tree for } M \quad \text{(if } \overline{M} \downarrow),$$

which can be defined directly as follows.
3. Complexity measures on recursive programs

**Lemma 3A.1.** Fix a $\Phi$-algebra $A$ and a $\Phi$-program $E$. There is exactly one function $D$ which is defined for every closed, convergent $\text{sig}(E)[A]$-term $M$ and satisfies the following conditions:

- (D1) $D(0) = D(1) = D(x) = 0$.
- (D2) $D(\phi(M_1, \ldots, M_n)) = \max\{D(M_1), \ldots, D(M_n)\} + 1$.
- (D3) If $M \equiv (\text{if } M_0 = 0 \text{ then } M_1 \text{ else } M_2)$, then
  
  \[ D(M) = \begin{cases} 
  \max\{D(M_0), D(M_1)\} + 1, & \text{if } M_0 = 0, \\
  \max\{D(M_0), D(M_2)\} + 1, & \text{if } M_0 \downarrow \text{ and } M_0 \neq 0. 
  \end{cases} \]

- (D4) If $f$ is a recursion variable of $E$,\(^\text{10}\) then
  
  \[ D(f(M_1, \ldots, M_n)) = \max\{D(M_1), \ldots, D(M_n), \delta_f(M_1, \ldots, M_n)\} + 1, \]

\(^\text{10}\)If $f \equiv f_i$ is a recursion variable of $E$, we sometimes set $E_f \equiv E_i$ for the term which defines $f$ in $E$. It is a useful convention which sometimes saves chasing subscripts.

---

**Figure 3.** Computation trees.
where $d_{f}(\vec{w}) = D(E_{f}(\vec{w}, w_{1}, \ldots, w_{K}))$.

**Proof.** If $\overline{M} = \text{den}(M(x, f_{1}, \ldots, f_{K}))\downarrow$, then there is some $k$ such that $\overline{M}^{k} = \text{den}(M(x, f_{1}^{k}, \ldots, f_{K}^{k}))\downarrow$, by Lemma x2A.1. We define $D(M)$ by recursion on

$$\text{stage}(M) = \text{the least } k \text{ such that } \overline{M}^{k} \downarrow,$$

and recursion on the length of terms within this. We consider cases on the form of $M$.

(D1) If $M$ is 0, 1 or a parameter $x$, set $D(M) = 0$.

(D2) If $M \equiv \phi(M_{1}, \ldots, M_{n})$ for some $\phi \in \Phi$ and $\overline{M} \downarrow$, then

$$\text{stage}(M) = \max\{\text{stage}(M_{1}), \ldots, \text{stage}(M_{n})\},$$

and these subterms are all smaller than $M$, so we may assume that $D(M_{i})$ is defined for $i = 1, \ldots, n$; we set

$$D(M) = \max\{D(M_{1}), \ldots, D(M_{n})\} + 1.$$

(D3) If $M \equiv \text{if } (M_{0} = 0) \text{ then } M_{1} \text{ else } M_{2}$ and $\overline{M} \downarrow$, then either $M_{0} \downarrow$ or $M_{0} \downarrow$, $M_{0} \neq 0$ and $M_{2} \downarrow$. Assuming the first, we may assume that $D(M_{0}), D(M_{1})$ are both defined, since $M_{0}, M_{1}$ are proper subterms of $M$, and define $D(M)$ appropriately.

(D4) If $M \equiv f(M_{1}, \ldots, M_{n})$ with a recursive variable $f$ of $E$, $\overline{M} \downarrow$ and $k = \text{stage}(M)$, then

$$\overline{M}^{k} = \overline{f}^{k}(\overline{M}_{1}^{k}, \ldots, \overline{M}_{n}^{k}),$$

and so $\text{stage}(M_{i}) \leq k$ and we can assume that $D(M_{i})$ is defined for $i = 1, \ldots, n$, since these terms are smaller than $M$. Moreover, if $\overline{M}_{1} = w_{1}, \ldots, \overline{M}_{n} = w_{n}$, then

$$\overline{f}^{k}(w_{1}, \ldots, w_{n}) = \text{den}(E_{f}(w_{1}, \ldots, w_{n}, \overline{f}^{k-1}_{1}, \ldots, \overline{f}^{k-1}_{K}))\downarrow,$$

by the definition of the iterates in the proof of Lemma x1A.1, and so

$$\text{stage}(E_{f}(w_{1}, \ldots, w_{n}, f_{1}, \ldots, f_{K})) < k;$$

thus we may assume that $D(E_{f}(w_{1}, \ldots, w_{n}, f_{1}, \ldots, f_{K}))$ is defined, and define $D(M)$ so that (D3) in the Lemma holds.

The uniqueness of $D$ is proved by a simple induction on $\text{stage}(M)$, following the definition.

It is sometimes convenient to have $D(M)$ defined on all closed sig($E$)[A]-terms, so we set

$$D(M) = \infty \quad (\overline{M} \uparrow),$$

where $n < \infty$ for every $n \in \mathbb{N}$. 

R&C, Version 1.1, first draft, full of errors, October 26, 2010 41
Complexity measures on recursive programs

The tree-depth complexity of a program $E$ is that of its head term,

$$d_E(\vec{x}) = D(E_0(\vec{x}, f_1, \ldots, f_K)).$$

We can think of $D(M)$ as measuring the “depth” of the computation pictured by the computation tree, but these computations are very abstract indeed. For example, to attain

$$D(f(M)) = \max \{ D(M), D(E(\overline{M})) \} + 1,$$

we are supposed to compute in parallel $\overline{M}$ and $\overline{f(M)}$, which means that we should start on $\overline{f(M)}$ before we have completed the computation of $\overline{M}$. This cannot be implemented, except in terms of an “infinitely parallel” computation (if $A$ is infinite), which computes $\overline{f(x)}$ for every $x$, simultaneously, and uses only the computation of $\overline{f(M)}$ when we finally learn the value $\overline{M}$. In any case, our methods do not yield any interesting lower bounds for the complexity $D(M)$; but it is a very useful technical aid in the definition of other, more natural complexity measures, as follows.

3A.2. The number-of-calls complexity $C^*(M)$. For a fixed $\Phi$-algebra $A$ and a $\Phi$-program $E$, the call complexity $C^*(M)$ of each convergent $\text{sig}(E)[A]$-term $M$ is defined by the following recursion on $D(M)$:

\begin{enumerate}
    \item[(C*1)] $C^*(0) = C^*(1) = C^*(x) = 0$ \quad ($x \in A$).
    \item[(C*2)] $C^*(\phi(M_1, \ldots, M_n)) = C^*(M_1) + \cdots + C^*(M_n) + 1$.
    \item[(C*3)] If $M \equiv \text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2$, then

$$C^*(M) = \begin{cases} 
C^*(M_0) + C^*(M_1), & \text{if } \overline{M}_0 = 0, \\
C^*(M_0) + C^*(M_2), & \text{if } \overline{M}_0 \downarrow \& \overline{M}_0 \neq 0.
\end{cases}$$

    \item[(C*4)] If $f$ is a recursive variable of $E$, then

$$C^*(f(M_1, \ldots, M_n)) = C^*(M_1) + \cdots + C^*(M_n) + c^*_f(\overline{M}_1, \ldots, \overline{M}_n),$$

where $c^*_f(\vec{x}) = C^*(E(\vec{x}, f_1, \ldots, f_K)).$

We have incidentally defined the complexity $c^*_f(\vec{x})$ of each of the mutual least fixed points of $E$, and the calls-complexity of $E$ is that of its head term,

$$C^*_E(\vec{x}) = c^*_E(\vec{x}) = C^*(E_0(\vec{x}, f_1, \ldots, f_K)).$$

When we need to make explicit the algebra or the program for which the calls complexity is computed, we will write

$$C^*(A, E, M) \text{ or } c^*_E(A, \vec{x}).$$

This is a very natural complexity measure: $C^*(M)$ counts the number of calls to the primitives which are required for the computation of $\overline{M}$ using “the algorithm expressed” by the program $E$ and disregarding the “logical steps”, tests for the conditional and recursive calls. It does not distinguish between parallel and sequential implementations of $E$, although
it is probably more directly relevant to the second—so we will sometimes refer to it as the sequential calls complexity. Notice that $E$ may (stupidly) call many times for the same value of one of the givens, and all these calls will be counted separately by $C^s(M)$. We will introduce further down more sophisticated complexity measures which do not count duplicate calls.

**3A.3. The depth-of-calls complexity measure $C^p(M)$**. For a fixed $\Phi$-algebra $A$ and a $\Phi$-program $E$, the depth-of-calls complexity $C^p(M)$ of each convergent $\text{sig}(E)[A]$-term $M$ is defined by the following recursion on $D(M)$:

**($C^p1$)** $C^p(0) = C^p(1) = C^p(x) = 0 \quad (x \in A)$.

**($C^p2$)** $C^p(\varphi(M_1, \ldots, M_n)) = \max\{C^p(M_1), \ldots, C^p(M_n)\} + 1$.

**($C^p3$)** If $M \equiv (\text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2)$, then

$$C^p(M) = \begin{cases} \max\{C^p(M_0), C^p(M_1)\}, & \text{if } \overline{M}_0 = 0, \\ \max\{C^p(M_0), C^p(M_2)\}, & \text{if } \overline{M}_0 \downarrow & \overline{M}_0 \neq 0. \end{cases}$$

**($C^p4$)** If $f$ is a recursive variable of $E$, then

$$C^p(f(M_1, \ldots, M_n)) = \max\{C^p(M_1), \ldots, C^p(M_n)\} + c_f(\overline{M}_1, \ldots, \overline{M}_n),$$

where $c^f_p(\vec{x}) = C^p(E_f(\vec{x}, f_1, \ldots, f_K))$.

We have again included the definition of the complexity $c^p_f(\vec{x})$ of each of the mutual least fixed points of $E$, and the calls-depth complexity of $E$ is that of its head term,

**($C^p$)** $c^p_E(\vec{x}) = c^p_f(\vec{x}) = C^p(E_0(\vec{x}, f_1, \ldots, f_K))$.

The number $C^p(M)$ counts the maximal depth of nested calls to the primitives in the computation of $\overline{M}$ by the algorithm expressed by $E$, and it is more directly relevant to parallel implementations—which is why we will sometimes call it the parallel calls complexity. Like the tree depth complexity $D(M)$, this $C^p(M)$ cannot be attained by actual implementations, but (as we will see) it is majorized by the natural complexity measures of all reasonable implementations of recursive programs, and so lower bound results about it have wide applicability. Most of the lower bound results we will derive are for the parallel call complexity $C^p(M)$.

Next we turn to complexity measures which count the logical steps (tests for the conditional and recursive calls) as well as calls to the primitives.

**3A.4. The sequential logical complexity $L^s(M)$**. For a fixed $\Phi$-algebra $A$ and a $\Phi$-program $E$, the sequential logical complexity $L^s(M)$ of each convergent $\text{sig}(E)[A]$-term $M$ is defined by the following recursion on $D(M)$:

**($L^s1$)** $L^s(0) = L^s(1) = L^s(x) = 0 \quad (x \in A)$.

**($L^s2$)** $L^s(\varphi(M_1, \ldots, M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$. 

R&C, Version 1.1, first draft, full of errors, October 26, 2010 43
3. Complexity measures on recursive programs

\((L^3)\) If \(M \equiv (\text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2)\), then

\[
L^s(M) = \begin{cases} 
L^s(M_0) + L^s(M_1) + 1 & \text{if } M_0 = 0, \\
L^s(M_0) + L^s(M_2) + 1 & \text{if } M_0 \downarrow \text{ and } M_0 \neq 0.
\end{cases}
\]

\((L^4)\) If \(f\) is a recursive variable of \(E\), then

\[
L^s(f(M_1, \ldots, M_n)) = L^s(M_1) + \cdots + L^s(M_n) + l^s_f(M_1, \ldots, M_n) + 1
\]

where \(l^s_f(\vec{x}) = L^s(E_f(\vec{x}, f_1, \ldots, f_K))\).

The sequential logical complexity

\((L^s)\)

\[l^s_E(\vec{x}) = l^s_{f_0}(\vec{x}) = L^s(E_0(\vec{x}))\]

counts the number of steps that a sequential (call-by-value) implementation of the program would execute, and is, in some sense, the most natural (realistic) time complexity measure for recursive programs.

3A.5. The parallel logical complexity \(L^p(M)\). For a fixed \(\Phi\)-algebra \(A\) and a \(\Phi\)-program \(E\), the parallel logical complexity \(L^p(M)\) of each convergent \(\text{sig}(E)[A]\)-term \(M\) is defined by the following recursion on \(D(M)\):

\((L^p1)\) \(L^p(0) = L^p(1) = L^p(x) = 0\) \((x \in A)\).

\((L^p2)\) \(L^p(\phi(M_1, \ldots, M_n)) = \max\{L^p(M_1), \ldots, L^p(M_n)\} + 1\).

\((L^p3)\) If \(M \equiv (\text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2)\), then

\[
L^p(M) = \begin{cases} 
\max\{L^p(M_0), L^p(M_1)\} + 1 & \text{if } M_0 = 0, \\
\max\{L^p(M_0), L^p(M_2)\} + 1 & \text{if } M_0 \downarrow \text{ and } M_0 \neq 0.
\end{cases}
\]

\((L^p4)\) If \(f\) is a recursive variable of \(E\), then

\[
L^p(f(M_1, \ldots, M_n)) = \max\{L^p(M_1), \ldots, L^p(M_n)\} + l^p_f(\overline{M_1}, \ldots, \overline{M_n}) + 1,
\]

where \(l^p_f(\vec{x}) = L^p(E_f(\vec{x}, f_1, \ldots, f_K))\).

The parallel logical complexity

\((L^p)\)

\[l^p_E(\vec{x}) = l^p_{f_0}(\vec{x}) = L^p(E_0(\vec{x}))\]

measures time in an abstract, call-by-value, parallel implementation of the program.

**Problems for Section 3A**

The first problem is a more detailed version of Lemma x2B.1, which relates the time complexity of an iterator with the calls complexity of the associated tail recursion.
For each iterator $i$ and the associated recursive program $E \equiv E_i$ on $A = A_i$ and for all $x \in X$,

\[
\tilde{i}(x) = \tilde{T^A}(x),
\]

\[
\text{Time}_i(x) = c^E_i(x) \quad (\tilde{i}(x)).
\]

This exact equality of the time complexity $\text{Time}_i(x)$ with the sequential complexity $c^E_i(x)$ of the associated recursive program is due partly to some choices we made in defining $\text{Time}_i(x)$—we could, for example, not “charge” for the calls to input($x$) and output($s$) and end up with a time complexity two units smaller. The precise definitions of time complexity for specific computation models can be quite complex, and we cannot expect a result as neat as this Lemma. It is always the case, however, that with each computation model $\epsilon$ there is a natural associated algebra $A_\epsilon$, whose primitives are the primitives of the model—not always explicitly identified; and an associated recursive program $E \equiv E_\epsilon$ on $A_\epsilon$ so that

\[
\text{Time}_\epsilon(x) = \Theta(c^E_\epsilon(x)),
\]

i.e., these two complexity measures are (essentially) linearly related. The same is true of all the other, natural complexity measures associated with computation models, which are similarly related to one or another of the measures we introduced in this section. We will not go into results of this type here, although they can be very interesting.\footnote{Especially because they require identifying exactly the data structures of the model and what functions on them are assumed as “given”, besides iteration and branching.}

### 3B. Complexity inequalities

Next we derive the expected inequalities that relate these complexity measures.

**Proposition x3B.1.** For each $\Phi$-algebra $A$, each $\Phi$-program $E$ and each closed, convergent $\text{sig}(E)[A]$-term $M$:

\[
C^*(M) \subseteq C^p(M) \subseteq L^*(M),
\]

\[
L^p(M).
\]
and, in particular, for all \( \vec{x} \) such that \( \overline{f}_E(\vec{x}) \downarrow \),

\[
\begin{align*}
c^s_E(\vec{x}) & \leq c^p_E(\vec{x}) \\
c^p_E(\vec{x}) & \leq l^p_E(\vec{x}).
\end{align*}
\]

**Proof.** All four claimed inequalities are very easy to check, by induction on \( D(M) \).

The difference between \( l^s_E(\vec{x}) \) and \( l^p_E(\vec{x}) \) measures (in some vague sense) how “parallel” the algorithm expressed by \( E \) is and it is no more than exponential by the next result. The base of the exponential is one more than the total arity of the program, which we have already defined,

\[
\ell(E) = \max\{\max\{\text{arity}(\phi) \mid \phi \in \Phi\}, \text{arity}(f_0), \ldots, \text{arity}(f_K)\} \geq 1.
\]

(48) \( \ell(E) = \max\{\max\{\text{arity}(\phi) \mid \phi \in \Phi\}, \text{arity}(f_0), \ldots, \text{arity}(f_K)\} \geq 1. \)

**Theorem x3B.2.** For each \( \Phi \)-algebra \( A \), each \( \Phi \)-program \( E \) of total arity \( \ell \), and each closed, convergent \( \text{sig}(E)[A] \)-term \( M \):

\[
L^s(M) \leq (\ell + 1)L^p(M),
\]

and hence, for all \( \vec{x} \) such that \( \overline{E}(\vec{x}) \downarrow \),

\[
l^p_E(\vec{x}) \leq (\ell + 1)^{l^p_E(\vec{x})}.
\]

**Proof.** If \( \ell = 1 \), then a somewhat stronger inequality holds, and we will leave that part for Problem x3D.6. Here we assume \( \ell \geq 2 \) and we show the result by induction on \( D(M) \).

**Case 1.** \( D(M) = 0 \). In this case \( M \) is 0, 1 or a parameter \( x \in A \), so that \( L^p(M) = L^s(M) = 0 \).

**Case 2.** \( M \equiv \phi(M_1, \ldots, M_n) \). The induction hypothesis gives us the result for \( M_1, \ldots, M_n \), and we compute:

\[
\begin{align*}
L^s(M) &= L^s(M_1) + \cdots + L^s(M_n) + 1 \\
&\leq (\ell + 1)L^p(M_1) + \cdots + (\ell + 1)L^p(M_n) + 1 \\
&\leq \ell(\ell + 1)^A + 1 \\
&= \ell(\ell + 1)^A + (\ell + 1)^A \\
&= (\ell + 1)^{A+1} = (\ell + 1)^{L^p(M)}.
\end{align*}
\]

**Case 3.** \( M \equiv \text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2 \). Assume, for definiteness that \( M_0 = 0 \), and the induction hypothesis for \( M_0 \) and \( M_1 \) and compute

\[
L^s(M) = L^s(M_1) + \cdots + L^s(M_n) + 1 \\
\leq (\ell + 1)L^p(M_1) + \cdots + (\ell + 1)L^p(M_n) + 1 \\
\leq \ell(\ell + 1)^A + 1 \\
\leq \ell(\ell + 1)^A + (\ell + 1)^A \\
= (\ell + 1)^{A+1} = (\ell + 1)^{L^p(M)}.
\]

\[\text{We assume that } \ell(E) \geq 1, \text{ as programs with } \ell(E) = 0 \text{ are not that interesting.}\]
3B. Complexity inequalities

as in Case 2:

\[ L^s(M) = L^s(M_0) + L^s(M_1) + 1 \]
\[ \leq (\ell + 1)^{L^p(M_0)} + (\ell + 1)^{L^p(M_1)} + 1 \]
\[ \leq 2(\ell + 1)^A + 1 \quad (A = \max\{L^p(M_0), L^p(M_1)\}) \]
\[ \leq (\ell + 1)^A + 1 \quad \text{(because } 2 \leq \ell\text{)} \]
\[ \leq (\ell + 1)^{A+1} = L^p(M). \]

Case 4, \( M \equiv f_i(M_1, \ldots, M_n) \). Now

\[ L^s(M) = L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(M_1, \ldots, M_n)) + 1. \]

If \( L^p(M_1) = \cdots = L^p(M_n) = 0 \), then each \( M_i \) is a constant, so their sequential logical complexities are also = 0, and then, using the induction hypothesis:

\[ L^s(M) = L^s(E_i(M_1, \ldots, M_n)) + 1 \]
\[ \leq (\ell + 1)^{L^p(E_i(M_1, \ldots, M_n))} + 1 \leq (\ell + 1)^{L^p(E_i(M_1, \ldots, M_n))} = (\ell + 1)^{L^p(M)}. \]

In the opposite case, setting \( A = \max\{L^p(M_1), \ldots, L^p(M_n)\} \geq 1 \), \( B = L^p(E_i(M_1, \ldots, M_n)) \), we can compute as above:

\[ L^s(M) = L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(M_1, \ldots, M_n)) + 1 \]
\[ \leq (\ell + 1)^{L^p(M_1)} + \cdots + (\ell + 1)^{L^p(M_n)} + (\ell + 1)^B + 1 \]
\[ \leq (\ell + 1)^A + (\ell + 1)^{B+1}, \]

and it sufficed to prove that for all \( A \geq 1 \) and all \( B \in \mathbb{N} \),

\[ \ell(\ell + 1)^A + (\ell + 1)^B+1 \leq (\ell + 1)^{A+B+1}; \]

but this inequality is equivalent to

\[ \frac{\ell}{(\ell + 1)^{B+1}} + \frac{1}{(\ell + 1)^A} \leq 1, \]

which is obvious for \( A \geq 1 \).

The corresponding result relating the calls-complexities (\( L^p(\vec{x}) \) and \( c^p(\vec{x}) \)) is not so simple, and depends on some properties of the computation tree \( T(M) \) associated with each convergent \( \text{sig}(E)[A] \)-term, which we have avoided using up until now. The results are due to Anush Tserunyan who wrote the remainder of this section.

We fix a \( \Phi \)-algebra \( A \) and a \( \Phi \)-program \( E \) with \( \text{sig}(E) = \Phi \cup \{f_0, \ldots, f_K\} \) and total arity \( \ell \geq 1 \). Recall that “the parameters which occur in a term
Complexity measures on recursive programs

Let $t = t(E) = \text{the number of distinct subterms of the terms in } E$
including the simple $f_i(v_1, \ldots, v_m)$ on the left-hand-sides of the equations,
and let

$$t = t(E) = \text{the number of distinct subterms of the terms in } E,$$

and let

$$H = t(\ell + 2)^\ell.$$ 

It is clear that for any $x_1, \ldots, x_\ell \in A$, there are at most $H$ distinct terms
which can be constructed by substituting $0, 1, x_1, \ldots x_\ell$ for the variables in
some subterm of $E$. We will call these closed terms extended $E$-subterms
and denote them by $M, M_1, \ldots, N, N_1, \ldots$.

**Lemma 3B.3 (Forking and property T).** An extended subterm $M$ is forking if
$M = f_i(M_1, \ldots, M_n)$ with

$$\max_{1 \leq j \leq n} C^p(M_j) > 0 \text{ and } C^p(E_i(\overline{M}_1, \ldots, \overline{M}_n)) > 0;$$

it has property T if it is either a constant or forking.

**Proof.** Let the constants of $M_1$ be $x_1, \ldots, x_k$, hence $k \leq \ell + 2$. Each
$M_i$ is an extended subterm of $E$ that uses $x_1, \ldots, x_k$ as parameters. There
are at most $H$ such distinct terms. So, since for $i \neq j$, $M_i \not \equiv M_j$ (otherwise
$M_i \not \equiv M_j$), we have that $n \leq H$.

**Lemma 3B.5.** If $C^p(M) = 0$ then $\overline{M}$ occurs in $M$.

**Proof.** Straightforward induction on $D(M)$.

**Lemma 3B.6.** Let $(M_1, \ldots, M_n) \in T(M_1)$ such that no $M_i$ is forking.
Then $n \leq 2H$.

**Proof.** If $(M_1, \ldots, M_n)$ satisfies the hypothesis of lemma 3B.4, then we are done. Otherwise, let $i$ be least such that $M_{i+1}$ uses constants that
don’t occur in $M_1$. Then necessarily $M_i = f_i(N_1, \ldots, N_k)$ and $M_{i+1} = E_i(\overline{N}_1, \ldots, \overline{N}_k)$. Moreover, $\max\{C^p(N_j) \mid 1 \leq j \leq k\} > 0$ since otherwise,
by Lemma 3B.5, $\overline{N}_1, \ldots, \overline{N}_k$ would occur in $M_i = f_i(N_1, \ldots, N_k)$
and hence in $M_1$. But $M_i$ is non-forking, so $C^p(M_{i+1}) = C^p(E_i(\overline{N}_1, \ldots, \overline{N}_k)) = 0$.
Hence for all $j \geq i + 1$, $C^p(M_j) = 0$ and thus, again by lemma 3B.5,
all constants of $M_j$ occur in $M_{i+1}$. Therefore, by lemma 3B.4, $i \leq H$
and $n - (i + 1) + 1 \leq H$, and thus $n \leq 2H$.

Let

$$v(M) = |\{(M_1, \ldots, M_n) \in T(M) : \forall i, M_i \text{ does not have property-T}\}|.$$
Because of Lemma x3B.6 and the fact that every node in $T(M)$ has at most $l + 1$ successors, $v(M) \leq V$, where

$$V = (l + 1)^{2H} - 1 = (l + 1)^{2H} - 1.$$  

Note that this is true for all convergent extended subterms of $E$.

**Lemma x3B.7.** If $C_p(M) = 0$, then $L^p(M) \leq v(M)$.

**Proof.** By straightforward induction on $D(M)$, it is easy to verify that $L^p(M) \leq$ the number of non-terminal vertices of $T(M)$. But the latter is equal to $v(M)$ since $C_p(M) = 0$.

**Proposition x3B.8.** For any extended subterm $M$, there exists a descendant (not necessarily a successor) $N$ of $M$ in $T(M)$ such that $N$ has property-T and $L^p(M) \leq v(M) + L^p(N)$.

In particular $L^p(M) \leq V + L^p(N)$.

**Proof.** We do induction on $D(M)$. If $M$ has property-T, then we are done. Assuming otherwise, we only prove one of the cases since the rest are similar. Let $M = M_1, \ldots, M_n$ with $\max \{C_p(M_j) \mid 1 \leq j \leq n\} > 0$ and $C_p(M') = 0$, where $M' = E_i(M_1, \ldots, M_n)$. Then, by lemma x3B.7, $L^p(M') \leq v(M')$. By induction hypothesis, for each $i$, there is a descendant $N_i$ of $M_i$ with property-T and such that $L^p(M_i) \leq v(M_i) + L^p(N_i)$. Thus

$$L^p(M) = \max_{1 \leq j \leq n} L^p(M_j) + L^p(M') + 1 
\leq \max_{1 \leq j \leq n} (v(M_j) + L^p(N_j)) + v(M') + 1
\leq \max_{1 \leq j \leq n} L^p(N_j) + \sum_{1 \leq j \leq n} v(M_j) + v(M') + 1
= \max_{1 \leq j \leq n} L^p(N_j) + v(M).$$

Let $k$ be such that $L^p(N_k) = \max_{1 \leq j \leq n} L^p(M_j)$. Then

$$L^p(M) \leq v(M) + L^p(N_k),$$

and we are done.

We introduce a new complexity notion $F(M)$ that represents the number of “non-parallel appearances” of forking terms in the computation tree of $M$. We give the formal definition.

**Proposition x3B.9 (Tserunyan).** By recursion on $D(M)$ define $F(M)$ as follows:

(i) if $M \in A \cup \{0, 1\}$ then $F(M) = 0$;

(ii) if $M = \phi(M_1, \ldots, M_n)$ then $F(M) = \max_{1 \leq j \leq n} F(M_j)$;

(iii) if $M = (\text{if } M_0 = 0 \text{ then } M_1 \text{ else } M_2)$ then

$$F(M) = \left\{ \begin{array}{ll} 
\max \{F(M_0), F(M_1)\}, & \text{if } M_0 = 0 \\
\max \{F(M_0), F(M_1)\}, & \text{if } M_0 \downarrow \text{ and } M_0 \neq 0.
\end{array} \right.$$
(iv) if $M = f_i(M_1, \ldots, M_n)$ and $M$ is non-forking, then

$$F(M) = \max_{1 \leq j \leq n} F(M_j) + F(E_i(M_1, \ldots, M_n));$$

(v) if $M = f_i(M_1, \ldots, M_n)$ and $M$ is forking, then

$$F(M) = \max_{1 \leq j \leq n} F(M_j) + F(E_i(M_1, \ldots, M_n)) + 1.$$

Intuitively, $F(M)$ measures the “non-parallelness” of the computation of $M$ (as far as calls are concerned) since it is equal to the number of times one has to actually add the $C_p$ complexities of the descendant branches (during the recursive calculation of $C_p$) instead of taking their maximum (ideally, by parallel complexity we mean the maximum of the complexities of the branches). In particular

if $F(M) = 0$, then $C_p(M) = \max \{|\sigma|_c \mid \sigma \in T(M)\},$

where if $\sigma = (L_0, \ldots, L_n)$ then $|\sigma|_c$ is the number of indices $i$ such that $L_i = \phi(M_1, \ldots, M_n)$ for some $\phi \in \Phi$.

The crucial point is that $F(M)$ is no more than $C_p(M)$.

**Proposition x3B.10.** $F(M) \leq \max\{C_p(M) - 1, 0\}$.

**Proof.** We do induction on $D(M)$. The only case that needs attention is case (v) of definition x3B.9 because it is the only case at which $F(M)$ grows. So, suppose $M$ is as in (v) of definition x3B.9. Then, $F(M) = \max_{1 \leq j \leq n} F(M_j) + F(M') + 1$, where $M' = E_i(M_1, \ldots, M_n)$. Also, $\max_{1 \leq j \leq n} C_p(M_j) > 0$ and $C_p(M') > 0$. Hence, by induction hypothesis,

$$\max_{1 \leq j \leq n} F(M_j) \leq \max_{1 \leq j \leq n} \max\{C_p(M_j) - 1, 0\}$$

$$= \max\{\max_{1 \leq j \leq n} C_p(M_j) - 1, 0\} = \max_{1 \leq j \leq n} C_p(M_j) - 1,$$

and $F(M') \leq C_p(M') - 1$. Thus

$$F(M) \leq (\max_{1 \leq j \leq n} C_p(M_j) - 1) + (C_p(M') - 1) + 1 = C_p(M) - 1. \quad \square$$
3B. Complexity inequalities

Figure 4. Weighted tree $S$

**Theorem 3B.11 (Tserunyan).** For every extended subterm $M$:

1. If $M$ has property-T, then $L^p(M) \leq (2V + 1)F(M)$, and
2. If $M$ does not have property T, then $L^p(M) \leq (2V + 1)F(M) + V$.

**Proof.** We prove (1) and (2) together, by induction on $D(M)$.

If $M$ is a constant, there is nothing to prove. If $M$ is a forking term with $M = f_i(M_1, \ldots, M_n)$, then by induction hypothesis, letting $M' \equiv E_i(M_1, \ldots, M_n)$,

$$L^p(M) = \max_{1 \leq j \leq n} L^p(M_j) + L^p(M') + 1$$

$$\leq \max_{1 \leq j \leq n} ((2V + 1)F(M_j) + V) + (2V + 1)F(M') + V + 1$$

$$= (2V + 1)(\max_{1 \leq j \leq n} F(M_j) + F(M')) + (2V + 1)$$

$$= (2V + 1)(F(M) - 1) + (2V + 1) + 1$$

$$= (2V + 1)F(M).$$

Now assume $M$ doesn’t have property-T, and let $N$ be provided by Proposition 3B.8 for $M$. Then necessarily $M \neq N$ and hence $D(N) < D(M)$. Thus by induction hypothesis and the choice of $N$ we have

$$L^p(M) \leq L^p(N) + V \leq (2V + 1)F(N) + V \leq (2V + 1)F(M) + V.$$

\[\square\]
3. Complexity measures on recursive programs

Corollary x3B.12. For any recursive program $E$, for all $\vec{x} \in A^n$ such that $\text{den}(E(\vec{x})) \downarrow$,

$$L_E^p(\vec{x}) \leq (2V + 1)c_E^p(\vec{x}) + V.$$ 

Proof. By theorem x3B.11 and proposition x3B.10.

x3B.13. The tree $S$ in Figure 4 illustrates the proof of theorem x3B.11. Black nodes are the forking nodes, gray nodes (except for the root) are successors of black nodes in $T(M)$ that witness the recursive calculation of $L^p$, and the successors of the gray nodes in $S$ are the terms provided by proposition x3B.8. Some edges and the black nodes are weighted so that $L^p$ of each node is not more than the total weight of the tree of descendants of the node. Thus it is clear that $L^p(M) \leq (2V + 1)F(M) + V$.

Theorem x3B.2 and Corollary x3B.12 together show that all the four, basic complexity measures we have associated with a recursive program $E$ do not differ by more than an exponential. For example,

$$c^s_E(\vec{x}) \leq l^s_E(\vec{x}) \leq (\ell + 1)^V E(\vec{x}) + V.$$ 

Here the total arity $\ell$ is an important invariant of $E$, independent of its size, but $V$ depends on the size of $E$ and, in fact, grows exponentially with it. It is not clear at this point what the best constants are for these bounds.

3C. Syntactic complexities of explicit terms

Since $\Phi$-terms can be viewed as recursive programs (with no “body”), it is natural to compare their complexities with the usual, syntactic measures of their sizes. We recall the most natural of these here for easy reference.

The length of a $\Phi[A]$-term $E$ is defined recursively by the clauses

$$\text{length}(0) = \text{length}(1) = \text{length}(v_i) = \text{length}(x) = 0,$$

$$\text{length}(\phi(E_1, \ldots, E_n)) = \text{length}(E_1) + \cdots + \text{length}(E_n) + 1,$$

$$\text{length}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) = \text{length}(E_1) + \text{length}(E_1) + \text{length}(E_2) + 1,$$

and its $\Phi$-length is defined by the same clauses when we count only $\Phi$-symbols:

$$\text{length}(0) = \text{length}(1) = \text{length}(v_i) = \text{length}(x) = 0,$$

$$\text{length}(\phi(E_1, \ldots, E_n)) = \text{length}(E_1) + \cdots + \text{length}(E_n) + 1,$$

$$\text{length}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) = \text{length}(E_1) + \text{length}(E_1) + \text{length}(E_2).$$
Similarly, the depth of a $\Phi[A]$-term $E$ is defined recursively by the clauses
\[
\begin{align*}
\text{depth}(0) &= \text{depth}(1) = \text{depth}(v_i) = \text{depth}(x) = 0, \\
\text{depth}(\phi(E_1, \ldots, E_n)) &= \max\{\text{depth}(E_1), \ldots, \text{depth}(E_n)\} + 1, \\
\text{depth}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) &= \max\{\text{depth}(E_1), \text{depth}(E_1), \text{depth}(E_2)\} + 1,
\end{align*}
\]
and its $\Phi$-depth (which only counts occurrences of symbols in the signature $\Phi$) is defined by the corresponding clauses
\[
\begin{align*}
\text{depth}(0) &= \text{depth}(1) = \text{depth}(v_i) = \text{depth}(x) = 0, \\
\text{depth}(\phi(E_1, \ldots, E_n)) &= \max\{\text{depth}(E_1), \ldots, \text{depth}(E_n)\} + 1, \\
\text{depth}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) &= \max\{\text{depth}(E_1), \text{depth}(E_1), \text{depth}(E_2)\}.
\end{align*}
\]

3D. Recursive vs. explicit definability

It is easy to verify that if $E(\vec{x})$ is a closed, convergent $\Phi[A]$-term, then
\[
\begin{align*}
C^p(E(\vec{x})) &\leq \text{depth}_\Phi(E(\vec{x})), & C^s(E(\vec{x})) &\leq \text{length}_\Phi(E(\vec{x})), \\
L^p(E(\vec{x})) &\leq \text{depth}(E(\vec{x})), & L^s(E(\vec{x})) &\leq \text{length}(E(\vec{x})),
\end{align*}
\]
see Problem x3D.8. Together with the results in the preceding section, this implies that if we derive a non-constant lower bound for $l^p_E(\vec{x})$ for any program $E$ which computes a certain $f : A^n \to A$, then no $\Phi$-term defines $f$ on $A$. We establish here the converse of this proposition, which shows that explicit definability is equivalent to computability with bounded logical complexity—or any other of the complexities we have associated with recursive programs, by the results of the preceding section.

**Theorem x3D.1.** Given $f : A^n \to A$, $S \subseteq \{\vec{x} \mid f(\vec{x}) \downarrow\}$ and partial functions $\Phi$ on $A$, the following are equivalent:

(a) There is a $\Phi$-term $E(\vec{x})$ which defines $f$ on $S$, i.e.,
\[
\vec{x} \in S \implies f(\vec{x}) = f^E(\vec{x}).
\]

(b) There is a $\Phi$-program $E$ and a number $k$ such that for every $\vec{x} \in S$,
\[
\forall \vec{x} \in S \quad \exists k \quad f^E(\vec{x}) = f(\vec{x}) \text{ and } l^p_E(\vec{x}) \leq k.
\]

In particular, a function $f : A^n \to A$ is definable by a $\Phi$-term if and only if it is computed by a $\Phi$-program $E$ with bounded logical complexity.
3. Complexity measures on recursive programs

Proof. (a) \( \Rightarrow \) (b) follows immediately from (52), with \( E \) the term \( E(\vec{x}) \) (as a program with no body) and \( k = \text{depth}(E(\vec{x})) \). For the converse implication we need some preliminary work.

It is convenient for this proof to add a nullary function constant \( \emptyset \) to the language which provides a name for the nullary, undefined partial function. The semantics of these extended (\( \Phi, \emptyset \))-terms and the corresponding (\( \Phi, \emptyset \))[\( A \)]-terms (where we allow parameters from \( A \)) are obvious, and we will need only one (monotonicity) property of them, which is established by a trivial induction on \( M \):

Lemma 1. If \( M(\emptyset) \) is a closed (\( \Phi, \emptyset \))[\( A \)]-term such that \( \text{den}(M(\emptyset)) \downarrow \), \( N \) is any closed (\( \Phi, \emptyset \))[\( A \)]-term, and \( M(N) \) is the \( \Phi[A] \)-term constructed by replacing every occurrence of \( \emptyset \) in \( M(\emptyset) \) by \( N \), then \( \text{den}(M(\emptyset)) = \text{den}(M(N)) \). In particular,

\[ \text{if } \text{den}(M(\emptyset)) \downarrow, \text{ then } \text{den}(M(\emptyset)) = \text{den}(M(0)). \]

Next we associate with each \( \Phi \)-program \( E \), each \( \text{sig}(E) \)-term

\[ M \equiv M(\vec{x}, \vec{f}) \]

and each \( k \), a (\( \Phi, \emptyset \))-term \( [M]^{(k)}(\vec{x}) \) with the same individual variables (the term approximation of \( M \) to depth \( k \)) by the following recursion on \( k \), and within this, recursion on depth(\( M \)):

1. If \( M \) is a variable, or 0, or 1, then \( [M]^{(k)} \equiv M \).
2. \( [M]^{(0)}(\vec{x}) \equiv M(\vec{x}, \emptyset) \), where the substitution \( f_i \mapsto \emptyset \) means that every subterm of \( M(\vec{x}, \vec{f}) \) of the form \( f_i(M_1, \ldots, M_n) \) is replaced by \( \emptyset \).
3. If \( M \equiv \phi(M_1, \ldots, M_n) \), then \( [M]^{(k+1)} \equiv \phi([M_1]^{(k)}, \ldots, [M_n]^{(k)}) \).
4. If \( M \equiv \) if \( (M_0 = 0) \) then \( M_1 \) else \( M_2 \), then

\[ [M]^{(k+1)} \equiv \text{if } ([M_0]^{(k+1)} = 0) \text{ then } [M_1]^{(k+1)} \text{ else } [M_2]^{(k+1)}. \]
5. If \( M \equiv f_i(M_1, \ldots, M_n) \), then

\[ [M]^{(k+1)} \equiv \text{if } ([M_1]^{(k)} \& \cdots \& [M_n]^{(k)} \downarrow) \text{ then } [E_i]^{(k)}([M_1]^{(k)}, \cdots, [M_n]^{(k)}) \text{ else } \emptyset. \]

The complex conditional in the last clause abbreviates a nested sequence of conditionals which insures that the term on the right will not converge unless all the terms \( [M_j]^{(k)} \) converge.\(^{13}\) The definition of \( [M]^{(k)} \) depends

\[ ^{13}\text{For any sequence of terms } N_1, \ldots, N_n, \text{ the term } N_1 \downarrow \& \cdots \& N_n \downarrow \text{ is defined by recursion on } n \text{ so that it converges (and has value 0) exactly when all } N_i \text{ converge:} \]

\[ N_1 \downarrow \equiv \text{if } (N_1 = 0) \text{ then } 0 \text{ else } 0, \]

\[ N_1 \downarrow \& \cdots \& N_{n+1} \downarrow \equiv \text{if } (N_{n+1} = 0) \text{ then } (N_1 \downarrow \& \cdots \& N_n \downarrow) \]

\[ \text{else } (N_1 \downarrow \& \cdots \& N_n \downarrow). \]
on the program $E$ (so that we should properly write $[M]^{(k)}_E$, but not on any specific $\Phi$-algebra $A$.

**Lemma 2.** For each $\Phi$-program $E$, each $\text{sig}(E)$-term $M(\vec{x})$, each $k$ and any $\vec{x} \in A^n$:

(i) If $\text{den}([M]^{(k)}(\vec{x})) \downarrow$, then $\text{den}([M]^{(k)}(\vec{x})) = \text{den}([M]^{(k+1)}(\vec{x}))$.

(ii) If $L^p(M(\vec{x})) \leq k$, then $\text{den}([M]^{(k)}(\vec{x})) = M(\vec{x})$.

**Proof.** (i) is verified by induction on $k$, and within this by induction on depth($M$).

*Basis, $k = 0$. The result is obvious when $M$ is 0, 1 or a variable, and there is nothing to prove if $M \equiv f_i(M_1, \ldots, M_n)$, since in that case $\text{den}([f_i(M_1, \ldots, M_n)]^{(0)}(\vec{x})) = \text{den}(\emptyset) \uparrow$.

For one of the other two cases where the hypothesis may hold, (i) follows immediately from the definitions:

$$[\phi(M_1, \ldots, M_n)]^{(1)} \equiv \phi([M_1]^{(0)}, \ldots, [M_n]^{(0)})$$

$$\equiv \phi(M_1(\vec{x}, \emptyset), \ldots, M_n(\vec{x}, \emptyset)) \equiv [\phi(M_1, \ldots, M_n)]^{(0)};$$

so $[M]^{(0)}$ and $[M]^{(1)}$ certainly have the same denotations. Finally, when $M$ is a conditional we use again the induction hypothesis on the depth of $M$: assuming that $\text{den}([M_0]^{(0)} = 0$ and $[M_1]^{(0)} \downarrow$, we compute:

$$\text{den}([M]^{(1)}) = \text{den}([M_1]^{(1)}) = \text{den}([M_1]^{(0)}) = \text{den}([M]^{(0)}).$$

*Induction Step.* We use again induction on the depth of $M$ and the argument is exactly like that in the basis, except for the new case of $M \equiv f_i(M_1, \ldots, M_n)$ which may now arise. In this case, the hypothesis gives us that $\text{den}([M_j]^{(k)}(\vec{x})) \downarrow$ for all $j$, and so by the definition and the induction hypothesis

$$\text{den}([M]^{(k+1)}(\vec{x})) = \text{den}([E_i]^{(k)}([M_1]^{(k)}(\vec{x}), \ldots, [M_n]^{(k)}(\vec{x})) =\text{den}([E_i]^{(k+1)}([M_1]^{(k+1)}(\vec{x}), \ldots, [M_n]^{(k+1)}(\vec{x})]) = \text{den}([M]^{(k+2)}(\vec{x})).$$

(ii) is proved by induction on $L^p(M)$, and it is obvious when $M$ is 0, 1 or a variable and quite routine in the induction step, with the help of the monotonicity properties established in (i), as follows.

If $M \equiv \phi(M_1, \ldots, M_n)$, then

$$L^p(M) = \max\{L^p(M_1), \ldots, L^p(M_n)\} + 1 = k + 1,$$

and by the induction hypothesis

$$\text{den}([M_i]^{(k)}) = M_i,$$
so that
\[ \text{den}([M]^{(k+1)}) = \phi(\text{den}([M_1]^{(k)}), \ldots, \text{den}([M_n]^{(k)})) = \phi(\overline{M_1}, \ldots, \overline{M_n}) = \overline{M}. \]

The argument is similar for conditionals, and if \( M \equiv f_i(M_1, \ldots, M_n), \) then
\[ k + 1 = L^p(M) = \max \{ L^p(M_1), \ldots, L^p(M_n) \} + 1, \]
so that the induction hypothesis applies to \( M_1, \ldots, M_n, E_i(\overline{M_1}, \ldots, \overline{M_n}) \) and yields
\[ \text{den}([M_i]^{(k)}) = \overline{M_i}, \quad \text{den}([E_i]^{(k)}(\overline{M_1}, \ldots, \overline{M_n})) = E_i(\overline{M_1}, \ldots, \overline{M_n}) = \overline{M}; \]
the required \( \text{den}([M_i]^{(k+1)}) = \overline{M} \) now follows by (i). ☐

(52) To prove that (b) \( \implies \) (a) in the Theorem, take
\[ E(\vec{x}) \equiv [E_0(\vec{x})]^{(k)} \{ \emptyset : \equiv 0 \}, \]
where \( E_0(\vec{x}) \) is the head of the given program \( E, \)
\[ k = \max \{ L^p_E(\vec{x}) \mid \vec{x} \in S \}, \]
and \([E_0(\vec{x})]^{(k)} \{ \emptyset : \equiv 0 \} \) is the \( \Phi \)-term constructed by replacing \( \emptyset \) by 0 in \([E_0(\vec{x})]^{(k)}\).

Problems for Section 3D

For any two tuples
\[ \vec{x} = (x_1, \ldots, x_n) \in A^n, \quad \vec{y} = (y_1, \ldots, y_n) \in A^n, \]
we set
\[ \vec{x} \sim \vec{y} \iff x_i = y_i \text{ for all } i \text{ such that } x_i \in \{0, 1\} \text{ or } y_i \in \{0, 1\}. \]

(53) Show that for a fixed recursive program on a partial algebra \( A, \)
\[ \text{if } C^p(M(\vec{x})) = 0 \text{ and } \vec{x} \sim \vec{y}, \text{ then } \overline{M(\vec{x})} \sim \overline{M(\vec{y})}. \]

Infer that if \( \overline{E(\vec{x})} = 1 \) but \( \overline{E(\vec{y})} = 0 \text{ for some } \vec{y} \sim \vec{x}, \text{ then} \)
\[ c^p_E(\vec{x}) \geq 1. \]

HINT: Use induction on \( D(M) \).

This is a funny-looking result, but it often allows us to infer \( c^p_E(\vec{x}) \geq 1 \)
without doing any work: for example, if \( E \) decides primality, then for every
prime \( p, \)
\[ c^p_E(p) \geq 1 \text{ simply because } \overline{E(p)} = 1 \text{ and } \overline{E(2p)} = 0, \text{ while} \]
\( p \sim 2p. \) (We will often appeal silently to this observation to simplify the
form of explicit, lower bounds.)

(54) \[ c^p_E(\vec{x}) \geq 1. \]

x3D.3. Compute (up to a multiplicative constant) \( c^p_E(x, y) \) for the
program defined (informally) in Problem x1A.16.

x3D.4. Compute (up to a multiplicative constant) \( c^p_E(x, y) \) for the
program defined (informally) in Problem x1A.17.
3D. Recursive vs. explicit definability

x3D.5. Compute (up to a multiplicative constant) $c^E_p(x,y)$ for the program in Problem x1A.18.

x3D.6. Fix a $\Phi$-algebra $A$ and a $\Phi$-program $E$ of total arity $\ell = 1$. Prove that for every closed, convergent $\text{sig}(E)[A]$-term $M$,

$$L^*(M) \leq 2^{L_p(M)} - 1 < 2^{L_p(M)}.$$

x3D.7. Consider the program $E$ with the single equation $f(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } f(\text{Pd}(x)) + f(\text{Pd}(x))$ in the algebra $(\mathbb{N}, 0, 1, \text{Pd}, +)$. Determine the function $f_E(x)$ computed by this program, as well as the complexity measures $c^E_p(x)$ and $c^E_s(x)$. Verify that for some constant $k > 1$,

$$c^E_s(x) \geq k c^E_p(x) \quad (x > 0).$$

x3D.8. Prove the inequalities in (52). Hint: Show first that if $E(x)$ is a $\Phi$-term in which the indicated variable $x$ may occur, $F$ is a $\Phi$-term, and $E(F)$ is the result of replacing each occurrence of $x$ in $E(x)$ by $F$, then

$$\text{depth}(E(F)) \leq \text{depth}(E(x)) + \text{depth}(F)$$
and

$$\text{depth}_\Phi(E(F)) \leq \text{depth}_\Phi(E(x)) + \text{depth}_\Phi(F).$$

Trees. Fix an algebra $A$ and a recursive program $E$ in the same signature. A label is any closed $\text{sig}(E)[A]$-term $M$, and a node is any finite, non-empty sequence $(L_1, L_2, \ldots, L_n)$ of labels. The depth of the node $(L_1, L_2, \ldots, L_n)$ is $n - 1$. A (finite, rooted) tree (of labels) is any finite, non-empty set $T$ of nodes, which contains all (proper) initial segments of its nodes and only one node of depth 0, its root. The leaves of $T$ are the nodes which have no proper extension in $T$, and the depth of $T$ is the maximum of the depths of its nodes.

Operations on trees. If $M$ is a closed $\text{sig}(E)[A]$ term and $T_1, \ldots, T_k$ are trees, then $F(M, T_1, \ldots, T_k)$ is the tree with root $M = w$ and $T_1, \ldots, T_k$ immediately below it. Formally:

$$F(M, T_1, \ldots, T_k) = \{ (M, L_1, \ldots, L_n) \mid \text{ for some } i, (L_1, \ldots, L_n) \in T_i \}. \quad (55)$$

x3D.9. Define precisely the computation tree $T(M)$ for a convergent, closed $\text{sig}(E)[A]$-term $M$ using the operation in (55), as it is illustrated in Figure 3, and prove that

$$D(M) = \text{the depth of } T(M).$$

---

14Here, for once, we use the notation $E(x)$ without assuming that $x$ is the only variable which occurs in $E$. 

R&C, Version 1.1, first draft, full of errors, October 26, 2010 57