Incompleteness in Linear Time

Salvatore Caporaso
(Dipartimento di Informatica dell'Università di Bari
caporaso@di.uniba.it)
Giovanni Pani
(Dipartimento di Informatica dell'Università di Bari
pani@di.uniba.it)
Emanuele Covino
(Dipartimento di Informatica dell'Università di Bari
covino@di.uniba.it)

Abstract: A class $\mathcal{LT}_0$ of functions computable in a proper sub-class of Lintime is defined, and formalized in a system $\mathbf{LT}_0$ of monadic and atomic (quantifier-free) logic. In spite of its poor computational complexity power and logical apparatus, this system has enough power to describe its own proof-predicate. Therefore it might qualify as smallest known system in which Gödel-like diagonalization can be applied. A proof is given that the identically true functions of $\mathcal{LT}_0$ are productive. Hence this incompleteness phenomenon doesn't depend on the technicalities adopted to show it.

Key Words: Incompleteness, Linear time, Computational Complexity
Category: F.4.1

1 Introduction

While the earlier proofs of the first Gödel theorem were involving Grzegorczyk classes $\mathcal{E}_n$ at the elementary level 3 or above, the time complexity of the syntactic algorithms in Rose [Rose, 84] and Smullyan [Smullyan, 61] is exponential. The former uses a system of polyadic, atomic logic for the (possibly proper, according to an old open problem) sub-class $\mathcal{L}$ of $\mathcal{E}_2$, which is obtained by replacing limited PR with limited summation. The latter works with the so-called *rudimentary* relations, a word free algebra with limited quantification and concatenation, which is known to be, in turn, a (possibly proper too) sub-class of $\mathcal{L}$. Incompleteness of an *inference system* DL, based on lists instead of numbers or notations, has been proved by N. Jones, by showing, via the halting problem, that the true formulæ of DL are not recursively enumerable. An exponential time complexity is implicit in an existential quantification of the computation *traces* (see [Jones, 97, p.201-203], for a contrast between this approach and *traditional* incompleteness proofs, which partially applies to this paper too).

We introduce here a class $\mathcal{LT}_0$ of (definitions of) unary functions $f(x)$ on the elements (*lists*) of a tree free algebra, which can be simulated in time $an+b$ and in space $n+c$, for some numerical constants a, b, c. This class, whose naturalness is defended in Note 10, is essentially a fragment of Lisp, obtained by a rather drastic restriction to recursion, and by reducing to 1 the number of the atoms. A system $\mathbf{LT}_0$ of monadic and atomic (quantifier-free) logic follows. All functions in $\mathcal{LT}_0$ can be *list-wise* represented in $\mathbf{LT}_0$. An analogue of Hilbert syntactic condition D1 ($d \vdash_S A$ implies $\vdash_S prov(\ulcorner d\urcorner, \ulcorner A\urcorner)$) holds for $\mathbf{LT}_0$, and its incompleteness follows. We believe that this system might qualify as smallest system known in which Gödel-like diagonalization can be applied.

The class of (the codes for) all $f(x)$ identically true is proved to be productive, and, therefore, not axiomatizable. Hence this result doesn't depend on the particular (and perhaps slightly odd) system $\mathbf{LT}_0$, or on the peculiarities of our codes.

We feel that the incompleteness of $\mathbf{LT}_0$ concerns, in its present form, several formal systems. Indeed, on one hand, the language and deductive machinery of many first-order theories S can be represented in a Polish prefix form; and condition D1 for any such S can be implemented in $\mathbf{LT}_0$. On the other hand, $\mathbf{LT}_0$, because of its poor logic and computational complexity, is, in a sense, a natural sub-system of most formal systems.
Let us observe that the present result implies that the *no-man's land* between Presburger and Gödel domains is, from the point of view of a mere evaluation of the computational power involved, a sub-class of the linear-time functions which lies between, on one side, addition and words as data-type and, on the other side, trees, addition, con/de-structors and the present restricted boolean recursion.
A historical remark. Sometimes people talking about Leibnitz appear to forget that all his work has been carried out in monadic logic, and that he was maintaining that all n-ary relations are reducible to unary properties ("A is larger than B" means that "A is big insofar as B is small" — see Kneale [Kneale, 62, Ch.5.2]). In the context of speculations of this kind, one might observe that the computation resources taking the complete and decidable monadic pure logic into an incomplete theory is rather modest — a time complexity below an, where a is a constant which can be expressed with two decimal digits (with three tapes and a ternary alphabet). See Note 33 for a justification of "modest", on the grounds of a result by Jones [Jones, 93].

2 A class of linear time functions

Definition 1. 1. 0 is a *(ternary) list*; all other lists are in the form $[X, Y, Z]$, where X, Y and Z are lists.
2. The *destructors* x_1, x_2, x_3 return respectively the *head*, the *body*, and the *tail* of x. That is:
$0_i =_{df} 0$ $(i = 1, 2, 3)$; $[X, Y, Z]_1 =_{df} X$; $[X, Y, Z]_2 =_{df} Y$; $[X, Y, Z]_3 =_{df} Z$.
3. Notation. $U, \ldots, Z$ are generic lists. x is our only informal variable defined on the lists. Subscripts appended to $U, \ldots, Z$ and x are always meaning conveying information. In particular, x_{ij} is $(x_i)_j$.
$[U, Y]$ is short for $[U, Y, 0]$, and $[U]$ for $[U, 0, 0]$.
4. The *unary numerals* $\mathbf{n}$ are given by $\mathbf{0} =_{df} 0$; $\mathbf{n+1} =_{df} [0, \mathbf{n}]$.
The *truth-values* are 0 (*true*) and $\mathbf{1}$ (*false*).

Definition 2. The *basic functions* are x_1, x_{21} and x_{31} (see Note 10 for the rationale of this choice). The *initial functions* are the closure of the basic ones under the destructors.

Complex functions are usually built-up from simpler ones by substitution. For example, having previously defined a function $EQ(y, z)$ for equality, one moves by substitution to $EQ(y + u, z + u)$. Since this wouldn't work with unary functions, we use (a) a finite basis of *defined schemes* (see Def. 4); and (b) a potential infinity of *compiled schemes* (see Def. 5).
Notation 3. An *n-ary scheme* Σ_n takes n functions $f_1, \ldots, f_n$ into a new function $\Sigma_n(f_1, \ldots, f_n)(x)$. Sometimes $f(x)\,\Sigma\, g(x)$ stands for $\Sigma(f, g)(x)$, and $\Sigma f(x)$ for $\Sigma(f)(x)$ (cf. for example schemes *or* , $\simeq$, and *not* below).

Definition 4. The *basic schemes* are: (a) the *assignment* of the constant Y to x in $f(x)$; and (b) the *equality*, the *negation*, and the *disjunction*, such that

$$\begin{aligned}
f(x) \simeq g(x) &=_{df} 0 \text{ iff } f(x) = g(x);\\
f(x) \simeq g(x) &=_{df} \mathbf{1} \text{ iff } f(x) \neq g(x);\\
\mathit{not}\ f(x) &=_{df} 0 \text{ iff } f(x) \neq 0;\\
\mathit{not}\ f(x) &=_{df} \mathbf{1} \text{ iff } f(x) = 0;\\
f(x)\ \mathit{or}\ g(x) &=_{df} 0 \text{ iff } f(x) = 0 \text{ or } g(x) = 0;\\
f(x)\ \mathit{or}\ g(x) &=_{df} \mathbf{1} \text{ iff } f(x) \neq 0 \text{ and } g(x) \neq 0.
\end{aligned}$$

Definition 5. 1. For each Y, define the *constant function* "Y"(x) by assignment of the constant $[Y]$ to function x_1.
Notation. We identify the constant functions with their values, writing Y for "Y"(x). $f(Y)$ is the closed function obtained from a given f (not necessarily closed) by assignment of Y to x.
2. Define, by schemes *not* and *or* in $f, g, 0, \mathbf{1}$, the *compiled scheme* *impl* (f, g) by $f(x)$ *impl* $g(x) =_{df}$ *not* $f(x)$ *or* $g(x)$.
Define in a similar way $f(x)$ *and* $g(x)$, $f(x)$ *iff* $g(x)$.

Definition 6. Given a function $g(x)$ and an initial function $h(x) = x_{\ldots}$, function $f = R(g, h)$ is defined by the *(boolean) recursion scheme* R *in* the *step-function* g and in the *substituted function* h if we have

$$\begin{cases} f^*(x) = 0 & \text{if } x \text{ is } 0 \\ f^*(x) = g(x) \ \mathit{and}\ f(x_2) \ \mathit{and}\ f(x_3) & \text{otherwise;} \\ f(x) \ \ = f^*(h(x)). & \end{cases}$$

Note 7. 1. The occurrences of x_2 and x_3 in last definition are just ways to show the values for the recursive calls, not functions of x (which don't belong to $\mathcal{LT}_0$).
2. Let I denote the *identity* function. For the sake of perspicuity, some definitions by recursion will be given in two steps: in the first, we put $f(x) =_{df} R(g, I)(x)$; in the second, we use f in the form $f(x_{\ldots})$. However, since I doesn't belong to the class $\mathcal{LT}_0$ defined below, this will stand for a single definition of the form $f = R(g, h)(x)$, where $h(x) = x_{\ldots}$.

Example 1. Let us call *tape* a not empty word over $\{0, 1\}$. Define the *code* $\overline{T}$ of T by

$$\overline{0T} =_{df} [\mathbf{1}, \overline{T}]; \quad \overline{1T} =_{df} [\mathbf{2}, \overline{T}],$$

where if T is absent, then $\overline{T}$ is absent too. Thus we have for example $\overline{110} = [\mathbf{2}, [\mathbf{2}, [\mathbf{1}]]]$. Tapes are recognized by

$$\begin{aligned} tp(x) = {} & ((x_1 \simeq \mathbf{1} \ \mathit{or}\ x_1 \simeq \mathbf{2}) \ \mathit{and}\ (x_{21} \simeq \mathbf{1} \ \mathit{or}\ x_{21} \simeq \mathbf{2} \ \mathit{or}\ x_{21}) \ \mathit{and}\ x_{31}) \\ & \mathit{and}\ tp(x_2) \ \mathit{and}\ tp(x_3). \end{aligned}$$

To satisfy our syntax (cf. Note 7) we may use tp only in expressions like $tp(x_1), tp(x_{2111},$ etc. (cf. proof of Lemma 13).

Definition 8. 1. Class $\mathcal{LT}_0^{cf}$ (read: *cons-free* $\mathcal{LT}_0$) is the closure of the initial functions under the basic and the recursion schemes.
2. Define the *constructors* by $cons_{Y,Z}(x) =_{df} [Y, x, Z]$.
3. $\mathcal{LT}_0$ is the closure under the basic schemes of $\mathcal{LT}_0^{cf}$ plus all functions *cons*.
4. A function $f(x)$ is *closed* if it begins with an assignment, or if it is built-up from closed functions.
5. $TRUE(\mathcal{LT}_0)$ is the class of all $f(x) \in \mathcal{LT}_0$ such that $f(Y) = 0$ for all Y.

Definition 9. The *height* $ht(X)$ and *length* $|X|$ of X are given by

$$\begin{aligned} ht(0) &=_{df} 0; \quad & ht([X, Y, Z]) &=_{df} 1 + \max(ht(X), ht(Y), ht(Z)); \\ |0| &=_{df} 1; \quad & |[X, Y, Z]| &=_{df} 1 + |X| + |Y| + |Z|. \end{aligned}$$

The *length* $lh(f)$ of f is the number of schemes occurring in its definition, plus the overall length of the constants occurring in its assignments.

Note 10. 1. By induction on $lh(f)$ (using the fact that R is a boolean scheme) one shows that $|f(X)| \leq lh(f) + |X|$ for all $f \in \mathcal{LT}_0$ and X.
2. When proving that $\mathcal{LT}_0$ is contained in Lintime, we shall use the fact that every $f \in \mathcal{LT}_0$ is obtained either from a function $g(u, y, z)$ by defining $f(x) = g(x_1, x_{21}, x_{31})$; or else from a recursive function $g(y)$ by defining $f(x) = g(x_{\ldots})$.
3. Let us represent a binary tree τ by a list x of the form [root, left sub-tree, right subtree]. The role of x_1, x_{21} and x_{31} is then important since they return the three

lowest nodes of τ. When designing $\mathcal{LT}_0$ our purpose was to allow an easy description of linear time algorithms based on *breadth first* visits to binary trees described in this way. An obvious generalization of $\mathcal{LT}_0$ to n-ary trees is obtained by: defining $n-3$ new basic functions $x_{41}, \dots, x_{n1}$, and by allowing the recursive calls $x_4, \dots, x_n$ in definition of scheme R.

We place here next section in order to supply further examples of our functions.

3 Recognition of terminating computations

A variant of TM's. When complexity doesn't matter, we may restrict ourselves, without any loss of generality (see our [Caporaso, 79]), to *structured TM's* M with a single (infinite to the right) semi-tape over the binary alphabet $\mathbf{B}$, generated by the following grammar

$$\langle \text{TM} \rangle := l | r | w | e | \langle \text{TM} \rangle \langle \text{TM} \rangle | (\langle \text{TM} \rangle),$$

where l, r, w, e are *elementary* TM's (meaning: move l-eft, r-ight, w-rite 1, e-rase); where $M_1 M_2$ is the sequence composition of M_1 (executed first) with M_2; and where (M) is the repetition of M *while* the observed symbol is 1.

Notation 11. 1. $T_1, T_2, \dots$ are not-empty words over alphabet $\mathbf{B}$.

2. Given T_1 and T_2, we write $T_1 T_2$ for the contents of the first $|T_1 T_2|$ cells of a tape, whose observed symbol is the leftmost bit of T_2, and whose other cells contain a zero.
3. An *instantaneous description* (ID) is a triple T_1, M, T_2, which is said to be *terminal* if M is absent. When M is not absent, it says that M is placed over $T_1 T_2$.
4. An *atom* is in the form $T_1\ M\ T_2 \models T_1^*\ M^*\ T_2^*$. It means that M, by input $T_1\ T_2$, yields the same output as M^* by input $T_1^*\ T_2^*$; when M^* is absent, it says that M by input $T_1\ T_2$ yields $T_1^*\ T_2^*$.

We may represent the behaviour of our structured TM's by means of the following rules (the idea is that the leftmost elementary TM or (M) is executed first, and cancelled; in rules 5, (M_1) is executed by copying M_1 at the left of (M_1) if the observed symbol is 1)

$$\begin{array}{lll}
\text{1a} & T_1 b\, l M\, T_2 & \models T_1\ M\ bT_2 \quad (b \in \mathbf{B}); \\
\text{1b} & b\, l M\, T_2 & \models 0\ M\ bT_2; \\
\text{2a} & T_1\ r M\ bT_2 & \models T_1 b\ M\ T_2; \\
\text{2b} & T_1\ r M\ b & \models T_1 b\ M\ 0; \\
3 & T_1\ w M\ bT_2 & \models T_1\ M\ 1T_2; \\
4 & T_1\ e M\ bT_2 & \models T_1\ M\ 0T_2; \\
\text{5a} & T_1\ (M_1) M\ 1T_2 & \models T_1\ M_1 (M_1) M\ 1T_2; \\
\text{5b} & T_1\ (M_1) M\ 0T_2 & \models T_1\ M\ 0T_2.
\end{array}$$

Definition 12. A *computation* of M taking $T_1\, T_2$ into $T_1^*\, T_2^*$ is a sequence $C = I_1, \dots, I_n$ of ID's, such that $I_1 = T_1\ M\ T_2$; $I_n = T_1^*\ T_2^*$; and for all $i < n$ we have $I_i \models I_{i+1}$, by one of the rules above. I_1 is the *initial* ID of C.

Codes. 0ur codes are based on the following method:

(a) a *terminal* alphabet $\mathbf{T}$ is adopted, and an arity $0 \leq ari(L) \leq 2$ is assigned (tacitly when obvious) to its letters;

(b) all expressions E to be coded are *translated* into elements E^* of a context-free language $\mathbf{TP}$ in Polish prefix form over $\mathbf{T}$;

(c) $\overline{L} =_{df} \mathbf{i}$ codes the i-th letter L of T, while 0 codes an absent expression;

(d) if E^* is $L\ E_1^* \dots E_n^*$, then $\overline{E} = \overline{E^*} =_{df} [\overline{L}, \overline{E_1^*}, \overline{E_2^*}]$.

To code the computations, define $\mathbf{M} =_{df} \{0, 1, l, r, w, e, while, \circ\}$. The code for the TM M is then $[\overline{M}]$ if M is one of the four elementary machines; is $[\overline{while}, \overline{M_1}]$ if M is (M_1) and is $[\overline{\circ}, \overline{M_1}, \overline{M_2}]$ if M is $M_1 M_2$.

Following a method dating back to [Hao Wang, 57], we define the code for the ID $I = T_1\ M\ T_2$ by $\overline{I} =_{df} [\overline{T_1}, \overline{M}, \overline{T_2^R}]$, where the code for the binary words has been introduced in Ex. 1, and where W^R denotes, as usual, word W read in reverse order. We may now code the computations by

$$\overline{(ID_1, \ldots, ID_n)} =_{df} [[\overline{ID_1}, [\overline{ID_2}, [\ldots, [\overline{ID_n}] \ldots]]]].$$

Lemma 13. A recognizer cm for the computations may be defined in $\mathcal{LT}_0$.

Proof. The following function recognizes the TM codes

$$\begin{aligned} tm(x) = ((&x_1 \simeq \overline{while}\ \mathit{and}\ \ \mathit{not}\ x_{21}\ \mathit{and}\ x_{31}) \\ \mathit{or}\ (&x_1 \simeq \overline{o}\ \mathit{and}\ \ \mathit{not}\ x_{21}\ \mathit{and}\ \ \mathit{not}\ x_{31}) \\ \mathit{or}\ (&x_1 \simeq \overline{l}, \overline{r}, \overline{w}, \overline{e}\ \mathit{and}\ x_{21}\ \mathit{and}\ x_{31}))\ \mathit{and}\ tm(x_2)\ \mathit{and}\ tm(x_3). \end{aligned}$$

The ID's are recognized by $id(x) =_{df} tm(x_2)\ \mathit{and}\ tp(x_1)\ \mathit{and}\ tp(x_3)$, where tp was defined in Ex. 1. We may now define a function nx which accepts all X in the form $[\overline{I}, [\overline{J}, U], W]$ such that $I \models J$ by one of the rules 1a-5b above.

$$nx(x) = id(x_1)\ \mathit{and}\ id(x_{21})\ \mathit{and}\ (st_{1a}(x)\ \mathit{or}\ \ldots\ \mathit{or}\ st_{5b}(x)),$$

and for example a recognizer for rule 1a is defined by

$$\begin{aligned} st_{1a}(x) = x_{121} \simeq \overline{l}\ &\mathit{and}\ x_{122} \simeq x_{212}\ \mathit{and}\ x_{112} \simeq x_{211} \\ &\mathit{and}\ x_{13} \simeq x_{2132}\ \mathit{and}\ x_{111} \simeq x_{2131}. \end{aligned}$$

(Notice that the syntax of R is respected in our use of function tp, because all occurrences of tp (via function id) in nx are in the form x_I for $I = 11, 13, 211, 213$ — cf. Note 7). The result follows by defining $cm0(x) = nx(x)\ \mathit{and}\ cm0(x_2)\ \mathit{and}\ cm0(x_3)$, and $cm(x) = cm0(x_1)$ (cf. Notat. 7 for the reason of the distinction between $cm0$ and cm).

Note 14. Clauses like $\mathit{not}\ x_{21}\ \mathit{and}\ x_{31}$ (in the first line of the definition of function tm and at many other places throughout this paper) check that the arity of the current terminal letter ($\overline{while}$, in this case) is respected.

4 The formal system $\mathbf{LT_0}$

4.1 Language

Let $f(x)$ be obtained by substitution of $g(x)$ for x in $h(x)$. If we write f in the usual *explicit form* which is obtained by replacing all occurrences of x in h by g, we have that substitution cannot be performed in linear time, since $|f(x)|$ is bounded above by $|g(x)||h(x)|$. In our formal system functions are separated from their arguments, by following Gödel's *implicit* presentation [Gödel, 34] of the PR functions (see also Rose [Rose, 84, Ch. 6] or Kleene [Kleene, 52, p. 221]). The same difficulty is met if *formal variables* for the functors are used, and the related substitution rule is postulated. Hence we give to $F, G, \ldots$ the status of *syntactic variables* defined on the functors. An expression in which $F, G, \ldots$ occur is a syntactic or metamathematical *scheme*, which yields a formal object when $F, G, \ldots$ are assigned with a system of (formal) functors.

Definition 15.
1. The *(formal) constants* are $\dot{0}$ and all expressions of the form $[\dot{X}, \dot{Y}, \dot{Z}]$, where $\dot{X}, \dot{Y}, \dot{Z}$ are constants.
2. The *basic functors* are $\mathbf{D}_1, \mathbf{D}_{21}, \mathbf{D}_{31}, C^b_{Y,Z}$. The *functor schemes* Σ_n are: $\mathbf{H}, B, T, C^s_{Y,Z}, N, A_Y$, for $n = 1$; E, D, R, for $n = 2$. (mnemonic: $\mathbf{D}$-estructor, B-ody, H-ead, T-ail, C-onstruction, N-egation, A-ssignment, E-quality, D-isjunction, R-ecursion).
3. A functor $F, G, H, \ldots$ is a basic functor, or is an expression of the form $\Sigma_n(F_1, \ldots, F_n)$, submitted to Restriction 6 below.

4. a is the only formal variable. The *terms* are in the form $F(a)$.
5. Define the *intended interpretation* $\Im$ in the most obvious way.
6. Restriction. (a) The *initial functors* are built-up from the basic functors by means of the destructors. Only initial functors may occur in the scope of a destructor. (b) G should be *cons-free*, and H should be an initial functor in all $R(G, H)$.

Note 16. In principle, we derive terms; in practice, the form of such terms is $E(F, G)(a)$, and their interpretation is $f(X) \simeq g(X)$ for all X, and for $f = \Im(F),\ g = \Im(G)$.

Notation 17. 1. F stands for $F(a)$ if there is no room for confusion between terms and atoms. We write $F = G$ for $E(F, G)$ when we believe that the scope and the formal status of the indicated equality are clear.
2. $F(\dot{Y})$ is short for $A_Y(F)$.
3. We write $\neg F$ and $F \vee G$ for $N(F)$ and $D(F, G)$. $F \wedge G,\ F \rightarrow G,\ F \leftrightarrow G$ are the functor compiled schemes corresponding to *and* , *impl* , *iff* . $F \neq G$ is $N(E(F, G))$. We often omit the parentheses redundant with respect to the following priorities: $=, \neq, \neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

4.2 Derivations

Proper axioms. The proper axioms of system $\mathbf{LT}_0$ are obtained by assigning in the following *proper axiom schemes* any constant to the syntactic variables X, Y, Z, and any functor to F, G, H, K

name	*axiom*
De-0	$\mathbf{H}(\dot{0});\quad B(\dot{0});\quad T(\dot{0})$
De-st	$\mathbf{H}([\dot{X}, \dot{Y}, \dot{Z}]) = \dot{X};\quad B([\dot{X}, \dot{Y}, \dot{Z}]) = \dot{Y};\quad T([\dot{X}, \dot{Y}, \dot{Z}]) = \dot{Z}$
C	$C^b_{Y,Z}(\dot{X}) = [\dot{Y}, \dot{X}, \dot{Z}]$
R-0	$H \rightarrow R(G, H)$
R-st	$\neg H \rightarrow (R(G, H) = (G(\mathbf{H}(H)) \wedge R(G, B(H)) \wedge R(G, T(H))))$
Zero	$F \leftrightarrow (\dot{0} = F)$
N	$(\neg\dot{0}) = \dot{\mathbf{1}}$
Eq	$G = H \leftrightarrow ((G \wedge H) \vee (\neg G \wedge \neg H \wedge$ $\mathbf{H}(G) = \mathbf{H}(H) \wedge B(G) = B(H) \wedge T(G) = T(H)))$
Trans	$(F = G\ \wedge\ F = H) \rightarrow G = H$
A-Σ_1	$F = \dot{X} \rightarrow \Sigma_1(F) = \Sigma_1(\dot{X})$
A-Σ_2	$(F = \dot{X} \wedge G = \dot{Y}) \rightarrow \Sigma_2(F, G) = \Sigma_2(\dot{X}, \dot{Y})$
A-Q	$F \rightarrow F(\dot{Y})$

Logic. We adapt the sentential fragment of the system in *Introduction to Metamathematics*. Thus, *modus ponens* is the only *rule*, and the logic axioms are all instances of the following *axiom schemes*:

1 $F \rightarrow (G \rightarrow F)$	2 $(F \rightarrow G) \rightarrow ((F \rightarrow (G \rightarrow H)) \rightarrow (F \rightarrow H))$
3 $F \rightarrow F \vee G$	4 $F \rightarrow G \vee F$
5 $F \rightarrow (G \rightarrow F \wedge G)$	6 $(F \rightarrow H) \rightarrow ((G \rightarrow H) \rightarrow (F \vee G \rightarrow H))$
7 $F \wedge G \rightarrow F$	8 $F \wedge G \rightarrow G$
9 $\neg\neg F \rightarrow F$	10 $(F \rightarrow G) \rightarrow ((F \rightarrow \neg G) \rightarrow \neg F)$.

We take for granted that all proper axioms are true under $\Im$, and that, therefore, our deductive machinery yields a consistent system.

Lemma 18. 1. Reflexivity and symmetry of equality can be derived in $\mathbf{LT}_0$.
2. We have $\vdash (E(\dot{X}, \dot{Y}) = \dot{\mathbf{1}}) \leftrightarrow \neg E(\dot{X}, \dot{Y})$.
3. All functions $f \in \mathcal{LT}_0$ are *list-wise* representable. That is, for all $g, k \in \mathcal{LT}_0$, there are G and K such that for all Y, Z we have

$$\Im(G) = g, \Im(K) = k, \text{ and } g(Y) = k(Z) \text{ implies } \vdash G(\dot{Y}) = K(\dot{Z}).$$

Proof. 1. Reflexivity. By logic from the instance $\dot{0} = F \to (\dot{0} = F \to F = F)$ of axiom Trans, and from the instance $(\dot{0} = F) \to (\dot{0} = F)$ of law $G \to G$.
Symmetry. $(F = G) \to (G = F)$ follows by logic from $F = F$ and from instance $(F = G) \wedge (F = F) \to (G = F)$ of Trans.
In what follows justifications "*by logic*" will include the laws of equality.
2. We have $\vdash E(\dot{X}, \dot{Y}) = \mathbf{\dot{1}} \to \neg(E(\dot{X}, \dot{Y})) = \neg\mathbf{\dot{1}}$ (axiom A-N) $\to \neg E(\dot{X}, \dot{Y}) = \dot{0}$ (axiom N) $\to \neg E(\dot{X}, \dot{Y})$ (axiom Zero).
Moreover, we have $\neg E(\dot{X}, \dot{Y}) \to \neg E(\dot{X}, \dot{Y}) = \dot{0}$ (axiom Zero) $\to \neg\neg E(\dot{X}, \dot{Y}) = \mathbf{\dot{1}}$ (axioms A-N, N and logic) $\to E(\dot{X}, \dot{Y}) = \mathbf{\dot{1}}$ (axiom 9, logic).
3. Induction on $lh(k) + lh(g)$, and on $ht(Y)$ or $ht(Z)$ if g or k begins with R. We may assume that $g(Y)$ is $e(Y) \simeq k(Y)$ and that k is, say, "$\mathbf{1}$", since in all other cases the proof is like the standard proofs of numeralwise representability in number theory. We have to prove

$$\vdash E(G(\dot{Y}), K(\dot{Y})) = \mathbf{\dot{1}}. \tag{1}$$

The hypothesis $(e(Y) \simeq k(Y)) = 1$ implies that we have $e(Y) = U,\ k(Y) = W$ and $U \neq W$ for some U, W. By the ind. hyp. there are E, K such that $\Im(E) = e, \Im(K) = k$,

$$\text{(a)} \quad \vdash G(\dot{Y}) = \dot{U}; \quad \text{(b)} \quad \vdash K(\dot{Y}) = \dot{W}; \quad \text{(c)} \quad \vdash E(\dot{U}, \dot{W}) = \mathbf{\dot{1}}. \tag{2}$$

From the law of equality $a = b \wedge c = d \wedge a = c \to b = d$, we obtain

$$\vdash ((G(\dot{Y}) = \dot{U}) \wedge (K(\dot{Y}) = \dot{W}) \wedge (G(\dot{Y}) = K(\dot{Y}))) \to E(\dot{U}, \dot{W}). \tag{3}$$

From (2)(a),(b) and (3), modus ponens (twice), and contraposition, we obtain

$$\vdash \neg(E(\dot{U}, \dot{W}) \to \neg(E(G(\dot{Y}), K(\dot{Y}))). \tag{4}$$

The result now follows, since we have $\vdash E(\dot{U}, \dot{W}) = \mathbf{\dot{1}} \to \neg(E(G(\dot{Y}), K(\dot{Y})))$ (by (4) and part 2); $\vdash \neg(E(G(\dot{Y}), K(\dot{Y})))$ by (2)(c); $\vdash E(G(\dot{Y}), K(\dot{Y})) = \mathbf{\dot{1}}$ (by the other half of part 2).

5 Codes and syntactic functions

To code the elements of $\mathbf{LT}_0$ we use the *terminal alphabet*

$$\mathbf{T} =_{df} \{\cdot, :, \dagger, 0, D_1, D_{21}, D_{31}, \mathbf{H}, B, T, C^s, N, A, R, D, E, Ad, \to, \neq, J_i\},$$

where: (a) $\cdot, :, \dagger$ are special symbols used to force the ternary lists in an essentially binary coding system; (b) J_i is the name of the i-th axiom scheme; and (c) $Ad, \to, \neq$ are used for the special codes defined below.
Definition 19. 1. The code $\overline{L}$ for the i-th letter L of $\mathbf{T}$ is $\mathbf{i}$.
2. The code $\overline{X}$ for the constant $\dot{X}$ is $[\,\overline{0}\,]$ if X is $\dot{0}$, and is $[\bar{\cdot}, [\bar{:}, \overline{U}], [\bar{\dagger}, \overline{Y}, \overline{Z}]]$ if X is $[\dot{U}, \dot{Y}, \dot{Z}]$.
3. $[\,\overline{F}\,]$ codes functor F if it is: (a) one of the basic functors $\mathbf{D}_1, \mathbf{D}_{21}, \mathbf{D}_{31}$; (b) an initial functor not occurring in the scope of a destructor; (c) a C-free functor occurring in the scope of a C.
4. $[[\overline{C^t}, \overline{Y}, \overline{Z}], \overline{F}]$ $(t = b, s)$ codes $C^t_{Y,Z}(F)$, with F absent for $s = b$; $[[\overline{A}, \overline{Y}], \overline{F}]$ codes $F(\overline{Y})$.
5. $[\overline{\Sigma}, \overline{F}]$ codes $\Sigma(F)$ for $\Sigma = \mathbf{H}, B, T, N$.
6. $[\overline{\Sigma}, \overline{G}, \overline{H}]$ codes $\Sigma(G, H)$ for $\Sigma = E, D, R$.
7. For every F, $[\overline{Ad}, \overline{F}]$ codes the *diagonal assignment* $A_{\overline{F}}(F)$ (cf. Lemma 23).
8. To have easier syntactic algorithms $[\overline{\to}, \overline{H}, \overline{G}]$ codes $G \to H$, and $[\overline{\neq}, \overline{H}, \overline{G}]$ codes $G \neq H$.

9. $[[\overline{F}, \overline{J_i}]]$ codes the derivation consisting of the instance F of axiom scheme J_i.
10. If derivation d yields F and ends with a modus ponens whose major and minor premisses are respectively the conclusions of d_1 and d_2, then $\overline{d} = [\overline{F}, \overline{d_1}, \overline{d_2}]$.

Example 2. $[\overline{E}, [\overline{D_1}], [[\overline{C^s}, \overline{Y}, \overline{Z}], [[\overline{\mathbf{H}}, [\overline{\mathbf{H}}, [\overline{D_{21}}]]]]]]$ codes $C^s_{Y,Z}(a_{2111}) = D_{21}$.
The different ways of parenthesising the two $\mathbf{H}$'s is due to the fact that the last one has no particular status, while the first *concludes* the construction of an initial functor, which, as a whole, is regarded as a leaf in the construction of the code (cf. clause 3(b) in last Definition).

Definition 20. Function $f \in \mathcal{LT}_0$ *recognizes* a class $\mathcal{S}$ of syntactic entities E if we have $\vdash f(\overline{E})$ iff $E \in \mathcal{S}$.

Lemma 21. A recognizer de for the derivations of $\mathbf{LT}_0$ can be defined in $\mathcal{LT}_0$.

Proof. By coding in our language (cf. Note 14) a number of rather standard syntactic algorithms, we may define the following functions which respectively recognize: the constants, the initial functors, the *cons-free* functors, and the functors

$$\begin{aligned}
co(x) &= c0(x) \ \mathit{and} \ x_1 \simeq \overline{0} \ \mathit{or} \ x_1 \simeq \bar{\cdot}, \qquad \text{where}\\
c0(x) &= ((x_1 \simeq \overline{0} \ \mathit{and} \ x_{21} \ \mathit{and} \ x_{31}) \ \mathit{or}\\
&\quad\ (x_1 \simeq \bar{\cdot} \ \mathit{and} \ x_{21} \simeq \overline{:} \ \mathit{and} \ x_{31} \simeq \overline{\dagger}) \ \mathit{or}\\
&\quad\ (x_1 \simeq \overline{:} \ \mathit{and} \ x_{21} \simeq \bar{\cdot} \ \mathit{and} \ x_{31}) \ \mathit{or}\\
&\quad\ (x_1 \simeq \overline{\dagger} \ \mathit{and} \ x_{21} \simeq \bar{\cdot} \ \mathit{and} \ x_{31} \simeq \bar{\cdot})) \ \mathit{and} \ c0(x_2) \ \mathit{and} \ c0(x_3)\\
in(x) &= ((x_1 \simeq \overline{D_1}, \overline{D_{21}}, \overline{D_{31}} \ \mathit{and} \ x_{21} \ \mathit{and} \ x_{31}) \ \mathit{or} \ (x_1 \simeq \overline{\mathbf{H}}, \overline{B}, \overline{T} \ \mathit{and} \ x_{31}))\\
&\qquad \mathit{and} \ in(x_2) \ \mathit{and} \ in(x_3)\\
cf(x) &= ((in(x_1) \ \mathit{and} \ x_{21} \ \mathit{and} \ x_{31}) \ \mathit{or} \ (x_{11} \simeq \overline{A} \wedge co(x_{12})) \ \mathit{or}\\
&\quad\ (x_1 \simeq \overline{N}, \overline{Ad} \ \mathit{and} \ x_{31}) \ \mathit{or} \ (x_1 \simeq \overline{E}, \overline{D}, \overline{\rightarrow}, \overline{\neq})\\
&\qquad \mathit{or} \ (x_1 \simeq \overline{R} \ \mathit{and} \ in(x_{31}))) \ \mathit{and} \ cf(x_2) \ \mathit{and} \ cf(x_3)\\
fc(x) &= ((cf(x_1) \ \mathit{and} \ x_{21} \ \mathit{and} \ x_{31}) \ \mathit{or} \ (x_{11} \simeq \overline{A} \ \mathit{and} \ co(x_{12}))\\
&\qquad \mathit{or} \ (x_1 \simeq \overline{N}, \overline{Ad} \ \mathit{and} \ x_{31})\\
&\qquad \mathit{or} \ (x_1 \simeq \overline{E}, \overline{D}, \overline{\rightarrow}, \overline{\neq})) \ \mathit{and} \ fc(x_2) \ \mathit{and} \ fc(x_3).
\end{aligned}$$

We may now define (recall that $[\overline{\rightarrow}, \overline{G}, \overline{F}]$ codes $F \to G$)

$$\begin{aligned}
de(x) = ((&x_1 \simeq x_{212} \ \mathit{and} \ x_{213} \simeq x_{31} \ \mathit{and} \ x_{211} \simeq \overline{\rightarrow})\\
&(\mathit{or} \ axm(x_1) \ \mathit{and} \ x_{21} \ \mathit{and} \ x_{31})) \ \mathit{and} \ de(x_2) \ \mathit{and} \ de(x_3)),
\end{aligned}$$

where the first line checks the inferences by modus ponens, and where

$$ax(x_1) = fc(x_{11}) \ \mathit{and} \ x_{13} \ \mathit{and} \ (ax\text{-}1(x_1) \ \mathit{or} \ \ldots \ \mathit{or} \ ax\text{-}\mathrm{D}(x_1))),$$

and for example the instances of Kleene's axiom scheme 1 are recognized by

$$ax\text{-}1(x_1) = (x_{111} \simeq x_{1121} \simeq \overline{\rightarrow}) \ \mathit{and} \ x_{1122} \simeq x_{113} \ \mathit{and} \ x_{12} \simeq \overline{J_1}.$$

6 Diagonalization

Notation 22. Hereinafter, to have less baroque symbols $\overline{X}$ will stand, in formal contexts, for $\dot{\overline{X}}$. To improve readability, we write $\langle Y, f(a), Z\rangle$ for $cons_{Y,Z}(f(a))$.
If $\Im(F) = f$ then $\overline{F}$ is the code $\overline{f}$ for f.

Lemma 23. 1. For every functor F there is a functor G such that

$$\vdash G(a) \leftrightarrow (F(a) \to \mathbf{H}(a) \neq \overline{G}).$$

2. For all $f(x) \in \mathcal{LT}_0$ there is $g(x) \in \mathcal{LT}_0$ such that

$$(g(x) \ \mathit{iff} \ (f(x) \ \mathit{impl} \ \ \mathit{not} \ x_{11} \simeq \overline{g})) \ \in \ \mathit{TRUE}(\mathcal{LT}_0).$$

Proof. We show part 1 only, since the proof of part 2 is similar and easier. Define $sb(x) =_{df} \langle \overline{Ad}, x, 0\rangle$ (cf. Notat. 22). By part 7 of Def. 19 we have

$$sb(\overline{K}) = [\overline{Ad}, \overline{K}] = \overline{K(\overline{K})}, \quad \text{for all functors } K \tag{1}$$

and, by Lemma 18, there exists a functor SB such that

$$\vdash SB(\overline{K}) = \overline{K(\overline{K})}, \quad \text{for all functors } K. \tag{2}$$

Given F, define function sb_F by

$$sb_F(x) =_{df} \langle \overline{\rightarrow}, \langle \overline{\neq}, \langle \overline{A}, x, \overline{SB}\rangle, \overline{\mathbf{H}}\rangle, \overline{F}\rangle. \tag{3}$$

By part 8 of Def. 19, we have $sb_F(\overline{K}) = \overline{F \rightarrow \mathbf{H} \neq SB(\overline{K})}$. By (1) and again by Lemma 18 there exists a functor SB_F such that we have

$$\vdash SB_F(\overline{K}) = \overline{F \rightarrow \mathbf{H} \neq SB(\overline{K})}, \quad \text{for all functors } K. \tag{4}$$

Now define

$$G =_{df} F \rightarrow \mathbf{H} \neq SB(\overline{SB_F}). \tag{5}$$

By (4) we have

$$\vdash SB(\overline{SB_F}) = \overline{F \rightarrow \mathbf{H} \neq SB(\overline{SB_F})},$$

that is, by (5),

$$\vdash SB(\overline{SB_F}) = \overline{G}. \tag{6}$$

From (6), by repeated applications of axioms A-Σ we may now replace $SB(\overline{SB_F})$ in $\vdash G \leftrightarrow G$ (that is in $\vdash G \leftrightarrow F \rightarrow \mathbf{H} \neq SB(\overline{SB_F})$) with $\overline{G}$. This yields $\vdash G \leftrightarrow (F \rightarrow \mathbf{H} \neq \overline{G})$.

7 Incompleteness

Definition 24. Let DE be the functor which, by Lemma 18, represents function de in $\mathbf{LT}_0$. Functor GOE is the "G" associated with DE by Lemma 23.

Theorem 25. Functor GOE is underivable though true.

Proof. By definition of GOE and last lemma we have

$$\vdash GOE(a) \leftrightarrow (DE(a) \rightarrow \mathbf{H}(a) \neq \overline{GOE}). \tag{1}$$

Thus we have

$$GOE(a) \in TRUE(\mathcal{LT}_0) \quad \text{iff} \quad \text{not} \vdash GOE(a). \tag{2}$$

Assume (ad absurdum) $d \vdash GOE$ for some derivation d. We have

$\vdash DE(\overline{d}) \wedge \mathbf{H}(\overline{d}) = \overline{GOE}$	$\vdash GOE$, def. of DE, Lemma 18
$\vdash DE(a) \rightarrow \mathbf{H}(a) \neq \overline{GOE}$	(1), $\vdash GOE$, logic
$\vdash DE(\overline{d}) \rightarrow \mathbf{H}(\overline{d}) \neq \overline{GOE}$	axiom A-Q, with $\overline{d}$ as $\dot{Y}$
$\vdash (\mathbf{H}(\overline{d}) \neq \overline{GOE}) \wedge (\mathbf{H}(\overline{d}) = \overline{GOE})$	logic.

Contradiction with the consistency of $\mathbf{LT}_0$ (see the assertion before Lemma 18).

8 Productivity

The incompleteness of last section does not depend on the peculiarities of $\mathbf{LT}_0$, since we now show that no axiomatizable system can exhaust $TRUE(\mathcal{LT}_0)$.

Notation 26. 1. For every Z coding a (structured) TM M, W_Z is the set of all lists Y such that M by input $Y0$ stops operating.
2. Define $TR =_{df} \{\overline{f} : f(x) \in TRUE(\mathcal{LT}_0)\}$.

Lemma 27. For all Z a function $p_Z \in \mathcal{LT}_0$ can be defined such that

$$X \in W_Z \quad \text{iff} \quad p_Z(U) = 0 \text{ and } U_{11} = X \text{ for some } U.$$

Proof. Define $p_Z(x) =_{df} cm(x)$ *and* $x_{12} \simeq Z$. If $X \in W_Z$ there exists a computation code $U = [[X, Z, 0], \ldots]$ which is accepted by *co* and by p_Z.

Definition 28. For all Z, $q_Z \in \mathcal{LT}_0$ is the function (associated by Lemma 23 with function $p_Z(x)$ of last lemma) such that

$$(q_Z(x) \textit{ iff } (p_Z(x) \textit{ impl not } (x_{11} \simeq \overline{q_Z}))) \in TRUE(\mathcal{LT}_0). \tag{1}$$

Theorem 29. (The set of all codes of the elements of) TR is productive.

Proof. Let $W_Z \subseteq TR$ be given. We show that we have $\overline{q_Z} \in TR \setminus W_Z$.
1. Assume first (ad absurdum) $q_Z(Y) = \mathbf{1}$ for some Y. By (1) we then have

$$p_Z(Y) = 0 \text{ and } (Y_{11} \simeq \overline{q_Z}) = 0. \tag{2}$$

However (2) implies (by Lemma 27) $\overline{q_Z} \in W_Z$ — a contradiction with the hyp. ad abs., since we have $W_Z \subseteq TR$. Thus we actually have $\overline{q_Z} \in TR$.
2. To obtain $\overline{q_Z} \notin W_Z$, assume now (again ad absurdum) $\overline{q_Z} \in W_Z$. By lemma 27 we then have $p_Z(U) = 0$ and $U_{11} = \overline{q_Z}$ for some U. By (1) this implies $q_Z(U) = \mathbf{1}$ — a contradiction, since $\overline{q_Z} \in TR$ was proved in the first part of this proof.

9 Complexity

We restrict ourselves to TM's with 3 semi-tapes T_i over the alphabet Γ which, besides the *blank* symbol ϕ, consists of the following four symbols: zero, brackets, comma.

Notation 30. u, v, w, y, z are words (possibly empty) over Γ. $T_i = u \uparrow w$ means that cells $1, \ldots, |uw|$ of T_i contain uw, cell $|u| + 1$ is observed, and all other cells are blank. We omit $\uparrow$ when in the rightmost position.

Definition 31. M by input y *standard computes* (s-computes) $F(y)$ within time $T(n)$ and space $S(n)$ if for all u, v, w, y we have that M (a) starts operating with

$$T_1 = u\phi y; \quad T_2 = v; \quad T_3 = w;$$

(b) stops operating with

$$T_1 = u\phi y\phi F(y); \quad T_2 = v; \quad T_3 = w;$$

(c) after $T(|y|)$ steps, without ever visiting cells $|u| + S(|y|), |v| + S(|y|), |w| + S(|y|)$).
LTSC_0 is the class of all functions which are standard computed by a TM in the form above within time $an + b$ and space $n + c$, for some constants a, b, c.

Example 3. 1. Let U^R stand for U read in reverse order, and let $U^{(i)}$ denote the i-th occurrrence of U in a given word. The form of every instance W of Kleene'axiom 1 is in the form $[X^{(1)}, \ldots, Y, \ldots, X^{(2)}, \ldots]$, where the ellipses stand for constant patterns of connectives and brackets that we omit for simplicity. They are standard decided in time $2n$ and space n by a TM which: (a) moving from right to left on T_1 and from left to right on T_2, writes $X^{(2)\,R}$ on T_2 (using T_3 to record and balance the brackets); (b)

moving back from the left end of W checks for equality $X^{(1)}$ with $X^{(2)\,R}$ and, in the meantime, erases $X^{(2)\,R}$.
2. The instances W of axiom 2 are in the form (again we omit all connectives and some brackets)

$$[X^{(1)},\ldots,Y^{(1)},\ldots,X^{(2)},\ldots,Y^{(2)},\ldots,Z^{(1)},\ldots,X^{(3)},\ldots,Z^{(2)},\ldots].$$

They are standard decided in time $2n$ and space n by a TM which:
(a) moving from right to left on T_1, writes $Z^{(2)\,R}Y^{(2)\,R}$ in T_2, and $X^{(3)\,R}$ in T_3;
(b) writes $X^{(2)\,R}$ in T_2, and, in the mean time, checks for equality $X^{(2)\,R}$ with $X^{(3)\,R}$ and erases $X^{(3)\,R}$ in T_3; thus entering a configuration of the form

$$T_1 = u_1\phi[X,Y,\uparrow X,Y,Z,X,Z];\quad T_2 = u_2\phi Z^R\phi Y^R\phi X^R\uparrow\phi;\quad T_3 = u_3\uparrow\phi;$$

(c) moving back from the left end of W, checks for equality $X^{(1)}, Y^{(1)}, Z^{(1)}$ with $X^{(2)\,R}$, $Y^{(2)\,R}$, $Z^{(2)\,R}$, and, in the mean time, erases the indicated reversed copies from T_2 and T_3.
3. One sees that this procedure (with the same time and space complexities) can be applied in order to s-recognize the other axioms and the inferences by modus ponens.

Theorem 32. $\mathcal{LT}_0 \subseteq \mathrm{LTSC}_0$.

Proof. Let f be given, and assume first $f \in \mathcal{LT}_0^{cf}$. Define $y = x$ if f begins with R and $y = x_1, x_{21}, x_{31}$ otherwise. We show that there is a TM M_f which, by input y s-computes $f(x)$ in time $a|x|$ and space $|x|$, for some a. Induction on $lh(f)$.

Construction of M_f. Case 1. f is an initial function, consisting of some destructors applied to x_1, or to x_{21}, or to x_{31}; or f is $g \simeq h$, where g and h are initial functions. Define M_f by methods like in Ex. 3.
Case 2. f doesn't begin with R and is not like under case 1. Assume for example that f is $g(x)$ *or* $h(x)$. A TM M_f can be decided, which: (a) applies the TM M_g (associated with g by the inductive construction); (b) stores in its finite control the truth-value corresponding to the value of $g(x)$, and erases $g(x)$ from T_1; (c) applies M_h; (d) replaces the value of $h(x)$ in T_1 by $f(x)$.
Case 3. $f = R(g,h)$. A TM can be defined which behaves in the following way

$z := h(y)$; $q := true$;
while z not empty and q do MAINCYCLE
if q then accept else reject,

where procedure MAINCYCLE is described in next figure by means of some re-write rules (M_g accepts/rejects is short for g accepts/rejects the three rightmost elements of T_1)

T_1	T_2	M_g	$\Rightarrow$	T_1	T_2	q
u	$z\uparrow[Xw$		$\Rightarrow$	$u\phi X$	$z\uparrow w$	
u	$z\uparrow,0w$		$\Rightarrow$	$u\phi 0$	$z\uparrow w$	
u	$z\uparrow,[Xw$		$\Rightarrow$	$u\phi X$	$z\uparrow w$	
$u\phi X\phi Y\phi Z$	z	accepts	$\Rightarrow$	$u\phi X$	z	
$u\phi X\phi Y\phi Z$	z	rejects	$\Rightarrow$	$u\phi X$	z	*false*

Assume given an argument X for f, and define $Y = h(X)$. M_f scans and erases Y from left to right; in the meantime, it uses T_1 as a stack of arguments for M_g. To see the algorithm, imagine that Y is a binary tree τ, represented in the way outlined under Note 10. At each $\uparrow[U$, M_f understands that U is a *parent*, and stores it in T_1; at each $\uparrow,V$, M_f takes V as a *son*, and stores it in T_1 too. When a $]$ is met, a sub-tree of U has been entirely visited and M_g can be applied to check the relation *parent-sons* among the last three elements U,V,W in T_1. If M_g accepts, M_f may forget the sons, though the parent U has still to be checked in its role of son of a node stored at its left in T_1.

Complexity of M_f Assume that f begins by R, since else the result follows immediately by the ind. hyp. Time for M_f may be estimated by summing-up the amounts required for: (a) computing $h(x)$ in T_3; (b) push and pop operations on T_1; (c) calls to M_g. Time for part (a) is $2n$. Time for part (b) is $2n$ too, since every symbol of x is pushed into and popped from T_1 at most once. By the ind. hyp. runtime for M_g is an for some constant a; hence, since every symbol of x is processed by M_g at most twice (the first in the role of son, the second as parent) time for part (c) is $2an$. The overall time complexity of M_f is, therefore, $2(a+1)n$.

Conclusion of the proof The theorem is proved by the arguments above for all f beginning with R. If $f \in \mathcal{LT}_0^{cf}$ doesn't begin with R, the result follows by considering the amount of time $2|x|$, needed to take x into $y = x_1, x_{21}, x_{31}$. Trivial adaptations to the arguments above prove the result for all $f \in \mathcal{LT}_0 \setminus \mathcal{LT}_0^{cf}$.

Note 33. Let $lh_0(f)$ denote the length of f if the constants occurring in its *cons* and assignments are ignored; and let $nr(f)$ denote the level of nesting of its recursions. By an easy, though cumbersome, analysis of last proof one proves that the time complexity of M_f is $lh_0(n)2^{nr(f)}$. By Ex. 3 and considering that the level of recursion nesting in *de* is 4, this yields a rough estimate of $100n$ for the time complexity of *de*. Thus our syntactic algorithms are in Jones class LIN(cn) for c approximatively equal to 100. We report this fact because hierarchy LIN(abn) (i) is proper for a constant b; and (ii) is based on a time measure more "faithful to current programming practice", than the *trick* of obtaining a linear speed-up by an increase in the tape alphabet. These two points give sense to our claim that completeness is lost when a modest amount of computational resources is added to the sum operator.

References

[Caporaso, 79] S. Caporaso, *Consistency proof without transfinite induction for a formal system for Turing machines*. Arch. Math. Logik u. G. 19(1979)157-164.

[Gödel, 34] K.Gödel, *On undecidable propositions of formal mathematical systems*. In Fefermann et alii (eds)*Collected papers*, vol. I (Oxford University Press, 1986).

[Jones, 93] N.D. Jones, *Constants time factors do matter*. In S. Homer(Ed.) STOC93, Symposium on Theory of Computing. (ACM Press, 1993)

[Jones, 97] N.D. Jones, *Computability and Complexity from a programming perspective*. (MIT Press, 1997)

[Kleene, 52] S.C. Kleene, *Introduction to Metamathematics*. (North-Holland, Amsterdam, 1952).

[Kneale, 62] W.Kneale, and M. Kneale, *The development of logic*. (Clarendon Press, Oxford, 1962).

[Rose, 84] H.E. Rose, *Subrecursion: Functions and hierarchies* . (Oxford University Press, Oxford, 1984).

[Smullyan, 61] R.M.Smullyan, *Theory of formal systems*. Annals of mathematical studies 47. (Princeton University Press, 1961).

[Hao Wang, 57] Hao Wang, *A variant to Turing's theory of computing machines*. Journal of the ACM 4.1(1957).