Double Exponential Inseparability Of Robinson Subsystem Q_+ From The Unsatisfiable Sentences In The Language Of Addition

Giovanni Faglia   Paul Young

Technical Report 92-09-01
September, 1992

Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195

Double Exponential Inseparability Of Robinson Subsystem Q_+ From The Unsatisfiable Sentences In The Language Of Addition[†]

Giovanni Faglia *Paul Young*

One of the first and seminal results in the study of the complexity of logical theories is the work by Fischer and Rabin [FR74] which proves the double-exponential complexity of the set S_+ of true sentences of addition.

The set S_+ is a logical theory—namely the theory studied by Presburger in [Pre29] and proposed as a first significant stage in the foundation of mathematics by Hilbert and Bernays in [HB34]. In particular S_+ is a complete and decidable first order theory.

This interesting theory is in several ways very powerful. On the one hand all true statements of the arithmetic of addition can be proved in it. On the other hand, being a complete theory, S_+ is a maximal consistent set and cannot be extended to any stronger consistent system. The power of S_+ simplifies the mathematical constructions needed to prove its double exponential complexity, but somehow implies that the result may be limited.

How deeply does the Fischer and Rabin lower bound really affect the possibility of a feasible theory of arithmetic? How strong must the logical system be for the lower bound to apply, and how much is this due to being a logical theory instead of a more general set of sentences? In [You85] a finitely axiomatizable subtheory of S_+ called ADDAX is given for which *any set* of sentences that separates ADDAX from the logically false sentences in the language of addition is *exponentially* difficult. Compton and Henson obtained in [CH90] a hereditary lower bound for S_+, which implies that any subset of S_+ which contains all logically valid sentences cannot be recognized by a Turing machine working in double exponential time.

In this work we prove a *double* exponential time inseparability result for a *finitely* axiomatizable theory Q_+ that is weaker than ADDAX. *Every set that separates Q_+ from the logically false sentences of addition is not recognizable by any Turing machine working in double exponential time.* This implies also that any theory of addition that is consistent with Q_+— in particular any theory contained in S_+—is at least double exponential time difficult. The result also subsumes both the hereditary lower bounds in [CH90] and the single exponential inseparability in [You85].

[†] The results presented here will be included as part of the first author's doctoral dissertation [Fag93] written under the direction of Alberto Bertoni, Pierangelo Miglioli and Paul Young. The research was performed partly while Young was a visiting Professor at Università degli Studi di Milano—supported by Consiglio Nazionale delle Ricerche—and partly while Faglia—supported by an Italian Dottorato di Ricerca scholarship—was a visiting scholar in the Department of Computer Science and Engineering at the University of Washington, Seattle, USA.

Thus the inseparability result we give is an improvement on the known lower bounds for arithmetic theories. It is stronger than the inseparability in [You85] because the lower bound is higher and the inseparability wider (since Q_+ is a proper subset of ADDAX), and the computational model more general—alternating versus non-deterministic Turing machines.

It is also strictly stronger than the hereditary lower bound in [CH90]. The hereditary lower bound in [CH90] is equivalent to the double exponential inseparability of the *complete* theory S_+, (which contains Q_+), from the unsatisfiable sentences of the language, while the inseparability for Q_+ implies in particular that *every* consistent theory extending Q_+, (including S_+ in particular), is double exponentially inseparable from the unsatisfiable sentences.

Historically, our result is expecially pleasing. In early work, Raphael Robinson extended Gödel's famous results on the undecidability of Peano Arithmetic by showing that his weaker theory Q is recursively inseparable from the logically false sentences of the language of Peano Arithmetic, [Rob50]. The axioms for our theory Q_+ are obtained from the axioms for Robinson's system by simply eliminating all axioms which refer to multiplication. Our results are then *exactly* analogous to Robinson's: not only is Presburger Arithmetic double exponentially difficult, as shown by Fischer and Rabin, but the weaker theory Q_+ is double exponentially inseparable from the logically false sentences of the language of Presburger Arithmetic.

1. OVERVIEW

The material in this report is organized in the following way. As shown by a careful analysis in the first sections, a small set of properties of addition and successor are sufficient to define an $\exp(3, k)$-representation '$\underset{k}{\otimes}$' of the multiplication function in the sense of [FY92]. These properties can be expressed by a finite set of sentences, and these sentences are then taken as axioms of a simple theory Q_+.

A proof of double-exponential multiplicative inseparability in the sense of [You85]—which in turn implies the standard non-deterministic Turing machine lower bound—almost immediately follows for the stronger theory ADDAX from the $\exp(3, k)$-representability of the multiplication function.

Some proofs of the inseparability result for Q_+ are then shown in detail. All the construction that has been done in [FR74] for the complete theory S_+ can be done for Q_+, or for any other finitely axiomatizable system in which all of the necessary properties of addition are provable, with the major difference that this much weaker theory is not complete and we cannot refer to truth in a 'standard' model to decide membership to Q_+. A double-exponential alternating time and linear alternation lower bound for Q_+ is derived in this way.

A hereditary lower bound in the sense of [CH90] for any theory extending Q_+ can be immediately inferred as a by-product of the above construction.

Finally the result is proven by reducing the decision problem of the complete Presburger theory to the problem of deciding any separating set.

2. A SUFFICIENT SET OF AXIOMS

In this section the theory Q_+ is defined, and all the relevant properties that are needed in the following sections are described and proven.

First of all, Q_+ is a theory in the first order language $\mathcal{L}_+$ with two function symbols, that are '+' for addition and 'S' for successor, one constant symbol '0' and one predicate symbol, namely '=' for equality. Only + and = are strictly necessary, while S and 0 can be eliminated.

2.1 *A small set of axioms.*

Like S_+, Q_+ also has to be a theory with equality, that is a theory where the symbol = means standard identity of elements. The formulas corresponding to the desired properties of + and S—that in first order arithmetic are usually proven using the induction schema—here are taken as axioms (namely d. and e.):

a. $\forall x\; x = x$

b. $\forall xy\; x = y \rightarrow y = x$

c. $\forall xyz\; (x = y \wedge y = z) \rightarrow x = z$

d. $\forall xy\; x = y \rightarrow Sx = Sy$

e. $\forall x_1 x_2 y\; x_1 = x_2 \rightarrow (x_1 + y = x_2 + y \wedge y + x_1 = y + x_2)$

Also the noninductive Peano axioms for S and + are included in the axiomatization of Q_+:

f. $\forall x\; \neg Sx = 0$

g. $\forall xy\; Sx = Sy \rightarrow x = y$

h. $\forall x\; x + 0 = x$

i. $\forall xy\; x + Sy = S(x + y)$

One major property of the structure of natural numbers characterizes the theory Q_+, namely the existence of predecessor for positive numbers. This property implies that $x \le \overline{n}$ is equivalent to the disjunction $x = 0 \vee x = S0 \vee \cdots \vee x = \overline{n}$ of all finite possible cases and this will be used in several situations to eliminate quantifiers from formulas. It implies also that in every model each element denoted by a numeral is comparable to every other element—even if the order happens to be partial. This sort of totality is used in the construction of $x \underset{k}{\otimes} y = z$, where it guarantees that an element is uniquely determined by the property of being denoted by a numeral and of being minimal in a certain set. Finally the existence of predecessor implies that the theory decides every divisibility statement: for any integers m and n either $n|m$ or $n \not| m$ is a theorem.

The following sentence, which expresses the existence of predecessor, is the last axiom of the theory Q_+:[1]

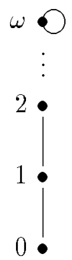
j. $\forall x\; x = 0 \vee (\exists y\, x = Sy)$.

---

[1] The theory ADDAX studied in [You85] can be given by the axiom system a.–j. plus the additional tricotomy axiom $\forall x \forall y\; x < y \vee x = y \vee y < x$, where the symbol $<$ may be considered to be either primitive or defined from + in a standard way. Since tricotomy is not a theorem of Q_+, the latter is strictly weaker than ADDAX.

Note that axioms a. through j. happen to be exactly the axioms for $+$ and S in the well-known Robinson system Q (cf. [Rob50])—hence the choice of the name Q_+. Our purpose has been to obtain from the Fischer and Rabin's work a result similar to the one obtained by Raphael Robinson from the Gödel theorem, and it is worth noting that we came out with the same kind of inseparability result for exactly the corresponding theory!

The theory Q_+ is contained in S_+, because all of its axioms are true in the standard model. On the other hand a very simple model of Q_+ that is not a model of S_+ can be constructed. Consider the normal model—i.e. a model in which '=' means identity—obtained adding a limit element ω on top of the natural numbers:



Let $S\omega$ be ω and $n + \omega = \omega + n = \omega + \omega = \omega$ for each integer n. Axioms a. through j. hold in this model, but the sentence $\forall x \neg x = Sx$ is false here, although it is true in the standard interpretation. Hence this model is not a model of S_+ and $Q_+ \neq S_+$.

The finite system Q_+ is then strictly smaller than the complete theory[2] S_+—indeed it is much weaker. A model of Q_+ similar to the preceding one falsifies commutativity and associativity of addition, which are not provable in Q_+:

$$Q_+ \nvdash \forall xy\ x + y = y + x$$
$$Q_+ \nvdash \forall xyz\ x + (y + z) = (x + y) + z$$

To see this, this time add three limit elements ω_1, ω_2 and ω_3 on top of the natural numbers, and for any integers i, j and n, $1 \leq i, j \leq 3$, let $S\omega_i = n + \omega_i = \omega_i + n = \omega_i$ and

$$\omega_i + \omega_j = \begin{cases} \omega_i & \text{if } i + j \text{ is odd} \\ \omega_j & \text{otherwise.} \end{cases}$$

Again, this is a model of Q_+. But $(\omega_1 + \omega_2) + \omega_3 = \omega_3 \neq \omega_1 = \omega_1 + (\omega_2 + \omega_3)$ and $\omega_1 + \omega_2 = \omega_1 \neq \omega_2 = \omega_2 + \omega_1$.

2.2 *Properties of* Q_+

Several well-known properties of Q_+ will be used to prove our inseparability result. Some of these are logical properties due to axioms a. through e. and some are simply due to Q_+ being a first order theory. These axioms guarantee that substitutivity of equality

$$x = y \rightarrow (\varphi(x, x) \rightarrow \varphi(x, y)) \text{ for all formulas } \varphi$$

[2] The complete theory S_+ in any case is not finitely axiomatizable (cf. [BS69]). We are grateful to Paola D'Aquino and Angus Macintyre for pointing out this reference.

holds in Q_+ (cf. [Men87, p. 76]), which is then by definition a *theory with equality*.

Some properties, as already mentioned, come instead from the arithmetical power of the theory, and most of them are already well studied in the literature (cf. [Rob50], [TMR53] or [Men87]).

Properties of the first kind are needed to apply several tools—which are commonly used in the field of the complexity of logical theories—for example to replace long formulas with shorter forms. Usually, in order for the substitution to operate soundly, each shorter form is shown to be equivalent to the original formula with semantical arguments.

These tools can be used when dealing with any first order theory with equality. In all first order theories the substitution of a subformula with an equivalent form gives rise to an equivalent formula, as stated by the *equivalence* and *replacement theorems.*

Theorem *(Equivalence and Replacement)*

Let $\Phi(X)$ be a formula of the first order language $\mathcal{L}(X)$ obtained by adjoining to the generic language $\mathcal{L}$ a new atomic 0-ary predicate symbol X, that works as place holder, and let β, γ be formulas of the plain $\mathcal{L}$. If $\{z_1, \ldots, z_n\} = \mathrm{bound}(\Phi(X)) \cap \mathrm{free}(\beta \leftrightarrow \gamma)$ is the set of bound variables in $\Phi(X)$ that are free in β or in γ, then the following statements hold for any first order theory T in the language $\mathcal{L}$:

$$T \vdash [\forall \underline{z}\, \beta \leftrightarrow \gamma] \;\rightarrow\; [\Phi(\beta) \leftrightarrow \Phi(\gamma)] \qquad \textit{(equivalence theorem)}$$
$$\textit{If}\;\; T \vdash \beta \leftrightarrow \gamma \;\;\textit{then}\;\; T \vdash \Phi(\beta) \leftrightarrow \Phi(\gamma) \qquad \textit{(replacement theorem)}$$

where $\Phi(\beta)$ [$\Phi(\gamma)$] is the result of replacing all occurences of X in Φ with β [γ].

Since we need to systematically abbreviate formulas, the preceeding theorem guarantees that each success that we obtain in one place extends immediately to many other places by replacement. Note in particular that for the antecedent of the replacement theorem to hold, it is not necessary that $\beta \leftrightarrow \gamma$ be logically valid; the formula need just be a logical consequence of the theory T.

Several ways to create equivalent shorter forms of a formula are known in theories with equality. It is worth noting that a common matrix can be ascribed to most of these equivalencies; indeed there is a well-known property valid for all languages and theories with equality:

Proposition

In any theory with equality $T_=$, for any formula $\varphi(x, \underline{y})$ and variable z not appearing in $\varphi(x, \underline{y})$ the following are theorems:

$$T_= \vdash \forall z\,[\varphi(z, \underline{y}) \leftrightarrow \forall x\,(x = z \rightarrow \varphi(x, \underline{y}))]$$
$$T_= \vdash \forall z\,[\varphi(z, \underline{y}) \leftrightarrow \exists x\,(x = z \wedge \varphi(x, \underline{y}))].$$

where $\varphi(z, \underline{y})$ is the result of replacing all occurences of x in φ with z. Hence for any formula $\varphi(x, \underline{y})$ and term t free for x in φ the following are theorems:

$$T_= \vdash \varphi(t, \underline{y}) \leftrightarrow \forall x\,(x = t \rightarrow \varphi(x, \underline{y}))$$
$$T_= \vdash \varphi(t, \underline{y}) \leftrightarrow \exists x\,(x = t \wedge \varphi(x, \underline{y})).$$

For example $\varphi(t_1) \wedge \varphi(t_2)$ is equivalent to $\forall x\,(x = t_1 \rightarrow \varphi(x)) \wedge \forall x\,(x = t_2 \rightarrow \varphi(x))$ and finally to $\forall x\,[(x = t_1 \vee x = t_2) \rightarrow \varphi(x)]$. The original formula can be twice as long as the last form, and this widely known technique will be used again and again in this work.

Let us now look at the arithmetical properties of Q_+, and list those that are used to prove the inseparability result.

Some common abbreviations have to be introduced at this point.

The expression $\alpha \leq \beta$ will be used as an abbreviation for $\exists u\,[\alpha + u = \beta]$, and $\alpha < \beta$ for $\exists u\,[\alpha + u = \beta \wedge \neg u = 0]$, taking u as a distinct variable that without loss of generality can be taken to appear only in such contexts. Note that $\leq$ does not necessarily have in Q_+ the properties of an order relation. For example, one of the previous models shows that $\leq$ is not necessarily anti-symmetric.

For any positive integer n and expression α, $n \cdot \alpha$ will be used instead of the term $\alpha + (\alpha + (\cdots + (\alpha + \alpha)\cdots)$ with n occurrences of α. Since commutativity and associativity of addition are not provable in Q_+, a different abbreviation is needed for terms like $((\alpha+\alpha)+\alpha)+(\alpha+\alpha)$ in which n occurrences of α appear but they are structured in a generic tree instead of a list, as is in $n \cdot \alpha$. Let $n \cdot_\tau \alpha$ be a notation for such terms; in each case it will be explicitly stated whether the notation stands for a particular term of that kind or for any such term. For a matter of homogeneity, in some expressions $0 \cdot \alpha$ and $0 \cdot_\tau \alpha$ will be used as another name of the closed term 0.

For any integer expression a—in the meta-language—with value n, $\overline{a}$ is an abbreviation for the numeral $SS \cdots S0$ with n occurrences of the successor symbol: e.g. if $q \cdot m + r = n$ then $\overline{q \cdot m + r}$ is an abbreviation for $S^n 0$.

<u>Proposition</u>

For all integers n, m, q and r the following are theorems of Q_+:

1. $Q_+ \vdash \forall z\,[\overline{n \cdot m} = n \cdot_\tau z \leftrightarrow z = \overline{m}]$ *for each term* $n \cdot_\tau z$
2. $Q_+ \vdash \forall z\,[\overline{q \cdot m + r} = m \cdot_\tau z + \overline{r} \leftrightarrow z = \overline{q}]$ *for each term* $m \cdot_\tau z$
3. $Q_+ \vdash \forall y\,[y \leq \overline{n} \leftrightarrow (y = 0 \vee y = S0 \vee \cdots \vee y = \overline{n})]$
4. $Q_+ \vdash \forall y\,[\overline{n} = m \cdot_\tau y \;\rightarrow\; (y = 0 \vee y = S0 \vee \cdots \vee y = \overline{n})]$, *for* $m > 0$
5. $Q_+ \vdash \neg \exists z\, \overline{n} = m \cdot_\tau z$ *if m does not divide n* *(non-divisibility)*
6. $Q_+ \vdash t = \overline{t}$ *for each closed term t, being $\overline{t}$ the numeral corresponding to the integer denoted by t in $\mathcal{N}$*
7. $Q_+ \vdash \neg t = \overline{s}$ *for each closed term t and numeral $\overline{s}$ distinct from $\overline{t}$*
8. $Q_+ \vdash \forall \underline{z}\,\{[\forall y\, y \leq t \rightarrow \varphi(y, \underline{z})] \leftrightarrow [\varphi(0, \underline{z}) \wedge \cdots \wedge \varphi(\overline{t}, \underline{z})]\}$ *for each closed term* t
9. $Q_+ \vdash \forall \underline{z}\,\{[\exists y\, y \leq t \wedge \varphi(y, \underline{z})] \leftrightarrow [\varphi(0, \underline{z}) \vee \cdots \vee \varphi(\overline{t}, \underline{z})]\}$ *for each closed term* t
10. $Q_+ \vdash z \leq \overline{n} \vee \overline{n} \leq z$ *(totality, with respect to numerals)*

<u>*proof*</u> Properties 1. and 2. are proven by a straightforward induction, using substitutivity of equality when necessary. Statement 3. has a simple proof by induction on n.

Property 4. can be proved by induction on m—simultaneously for each n. For $m = 1$ the proof is immediate. If $m > 1$ and $m \cdot_\tau y$ is $m_1 \cdot_\tau y + m_2 \cdot_\tau y$ then $Q_+ \vdash \overline{n} = m \cdot_\tau y \;\rightarrow\; \exists z \overline{n} = m_1 \cdot_\tau y + z$. Hence $Q_+ \vdash \overline{n} = m \cdot_\tau y \;\rightarrow\; (m_1 \cdot_\tau y = 0 \vee m_1 \cdot_\tau y = S0 \vee \cdots \vee m_1 \cdot_\tau y = \overline{n})$, and the inductive hypothesis can be applied to each disjunct $m_1 \cdot_\tau y = \overline{s}$.

The non-divisibility property is a direct consequence of the previous one. Property 6. can be easily proven by induction on the length of the term t, using axioms h. and i., and 7. is a consequence of 6. and axiom j.

By 3., 6. and the replacement theorem, the result of substituting the subformula $y \leq t$ with $(y = 0 \vee y = S0 \vee \cdots \vee y = \overline{t})$ in 8. or 9. is equivalent to the original form. Then both properties 8. and 9. can be proven from the properties of theories with equality stated in a previous proposition.

Totality of $\leq$ with respect to numerals is proven by n applications of the axiom j., noting that $\exists y\ z = S^n y$ implies $\overline{n} \leq z$. ■

Let $\varphi(\underline{z}) \in \mathcal{L}_+$ be a formula with free variables in the set $\{\underline{z}\}$, in which some quantified variables are bounded by a closed term t, like for example x in $\forall x[(x \leq S(S0+S0)) \rightarrow x + z_1 \leq z_2]$. Then there exists a formula $\sigma(\underline{z})$—with free variables in the set $\{\underline{z}\}$—that is equivalent to $\varphi(\underline{z})$ in Q_+:

$$Q_+ \vdash \forall \underline{z}\ [\varphi(\underline{z}) \leftrightarrow \sigma(\underline{z})]$$

in which the bounded quantifications have been replaced by the equivalent conjunctions or disjunctions of all possible values.

<u>Theorem</u>*(Bounded Quantifier Elimination)*

Let $\varphi(\underline{z}) \in \mathcal{L}_+$ be a formula with free variables in the set $\{\underline{z}\}$, such that each quantified variable in $\varphi(\underline{z})$ is bounded by some closed term t. Then there exists a quantifier free formula $\sigma(\underline{z})$—with free variables in the set $\{\underline{z}\}$—that is equivalent to $\varphi(\underline{z})$ in Q_+:

$$Q_+ \vdash \forall \underline{z}\ [\varphi(\underline{z}) \leftrightarrow \sigma(\underline{z})].$$

If $\{\underline{z}\}$ is $\emptyset$ and φ is a sentence, then either φ or $\neg\varphi$ is provable in Q_+:

$$Q_+ \vdash \varphi \textit{ if and only if } Q_+ \nvdash \neg\varphi.$$

3. THE DEFINITION OF PRODUCT

In this section a sequence of formulas is built that is intended to represent the product function in Q_+, simplifying and adapting the construction made in [FR74] for the complete theory S_+. The sequence has linear growth, i.e. there exists a positive integer c such that for all k the length of the k-th formula in this sequence, denoted by $x \underset{k}{\otimes} y = z$, is less than $c \cdot k$. The formula $x \underset{k}{\otimes} y = z$ has exactly three free variables and is equivalent in Q_+ to a quantifier free formula, namely:

$$Q_+ \vdash \forall xyz\ (x \underset{k}{\otimes} y = z \leftrightarrow \bigvee_{n,m \leq m_{k+4}} [x = \overline{n} \wedge y = \overline{m} \wedge z = \overline{n \cdot m}])$$

where $m_{k+4} >> \exp(3, k)$ is for each k the least common multiple of $\{2, \ldots, \exp(2, k+4) - 1\}$. In particular for all numerals $\overline{r}, \overline{s}$ and $\overline{n}$

$$Q_+ \vdash \overline{r} \underset{k}{\otimes} \overline{s} = \overline{n} \quad \text{iff} \quad Q_+ \nvdash \neg\overline{r} \underset{k}{\otimes} \overline{s} = \overline{n} \quad \text{iff} \quad [r, s \leq m_{k+4}\ \&\ n = rs].$$

3.1 *Definition of minor times*

The construction of $x \underset{k}{\otimes} y = z$ starts with the inductive definition of a weaker linear sequence $\{M_k(x, y, z)\}_{k \geq 0}$ that in Q_+ represents the product function provided the first argument x belongs to the initial segment $\{0 \cdots \exp(2, k) - 1\}$ of the natural numbers. It is shown in particular that a sequence like the one described in [FR74] for the complete theory S_+ is appropriate for the theory Q_+, because in this theory it is provably equivalent to a quantifier free formula:

$$Q_+ \vdash \forall xyz M_k(x,y,z) \leftrightarrow \bigvee_{s<\exp(2,k)} [x = \overline{s} \wedge z = s \cdot_\tau y].$$

for some terms of the form $s \cdot_\tau y$, one for each s.

The definition is given by induction:

$$M_0(x,y,z) \quad \text{is} \quad (x = 0 \wedge z = 0) \vee (x = S0 \wedge z = y).$$

Recall that for each integer $n < m^2$ there exists a triple $\langle s_1, s_2, s_3 \rangle$, $s_i < m$, such that $n = s_2 + s_3 + s_1 \cdot s_1$ and $s_2 \leq s_3 \leq s_1 \wedge s_3 - s_2 \leq 1$. This property is crucial to understand the inductive step of the definition. The step defines $M_{k+1}(x,y,z)$ in such a way that

$$\begin{aligned}
&\vdash \forall xyz \, \big[M_{k+1}(x,y,z) \quad \leftrightarrow \\
&\quad \exists u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 \\
&\qquad M_k(u_1,u_1,u_4) \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge x = u_2 + u_3 + u_4 \wedge \\
&\qquad M_k(u_1,y,u_5) \wedge M_k(u_1,u_5,u_6) \wedge M_k(u_2,y,u_7) \wedge M_k(u_3,y,u_8) \wedge z = u_6 + u_7 + u_8\big].
\end{aligned}$$

The formula on the right-hand side of the previous implication is logically equivalent to

$$\begin{aligned}
&\exists u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 \\
&\quad x = u_2 + u_3 + u_4 \wedge z = u_6 + u_7 + u_8 \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge \\
&\quad [\forall x'y'z'(x = x' \wedge y = y' \wedge z = z') \rightarrow \\
&\qquad M_k(u_1,u_1,u_4) \wedge M_k(u_1,y',u_5) \wedge M_k(u_1,u_5,u_6) \wedge M_k(u_2,y',u_7) \wedge M_k(u_3,y',u_8)]
\end{aligned}$$

which is logically equivalent to

$$\begin{aligned}
&\exists u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 \\
&\quad x = u_2 + u_3 + u_4 \wedge z = u_6 + u_7 + u_8 \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge \\
&\quad [\forall x'y'z'(x = x' \wedge y = y' \wedge z = z') \rightarrow \\
&\qquad \forall xyz[(x = u_1 \wedge y = u_1 \wedge z = u_4) \vee (x = u_1 \wedge y = y' \wedge z = u_5) \vee \\
&\qquad\quad (x = u_1 \wedge y = u_5 \wedge z = u_6) \vee (x = u_2 \wedge y = y' \wedge z = u_7) \vee \\
&\qquad\quad (x = u_3 \wedge y = y' \wedge z = u_8)] \rightarrow M_k(x,y,z)]
\end{aligned}$$

Define then $M_{k+1}(x,y,z)$ to be the formula above. As k increases, the length of $M_k(x,y,z)$ increases linearly in k.

<u>Theorem</u>

The formula $M_k(x,y,z)$ is equivalent in Q_+ to

$$\bigvee_{s<\exp(2,k)} [x = \overline{s} \wedge z = s \cdot_\tau y]$$

for some terms of the form $s \cdot_\tau y$, one for each s.

proof The base is straightforward. As for the induction step, as noticed above the formula $M_{k+1}(x, y, z)$ is logically equivalent to:

$$\begin{aligned}&\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9\\ &\quad M_k(u_1, u_1, u_4) \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge x = u_2 + u_3 + u_4 \wedge\\ &\quad M_k(u_1, y, u_5) \wedge M_k(u_1, u_5, u_6) \wedge M_k(u_2, y, u_7) \wedge M_k(u_3, y, u_8) \wedge z = u_6 + u_7 + u_8\end{aligned}$$

that is equivalent in Q_+, using the replacement theorem and the inductive hypothesis, to the following formula—for some terms $s_1 \cdot_\tau u_1$, $s'_1 \cdot_\tau y$, $s''_1 \cdot_\tau u_5$, $s_2 \cdot_\tau y$, $s_3 \cdot_\tau y$:

$$\begin{aligned}&\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9 \bigvee_{s_1<\exp(2,k)} [u_1 = \overline{s_1} \wedge u_4 = s_1 \cdot_\tau u_1] \wedge\\ &\quad x = u_2 + u_3 + u_4 \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge\\ &\quad \bigvee_{s'_1<\exp(2,k)} [u_1 = \overline{s'_1} \wedge u_5 = s'_1 \cdot_\tau y] \wedge \bigvee_{s''_1<\exp(2,k)} [u_1 = \overline{s''_1} \wedge u_6 = s''_1 \cdot_\tau u_5] \wedge\\ &\quad \bigvee_{s_2<\exp(2,k)} [u_2 = \overline{s_2} \wedge u_7 = s_2 \cdot_\tau y] \wedge \bigvee_{s_3<\exp(2,k)} [u_3 = \overline{s_3} \wedge u_8 = s_3 \cdot_\tau y] \wedge z = u_6 + u_7 + u_8.\end{aligned}$$

Using distributivity of $\wedge$ over $\vee$ the above formula is shown to be logically equivalent to

$$\begin{aligned}&\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9 \bigvee_{s_1<\exp(2,k)} \bigvee_{s'_1<\exp(2,k)} \bigvee_{s''_1<\exp(2,k)} \bigvee_{s_2<\exp(2,k)} \bigvee_{s_3<\exp(2,k)}\\ &\quad \big[u_1 = \overline{s_1} \wedge u_4 = s_1 \cdot_\tau u_1 \wedge x = u_2 + u_3 + u_4 \wedge u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge\\ &\quad u_1 = \overline{s'_1} \wedge u_5 = s'_1 \cdot_\tau y \wedge u_1 = \overline{s''_1} \wedge u_6 = s''_1 \cdot_\tau u_5 \wedge u_2 = \overline{s_2} \wedge\\ &\quad u_7 = s_2 \cdot_\tau y \wedge u_3 = \overline{s_3} \wedge u_8 = s_3 \cdot_\tau y \wedge z = u_6 + u_7 + u_8\big]\end{aligned}$$

As shown in the preceeding proposition (part 7.), Q_+ proves the negation of each disjunct corresponding to integers $s_1, s'_1, s''_1, s_2, s_3$ such that not $s_1 = s'_1 = s''_1$. From $\neg B \vdash [A \vee B] \leftrightarrow [(A \vee B) \wedge \neg B]$ and $\vdash [(A \vee B) \wedge \neg B] \leftrightarrow [A \wedge \neg B]$ it follows that the formula displayed above is equivalent in Q_+ to the following[3]:

$$\begin{aligned}&\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9 \bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[u_1 = \overline{s_1} \wedge u_2 = \overline{s_2} \wedge u_3 = \overline{s_3} \wedge\\ &\quad u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge u_4 = s_1 \cdot_\tau u_1 \wedge x = u_2 + u_3 + u_4 \wedge\\ &\quad u_5 = s_1 \cdot_\tau y \wedge u_6 = s_1 \cdot_\tau u_5 \wedge u_7 = s_2 \cdot_\tau y \wedge u_8 = s_3 \cdot_\tau y \wedge z = u_6 + u_7 + u_8\big].\end{aligned}$$

Since $\exists x\ [\alpha(x) \vee \beta(x)] \quad \leftrightarrow \quad [\exists x\alpha(x) \vee \exists x\beta(x)]$ is logically valid, the preceding is logically equivalent to:

$$\begin{aligned}&\bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9\ [u_1 = \overline{s_1} \wedge u_2 = \overline{s_2} \wedge u_3 = \overline{s_3} \wedge\\ &\quad u_1 = u_3 + u_9 \wedge (u_3 = u_2 \vee u_3 = Su_2) \wedge u_4 = s_1 \cdot_\tau u_1 \wedge x = u_2 + u_3 + u_4 \wedge\\ &\quad u_5 = s_1 \cdot_\tau y \wedge u_6 = s_1 \cdot_\tau u_5 \wedge u_7 = s_2 \cdot_\tau y \wedge u_8 = s_3 \cdot_\tau y \wedge z = u_6 + u_7 + u_8]\big].\end{aligned}$$

[3] Each term of the form $s'_1 \cdot_\tau \alpha$ or $s''_1 \cdot_\tau \alpha$ can be considered to be also of the form $s_1 \cdot_\tau \alpha$, if $s_1 = s'_1 = s''_1$.

Q_+ is a theory with equality, hence $Q_+ \vdash \varphi(t, \underline{y}) \leftrightarrow \exists x\,(x = t \wedge \varphi(x, \underline{y}))$ for any formula $\varphi(x, \underline{y})$ and term t free for x in φ. Applying this property to each disjunct, the formula displayed above can be shown equivalent to:

$$\bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[\exists u_4 u_5 u_6 u_7 u_8 u_9 \\ [u_4 = s_1 \cdot_\tau \overline{s_1} \wedge x = \overline{s_2} + \overline{s_3} + u_4 \wedge \overline{s_1} = \overline{s_3} + u_9 \wedge (\overline{s_3} = \overline{s_2} \vee \overline{s_3} = S\,\overline{s_2}) \wedge \\ u_5 = s_1 \cdot_\tau y \wedge u_6 = s_1 \cdot_\tau u_5 \wedge u_7 = s_2 \cdot_\tau y \wedge u_8 = s_3 \cdot_\tau y \wedge z = u_6 + u_7 + u_8]\big].$$

Applying again the same property to the quantified variables u_4 through u_8—in the natural order, so that u_6 is correctly eliminated after u_5—the following equivalent formula is derived:

$$\bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[\exists u_9\; x = \overline{s_2} + \overline{s_3} + (s_1 \cdot_\tau \overline{s_1}) \wedge z = s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y \wedge \\ (\overline{s_3} = \overline{s_2} \vee \overline{s_3} = S\,\overline{s_2}) \wedge \overline{s_1} = \overline{s_3} + u_9\big]$$

The last existential quantifier can be carried inside in order to obtain the logically equivalent:

$$\bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[x = \overline{s_2} + \overline{s_3} + (s_1 \cdot_\tau \overline{s_1}) \wedge z = s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y \wedge \\ (\overline{s_3} = \overline{s_2} \vee \overline{s_3} = S\,\overline{s_2}) \wedge \exists u_9\; \overline{s_1} = \overline{s_3} + u_9\big]$$

Using an instance of $Q_+ \vdash \forall y\,[y \leq \overline{n} \leftrightarrow (y = 0 \vee y = S0 \vee \cdots \vee y = \overline{n})]$ and replacement, the following formula can be shown equivalent in Q_+ to the one above:

$$\bigvee_{s_1,s_2,s_3<\exp(2,k)} \big[x = \overline{s_2} + \overline{s_3} + (s_1 \cdot_\tau \overline{s_1}) \wedge z = s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y \wedge \\ (\overline{s_3} = \overline{s_2} \vee \overline{s_3} = S\,\overline{s_2}) \wedge (\overline{s_3} = 0 \vee \overline{s_3} = S0 \vee \cdots \vee \overline{s_3} = \overline{s_1})\big]$$

Finally $Q_+ \vdash \neg\alpha$ for each disjunct α such that $s_3 \not\leq s_1$ or $(s_2 \neq s_3 \neq s_2 + 1)$, and as before the preceding formula can be shown equivalent in Q_+ to the following one—where the last two conjuncts of each disjunct have been suppressed, being provable in Q_+:

$$\bigvee_{\substack{s_1,s_2,s_3<\exp(2,k)\\ s_3\leq s_1,(s_3=s_2\vee s_3=s_2+1)}} \big[x = \overline{s_2} + \overline{s_3} + (s_1 \cdot_\tau \overline{s_1}) \wedge z = s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y\big]$$

For each triple $\langle s_1, s_2, s_3\rangle$, $Q_+ \vdash \overline{s_2} + \overline{s_3} + s_1 \cdot_\tau \overline{s_1} = \overline{s_2 + s_3 + s_1 \cdot s_1}$, and $s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y$ is by definition a term of the form $(s_1 \cdot s_1 + s_2 + s_3) \cdot_\tau y$. Hence each disjunct in the previous formula is equivalent in Q_+ to the corresponding disjunct in the following:

$$\bigvee_{\substack{s_1,s_2,s_3<\exp(2,k)\\ s_3\leq s_1,(s_3=s_2\vee s_3=s_2+1)}} \big[x = \overline{s_2 + s_3 + s_1 \cdot s_1} \wedge z \;=\; (s_1 \cdot s_1 + s_2 + s_3) \cdot_\tau y\big] \tag{$\dagger$}$$

For each $n < \exp(2, k)^2 = \exp(2, k+1)$ there exists exactly one triple $\langle s_1, s_2, s_3\rangle$, $s_i < \exp(2, k)$, such that $n = s_2 + s_3 + s_1 \cdot s_1$ and $s_3 \leq s_1 \wedge (s_3 = s_2 \vee s_3 = s_2 + 1)$. Then the previous formula can be written[4]

$$\bigvee_{n<\exp(2,k+1)} [x = \overline{n} \wedge z = n \cdot_\tau y] \;\blacksquare$$

[4] At this point it should be clear why the conjuncts:

$(\exists u_9\, u_1 = u_3 + u_9) \wedge (u_3 = u_2 \vee u_3 = Su_2)$, 3.2 *Definition of major times*

We are next going to raise the upper limit from a double to a triple exponential. Since we will need to refer to the number of primes $\pi(n)$ less than or equal to a given number n, we first report the derivation of a lower bound on $\pi(n)$ (namely $\sqrt{n} < \pi(n)$ if n is big enough) that is very simple and sufficient to our purposes. Let m be any integer and p a prime less than or equal to m and consider the factors p^h and p^k in the prime decomposition of m and $m!$—i.e. h is the greatest power of p that divides m and so is k for the factorial of m. Each multiple of p less than or equal to m appears in the product $m!$, and increments k of a unit; indeed some multiples of p are also multiples of p^2, and they contribute twice to k; some are multiples of p^3, and so on. The following expression can be seen to hold:

$$k = \left\lfloor \frac{m}{p} \right\rfloor + \left\lfloor \frac{m}{p^2} \right\rfloor + \cdots + \left\lfloor \frac{m}{p^s} \right\rfloor$$

where s equals h or is any integer greater than it—in this case the additional items after the h-th do not contribute to the overall sum.

Consider then the binomial

$$\binom{2n}{n} = \frac{(2n)!}{n!n!} = \frac{2n}{n} \cdot \frac{2n-1}{n-1} \cdot \ldots \cdot \frac{n+1}{1} > 2 \cdot 2 \cdot \ldots \cdot 2 = 2^n.$$

Let h_1, k_1 and h_2, k_2 be defined as above for the integers n, $2n$ and their factorials, and let k be the greatest power of p that divides $\binom{2n}{n}$. Then

$$k = k_2 - 2k_1 = \left\lfloor \frac{2n}{p} \right\rfloor + \left\lfloor \frac{2n}{p^2} \right\rfloor + \cdots + \left\lfloor \frac{2n}{p^{h_2}} \right\rfloor - 2\left(\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \cdots + \left\lfloor \frac{n}{p^{h_2}} \right\rfloor \right)$$

since $h_2 \geq h_1$. For all α, $\lfloor 2\alpha \rfloor - 2\lfloor \alpha \rfloor \leq 1$; then $k \leq h_2$. This means that the greatest power of p that divides $\binom{2n}{n} > 2^n$ is not greater than the greatest power of p that divides $2n$. In particular all factors in the prime decomposition of $\binom{2n}{n}$ are less than or equal to $2n$, and no prime greater than $2n$ can appear in it. The following inequality holds:

$$2^n < \binom{2n}{n} \leq (2n)^{\pi(2n)}$$

---

have been added to the original definition (cf. [FR74], [MY78, pp. 194–5], [HU79] and [You85]) of $M_{k+1}(x, y, z)$ for S_+. They guarantee uniqueness of decomposition of n as a sum $s_1 \cdot s_1 + s_2 + s_3$. If this were not the case, for each n less than $\exp(2, k)^2$ several disjuncts of the form:

$$x = \overline{s_2 + s_3 + s_1 \cdot s_1} \wedge z \;=\; (s_1 \cdot s_1 + s_2 + s_3) \cdot_\tau y$$

would appear in (†), one for each possible decomposition of n. Distinct terms $s_1 \cdot_\tau (s_1 \cdot_\tau y) + s_2 \cdot_\tau y + s_3 \cdot_\tau y$ correspond to different decompositions, and two distinct terms of the form $n \cdot_\tau y$ cannot be proved equal, since commutativity and associativity do not hold. (In order to see a countermodel e.g. to $x + (x + x) = (x + x) + x$—and to associativity and commutativity as well—add two limit elements ω_0 and ω_1 to N and define successor and addition as before, but for $\omega_i + \omega_j$ which is ω_{1-i} if $i = j$, ω_i otherwise: then $\omega_0 + (\omega_0 + \omega_0) = \omega_0 \neq \omega_1 = (\omega_0 + \omega_0) + \omega_0$.) Then we could not get rid of the duplicated disjuncts by just appealing to the tautology $A \leftrightarrow B \;\rightarrow\; [(A \vee B) \leftrightarrow A]$.

which implies

$$\frac{n}{\lceil \log_2 2n \rceil} < \pi(2n) \leq \pi(2n+1)$$

Every integer m is either odd or even and matches one of the two right-hand sides.

$$\pi(m) > \left\{ \begin{array}{cc} \frac{m-1}{2\lceil \log_2(m-1) \rceil} & \text{se} \quad m = 2n+1 \\ \frac{m}{2\lceil \log_2 m \rceil} & \text{se} \quad m = 2n \end{array} \right\} \geq \frac{m-1}{2\lceil \log_2(m-1) \rceil}$$

We are taking m to be odd in order to consider the worst case, i.e. the smallest possible right-hand side. In any case the following inequality holds:

$$\frac{m-1}{2(1+\log_2(m-1))} \leq \frac{m-1}{2\lceil \log_2(m-1) \rceil} < \pi(m).$$

The left-hand term is $\omega(\sqrt{m})$; in particular for $m \geq 2^9$: [5]

$$\sqrt{m} < \frac{m-1}{2(1+\log_2(m-1))} < \pi(m).$$

Consider now the least common multiple m_k of all positive integers less than $\exp(2,k)$. We know that the factorization of this number contains the greatest power less than $\exp(2,k)$ of each prime. If k is big enough the lower bound derived above guarantees that $\exp^{1/2}(2,k) < \pi(\exp(2,k))$—and $\exp^{1/2}(2,k) \leq \pi(\exp(2,k)-1)$. Indeed $k \geq 4$ is enough, as can be immediately verified. Since $\exp^{1/2}(2,k)$ happens to be $\exp(2,k-1)$, we know that the least common multiple m_k is the product of at least $\exp(2,k-1)$ factors each greater than 2—indeed greater than the square root $\exp^{1/2}(2,k) = \exp(2,k-1)$ but this is useless. Hence $\exp(3,k-1) < m_k$. Note that the factorial $\exp(2,k)!$ would not be a substantial improvement over m_k, since it cannot be greater than $\exp(3,k+1)$.

We have at least proved that $\exp(3,k+3) < m_{k+4}$, since $k+4 \geq 4$. Consider the following formula $\Lambda_k(z)$, whose length increases linearly in k

$$\forall w \left[w = z \leftrightarrow \left(0 < w \leq z \wedge \forall n \left\{ [\neg n = 0 \wedge M_{k+4}(n,0,0)] \rightarrow \exists v \, M_{k+4}(n,v,w) \right\} \right) \right].$$

The intended meaning of $\Lambda_k(z)$ is "z is minimal in the set of all elements that are divisible by all integers less than $\exp(2,k+4)$." Using the equivalence proven for the minor times and using the existence of predecessor it can be proved that the (element denoted by the numeral $\overline{m}_{k+4}$ corresponding to the) least common multiple of $\{2,\ldots,\exp(2,k+4)-1\}$ is minimal in this

[5] The first author learned this simple proof of the lower bound on the number of primes during a class given by Paul Beame at the University of Washington, (cf. [BHC86]). This argument proving an $\omega(n/\log n)$ lower bound on the number of primes is much simpler than other proofs, for example the proof using the Chebychev inequality. In [BHC86] the multiplication of n n-bit integers is efficiently performed solving a related system of equations modulo a sufficiently wide set of prime numbers, in a way similar to the construction of approximations for multiplication in [FR74], but this approach greatly simplifies the overall proofs.

set, i.e. $\Lambda_k(\overline{m}_{k+4})$ can be proven. On the other hand if totality of order holds (with respect to numerals) then uniqueness is provable, that is $\forall z[\Lambda_k(z) \leftrightarrow z = \overline{m}_{k+4}]$.

Finally consider the formula $x \underset{k}{\otimes} y = z$ below, whose length is again linear in k:

$$\begin{aligned}
&\forall u_1 u_2\, (\Lambda_k(u_1) \wedge \Lambda_{k+2}(u_2)) \,\rightarrow\, \big[(x \leq u_1) \wedge (y \leq u_1) \wedge (z < u_2) \wedge \\
&\quad \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\qquad \big[\big(\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle x,y,z,t\rangle}} \big[M_{k+6}(n, q_i, k_i) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge M_{k+6}(r_1, r_2, t)\big) \,\rightarrow\, r_3 = r_4\big]\big]
\end{aligned}$$

This formula asserts that for each number n, not necessarily a prime, that is less than $\exp(2, k+6)$ the values of x, y and z are such that the product of the remainders of x and y modulo n is equivalent modulo n to the remainder of z; in other words in the ring of residues Z_n the product $[x]_n[y]_n$ equals $[z]_n$.

The input variables x and y are bounded by $\overline{m}_{k+4}$ and the ouptut z by $\overline{m}_{k+6}$. Note that for all h $(m_h)^2 < m_{h+2}$, and the formula expresses multiplication for all inputs x and y up to $\overline{m}_{k+4}$. Indeed each prime power in the factorization of $(m_h)^2$ is the square of the same prime power in the factorization of m_h: since the latter is an integer less than $\exp(2, h)$, the former is less than $\exp(2, h)^2 = \exp(2, h+1)$. On the other hand the same prime appears also in the factorization of m_{h+2} as a prime power that is greater than the square root of $\exp(2, h+2)$, namely $\exp(2, h+1)$: this implies that for each factor in the factorization of $(m_h)^2$ there is a greater factor in the factorization of m_{h+2}.

Using the properties of Q_+, $x \underset{k}{\otimes} y = z$ can be proven equivalent in Q_+ to:

$$\bigvee_{n,m \leq m_{k+4}} \big[x = \overline{n} \wedge y = \overline{m} \wedge z = \overline{n \cdot m}\big].$$

3.3 *The formula* $\Lambda_k(x)$

In this section it is shown that the formula $\Lambda_k(x)$ is equivalent in Q_+ to $x = \overline{m}_{k+4}$. This is shown by using non-divisibility and total order with respect to numerals. We first replace in $\Lambda_k(x)$ the minor-times subformulas with the equivalent quantifier free disjunctions and then prove that this simplified expression has the required property. For each k, the formula $\Lambda_k(x)$

$$\forall w\, \big[w = x \,\leftrightarrow\, \big(0 < w \leq x \wedge \forall n\, \{[\neg n = 0 \wedge M_{k+4}(n, 0, 0)] \rightarrow \exists v\, M_{k+4}(n, v, w)\}\big)\big]$$

is equivalent to

$$\begin{aligned}
&\forall w \Big[w = x \,\leftrightarrow\, \Big\{0 < w \leq x \wedge \forall n\, \Big[\Big(\neg n = 0 \wedge \bigvee_{s<\exp(2,k+4)} [n = \overline{s} \wedge 0 = s \cdot_\tau 0]\Big) \rightarrow \\
&\qquad \exists v \bigvee_{u<\exp(2,k+4)} [n = \overline{u} \wedge w = u \cdot_\tau v]\Big]\Big\}\Big],
\end{aligned}$$

since $Q_+ \vdash M_k(x, y, z) \leftrightarrow \bigvee_{s<\exp(2,k)} [x = \overline{s} \wedge z = s \cdot_\tau y]$. For all integers s, $Q_+ \vdash 0 = s \cdot_\tau 0$, and this atomic formula can be dropped in the conjunction above. Using the tautology $\big(\bigvee A_i\big) \rightarrow$

$B \leftrightarrow \bigwedge(A_i \to B)$ and afterwards the property $\vdash \forall x \bigwedge \varphi_i(x) \leftrightarrow \bigwedge \forall x \varphi_i(x)$, the formula above is equivalent to the following:

$$\forall w \Big[w = x \leftrightarrow \Big(0 < w \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \forall n \ \Big(n = \overline{s} \to \exists v \bigvee_{u<\exp(2,k+4)} [n = \overline{u} \wedge w = u \cdot_\tau v]\Big)\Big)\Big].$$

Now remove the universal quantifier $\forall n$ substituting $\overline{s}$ for n in the quantified subformula, in order to obtain the logically equivalent:

$$\forall w \Big[w = x \leftrightarrow \Big(0 < w \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v \bigvee_{u<\exp(2,k+4)} [\overline{s} = \overline{u} \wedge w = u \cdot_\tau v]\Big)\Big].$$

If $s \neq u$, $Q_+ \vdash \neg \overline{s} = \overline{u}$; hence all disjuncts corresponding to u different from s can be dropped. Furthermore $\overline{s} = \overline{s}$ is provable and can be deleted from the rightmost conjunction. This finally gives the following expression, equivalent to $\Lambda_k(x)$ in Q_+:

$$\forall w \Big[w = x \leftrightarrow \Big(0 < w \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w = s \cdot_\tau v]\Big)\Big].$$

Using the above expression for $\Lambda_k(x)$ we can now prove the first direction of the desired equivalence, namely $\forall x \ [x = \overline{m}_{k+4} \to \Lambda_k(x)]$, that is logically equivalent to $\Lambda_k(\overline{m}_{k+4})$:

$$\forall w \big[w = \overline{m}_{k+4} \leftrightarrow \big(0 < w \leq \overline{m}_{k+4} \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w = s \cdot_\tau v]\big)\big].$$

Again, in order to prove this equivalence we split it in its two directions, and first prove:

$$\forall w \big[w = \overline{m}_{k+4} \to \big(0 < w \leq \overline{m}_{k+4} \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w = s \cdot_\tau v]\big)\big].$$

This is logically equivalent to

$$\big(0 < \overline{m}_{k+4} \leq \overline{m}_{k+4} \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[\overline{m}_{k+4} = s \cdot_\tau v]\big)$$

which is a logical consequence of the following theorem of Q_+

$$\big(\neg \overline{m}_{k+4} = 0 \wedge \overline{m}_{k+4} + 0 = \overline{m}_{k+4} \wedge \bigwedge_{0<s<\exp(2,k+4)} [\overline{m}_{k+4} = s \cdot_\tau \overline{m_{k+4}/s}]\big).$$

Next we prove the other direction:

$$\forall w \big[\big(0 < w \leq \overline{m}_{k+4} \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w = s \cdot_\tau v]\big) \to w = \overline{m}_{k+4}\big]$$

which is equivalent in Q_+ to

$$\forall w \big[\big(\neg w = 0 \wedge \bigvee_{u \leq m_{k+4}} [w = \overline{u}] \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w = s \cdot_\tau v]\big) \to w = \overline{m}_{k+4}\big].$$

Distribute $\bigvee$ over $\wedge$ and afterwards apply again the tautology $\left(\bigvee A_i\right) \rightarrow B \quad \leftrightarrow \quad \bigwedge\left(A_i \rightarrow B\right)$ and the property $\vdash \forall x \bigwedge \varphi_i(x) \quad \leftrightarrow \quad \bigwedge \forall x \varphi_i(x)$: the following logically equivalent formula is derived

$$\bigwedge_{u \leq m_{k+4}} \forall w\left[\left(\neg w=0 \wedge w=\overline{u} \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w=s \cdot_\tau v]\right) \rightarrow w=\overline{m}_{k+4}\right].$$

This formula is logically equivalent to

$$\bigwedge_{u \leq m_{k+4}} \forall w\left[w=\overline{u} \rightarrow \left[\left(\neg w=0 \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w=s \cdot_\tau v]\right) \rightarrow w=\overline{m}_{k+4}\right]\right]$$

and to

$$\bigwedge_{u \leq m_{k+4}} \left[\left(\neg \overline{u}=0 \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[\overline{u}=s \cdot_\tau v]\right) \rightarrow \overline{u}=\overline{m}_{k+4}\right].$$

For each positive integer u, if s does not divide u then for any term $s \cdot_\tau y$, $Q_+ \vdash \neg \exists y\, \overline{u}=s \cdot_\tau y$. Then the formula above can be derived in Q_+ from the theorem

$$\bigwedge_{0<u<m_{k+4}} \neg\left[\bigwedge_{0<s<\exp(2,k+4)} \exists v[\overline{u}=s \cdot_\tau v]\right].$$

Finally we prove $\forall x\, [\Lambda_k(x) \rightarrow x=\overline{m}_{k+4}]$. This sentence is equivalent—using totality of order with respect to numerals—to:

$$\forall x\ \left[\left\{[(\overline{m}_{k+4} \leq x) \vee (x \leq \overline{m}_{k+4})] \wedge \Lambda_k(x)\right\} \rightarrow x=\overline{m}_{k+4}\right]$$

and to

$$\forall x\ \left[\left\{(\overline{m}_{k+4} \leq x) \wedge \Lambda_k(x)\right\} \rightarrow x=\overline{m}_{k+4}\right] \wedge \forall x\ \left[\left\{(x \leq \overline{m}_{k+4}) \wedge \Lambda_k(x)\right\} \rightarrow x=\overline{m}_{k+4}\right]$$

using some tautology and logically valid property. The second conjunct is easily proved, since $Q_+ \vdash \neg\Lambda_k(\overline{u})$ if $u < m_{k+4}$. The first conjunct

$$\forall x\ \left[\left\{(\overline{m}_{k+4} \leq x) \wedge \forall w\left[w=x \leftrightarrow \left(0<w \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w=s \cdot_\tau v]\right)\right]\right\} \rightarrow x=\overline{m}_{k+4}\right]$$

is logically equivalent to

$$\forall x \exists w\ \left[\left\{(\overline{m}_{k+4} \leq x) \wedge \left[w=x \leftrightarrow \left(0<w \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[w=s \cdot_\tau v]\right)\right]\right\} \rightarrow x=\overline{m}_{k+4}\right]$$

since the variable w does not appear in the implication consequent $x=\overline{m}_{k+4}$. Then the first conjunct follows from the Q_+ theorem:

$$\forall x\ \left[\left\{(\overline{m}_{k+4} \leq x) \wedge \left[\overline{m}_{k+4}=x \leftrightarrow \left(0<\overline{m}_{k+4} \leq x \wedge \bigwedge_{0<s<\exp(2,k+4)} \exists v[\overline{m}_{k+4}=s \cdot_\tau v]\right)\right]\right\} \rightarrow x=\overline{m}_{k+4}\right].$$

This concludes the proof of $Q_+ \vdash \forall x\, [\Lambda_k(x) \leftrightarrow x=\overline{m}_{k+4}]$.

3.4 *The formula* $x \underset{k}{\otimes} y = z$

We prove in this section that the formula $x \underset{k}{\otimes} y = z$ is equivalent to the disjunction of all triples of arguments less than m_{k+4} and the corresponding product:

$$\bigvee_{n,m \leq m_{k+4}} \left[x = \overline{n} \wedge y = \overline{m} \wedge z = \overline{n \cdot m}\right].$$

Consider the definition of $x \underset{k}{\otimes} y = z$:

$$\begin{aligned}
&\forall u_1 u_2\, (\Lambda_k(u_1) \wedge \Lambda_{k+2}(u_2)) \rightarrow \big[(x \leq u_1) \wedge (y \leq u_1) \wedge (z < u_2) \wedge \\
&\quad \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\qquad \big[\big(\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle x,y,z,t\rangle}} \big[M_{k+6}(n, q_i, k_i) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge M_{k+6}(r_1, r_2, t)\big) \rightarrow r_3 = r_4\big]\big].
\end{aligned}$$

The above formula is logically equivalent to

$$\begin{aligned}
&(x \leq \overline{m}_{k+4}) \wedge (y \leq \overline{m}_{k+4}) \wedge (z < \overline{m}_{k+6}) \wedge \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\quad \big[\big(\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle x,y,z,t\rangle}} \big[M_{k+6}(n, q_i, k_i) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge M_{k+6}(r_1, r_2, t)\big) \rightarrow r_3 = r_4\big]
\end{aligned}$$

which is equivalent in Q_+ to

$$\begin{aligned}
&\bigvee_{\substack{r,m \leq m_{k+4} \\ s < m_{k+6}}} \big[(x = \overline{r}) \wedge (y = \overline{m}) \wedge (z = \overline{s}) \wedge \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\quad \big[\big(\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle x,y,z,t\rangle}} \big[M_{k+6}(n, q_i, k_i) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge M_{k+6}(r_1, r_2, t)\big) \rightarrow r_3 = r_4\big]\big].
\end{aligned}$$

Using the tautology $[A \wedge B] \leftrightarrow [A \wedge (A \rightarrow B)]$ and substitutivity three times, this is shown equivalent to

$$\begin{aligned}
&\bigvee_{\substack{r,m \leq m_{k+4} \\ s < m_{k+6}}} \big[(x = \overline{r}) \wedge (y = \overline{m}) \wedge (z = \overline{s}) \wedge \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\quad \big[\big(\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle \overline{r},\overline{m},\overline{s},t\rangle}} \big[\big(M_{k+6}(n, q_i, k_i)\big) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge M_{k+6}(r_1, r_2, t)\big) \rightarrow r_3 = r_4\big]\big]
\end{aligned}$$

which is equivalent in Q_+ to

$$\begin{aligned}
&\bigvee_{\substack{r,m \leq m_{k+4} \\ s < m_{k+6}}} \big[(x = \overline{r}) \wedge (y = \overline{m}) \wedge (z = \overline{s}) \wedge \\
&\quad \forall n t k_1 k_2 k_3 k_4 r_1 r_2 r_3 r_4 q_1 q_2 q_3 q_4 \\
&\qquad \big[\big\{\bigwedge_{\substack{i=1,2,3,4 \\ \alpha=\langle \overline{r},\overline{m},\overline{s},t\rangle}} \big[\big(\bigvee_{n_i < \exp(2,k+6)} [n = \overline{n_i} \wedge k_i = n_i \cdot_{\tau} q_i]\big) \wedge k_i + r_i = \alpha_i \wedge r_i < n\big] \wedge \\
&\qquad\quad \bigvee_{c < \exp(2,k+6)} [r_1 = \overline{c} \wedge t = c \cdot_{\tau} r_2]\ \big\} \quad \rightarrow \quad r_3 = r_4\big]\big]
\end{aligned}$$

Consider the last two formulas, and in particular each disjunct—corresponding to a particular choice of r, m and s—of the main disjunction. Each individual variable in the last of the four conjuncts can be shown to equal the numeral that it should mean, e.g. $q_1 = \overline{r/n}, r_4 = \overline{[r]_n \cdot [m]_n}$. This universal formula corresponds to a huge system of $\exp(2, k+6) - 1$ equations, saying that $[s]_n = [r]_n [m]_n$ for all integers n, $0 < n \leq \exp(2, k+6) - 1$. Since for each r, m the difference between each two solutions of the $\exp(2, k+6) - 1$ conditions specified is a multiple of m_{k+6}, then for each r, m there exists exactly one integer s less than m_{k+6} that simultaneously satisfies all the conditions specified, and furthermore, the integer $r \cdot m$ does this. So for all r, m and s, if $s \neq r \cdot m$ the negation of the corresponding main disjunct can be proved in Q_+—and the disjunct can be dropped. Otherwise the universal conjunct is provable and can be removed from the formula. At the end only the disjuncts corresponding to the correct s remain, and they have the form $(x = \overline{r}) \wedge (y = \overline{m}) \wedge (z = \overline{s})$, that is:

$$Q_+ \vdash \forall xyz \; x \underset{k}{\otimes} y = z \leftrightarrow \bigvee_{n,m \leq m_{k+4}} \left[x = \overline{n} \wedge y = \overline{m} \wedge z = \overline{n \cdot m} \right].$$

4. DOUBLE-EXPONENTIAL INSEPARABILITY

In this section we give our first proof of an inseparability result, for the stronger theory ADDAX, with respect to a certain programming system and a complexity measure that are more natural in this arithmetical framework than Turing machines and time complexity. We then show how this inseparability implies for ADDAX the one referring to non-deterministic Turing machines.

The technique applied in this section is derived from the work in [You85], where this double exponential inseparability is listed as an open problem. Some new technical questions had to be solved in order to raise the lower bound to a double exponential.

A *min-program* H (cf. [MY78] and [You85]) is a syntactic expression to which a number n of arguments and a partial function f_H from N^n to N are associated. The function f_H is the meaning of H—the function computed by the program.

For every integer n and $j \leq n$, P^n_j is a primitive program of n arguments; the meaning of P^n_j is a function from N^n to N that associates to each n-tuple its j-th component. Primitive *min-programs* of two arguments are $+, \times$ and $c_=$. The characteristic function of equality is associated to the program $c_=$, addition and multiplication are associated to $+$ and $\times$.

If H is a program of m arguments and G_1 through G_m are all programs of n arguments, then the composition $H(G_1, \ldots, G_m)$ is a program of n arguments. The associated function from N^n to N is the obvious composition of functions.

Finally if H is a program of $n+1$ arguments then min H is a program of n arguments, and the value that the partial function associates to $a_1, \ldots, a_n$ is the least integer m such that $f_H(a_1, \ldots, a_n, i)$ is defined and positive for all $i \leq m$ and $f_H(a_1, \ldots, a_n, m) = 0$. If no such m exists then $f_{\min H}$ is undefined on the arguments $a_1, \ldots, a_n$.

A coding of min-programs can be defined in such a way that to each program H an integer h is uniquely assigned. The binary representation of the coding h could be simply the binary representation of the string of symbols of H. But we prefer a slight variation of this coding. Let us define our coding to be the previous one, but padded by the prefix $0^{||H||^2}1$, where $||H||$ is the length of the string of symbols of H. In this way an expression whose length would be almost quadratic in $||H||$ will be linear in the length $|h|$ of the coding of H.

For each program H of n arguments, and each n-tuple of integers $\langle v_1, \ldots, v_n \rangle$, if H converges on $v_1, \ldots, v_n$ let $\Phi(H, v_1, \ldots, v_n)$ be the greatest number that is the argument of a multiplication during the computation of H on inputs $v_1, \ldots, v_n$, which is defined in the obvious way.[6]

Let Δ be the set of programs of one argument which converge on each input a and never multiplies any number greater than $\exp(3, |a|)$—where $|a|$ is the length of the binary representation of a. The function computed by each $H \in \Delta$ is then a total function of one argument, and for each a $\Phi(H, a) \leq \exp(3, |a|)$.

Let $\Delta_0 \subseteq \Delta$ be the set of programs H of one argument that on input h, their own coding, converge to $f_H(h) = 0$ without multipling numbers greater than $\exp(3, |h|)$, and let Δ_1 be the set of those that get value 1. By the Russell paradigm (cf. [You85, pp. 503–4]) no element in Δ can separate Δ_0 from Δ_1.

We will show an efficient reduction of the problem of separating Δ_0 from Δ_1 to the problem of separating ADDAX from the unsatisfiable sentences. In this way, if an efficiently recognizable set separating ADDAX from the unsatisfiable exists, than Δ_0 can be separated from Δ_1 by a program in Δ, a contradiction. The core of the inseparability result is then the following lemma, in which the reduction f_R is built.

<u>Lemma</u>

Let $\mathrm{Uns}\,\mathcal{L}_+$ *be the set of unsatisfiable sentences of* $\mathcal{L}_+$*. There exists an integer* c_1 $(c_1 > 1)$ *and a total function* $f_R : N \longrightarrow N$ *with the following properties:*

- *For each* $w \in N$ $f_R(w)$ *is the coding of a sentence* $\sigma(w)$ *such that:*

$$|\sigma(w)| \leq c_1 \cdot |w|$$

- *For each* $q \in N$ *that is not the coding of a program:*

$$\sigma(q) \in \mathrm{Uns}\,\mathcal{L}_+.$$

- *For each* $h \in N$ *that is the coding of some program* H*:*

$$H \in \Delta_0 \Rightarrow \sigma(h) \in \mathrm{ADDAX}$$
$$H \in \Delta_1 \Rightarrow \sigma(h) \in \mathrm{Uns}\,\mathcal{L}_+.$$

- *There exists a program* R *of one argument that has meaning* f_R *and is such that for every* $w \in N$*:*

$$\Phi(R, w) < \exp(3, |w|).$$

<u>*proof*</u> We know how to give a definition of representability of a function by open formulas of a theory with equality. For example we know that in the Peano arithmetic S all recursive functions are representable. For every recursive function f of n arguments there exists a formula in the language of arithmetic that has exactly $n+1$ free variables such that every instance

[6] This differs from the complexity measure chosen in [You85], where only the smaller argument of each multiplication was relevant.

corresponding to a $n+1$-tuple in f is a theorem of S, and it is provable in S that only one value corresponds to each n-tuple of arguments.

There are not as many functions representable in this sense in the quite small theory ADDAX. But many functions are representable, in a weaker sense, if formulas have to represent only initial segments and their instances are to be provable only on bounded arguments (cf. [FY92]). We say that a formula φ_k $\exp(3,k)$-represents a function when it is provable at least on arguments less than $\exp(3,k)$.

The reduction f_R will be built on the base of a function that effectively associates to the coding h of each min-program H a formula that represents the function computed by H if on all inputs H does not require too large multiplication. We have already proved the key fact, that $x \underset{k}{\otimes} y = z$ $\exp(3,k)$-represents multiplication in Q_+, and therefore in ADDAX as well. We can then recursively associate to each program H of n arguments a formula $\varphi_H(x_1,\ldots,x_n,z)$:

- $\varphi_{P^n_j}(x_1,\ldots,x_n,z)$ is $z = x_j$;
- $\varphi_+(x_1,x_2,z)$ is $z = x_1 + x_2$;
- $\varphi_\times(x_1,x_2,z)$ is $x_1 \underset{k}{\otimes} x_2 = z$;
- $\varphi_{c_=}(x_1,\ldots,x_2,z)$ is $(x_1 = x_2 \wedge z = 0) \vee (\neg x_1 = x_2 \wedge z = S0)$;
- $\varphi_{H(G_1,\ldots,G_m)}(x_1,\ldots,x_n,z)$ is

$$\begin{aligned}\exists y_1 \cdots \exists y_m \quad [\bigwedge_{i\le m} \forall z\,[y_i = z \;\; \rightarrow \;\varphi_{G_i}(x_1,\ldots,x_n,z)] \;\; \wedge \\ \forall x_1 \cdots \forall x_m \;\; \bigwedge_{i\le m} [y_i = x_i] \;\; \rightarrow \;\varphi_H(x_1,\ldots,x_m,z)]\end{aligned}$$

- $\varphi_{\min H}(x_1,\ldots,x_n,z)$ is

$$\forall x_{n+1}\,\{x_{n+1} \le z \rightarrow \exists z'\;[(z' = 0 \leftrightarrow z = x_{n+1}) \wedge \forall z(z = z' \rightarrow \varphi_H(x_1,\ldots,x_n,x_{n+1},z))]\}$$

The formula φ_H in general depends on the parameter k chosen for $\varphi_\times$. Let us write φ^k_H to stress this fact.

The proof that φ^k_H represents in ADDAX the meaning of H for all arguments $\underline{x}$ with $\Phi(H,\underline{x}) < \exp(3,k)$, proceeds by induction on the structure of H. A meaningful application of the tricotomy axiom is needed in order to prove functionality in the last clause of the step.[7]

The length of φ^k_H is $\omega(\|H\|\cdot k)$, and roughly $O(\|H\|\cdot[k+\log\|H\|])$, for there can be $O(\|H\|)$ occurrences of $\times$ in H and $O(\|H\|)$ new variables. Hence the length of the formula representing

[7] To avoid this, one could define $\varphi_{\min H}(x_1,\ldots,x_n,z)$ as

$$\begin{aligned}z \underset{k}{\otimes} 0 = 0 \wedge \forall x_{n+1}\,\{x_{n+1} \le z \rightarrow \exists z'\;[z' \le S0\, \wedge \\ (z' = 0 \leftrightarrow z = x_{n+1}) \wedge \forall z(z = z' \rightarrow \varphi_H(x_1,\ldots,x_n,x_{n+1},z))]\}\end{aligned}$$

in such a way that all quantifiers are bounded, and can be eliminated. Instead of considering the complexity measure Φ of the arguments of multiplication, we would need to take into account also the output of minimalization. In any case the class of programs would be powerful enough and the whole construction in this section could be done for Q_+, yielding an inseparability result for this weaker theory—which we will prove in a different way in the next section.

the meaning of H grows too quickly, because at the end of the construction f_R shall associate to each coding h a formula built on the base of $\varphi_H^{|h|}$, which would be $\omega(|h|)$ long.

Using the fact that ADDAX is a theory with equality—in which it is provable that two distinct elements exist, e.g. 0 and $S0$—we can modify the definition of φ_H^k in such a way that only two occurrences of the formula $x \underset{k}{\otimes} y = z$ do appear in it. Let $Q_1 u_1 \ \cdots \ Q_m u_m \psi_H^k$ be obtained from the prenex normal form of φ_H^k by replacing each occurrence of $\varphi_\times^k(u_1^i, u_2^i, u_3^i)$ in it with a shorter formula, namely the i-th occurrence with $v_i = 0$. In every model of ADDAX, the formula:

$$\exists v_1 \ldots v_n \quad \left[\bigwedge_{i \leq n} v_i = 0 \leftrightarrow u_1^i \underset{k}{\otimes} u_2^i = u_3^i \right] \wedge \psi_H^k$$

is equivalent to the matrix of the prenex formula from which the quantifier free formula ψ_H^k was derived. The following formula χ_H^k is then equivalent in ADDAX to φ_H^k (cf. [CH90, pp. 15–16] or [FR79, pp. 155–157]) and represents in ADDAX the meaning of H if $\Phi(H, \underline{x}) < \exp(3, k)$ for all possible arguments $\underline{x}$:

$$\begin{aligned} & Q_1 u_1 \ \cdots \ Q_m u_m \ \exists v_1 \ldots v_n \\ & \quad \left[\forall v\, xyz \left(\bigvee_{i \leq n} v = v_i \wedge x = u_1^i \wedge y = u_2^i \wedge z = u_3^i \right) \rightarrow (v = 0 \leftrightarrow x \underset{k}{\otimes} y = z) \right] \wedge \psi_H^k \end{aligned}$$

Two distinct expressions can be recognized, one depending only on H and the other one—namely $x \underset{k}{\otimes} y = z$—only on k. We have chosen to pad the coding h of H in such a way that the first expression grows as $O(|h|)$. The overall length of χ_H^k is then $O(|h| + k)$—indeed $O(\|H\| \log \|H\| + k)$.

For each integer a whose binary representation has length k or less, the formula $W_a^k(x)$:

$$\begin{aligned} & \exists x_0 \,(x_0 = \overline{a[k]} \wedge \exists x_1 \,(x_1 = 2 \cdot x_0 + \overline{a[k-1]} \wedge \exists x_0 \,(x_0 = 2 \cdot x_1 + \overline{a[k-2]} \wedge \cdots \\ & \qquad \wedge \exists x_i \,(x_i = 2 \cdot x_{1-i} + \overline{a[1]} \wedge x = 2 \cdot x_i + \overline{a[0]}) \cdots))), \end{aligned}$$

where $a[n]$ is the n-th bit in the binary representation of a, has length linear in k and is equivalent in ADDAX, as well as in Q_+, to $x = \overline{a}$.

Next consider for each coding h of a program H the $O(|h|)$ sentence $\sigma(h)$:

$$\delta_+ \wedge \forall x_1 \ W_h^{|h|}(x_1) \rightarrow [\exists z \ z = 0 \wedge \chi_H^{|h|}(x_1, z)]$$

where the sentence δ_+ is the conjunction of the finite axioms of ADDAX. The formula $\sigma(h)$ is equivalent in ADDAX to $\chi_H^{|h|}(\overline{h}, 0)$, which is provable in the theory if $f_H(h) = 0$ and $\Phi(H, h) < \exp(3, |h|)$. On the other hand, if $f_H(h) = 1$ and $\Phi(H, h) < \exp(3, |h|)$ then ADDAX $\vdash \chi_H^{|h|}(\overline{h}, S0)$ and then, using functionality of representation, ADDAX $\vdash \neg\chi_H^{|h|}(\overline{h}, 0)$. This implies that if $f_H(h) = 1$ then $\sigma(h)$ belongs to Uns $\mathcal{L}_+$.

To conclude the proof note that the formula $\chi_H^{|h|}(x_1, z)$ has a quite simple structure. From a computational standpoint, given an integer h it is quite simple to check that it is a correct coding of a program of one argument, and then compute $|h|$ and $\varphi_H^{|h|}$, its prenex form and the formula ψ and finally χ. It is even simpler to compute the coding of the formula $W_h^{|h|}(x_1)$.

On the other hand—as will be shown in the next section—the set of min-programs whose complexity is bounded by a triple exponential is quite powerful. All common string manipulation functions are computable by a program of this class, and the class of functions computed by programs in Δ contains all language in NTIME$(\exp(2, cn))$ for some $c > 0$. Then a program R matching the theorem statement can be built that computes on input h the formula $\sigma(h)$—which will be a fixed contradiction in case h is not the coding of a program of one argument. ∎

The existence of a translation f_{R} that is easily computable implies that ADDAX is not separable from Uns $\mathcal{L}_+$ by any min-program whose multiplicative complexity is smaller than $\exp(3, c \cdot |w|)$, for some $c > 0$.

Theorem *(Inseparability for Programs)*

There exists a constant $c_2 > 0$ such that, for each set $T \subseteq \mathcal{L}_+$, if T separates ADDAX *from* Uns $\mathcal{L}_+$ *then there is no program G such that:*

$$\forall w \quad \Phi(G, w) < \exp(3, c_2 \cdot |w|) \;\; \& \;\; f_G(w) = \begin{cases} 0 & \textit{if } w \in T \\ 1 & \textit{otherwise} \end{cases}$$

proof Let c_2 be $1/c_1$ and consider first the case in which $T \supseteq$ ADDAX and $T \cap \text{Uns}\,\mathcal{L}_+ = \emptyset$. The proof is by contradiction. Suppose that a min-program G exists such that

$$\forall w \quad \Phi(G, w) < \exp(3, c_2 \cdot |w|) \;\; \& \;\; f_G(w) = \begin{cases} 0 & \text{if } w \in T \\ 1 & \text{otherwise} \end{cases}$$

From the preceding lemma we know that there exists a program R such that

$$\forall w \quad \Phi(R, w) < \exp(3, |w|) \wedge f_R(w) = \sigma(w).$$

Consider the program $G(R)$:

$$\forall w \quad \Phi(G(R), w) < \exp(3, |w| \cdot \max(1, c_1 c_2)) = \exp(3, |w|).$$

For each $h \in N$ that is the coding of some program H:

$$H \in \Delta_0 \Rightarrow \sigma(h) \in \text{ADDAX} \Rightarrow f_G(\sigma(h)) = 0$$
$$H \in \Delta_1 \Rightarrow \sigma(h) \in \text{Uns}\,\mathcal{L}_+ \Rightarrow f_G(\sigma(h)) = 1.$$

For each $q \in N$ that is not the coding of a program:

$$\sigma(q) \in \text{Uns}\,\mathcal{L}_+ \Rightarrow f_G(\sigma(q)) = 1.$$

By the Russell paradigm, this is a contradiction.

The case $T \supseteq \text{Uns}\,\mathcal{L}_+$ and $T \cap \text{ADDAX} = \emptyset$ reduces to the previous one considering the complement $\mathcal{L}_+ \setminus T$. ∎

In order to show that this inseparability result for programs implies inseparability for Turing machines, we have to define many simple programs which compute the most common string manipulation functions, and we have to show that Turing machines may be simulated efficiently by min-programs. In the next section we show in particular that there exists a constant c_3 ($c_3 > 1$) such that for each language $L(M) \subseteq \Sigma^*$ accepted by a non-deterministic Turing machine M working in time $\exp(2, c \cdot n)$ for some $c > 0$, there exists a program H such that for all $w \in \Sigma^*$:

$$\Phi(H, w) < \exp(3, c_3 c \cdot |w|) \wedge f_H(w) = \begin{cases} 0 & \text{if } w \in L(M) \\ 1 & \text{otherwise} \end{cases}$$

If the existence of a Turing machine implies the existence of an equivalent program, then no Turing machine can separate ADDAX from the logically false sentences.

Theorem *(Inseparability for Non-Deterministic Turing Machines)*

There exists a constant $c_4 > 0$ such that for all $c \leq c_4$, for each set $T \subseteq \mathcal{L}_+$, if T separates ADDAX *from* Uns $\mathcal{L}_+$ *then there is no non-deterministic Turing machine M working in time* $\exp(2, c \cdot n)$ *such that $T = L(M)$.*

proof Let c_4 be a constant such that $c_3 \cdot c_4 \leq c_2$ where c_3 is the factor in the preceding statement and c_2 is the constant in the inseparability result for programs. If T were as in the hypothesis, then there would be a program G such that for all $w \in \Sigma^*$:

$$\Phi(G, w) < \exp(3, c_3 c_4 \cdot |w|) \wedge f_G(w) = \begin{cases} 0 & \text{if } w \in L(M) = T \\ 1 & \text{otherwise,} \end{cases}$$

a contradiction by the inseparability result for programs, since $\exp(3, c_3 c_4 \cdot |w|) \leq \exp(3, c_2 \cdot |w|)$ ∎

4.1 *The computational power of min-programs and a Turing machine simulation*

In this section it will be shown that string manipulations and exponential functions are computable by min-programs without multiplying very large numbers, and that a non-deterministic Turing machine can be efficiently simulated by a min-program.

After developing programs for some useful numeric functions and predicates, we recall the basic properties of the coding of strings introduced in [MY78]. We consider integers to be strings, as happens with binary notation, and show programs that compute several string manipulations without performing costly operations (i.e. requiring multiplication of big numbers) on any string longer than a linear factor of the inputs. In this way we can build a program for the exponentiation function through an application of the standard technique to remove a primitive recursion from a function definition. As soon as we have exponentiation at a cheap cost, we can simulate a non-deterministic Turing machine with a program of bounded multiplicative complexity.

Let us start by verifying that several numeric functions and predicates, including primality, can be efficiently computed by min-programs. It is easy to construct, for all integers n and m, a program of m arguments that computes the constant function n. We will use simply the number n, e.g. 0, to denote the corresponding program when necessary.

The logical operations $\mathtt{AND}, \mathtt{OR}, \mathtt{NOT}$ and $\mathtt{IMPLIES}$ are computed by programs that do not perform multiplication at all. Comparison between numbers and the inverse $\dot{-}$ of addition are computed at no cost as well, as shown by the three following programs:

$$\mathtt{GREATER\text{-}EQ} := \\ \mathtt{c}_=(\mathtt{P}^2_2, \min \mathtt{OR}(\mathtt{c}_=(\mathtt{P}^3_1, \mathtt{P}^3_3), \mathtt{c}_=(\mathtt{P}^3_2, \mathtt{P}^3_3)))$$

$$\mathtt{GREATER} := \\ \mathtt{AND}(\mathtt{GREATER\text{-}EQ}, \mathtt{NOT}(\mathtt{c}_=))$$

$$\mathtt{MINUS} := \\ \min \mathtt{GREATER\text{-}EQ}(+ (\mathtt{P}^3_3, \mathtt{P}^3_2), \mathtt{P}^3_1)$$

The functions $f_{\mathtt{GREATER\text{-}EQ}}, f_{\mathtt{GREATER}}$ and $f_{\mathtt{MIN}}$ are total and for all arguments n, m $\Phi(\mathtt{GREATER\text{-}EQ}, n, m) = \Phi(\mathtt{GREATER}, n, m) = \Phi(\mathtt{MIN}, n, m) = 0$.

The usual integer operations of division and remainder can be efficiently performed if multiplication is. This implies that primality can be checked efficiently:

Proposition

There exist min-programs which compute the following functions, within the specified multiplicative complexity bounds:

- $\mathtt{MOD}$, *of two arguments; for all integers* m, n, $\Phi(\mathtt{MOD}, m, n) \leq \max(n, m)$ *and* $f_{\mathtt{MOD}}(m, n) = m \bmod n$
- $\mathtt{DIV}$, *of two arguments; for all integers* m, n, $\Phi(\mathtt{DIV}, m, n) \leq \max(n, m)$ *and* $f_{\mathtt{DIV}}(m, n) = m \div n$
- $\mathtt{DIVIDES}$, *of two arguments; for all integers* m, n, $\Phi(\mathtt{DIVIDES}, m, n) \leq \max(n, m)$ *and* $f_{\mathtt{DIVIDES}}$ *is the characteristic function of the divisibility predicate*
- $\mathtt{IS\text{-}PRIME}$, *of one argument; for all integers* n, $\Phi(\mathtt{IS\text{-}PRIME}, n) \leq n$ *and* $f_{\mathtt{IS\text{-}PRIME}}$ *is the characteristic function of the 'n is prime' predicate*
- $\mathtt{IS\text{-}POWER}$, *of one argument; for all integers* p, n, $\Phi(\mathtt{IS\text{-}POWER}, p, n) \leq \max(p, n)$ *and* $f_{\mathtt{IS\text{-}POWER}}$ *is the characteristic function of the predicate 'p is prime and n is a power of p'*
- $\mathtt{IS\text{-}POWER}_m$, *of one argument; for all integers* p, n, $\Phi(\mathtt{IS\text{-}POWER}_m, p, n) \leq \max(p, n + c_m)$ *and* $f_{\mathtt{IS\text{-}POWER}_m}$ *is the characteristic function of the predicate 'p is prime and n is a power of* p^m*'*

proof We simply show the programs and analize their complexity:

$$\mathtt{DIV} := \\ \min \mathtt{GREATER\text{-}EQ}(+ (\times (\mathtt{P}^3_3, \mathtt{P}^3_2), \mathtt{P}^3_2), \mathtt{P}^3_1)$$

If the second argument is positive then the function $f_{\mathtt{DIV}}$ is defined and $\Phi(\mathtt{DIV}, m, n) \leq \max(m, n)$.

$$\mathtt{MOD} := \\ \min \mathtt{c}_=(\mathtt{P}^3_1, +(\mathtt{P}^3_3, \times(\mathtt{P}^3_2, \mathtt{DIV}(\mathtt{P}^3_1, \mathtt{P}^3_2))))$$

If the second argument is positive then the function $f_{\texttt{MOD}}$ is defined and $\Phi(\texttt{MOD}, m, n) \leq \max(m, n)$.

$$\begin{array}{l} \texttt{DIVIDES} := \\ \quad \texttt{c}_=(0, \texttt{MOD}(\texttt{P}_2^2, \texttt{P}_1^2)) \end{array}$$

If the first argument is positive then the function $f_{\texttt{DIVIDES}}$ is defined and $\Phi(\texttt{MOD}, n, m) \leq \max(m, n)$.

$$\begin{array}{l} \texttt{IS-PRIME} := \\ \quad \texttt{c}_=(\texttt{P}_1^1, \\ \qquad +(2, \min \texttt{OR}(\texttt{GREATER-EQ}(+(2, \texttt{P}_2^2), \texttt{P}_1^2), \\ \qquad\quad \texttt{DIVIDES}(+(2, \texttt{P}_2^2), \texttt{P}_1^2)))) \end{array}$$

The function $f_{\texttt{IS-PRIME}}$ is total and $\Phi(\texttt{IS-PRIME}, n) \leq n$.

$$\begin{array}{l} \texttt{IS-POWER} := \\ \quad \texttt{AND}(\texttt{IS-PRIME}(\texttt{P}_1^2), \texttt{c}_=(\texttt{P}_2^2, \\ \qquad +(2, \min \texttt{OR}(\texttt{c}_=(\texttt{P}_2^3, +(2, \texttt{P}_3^3)), \\ \qquad\quad \texttt{NOT}(\texttt{IMPLIES}(\\ \qquad\qquad \texttt{DIVIDES}(+(2, \texttt{P}_3^3), \texttt{P}_2^3), \\ \qquad\qquad \texttt{DIVIDES}(\texttt{P}_1^3, +(2, \texttt{P}_3^3)))))))) \end{array}$$

The function $f_{\texttt{IS-POWER}}$ is total and $\Phi(\texttt{IS-POWER}, n, m) \leq \max(n, m)$.

$$\begin{array}{l} \texttt{IS-POWER}_m := \\ \quad \texttt{AND}(\texttt{IS-POWER}, \texttt{c}_=(\texttt{P}_2^2, \\ \qquad \times(\times(\cdots \times(\texttt{P}_1^1, \texttt{P}_1^1)\cdots), \texttt{P}_1^1) \qquad \texttt{\%}\ m \quad \text{times} \\ \qquad\; (\min \texttt{GREATER-EQ}(\times(\times(\cdots \times(\texttt{P}_3^3, \texttt{P}_3^3)\cdots), \texttt{P}_3^3), \texttt{P}_2^3)))) \end{array}$$

If the first argument is positive, then the function $f_{\texttt{IS-POWER}_m}$ is defined. To bound $\Phi(\texttt{IS-POWER}_m, p, n)$ consider that the greatest integer multiplied during the computation of IS-POWER on input p, n, p a prime less than n, is in any case a^{m-1}, where a is such that $(a-1)^m < n \leq a^m$. Consider an a_0 such that $b^{m-1} < (b-1)^m$ for all $b \geq a_0$, and let c_m be ${a_0}^{m-1}$. Then $a^{m-1} \leq n + c_m$ and for all p, n $\Phi(\texttt{IS-POWER}_m, p, n) \leq \max(p, n + c_m)$. ∎

We will introduce and use a coding of strings different from the usual binary representation, or n-ary representation.

Definition *(Coding)*

Given a finite alphabet A, arbitrarily assign to each character in A a number between 1 and the set cardinality a. Denote by c_i the character corresponding to integer i. Then the function $C\colon A^* \longrightarrow N$ defined by

$$C(\alpha) = \begin{cases} 0 & \text{if } \alpha = \epsilon \\ \sum_{1 \leq j \leq n} i_j a^{n-j} & \text{if } \alpha = c_{i_1} \ldots c_{i_n} \end{cases}$$

is a coding of the strings in A^*.

The coding of α is then its position in the lexicographical order. This coding appears in [MY78] and will be useful for its property that any sequence of characters denotes an integer, and every integer is denoted by exactly one string of characters.

You should note that in the previous section all proofs refer to binary coding, for it is more familiar to everybody. Two characteristics of binary coding are used in those proofs, namely that an integer n, which is the coding of its binary representation, codes a string whose length is logarithmic in n, and that string manipulations are easily performed on the string codings. In this section, during the construction of min-programs computing string manipulations, it will be clear that our new coding have the same properties as the one associated with binary representations. In particular we will see that for each alphabet A of cardinality a the number a^k codes a string of length k. Then the result in the previous section should be read as if this coding were used instead of the one associated with binary representation. In any case it will become clear that there are efficient programs which translates each coding into the other.

Since we will encode different objects, such as function arguments and values, in single strings, it will be convenient to separate them with characters that do not appear in the original language. Thus we use two different alphabets, and it is useful to choose them in such a way that their cardinalities are successive powers of a prime number. We can add to a small alphabet some new distinct characters—e.g. '$\alpha, \beta, \ldots$'—so that the first alphabet that we choose has a prime power p^m of symbols, for example $a = 2^7$. As second alphabet we take a superset of the first that contains ap symbols, $ap = 2^8$ in the previous example, including for our convenience some character that can be used as separator, e.g. '#' and '\$'. The smaller alphabet will be called the '*ground alphabet*' and the larger one the '*work alphabet*'.

If the cardinality of the alphabet is a prime power, many string-manipulation functions are easily computed in that alphabet. It is also easy to switch between the coding of one string in two alphabets whose cardinalities are successive powers of a prime number. This is stated in the following:

<u>Proposition</u>

For each m there exists c_m (with $c_1 = 0$ and $c_j < c_{j+1}$) such that if the cardinality a of the alphabet is a power p^m of a prime number then there exist min-programs which compute the following functions, within the specified multiplicative complexity bounds:

- `EQ-LENGTH`*, of two arguments; for all integers v, w, $\Phi(\texttt{EQ-LENGTH}, v, w) \leq \min(v, w) + c_m$ and $f_{\texttt{EQ-LENGTH}}$ is the characteristic function of the predicate 'the strings with coding v and w have the same length.'*
- `PWR-FLOOR`*, of one argument; for all integers v, $\Phi(\texttt{PWR-FLOOR}, v) \leq (a-1) \cdot v + 1 + c_m \leq a^{|v|+2} + c_m$ and $f_{\texttt{PWR-FLOOR}}(v)$ is $a^{|v|}$, a to the length of v.*
- `APPEND`*, of two arguments; for all integers v, w, $\Phi(\texttt{APPEND}, v, w) \leq \max(v, (a-1) \cdot w + 1 + c_m) \leq a^{\max(|v|,|w|)+2} + c_m$ and $f_{\texttt{APPEND}}(v, w)$ is the coding of the concatenation of the strings with coding v and w.*
- `STRBEG`*, of two arguments; for all integers v, w, $\Phi(\texttt{STRBEG}, v, w) \leq \max(v, (a-1) \cdot w + 1 + c_m) \leq a^{\max(|v|,|w|)+2} + c_m$ and $f_{\texttt{STRBEG}}(v, w)$ is the characteristic function of the predicate 'the string with coding v is an initial segment of the string with coding w.'*
- `STREND`*, of two arguments; for all integers v, w, $\Phi(\texttt{STREND}, v, w) \leq \max(w, (a-1) \cdot v + 1 + c_m) \leq a^{\max(|v|,|w|)+2} + c_m$ and $f_{\texttt{STREND}}(v, w)$ is the characteristic function of the predicate*

'the string with coding v is a final segment of the string with coding w.'

- $\texttt{SUBSTR}$, *of two arguments; for all integers v, w, $\Phi(\texttt{SUBSTR}, v, w) \leq (a-1)\cdot\max(v, a^3 \cdot w) + 1 + c_m \leq a^{\max(|v|,|w|)+5} + c_m$ and $f_{\texttt{SUBSTR}}(v, w)$ is the characteristic function of the predicate 'the string with coding v is a substring of the string with coding w.'*
- $\texttt{WORK}$, *of one argument; for all integers v, $\Phi(\texttt{WORK}, v) \leq (a-1)v^2 + 1 + c_{m+1} \leq a^{2|v|+3} + c_{m+1}$ and $f_{\texttt{WORK}}$ is the function which, given an input number v, outputs the coding in the alphabet with $a \cdot p$ characters of the string whose coding in the alphabet with a characters is v.*
- $\texttt{IS-GROUND}$, *of one argument; for all integers v, $\Phi(\texttt{IS-GROUND}, v) \leq (ap-1)v + 1 + c_{m+1} \leq a^{|v|+3} + c_{m+1}$ and $f_{\texttt{IS-GROUND}}$ is the characteristic function of the predicate 'every character of the string which has coding v in the larger alphabet is also a character of the smaller one.'*
- $\texttt{GROUND}$, *of one argument; for all integers v, $\Phi(\texttt{GROUND}, v) \leq (ap-1)v + 1 + c_{m+1} \leq a^{|v|+3} + c_{m+1}$ and $f_{\texttt{GROUND}}$ is an inverse of $f_{\texttt{WORK}}$. More precisely if σ is the string which has coding v in the larger alphabet, then $f_{\texttt{GROUND}}(v)$ is 0 if some character greater than a appears in σ, otherwise it is the coding of σ in the smaller alphabet.*

proof For each integer s, the first coding f_s of a string of length s is the coding of $c_1c_1\ldots c_1$ while the last one l_s is the coding of $c_ac_a\ldots c_a$. Since $l_s = af_s$, two integers n and m are the coding of strings of equal length if and only if there exists an integer f which is the coding of a sequence of c_1's and such that $f \leq n, m \leq a \cdot f$.

In order to be able to verify that an integer is the coding of a sequence of c_1's we can reason in the following way. Consider the coding of an n-long string $\sigma = c_{i_1}\ldots c_{i_n}$:

$$C(\sigma) = i_1 \cdot a^{n-1} + i_2 \cdot a^{n-2} + \ldots + i_{n-k} \cdot a^k + \underbrace{i_{n-(k-1)} \cdot a^{k-1} + \ldots + i_n}_{k\text{ terms}}$$

We would like to access i_{n-k}, the $k+1$-st coefficient from the right-hand side of this expression, to verify that it is c_1. In general the value of $C(\sigma) \div a^k$ will depend also on the k rightmost terms in $C(\alpha)$, because their partial sum can be in many cases greater than a^k—e.g. for $a = 7$ and $k = 2$ consider $6 \cdot 49 + 7 \cdot 7 + 1$. So we cannot claim that in general i_{n-k} can be derived immediately from $[C(\sigma) \div a^k] \bmod a$, but we can do it in case the right-most coefficients are small enough that they do not give any disturbance—for example if they are all 1. Then the condition $\sigma \in \{c_1\}^*$ can be proven equivalent to $\forall k < n\ [C(\sigma) \div a^k] \bmod a = 1$ by induction on the length n of σ.

$\texttt{IS-C-ONE} :=$ % of two arguments, a string and a position
  $\texttt{IMPLIES}(\texttt{IS-POWER}_m(p, \texttt{P}^2_2), \texttt{c}_=(1, \texttt{MOD}(\texttt{DIV}(\texttt{P}^2_1, \texttt{P}^2_2), a)))$

$\texttt{IS-C-ONE-STAR} :=$ % of one argument
  $\texttt{IS-C-ONE}(\texttt{P}^1_1,$
  $\min \texttt{OR}(\texttt{c}_=(\texttt{P}^2_1, \texttt{P}^2_2),$
    $\texttt{NOT}(\texttt{IS-C-ONE}(\texttt{P}^2_1, \texttt{P}^2_2))))$

```
FIRST-STR :=            %    returns the right c1c1...c1, if any, a wrong string otherwise
  min OR(GREATER-EQ(P^3_3, P^3_1),
    GREATER-EQ(P^3_3, P^3_2),
    AND(GREATER-EQ(a · P^3_3, P^3_1),
      GREATER-EQ(a · P^3_3, P^3_2),
      IS-C-ONE-STAR(P^3_3)))
EQ-LENGTH :=
  AND(GREATER-EQ(P^2_1, FIRST-STR),
    GREATER-EQ(P^2_2, FIRST-STR),
    GREATER-EQ(a · FIRST-STR, P^2_1),
    GREATER-EQ(a · FIRST-STR, P^2_2),
    IS-C-ONE-STAR(FIRST-STR))
```

The function $f_{\texttt{EQ-LENGTH}}$ is total and it is the characteristic function of the predicate 'the strings of coding v and w have the same length.' All occurrences of a and p which appear in the programs above can be eliminated, substituing multiplications and divisions with addition and minimalization. The program `EQ-LENGTH` may be modified in this way in order to establish that, since all minimalizations are bounded by both v and w, for each v, w $\Phi(\texttt{EQ-LENGTH}, v, w) \leq \min(v, w) + c_m$.

In order to build the program `PWR-FLOOR`, which on input w computes the $|w|$-th power of a, consider that if n is positive, then

$$a^n = (a-1)a^{n-1} + (a-1)a^{n-2} + \ldots + (a-1)a + a.$$

This means that if $w > 0$ then $a^{|w|}$ is the coding of the $|w|$-long string $c_{a-1}c_{a-1}\ldots c_{a-1}c_a$, which is the unique string of this length that has coding a perfect power of a.

```
PWR-FLOOR :=
  min OR(AND(c=(P^2_1, 0), c=(P^2_2, 1)),
    AND(GREATER(P^2_1, 0), IS-POWER_m(p, P^2_2), EQ-LENGTH(P^2_1, P^2_2)))
```

The function $f_{\texttt{PWR-FLOOR}}$ is total, and for all v $\Phi(\texttt{PWR-FLOOR}, v) \leq (a-1) \cdot v + 1 + c_m$, removing the dependence of the complexity from the constant p.

The definitions of `APPEND`, `STRBEG` and `STREND` are now straightforward:

```
APPEND :=
  + ( × (P^2_1, PWR-FLOOR(P^2_2)), P^2_2)
STRBEG :=
  c=(P^2_2, APPEND(P^2_1,
      min GREATER-EQ(APPEND(P^3_1, P^3_3), P^3_2)))
STREND :=
  c=(P^2_2, APPEND(
      min GREATER-EQ(APPEND(P^3_3, P^3_1), P^3_2),
      P^2_1))
```

$$
\begin{array}{l}
\texttt{SUBSTR} := \\
\quad \texttt{STREND}(\\
\quad\quad \texttt{APPEND}(\texttt{P}^2_1, \\
\quad\quad\quad \min \texttt{OR}(\\
\quad\quad\quad\quad \texttt{c}_=(0, \texttt{P}^3_1), \\
\quad\quad\quad\quad \texttt{GREATER-EQ}(\texttt{P}^3_1, \texttt{P}^3_2), \\
\quad\quad\quad\quad \texttt{EQ-LENGTH}(\texttt{P}^3_1, \texttt{P}^3_2), \\
\quad\quad\quad\quad \texttt{GREATER-EQ}(\texttt{APPEND}(\texttt{P}^3_1, \texttt{P}^3_3), \texttt{P}^3_2), \\
\quad\quad\quad\quad \texttt{STREND}(\texttt{APPEND}(\texttt{P}^3_1, \texttt{P}^3_3), \texttt{P}^3_2))) \\
\quad\quad \texttt{P}^2_2)
\end{array}
$$

The functions $f_{\texttt{APPEND}}, f_{\texttt{STRBEG}}, f_{\texttt{STREND}}$ and $f_{\texttt{SUBSTR}}$ are total. For all v, w $\Phi(\texttt{APPEND}, v, w) \leq \max(v, (a-1)\cdot w+1+c_m)$, $\Phi(\texttt{STRBEG}, v, w) \leq \max(v, (a-1)\cdot w+1+c_m)$ and $\Phi(\texttt{STREND}, v, w) \leq \max(w, (a-1)\cdot v+1+c_m)$. In case $v \leq w$, $\Phi(\texttt{STRBEG}, v, w), \Phi(\texttt{STREND}, v, w) \leq (a-1)\cdot w+1+c_m$.

Consider the minimalization in the definition of $\texttt{SUBSTR}$. On input v, w, if $0 < |v| < |w|$, the increasing argument $\texttt{P}^3_3$ is always bounded by w, but $\texttt{APPEND}(\texttt{P}^3_1, \texttt{P}^3_3)$ may eventually produce a string t such that $|t| = |w|+1$. In any case $a^{|w|-1} < w < t < a^{|w|+2}$ and $t \leq a^3 \cdot w$. Otherwise $\texttt{P}^3_3$ takes only value 0. In any case

$$
\begin{array}{l}
\Phi(\texttt{SUBSTR}, v, w) \leq \\
\quad \max(\Phi(\texttt{EQ-LENGTH}, v, w), \Phi(\texttt{APPEND}, v, w), \Phi(\texttt{STREND}, v, w), \Phi(\texttt{STREND}, a^3\cdot w, w)) \leq \\
\quad \leq \max(\,\min(v, w) + c_m, \max(v, (a-1)\cdot w + 1 + c_m), \max(w, (a-1)\cdot v + 1 + c_m), \\
\quad\quad\quad \max(w, (a-1)a^3 \cdot w + 1 + c_m)) \leq \\
\quad \leq (a-1)\cdot \max(v, a^3 \cdot w) + 1 + c_m
\end{array}
$$

Some utilities are now introduced in order to build the programs $\texttt{WORK}$ and $\texttt{GROUND}$ which let us shift between the two adiacent codings relative to the alphabets with a and $a\cdot p$ characters. The first one is $\texttt{ARE-SAME-PWR}$ of two arguments v, and w, which computes the characteristic function of the predicate 'exists q such that $v = a^q$ and $w = (ap)^q$.' Since this is equivalent to 'exists u such that $u = p^q$ is a power of p and $v = u^m, w = u^{m+1}$,' the program $\texttt{ARE-SAME-PWR}$ and its complexity closely resembles the definition of $\texttt{IS-POWER}_m$:

$$
\begin{array}{l}
\texttt{MTH-POWER} := \\
\quad \times(\,\times(\cdots \times(\texttt{P}^1_1, \texttt{P}^1_1)\cdots), \texttt{P}^1_1) \qquad \texttt{\%}\ m \quad \text{times}
\end{array}
$$

$$
\begin{array}{l}
\texttt{ARE-SAME-PWR} := \\
\quad \texttt{AND}(\texttt{IS-POWER}(p, \texttt{P}^2_1), \\
\quad\quad \texttt{c}_=(\texttt{P}^2_1, \texttt{MTH-POWER}(\,\min \texttt{GREATER-EQ}(\texttt{MTH-POWER}(\texttt{P}^3_3), \texttt{P}^3_2))) \\
\quad\quad \texttt{c}_=(\texttt{P}^3_1, \times(\texttt{P}^3_2, \min \texttt{GREATER-EQ}(\texttt{MTH-POWER}(\texttt{P}^3_3), \texttt{P}^3_2))))
\end{array}
$$

The computed function is total, and $\Phi(\texttt{ARE-SAME-PWR}, v, w) \leq v + c_m$. The following program $\texttt{CH-CHECK}$ of three arguments v, w and n, checks that the strings of coding v and w, respectively in the ground and work alphabet, match in the k-th position, if $n = a^k$ is a power of a. It uses a program of two arguments that computes the initial segment of the first argument that is as long as the second one.

$$
\begin{array}{l}
\texttt{CH-CHECK} := \\
\quad \texttt{IMPLIES}(\texttt{IS-POWER}_m(p, \texttt{P}^3_3), \\
\qquad \texttt{c}_=(\min \texttt{AND}(\texttt{GREATER}(\texttt{P}^4_4, 0), \\
\qquad\qquad\qquad \texttt{c}_=(\texttt{MOD}(\texttt{P}^4_4, a), \texttt{MOD}(\texttt{INITIAL}(\texttt{P}^4_1, \texttt{P}^4_3), a))), \\
\qquad\quad \texttt{MOD}(\texttt{INITIAL}_{ap}(\texttt{P}^3_2, \min \texttt{ARE-SAME-PWR}(\texttt{P}^4_3, \texttt{P}^4_4)), ap)))
\end{array}
$$

$$
\begin{array}{l}
\texttt{INITIAL} := \min \texttt{OR}(\texttt{AND}(\texttt{GREATER}(\texttt{P}^3_2, \texttt{P}^3_1), \texttt{NOT}(\texttt{EQ-LENGTH}(\texttt{P}^3_2, \texttt{P}^3_1))), \\
\qquad \texttt{AND}(\texttt{EQ-LENGTH}(\texttt{P}^3_3, \texttt{P}^3_2), \texttt{STRBEG}(\texttt{P}^3_3, \texttt{P}^3_1)))
\end{array}
$$

For each input v, w $\Phi(\texttt{INITIAL}, v, w) \leq (a-1)v+1+c_m$ because in any case only two relevant instructions are executed, namely $\texttt{EQ-LENGTH}$ on arguments v, w and $\texttt{STRBEG}$ on arguments v, v. This implies that $\Phi(\texttt{CH-CHECK}, v, w, n) \leq \max((a-1)v+1+c_m, (a-1)w+1+c_{m+1}, u+c_m)$.

We are now able to specify and analize the programs $\texttt{WORK}$,$\texttt{IS-GROUND}$ and $\texttt{GROUND}$.

$$
\begin{array}{l}
\texttt{WORK} := \\
\quad \min \texttt{CH-CHECK}(\texttt{P}^2_1, \texttt{P}^2_2, \\
\qquad \min \texttt{OR}(\texttt{c}_=(\texttt{P}^3_1, 0), \\
\qquad\quad \texttt{AND}(\texttt{IS-POWER}_m(p, \texttt{P}^3_3), \texttt{EQ-LENGTH}(\texttt{P}^3_3, \texttt{P}^3_1)), \\
\qquad\quad \texttt{NOT}(\texttt{CH-CHECK}(\texttt{P}^3_1, \texttt{P}^3_2, \texttt{P}^3_3))))
\end{array}
$$

During the computation of $\texttt{WORK}$ on argument v, the increasing value $\texttt{P}^2_2$ in the outer minimalization is bounded by the coding in the work alphabet of the string of coding v—in any case it is bounded by v^2. The increasing value $\texttt{P}^3_3$ in the inner minimalization is bounded by the first power of a that codes a string as long as $|v|$, which is in turn bounded by $(a-1)v+1$. Hence $\Phi(\texttt{WORK}, v) \leq \Phi(\texttt{CH-CHECK}, v, v^2, (a-1)v+1) \leq (a-1)v^2+1+c_{m+1}$.

$$
\begin{array}{l}
\texttt{IS-GROUND} := \\
\quad \texttt{GREATER-EQ}(a, \texttt{MOD}(\texttt{INITIAL}_{ap}(\texttt{P}^1_1, \\
\qquad \min \texttt{OR}(\texttt{c}_=(\texttt{P}^2_1, 0), \\
\qquad\quad \texttt{AND}(\texttt{IS-POWER}_{ap}(p, \texttt{P}^2_2), \texttt{EQ-LENGTH}_{ap}(\texttt{P}^2_2, \texttt{P}^2_1)), \\
\qquad\quad \texttt{GREATER}(\texttt{MOD}(\texttt{INITIAL}_{ap}(\texttt{P}^2_1, \texttt{P}^2_2), ap), a)), ap)))
\end{array}
$$

Computing on input argument v, the increasing value $\texttt{P}^2_2$ in the minimalization is bounded by the first power of ap that codes a string as long as $|v|_{ap}$, which is bounded by $(ap-1)v+1$. Hence $\Phi(\texttt{IS-GROUND}, v) \leq \Phi(\texttt{IS-POWER}_{ap}, p, (a-1)v+1) \leq (ap-1)v+1+c_{m+1}$.

$$
\begin{array}{l}
\texttt{GROUND} := \\
\quad \min \texttt{OR}(\texttt{NOT}(\texttt{IS-GROUND}(\texttt{P}^2_1)), \texttt{CH-CHECK}(\texttt{P}^2_2, \texttt{P}^2_1, \\
\qquad \min \texttt{OR}(\texttt{NOT}(\texttt{IS-GROUND}(\texttt{P}^3_1)), \texttt{c}_=(\texttt{P}^3_1, 0), \\
\qquad\quad \texttt{AND}(\texttt{IS-POWER}_m(p, \texttt{P}^3_3), \texttt{EQ-LENGTH}(\texttt{P}^3_3, \texttt{P}^3_2)), \\
\qquad\quad \texttt{NOT}(\texttt{CH-CHECK}(\texttt{P}^3_2, \texttt{P}^3_1, \texttt{P}^3_3)))))
\end{array}
$$

On argument v, the increasing value $\texttt{P}^2_2$ in the outer minimalization is bounded by the coding in the ground alphabet of the string that has coding v in the work alphabet—in any case it is bounded by v. The increasing value $\mathtt{P}_3^3$ in the inner minimalization is then bounded by $(a-1)v+1$. Hence $\Phi(\mathtt{GROUND}, v) \leq \Phi(\mathtt{IS\text{-}GROUND}, v) \leq (ap-1)v+1+c_{m+1}$. This concludes the proof of the theorem. ∎

Before going on and building programs that compute the exponentiation function and simulate a Turing machine computation, in order to simplify the complexity analyses to come, it is useful to consider the preceding theorem in a different way, as bounding the length of the greatest string involved in the computation of any ordinary string manipulation. We can assert that for each prime p and positive integer m, there exists a constant r_a depending on $a = p^m$ such that each string manipulation among those specified in the previous statement does not involve multiplication on arguments greater than $2^{r_a|v|+r_a}$, v being the greatest argument to the program. All string manipulation programs that are listed in the theorem above involve costly operations on strings whose length is at most linear in the maximum length of the input arguments, and they also output strings whose length is at most linear in the sum of the length of the inputs.

Suppose that H is a program of n arguments which computes a total function f_H using the string manipulations above only on strings whose length is bounded by some monotone function g of the n arguments. The contribution of the string manipulations to the multiplicative complexity $\Phi(H)$ of H is not greater than $2^{r_a \cdot [dg(v_1,\ldots,v_n)+d]+r_a}$—where d is derived from the linear bound on the output of manipulations. This observation will simplify the complexity analyses of the following programs, because bounding the length of the longest string involved in a computation is often straightforward. For example we are going to build a program $\mathtt{EXP}$ which computes the exponentiation function, and we are going to bound its complexity simply by noting that the length of the longest string checked in the main minimalization is proportional to the square of the value v of the input argument—which will yield a $2^{r_a \cdot (cv^2)+r_a}$ upper bound.

We are going to use the exponentiation function in order to simulate a Turing machine computation working in double exponential time, the reason will become clear later.

The recursive definition of $\exp(1,n) = 2^n$ is an application of the recursive operator to the multiplication function. We do not have a primitive program for $\exp(1,n)$ and we do not have recursion among our program constructors. But still we can build a program $\mathtt{EXP}$ with an application of the standard technique for removing primitive recursion from function definition.

<u>Proposition</u>

There exists a program $\mathtt{EXP}$ *of one argument and a constant* c *such that for all* n $f_{\mathtt{EXP}}(n) = \exp(1,n)$ *and* $\Phi(\mathtt{EXP}, n) < \exp(1, c \cdot n^2 + c)$.

<u>*proof*</u> Let us fix an $a = p^m$, e.g. $a = 3$, and use both the ground alphabet with a symbols and the work alphabet with $ap \geq a+2$ symbols, containing the symbols # and \$ not appearing in a. Using these alphabets we build a program $\mathtt{EXP}$, which will be safely used in several contexts where a different alphabet has been chosen, without affecting the constant c in the complexity upper bound on $\mathtt{EXP}$ that we are going to establish.

For each integer n, the value 2^n can be computed by multiplying n times the unity by 2. A computation of this kind can be represented in the work alphabet by a string m of the following form:

$$\#\$a_1\#a_1\$a_2\#a_2\$a_1a_1\#a_3\$a_2a_2\#a_1a_1\$a_1a_2a_1\#\ldots\#f_{\mathtt{WORK}}(k)\$f_{\mathtt{WORK}}(2^k)\ldots \qquad (\dagger)$$

Each portion $\#v\$w\#$ is such that v is an integer between 0 and n—represented by the string in the work alphabet corresponding to the string that has coding v in the ground alphabet—and w is 2^v—represented in the same way.

The program `EXP` on argument n computes the minimum integer that contains a string like (†) and that terminates with $\#n\$w$, for some w; then it outputs w. Let us suppose that `EXP-COMP` is a program of one argument that given z, the coding in the work alphabet of a ground string v, computes (†). Then `EXP` is simply the program:

$$\texttt{EXP} := \texttt{GROUND}(\min \texttt{STREND}_{ap}(\texttt{APPEND}_{ap}(\text{“}\#\text{”}, \texttt{WORK}(\texttt{P}^2_2)), \texttt{EXP-COMP}(\texttt{WORK}(\texttt{P}^2_1))))$$

The program `EXP-COMP` on argument n will compute the shortest string that contains a string like (†) and terminates with $\#n\$w$. In order to build `EXP-COMP` we need another auxiliary program. The program `EXP-NEXT` of two arguments, which on inputs v and w of the form specified above shall output the string $\#v\$w\#v'\$w'$, where v' and w' are the strings corresponding to $v+1$ and $2 \cdot w$.

$\texttt{EXP-COMP} :=$
$\quad \min \texttt{AND}($
$\quad\quad \texttt{STRBEG}_{ap}(\text{“}\#\$a_1\text{”}, \texttt{P}^2_2),$
$\quad\quad \texttt{SUBSTR}_{ap}(\texttt{APPEND}_{ap}(\texttt{APPEND}_{ap}(\text{“}\#\text{”}, \texttt{P}^2_1), \text{“}\$\text{”}), \texttt{P}^2_2),$
$\quad\quad \texttt{c}_=(\texttt{P}^2_2,$
$\quad\quad\quad \min \texttt{OR}(\texttt{c}_=(\texttt{P}^3_3, \texttt{P}^3_2),$
$\quad\quad\quad \texttt{NOT}(\texttt{c}_=(\texttt{P}^3_2,$
$\quad\quad\quad \min \texttt{OR}(\texttt{c}_=(\texttt{P}^4_4, \texttt{P}^4_2),$
$\quad\quad\quad \texttt{NOT}(\texttt{IMPLIES}($
$\quad\quad\quad\quad \texttt{AND}($
$\quad\quad\quad\quad\quad \texttt{IS-GROUND}(\texttt{P}^4_3),$
$\quad\quad\quad\quad\quad \texttt{GREATER}(\texttt{P}^4_1, \texttt{P}^4_3),$
$\quad\quad\quad\quad\quad \texttt{IS-GROUND}(\texttt{P}^4_4),$
$\quad\quad\quad\quad\quad \texttt{SUBSTR}_{ap}(\texttt{APPEND}_{ap}($
$\quad\quad\quad\quad\quad\quad \texttt{APPEND}_{ap}(\text{“}\#\text{”}, \texttt{P}^4_3),$
$\quad\quad\quad\quad\quad\quad \texttt{APPEND}_{ap}(\texttt{APPEND}_{ap}(\text{“}\$\text{”}, \texttt{P}^4_4), \text{“}\#\text{”})),$
$\quad\quad\quad\quad\quad \texttt{P}^4_2),$
$\quad\quad\quad\quad \texttt{SUBSTR}_{ap}(\texttt{EXP-NEXT}(\texttt{P}^4_3, \texttt{P}^4_4), \texttt{P}^4_2)))))))))$

On input argument z, this program computes the shortest string σ that correctly begins with $\#0\$2^0$ and contains $\#z\$$, and which is such that for every substring of the form $\#v\$w\#$, with $v < z$ and v, w containing only ground symbols, also the string $\#v\$w\#v+1\$2w$ is contained in σ. The first string with these properties is a computation like (†) that terminates with $\$z\#f_{\texttt{WORK}}(2^{f_{\texttt{GROUND}}(z)})$. The definition of `EXP-NEXT` follows:

$\texttt{EXP-NEXT} :=$
$\quad \texttt{APPEND}_{ap}($
$\quad\quad \texttt{APPEND}_{ap}(\texttt{APPEND}_{ap}(\text{“}\#\text{”}, \texttt{P}^2_1), \texttt{APPEND}_{ap}(\text{“}\$\text{”}, \texttt{P}^2_2)),$
$\quad\quad \texttt{APPEND}_{ap}($
$\quad\quad\quad \texttt{APPEND}_{ap}(\text{“}\#\text{”}, \texttt{WORK}(+(1, \texttt{GROUND}(\texttt{P}^2_1)))),$
$\quad\quad\quad \texttt{APPEND}_{ap}(\text{“}\$\text{”}, \texttt{WORK}(\times(2, \texttt{GROUND}(\texttt{P}^2_2))))))$

In order to bound the multiplicative complexity of $\mathtt{EXP}$ we may bound the length of the string (†). On input $v > 0$, v strings (in the work alphabet) of the form $\#z\$w$ are to be appended to $\#\$a_1$ in order to give (†). Each z is a string in the work alphabet such that $f_{\mathtt{GROUND}}(z)$ is an integer between 1 and v; z is as long as $f_{\mathtt{GROUND}}(z)$, hence $|z|_{ap} \leq \lceil \log_a v \rceil$. The string w is such that $f_{\mathtt{GROUND}}(w) = \exp(1, f_{\mathtt{GROUND}}(z)) \leq a^v$; hence $|w|_{ap} \leq v$.

On input $v > 0$, the longest string appearing in the computation of $\mathtt{EXP}$ has length bounded by $2v^2 + 3$. Take $c > 3$ to be a constant greater than $\Phi(\mathtt{EXP}, 0)$ and greater than $2d(r_{ap} + 1), 2d(r_a + 1)$; then for all v $\Phi(\mathtt{EXP}, v) \leq 2^{cv^2+c}$ ∎

For each $n > 2$ there exists a program $\mathtt{POWER}_n$ of one argument which on input w computes n^w. The definition of $\mathtt{POWER}_n$ differs from $\mathtt{EXP}$ just in the constant appearing in the $\mathtt{EXP\text{-}NEXT}$ program, which will be n instead of 2. The complexity of $\mathtt{POWER}_n$ is bounded by 2^{cv^2+c}, c being a constant dependent only on n, since the corresponding (†) is bounded by $2\lceil \log_2 n \rceil v^2 + 3$.

Having constructed a program that computes exponentiation, we are able to measure the length of strings and to extract the character in any given position of a string. A program $\mathtt{LENGTH}$ of one argument can be defined which on input w computes the length $|w|$ of the string whose coding is w in the alphabet with a characters:

$$
\begin{aligned}
&\mathtt{LENGTH} := \\
&\quad \min \mathtt{OR}(\mathtt{c}_=(\mathtt{P}^2_1, 0), \\
&\qquad \mathtt{AND}(\mathtt{NOT}(\mathtt{c}_=(\mathtt{P}^2_1, 0)), \mathtt{EQ\text{-}LENGTH}(\mathtt{POWER}_a(\mathtt{P}^2_2), \mathtt{P}^2_1)))
\end{aligned}
$$

The complexity $\Phi(\mathtt{LENGTH}, v)$ is bounded by $2^{c|v|^2+c}$ for some c depending only on a, since $|v|$ is the greatest argument to the $\mathtt{POWER}_a$ program. Finally, a program $\mathtt{STR}$ of two arguments can be defined that on input arguments v, w—with w in the range $0 \leq w < |v|$—computes the $w+1$-st character $v[w]$ from the left[8]

$$
\begin{aligned}
&\mathtt{STR} := \min \mathtt{OR}(\mathtt{GREATER\text{-}EQ}(\mathtt{P}^3_2, \mathtt{LENGTH}(\mathtt{P}^3_1)), \\
&\quad \mathtt{AND}(\mathtt{GREATER}(\mathtt{P}^3_3, 0), \mathtt{c}_=(\mathtt{MOD}(\mathtt{P}^3_3, a), \mathtt{MOD}(\\
&\qquad \min \mathtt{OR}(\mathtt{GREATER\text{-}EQ}(\mathtt{P}^4_2, \mathtt{LENGTH}(\mathtt{P}^4_1)), \\
&\qquad\quad \mathtt{AND}(\mathtt{STRBEG}(\mathtt{P}^4_4, \mathtt{P}^4_1), \mathtt{EQ\text{-}LENGTH}(\mathtt{P}^4_4, \\
&\qquad\qquad \mathtt{POWER}_a(\min \mathtt{OR}(\mathtt{GREATER\text{-}EQ}(\mathtt{P}^5_2, \mathtt{LENGTH}(\mathtt{P}^5_1)), \mathtt{c}_=(\mathtt{P}^5_5, +(\mathtt{P}^5_2, 1))))))), \\
&\qquad a))))
\end{aligned}
$$

The function $f_{\mathtt{STR}}$ is total, and takes value 0 in case the second argument specifies a position that is out of range. The complexity $\Phi(\mathtt{STR}, v, w)$ is bounded by $2^{c|v|^2+c}$ for some c depending on a, because the argument to the $\mathtt{POWER}_a$ instruction is either 0 or $w < |v|$.

For each string manipulation function in the set of functions that are needed to simulate a Turing machine computation, a min-program has been shown which computes that function without multipling too large a number. We can no longer delay dealing with the cumbersome details of a standard simulation of a non-deterministic Turing machine working in bounded time:

[8] That is the character that is the coefficient of the higher power of a in the coding of v. This unfamiliar use may seem odd, but it is consistent with the fact that the characters appearing in an initial segment of v are those that more heavily contribute to the coding v.

Theorem

There exists a constant c_3 ($c_3 > 1$) such that for each language $L(M) \subseteq \Sigma^*$ accepted by a non-deterministic Turing machine M working in time $\exp(2, c \cdot n)$ for some $c > 0$, there exists a program H such that for all $w \in \Sigma^*$:

$$\Phi(H, w) < \exp(3, c_3 c \cdot |w|) \wedge f_H(w) = \begin{cases} 0 & \textit{if } w \in L(M) \\ 1 & \textit{otherwise} \end{cases}$$

proof Let M be a non-deterministic Turing machine, with one tape, infinite in both directions, working alphabet Σ, states Q and transition relation δ. The machine starts in an initial state q_0 and computes according to δ, that is a set of five-tuples. Each tuple $\langle q, r, H, r', q' \rangle$ in δ specifies that the machine being in state q and reading the bit r, may (non-deterministically) write the bit r', enter state q' and move the head according to H, either to the left, L, or to the right, R, or may keep it unchanged, S. We can suppose without loss of generality that M has only one final state f, and that the transition relation specifies that when M enters state f, it does not change its configuration anymore.

We may code a description of a possible computation of M, on input w and length $\exp(2, c|w|)$, with a couple of strings $\langle \tau, \sigma \rangle$, $\tau \in \Sigma^*$ and $\sigma \in (Q \cup \{\#\})^*$.

The first string τ will describe the tape contents at each instant of the computation, and will simply be a concatenation of $\exp(2, c|w|)$ tape images, each of length $2\exp(2, c|w|)$—because $\exp(2, c|w|)$ tape cells are relevant on each side of the starting head position. Hence τ has length $2\exp(2, c|w|)^2 = 2\exp(2, c|w| + 1)$.

The string σ contains a description of the head position and of the machine state at each instant of the computation. It is a concatenation of $\exp(2, c|w|)$ strings of length $2\exp(2, c|w|)$, each concerning a particular step of the computation. The character $\sigma[m]$, $0 \leq m < 2\exp(2, c|w|)^2$, is q if and only if at the n-th step of the computation, $n = m \div 2\exp(2, c|w|)$, the head is in position $h = m \bmod 2\exp(2, c|w|)$, and the machine is in the state q—otherwise $\sigma[m]$ =‘#’. In this way, since # is not in Q, σ is # everywhere but in the head positions, one for each step of the computation, where it evaluates to the current machine state q.

Then τ and σ are obtained concatenating $\exp(2, c|w|)$ snapshots of the machine configuration at successive steps, each one being $2\exp(2, c|w|)$ long.

The machine M accepts w in time $\exp(2, c|w|)$ if and only if there exists a couple of strings of length $\exp(2, 2c|w| + 2) \geq 2\exp(2, c|w| + 1)$ that codes an accepting computation of M of length greater than $\exp(2, c|w|)$ starting on the input w—because we know that M cannot accept after the $\exp(2, c|w|)$-th step.

We choose as alphabet a superset of $\Sigma \cup Q \cup \{\#\}$ with a prime number of characters and build a program H' of one argument which on input $w \in \Sigma^*$ generates all strings that are at most $\exp(2, 2\lceil c|w| \rceil + 2)$ long and check whether they code an accepting computation of M with input the string of coding w. In the worst case, when M does not accept w, a string as long as $\exp(2, 2\lceil c|w| \rceil + 2)$ is generated. Hence $\Phi(H', w) \leq \exp(1, c_a(d\exp(2, 2\lceil c|w| \rceil + 2) + d) + c_a)$ which is in any case definitively less than $\exp(3, c_3 c|w|)$ for some constant c_3 independent of a and M. We can then modify H' in such a way that all input exceptions are treated separately without any multiplication but using only the program $\mathtt{c}_=$ and the logical operators; in the modified program H all minimalizations will also perform an initial check which truncates the computation at the first step whenever the input argument is treated as an exception, and all inputs to expensive operations will be forced to 0—in the same way we implemented in the definition of $\mathtt{STR}$—whenever the input is among the exceptions. Then the modified program H will have complexity $\Phi(H, v) \leq \exp(3, c_3 c|w|)$ and $f_H = f_{H'}$.

In the rest of the proof we build the program H'. We need a program $\mathtt{INITIAL\text{-}SNP}$ of two arguments, which returns an initial machine snapshot. The first argument is the input to the Turing machine, and is used to select the right configuration length, $2\exp(2, c|w|)$. The second argument is used to initialize the right-hand side of the snapshot—it is meant to be either the input or the initial state.

```
INITIAL-SNP :=
  APPEND(
    min AND(
      c=(LENGTH(P^3_3), EXP2(LENGTH(P^3_1))),
      IS-BLANK(P^3_3)),
    APPEND(P^2_2,
        min AND(
          c=(LENGTH(APPEND(P^3_2, P^3_3)), EXP2(LENGTH(P^3_1))),
          IS-BLANK(P^3_3))))
```

The program $\mathtt{IS\text{-}BLANK}$ of one argument v is intended to compute the characteristic function of the predicate 'v is a blank string'. Let c be $\frac{n}{m}$. The program $\mathtt{MAX\text{-}COMPUTATION}_c$ of one argument, the Turing machine input w, computes $\exp(2, 2c|w| + 2)$, without multiplying too large numbers:

$$\mathtt{MAX\text{-}COMPUTATION}_c := \mathtt{EXP}(\mathtt{EXP}(+(2, \mathtt{DIV}(\times(2n, \mathtt{LENGTH}), m))))$$

The program H' is made of two nested cycles of minimalization. Together they generate all couples of strings $\langle \sigma, \tau \rangle$ of appropriate length and check whether there is one that codes an accepting computation of M on input w. In this case the program will output 0, otherwise it will output 1.

```
NOT(c_=(MAX-COMPUTATION,
  LENGTH(
   min OR(c_=(LENGTH(P^2_2), MAX-COMPUTATION(P^2_1)),
    AND(
      SUBSTR("f", P^2_2),
      STRBEG(INITIAL-SNP(P^2_1, "q_0"), P^2_2),
      LESS-EQ(LEFT-DISPLACEMENT(P^2_1), LENGTH(
        min AND(
          IS-BLANK(P^3_3),
          OR(
            SUBSTR(APPEND("a_0", APPEND(P^3_3, "a_0")), P^3_2),
            SUBSTR(APPEND("a_0", APPEND(P^3_3, "a_1")), P^3_2),
                ⋮
            SUBSTR(APPEND("a_n", APPEND(P^3_3, "a_n")), P^3_2))))),
      NOT(c_=(MAX-COMPUTATION(P^3_1),
        LENGTH(
          min OR(c_=(LENGTH(P^3_3), MAX-COMPUTATION(P^3_1)),
            AND(
              STRBEG(INITIAL-SNP(P^3_1, P^3_1), P^3_3),
              c_=(MAX-COMPUTATION(P^3_1),
                min OR(c_=(P^4_4, MAX-COMPUTATION(P^4_1)),
                  NOT(IMPLIES(
                    c_=(STR(P^4_2, P^4_4), BLANK),
                    c_=(STR(P^4_3, P^4_4),
                        STR(P^4_3, +(STILL-DISPLACEMENT(P^4_1), P^4_4)))))),
                  NOT(OR(
                    IS-MOVE_δ_0,
                        ⋮
                    IS-MOVE_δ_n))))))))))))
```

```
IS-MOVE_⟨q,r,H,r',q'⟩ :=
  AND(
    c_=("q", STR(P^4_2, P^4_4)),
    c_=("r", STR(P^4_3, P^4_4)),
    c_=("q'", STR(P^4_2, +(P^4_4, H-DISPLACEMENT(P^4_1)))),
    c_=("r'", STR(P^4_3, +(P^4_4, STILL-DISPLACEMENT(P^4_1)))))
```

The programs `LEFT-DISPLACEMENT`, `STILL-DISPLACEMENT`, and `RIGHT-DISPLACEMENT` of one argument, on input w compute the numbers $2\exp(2, c|w|)-1, 2\exp(2, c|w|)$, and $2\exp(2, c|w|)+1$, and are defined in the obvious way.

This concludes the proof that min-programs can "efficiently" simulate non-deterministic Turing machines, and hence also the proof of our first inseparability result. ∎

5. A CLASSICAL PROOF USING TURING MACHINES

In [FR74], Fischer and Rabin proved the difficulty of the complete Presburger theory S_+ by showing that for each non-deterministic Turing machine M and input w there exists a short and easy sentence $\varphi_{M,w}$ in the language of addition that is true in the standard model—i.e. provable in S_+—if and only if M halts on input w within $\exp(2,|w|)$ steps. In this section some steps of that construction are carried out for the theory Q_+ in order to show that the whole work can be done in this system, yielding an inseparability result.

Some modifications are necessary to obtain this goal. The first one is closely related to the fact that we are not dealing with a complete theory, and hence whenever the behavior of a Turing machine must be linked to the provability of a sentence in the theory, the arguments cannot rely upon the system's ability to prove the negation of false statements.

In order to overcome this, we restrict ourselves to using only formulas with bounded quantifiers, so that each formula is equivalent in Q_+ to a quantifier free matrix. In particular the formulas representing functions, such as $x \underset{k}{\otimes} y = z$, are equivalent to the disjunctions of all relevant input/output descriptions.

For each non-deterministic Turing machine M, polynomial g and input w it will be shown that a sentence $\varphi_{M,g,w}$ can be constructed as in [FR74], in which only bounded quantifiers occur. In this way M accepts w within $\exp(2,g(|w|))$ steps if and only if $Q_+ \vdash \varphi_{M,g,w}$, and M does not accept w within $\exp(2,g(|w|))$ steps if and only if $Q_+ \vdash \neg\varphi_{M,g,w}$—since bounded quantifiers can be eliminated.

The multiplication function, as represented by $x \underset{k}{\otimes} y = z$, is not the sole function that we need in order to simulate a Turing machine in the way of [FR74]. We need functions to manipulate strings, considered as binary representations of integers, predicates to express bounds and a way to shorten representation of integers, for numerals are too lengthy for our purposes.

This last matter has an immediate solution, since for each integer a whose binary representation has length k or less, the formula $W_k^a(x)$:

$$\exists x_0\,(x_0 = \overline{a[k]} \wedge \exists x_1\,(x_1 = 2\cdot x_0 + \overline{a[k-1]} \wedge \exists x_0\,(x_0 = 2\cdot x_1 + \overline{a[k-2]} \wedge \cdots \\ \wedge \exists x_i\,(x_i = 2\cdot x_{1-i} + \overline{a[1]} \wedge x = 2\cdot x_i + \overline{a[0]})\cdots)))$$

has length linear in k and is equivalent in Q_+ to $x = \overline{a}$.

Using $x \underset{k}{\otimes} y = z$ and $M_k(x,y,z)$, we can easily build sequences of formulas that express the necessary predicates, which happen to be just $x < \exp(2,k)^2$, $x = \exp(2,k)$ and $\exp(2,k)|x \wedge x < \exp(2,k)^2$. Let $I_k(x)$ be the sequence $M_{k+1}(x,0,0)$, $J_k(x)$ be $\exists y\,[M_k(y,0,0) \wedge Sy = x \wedge \neg M_k(x,0,0)]$ and $Z_k(x)$ be $\exists su\,[J_k(u) \wedge s < u \wedge M_{k+2}(s,u,x)]$. Replacing $M_k(x,y,z)$ with the disjunction that is equivalent to it in Q_+, we see that

$$Q_+ \vdash J_k(x) \leftrightarrow x = \overline{\exp(2,k)}$$

$$Q_+ \vdash I_k(x) \leftrightarrow \bigvee_{s<\exp(2,k)^2} x = \overline{s}$$

$$Q_+ \vdash Z_k(x) \leftrightarrow \bigvee_{s<\exp(2,k)} x = \overline{s\cdot\exp(2,k)}$$

This guarantees that I_k, J_k and Z_k actually have in Q_+ their intended meaning.

Only one string-manipulation function is indeed necessary to simulate a Turing machine in the way of [FR74], the function $S(a,i)$ that for all integers a and i returns $a[i]$, the i-th bit in the binary representation of a. To express this function we need to compute the exponentiation function 2^i. The sequence that in Q_+ expresses exponentiation cannot be immediately built from the sequence $x \underset{k}{\otimes} y = z$, but the construction is not very long either.

Using the definition of the major times a sequence of formulas $E_k(x,y,z)$—whose length is linear in n—has to be defined such that

$$E_k(x,y,z) \leftrightarrow \bigvee_{\substack{a<\exp(2,k)\\ b,b^a \le m_{k+4}}} [x=\overline{a} \wedge y=\overline{b} \wedge z=\overline{b^a}].$$

The construction of $E_k(x,y,z)$ starts with the inductive definition of formulas $E_i^k(x,y,z,u,v,w)$ for all i and k, with $0 \le i \le k$. Roughly speaking, the meaning of these formulas is:

$$x < 2^{2^i} \quad \text{and} \quad z = y^x \le m_{k+4} \quad \text{and} \quad u \underset{k}{\otimes} v = w.$$

The formulas $E_i^k(x,y,z,u,v,w)$ are built inductively on i for each k, and it is proved that each E_i^k is equivalent in Q_+ to

$$\bigvee_{\substack{a<\exp(2,i)\\ b,b^a \le m_{k+4}}} [x=\overline{a} \wedge y=\overline{b} \wedge z=\overline{b^a}] \wedge \bigvee_{A,B\le m_{k+4}} [u=\overline{A} \wedge v=\overline{B} \wedge w=\overline{A\cdot B}]. \tag{$\dagger$}$$

Afterwards a straightforward definition of $E_k(x,y,z)$ can be $E_k^k(x,y,z,0,0,0)$, i.e. the following sequence of formulas is used for the exponentiation function:

$$E_0^0(x,y,z,0,0,0), E_1^1(x,y,z,0,0,0), E_2^2(x,y,z,0,0,0), \ldots$$

For each k, two clauses inductively define $E_i^k(x,y,z,u,v,w)$:
$E_0^k(x,y,z,u,v,w)$ is:

$$[(x=0 \wedge z=S0) \vee (x=S0 \wedge z=y)] \wedge u \underset{k}{\otimes} v = w \wedge y \underset{k}{\otimes} 0 = 0 \wedge z \underset{k}{\otimes} 0 = 0,$$

and $E_{i+1}^k(x,y,z,u,v,w)$, $i<k$, is:

$$\begin{aligned}&\exists u_1u_2u_3u_4u_5u_6u_7u_8u_9\ x = u_2+u_3+u_4 \wedge E_i^k(u_1,y,u_5,u_1,u_1,u_4) \wedge \\ &\qquad E_i^k(u_1,u_5,u_6,u_6,u_7,u_9) \wedge E_i^k(u_2,y,u_7,u_8,u_9,z) \wedge E_i^k(u_3,y,u_8,u,v,w)\end{aligned}$$

The formula $E_{i+1}^k(x,y,z,u,v,w)$ expresses the power y^x in terms of the product $(y^{u_1})^{u_1} \times y^{u_2} \times y^{u_3}$, giving $x = u_1^2+u_2+u_3$ in any decomposition of x into smaller elements $u_j < 2^{2^i}$. The four occurences of E_i^k can be reduced to one with the same transformation that was used for the minor times, in order to have a sequence $E_i^k(x,y,z,u,v,w)$ whose elements have length that grows linearly in $k+i$.

The proof of the equivalence of $E_i^k(x, y, z, u, v, w)$ with the disjunction (†) can be carried out by induction on i. The base is straightforward, because

$$[(x = 0 \wedge z = S0) \vee (x = S0 \wedge z = y)] \wedge u \underset{k}{\otimes} v = w \wedge y \underset{k}{\otimes} 0 = 0 \wedge z \underset{k}{\otimes} 0 = 0$$

is equivalent in Q_+, by replacement, to

$$\bigvee_{\substack{a < \exp(2,0) \\ b, b^a \le m_{k+4}}} [x = \overline{a} \wedge y = \overline{b} \wedge z = \overline{b^a}] \wedge \bigvee_{A, B \le m_{k+4}} [u = \overline{A} \wedge v = \overline{B} \wedge w = \overline{A \cdot B}].$$

Note now the role of the bounds on y and z expressed by $y \underset{k}{\otimes} 0 = 0 \wedge z \underset{k}{\otimes} 0 = 0$.

The step can be proved checking that in the next expression—which is equivalent in Q_+ to $E_{i+1}^k(x, y, z, u, v, w)$—all trailing existential quantifiers can be eliminated, reducing it to the desired disjunction as has been done in the analogous proof for the minor times.

$$\begin{aligned}
&\exists u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 \; x = u_2 + u_3 + u_4 \wedge \\
&\bigvee_{\substack{a_1 < \exp(2,i) \\ b_1, b_1^{a_1} \le m_{k+4}}} [u_1 = \overline{a_1} \wedge y = \overline{b_1} \wedge u_5 = \overline{b_1^{a_1}}] \wedge \bigvee_{A_1, B_1 \le m_{k+4}} [u_1 = \overline{A_1} \wedge u_1 = \overline{B_1} \wedge u_4 = \overline{A_1 \cdot B_1}] \wedge \\
&\bigvee_{\substack{a_2 < \exp(2,i) \\ b_2, b_2^{a_2} \le m_{k+4}}} [u_1 = \overline{a_2} \wedge u_5 = \overline{b_2} \wedge u_6 = \overline{b_2^{a_2}}] \wedge \bigvee_{A_2, B_2 \le m_{k+4}} [u_6 = \overline{A_2} \wedge u_7 = \overline{B_2} \wedge u_9 = \overline{A_2 \cdot B_2}] \wedge \\
&\bigvee_{\substack{a_3 < \exp(2,i) \\ b_3, b_3^{a_3} \le m_{k+4}}} [u_2 = \overline{a_3} \wedge y = \overline{b_3} \wedge u_7 = \overline{b_3^{a_3}}] \wedge \bigvee_{A_3, B_3 \le m_{k+4}} [u_8 = \overline{A_3} \wedge u_9 = \overline{B_3} \wedge z = \overline{A_3 \cdot B_3}] \wedge \\
&\bigvee_{\substack{a_4 < \exp(2,i) \\ b_4, b_4^{a_4} \le m_{k+4}}} [u_3 = \overline{a_4} \wedge y = \overline{b_4} \wedge u_8 = \overline{b_4^{a_4}}] \wedge \bigvee_{A_4, B_4 \le m_{k+4}} [u = \overline{A_4} \wedge v = \overline{B_4} \wedge w = \overline{A_4 \cdot B_4}]
\end{aligned}$$

Now that we have built $E_k(x, y, z)$ which represents exponentiation in Q_+, we can construct a sequence $S_k(x, y, z)$ for the function $S(a, i)$. Roughly, the meaning of $S_k(x, y, z)$ has to be:

$$|x|, y + 1 \le \left(2^{2^k}\right)^2 \quad \text{and} \quad z = x[y]$$

and the value $z = S(x, y)$ can be simply taken to be $(x \div 2^y) \bmod 2$—because strings are represented in binary notation. In the following, let $\alpha < \exp(3, k)$ stand for $\exists uv[J_k(u) \wedge E_{k+1}(u, \overline{2}, v) \wedge \alpha < v]$, and define $S_k(x, y, z)$ to be:

$$\begin{aligned}
&[x < \exp(3, k+1) \wedge I_k(y) \wedge (z = 0 \vee z = S0)] \wedge \\
&\quad \exists mqrtw \, [m < \exp(3, k+1) \wedge \cdots \wedge w < \exp(3, k+1) \wedge E_{k+1}(y, \overline{2}, m) \wedge \\
&\qquad x = w + r \wedge q \underset{k+1}{\otimes} m = w \wedge r < m \wedge q = 2 \cdot t + z].
\end{aligned}$$

Then $S_k(x, y, z)$ is equivalent in Q_+ to

$$\bigvee_{\substack{a = 0, 1, 10, 11, \ldots, \exp(3, k+1) - 1 \\ n = 0, 1, 2, \ldots, \exp(2, k+1) - 1 \\ c = a[n]}} x = \overline{a} \wedge y = \overline{n} \wedge z = \overline{c},$$

which implies that $S_k(x, y, z)$ has the intended meaning in Q_+.

We have now collected enough material to simulate a non-deterministic Turing machine within the theory Q_+.

Lemma

There exists a constant d and a polynomial $p(n)$ such that for each polynomial $g(n)$ and non-deterministic Turing machine M there exists a $p(n+g(n))$-time $O(n+g(n))$-space deterministic Turing machine P_g^M such that for each input w for M, $P_g^M(w)$ is a sentence $\varphi_{M,g,w}$ such that $|\varphi_{M,g,w}| \leq d \cdot [|w| + g(|w|)]$ and

$$M \text{ accepts } w \text{ within } \exp(2, g(|w|)) \text{ steps} \quad \Longleftrightarrow \quad Q_+ \vdash \varphi_{M,g,w} \quad \Longleftrightarrow \quad Q_+ \not\vdash \neg\varphi_{M,g,w}\,.$$

proof Once the function S and the predicates I, J, Z and W have been defined, the proof follows the line of the construction given in [FR74] for the complete theory S_+.

Let M be a non-deterministic Turing machine, with one tape, working alphabet $\Sigma = \{0, 1\}$, set of states Q and transition relation δ. The machine starts in an initial state q_0 and computes according to δ, that is a set of five-tuples. Each tuple $\langle q, r, H, r', q' \rangle$ in δ specifies that the machine being in state q and reading the bit r, may (non-deterministically) write the bit r', enter state q' and move the head according to H, either to the left, L, or to the right, R, or let it stand still, S. We can suppose without loss of generality that M has only one final state f, and that the transition relation specifies that when M enters state f, it does not change its configuration anymore.

Let k equal $g(|w|)$. We may code a description of a possible computation of M, on input w and length $\exp(2, g(|w|)) = \exp(2, k)$, with a tuple $\langle \tau, \sigma_0, \ldots, \sigma_{l-1} \rangle$ of $1 + \lceil \log(1 + |Q|) \rceil$ strings in $\{0, 1\}^*$.

The first string, τ will describe the tape contents at each instant of the computation, and will simply be a concatenation of $\exp(2, k)$ tape images, each of length $\exp(2, k)$. Hence τ has length $\exp(2, k)^2 = \exp(2, k + 1)$.

The remaining $l = \lceil \log(1 + |Q|) \rceil$ strings contain altogether a description of the head position and the machine state at each instant of the computation. Each σ_i is a concatenation of $\exp(2, k)$ strings of length $\exp(2, k)$, each concerning a particular step of the computation. In some unique fashion associate a number in $\{1, \ldots, 1 + |Q|\}$ to each machine state. The bit $\sigma_i[m]$, $0 \leq m < \exp(2, k)^2$, is 1 if and only if at the n-th step of the computation, $n = m \div \exp(2, k)$, the head is in position $h = m \bmod \exp(2, k)$ and the machine is in a state q such that the $i+1$-st bit in the binary representation of the integer associated to q is 1. In this way, since 0 is not associated to any $q \in Q$, we can look at $\langle \sigma_{l-1}, \ldots, \sigma_0 \rangle$ as a unique string σ, in the alphabet $\Lambda = \{0, \ldots, 2^l - 1\} \supseteq \{\underline{0}\} \cup Q$, that is $\underline{0}$ everywhere but in the head positions, one for each step of the computation, where it evaluates to the current machine state q.

Then τ and σ are obtained concatenating $\exp(2, k)$ snapshots of the machine configuration at successive steps, each one being $\exp(2, k)$ long.

M accepts w in time $\exp(2, g(|w|))$ if and only if there exists a tuple of strings that codes an accepting computation of M of exact length $\exp(2, g(|w|))$, starting on the input w. Let us write $M_w(\tau, \sigma)$ to mean that τ and σ code such a computation.

Using the string manipulation function S, it is possible to express in Q_+ that a certain tuple effectively encodes an accepting computation of the appropriate length. More specifically, formulas with free variables $t, s_0, \ldots, s_{l-1}$ can be designed whose meaning is that the values of t and $\underline{s}$ encode an accepting computation of M on input w:

- Let $\varphi_a(t, \underline{s})$ specify that t and $\underline{s}$ are binary strings of the appropriate length:

$$\forall i \; I_k(i) \rightarrow [\exists d \; S_k(t, i, d) \wedge \exists \underline{d} \; S_k(\underline{s}, i, \underline{d})].$$

  Since $I_k(i)$ means that i is in the correct range $\{0, \ldots, \exp(2,k)^2 - 1\}$, the formula φ_a says that for each position i there is a corresponding character in t and $\underline{s}$—$S_k(\underline{s}, i, \underline{d})$ being an abbreviation for $\bigwedge_{n<l} S_k(s_n, i, d_n)$. It is equivalent in Q_+ to the disjunction:

$$Q_+ \vdash \forall t \, \underline{s} \; \varphi_a(t, \underline{s}) \leftrightarrow \bigvee_{\substack{\tau, \sigma_0, \ldots, \sigma_{l-1} = \\ 0,1,10,11,\ldots,\exp(3,k+1)-1}} t = \overline{\tau} \wedge s_0 = \overline{\sigma}_0 \wedge \ldots \wedge s_{l-1} = \overline{\sigma}_{l-1}.$$

  We write this in the following more compact way:

$$Q_+ \vdash \forall t \, \underline{s} \; \varphi_a(t, \underline{s}) \leftrightarrow \bigvee_{\substack{\tau \epsilon \Sigma^*, \sigma \epsilon \Lambda^* \\ |\tau| = |\sigma| = \exp(2,k)^2}} t = \overline{\tau} \wedge \underline{s} = \overline{\sigma}.$$

- Let $\varphi_b(t, \underline{s})$ specify that t and $\underline{s}$ start with an initial configuration of M on input w:

$$\begin{aligned}
&\varphi_a(t, \underline{s}) \wedge S_k(\underline{s}, 0, q_0) \wedge \\
&\forall uvx \; \{(W^w_{|w|}(x) \wedge J_k(u) \wedge v < u) \rightarrow \forall j \, [S_k(x, v, j) \leftrightarrow S_k(t, v, j)]\},
\end{aligned}$$

  with q_0 the initial state of M. Since $J_k(j)$ is equivalent to $j = \overline{\exp(2,k)}$, that is the length of a snapshot description in the strings τ and σ, the formula φ_b requires that q_0 be the first character of σ, and that each tape cell in the first computation step match the input w. The formula is equivalent to an appropriate disjunction, where t and $\underline{s}$ assume values τ and σ such that τ starts with a snapshot of the tape corresponding to the input w, and $\sigma[0] = q_0$. Let $\Phi_b(\tau, \sigma)$ be an abbreviation for this statement about τ and σ; then:

$$Q_+ \vdash \forall t \, \underline{s} \; \varphi_b(t, \underline{s}) \leftrightarrow \bigvee_{\substack{|\tau| = |\sigma| = \exp(2,k)^2 \\ \Phi_b(\tau, \sigma)}} t = \overline{\tau} \wedge \underline{s} = \overline{\sigma}.$$

- Let $\varphi_c(t, \underline{s})$ specify that t and $\underline{s}$ encode a sequence of $\exp(2,k)$ correct moves of M:

$$\begin{aligned}
&\varphi_a(t, \underline{s}) \wedge \\
&\forall i \, j \, j_L \, j_S \, j_R \, \big([I_k(i) \wedge J_k(j) \wedge SSj_L = Sj = Sj_S = j_R] \rightarrow \\
&\qquad \big\{[S_k(\underline{s}, i, \underline{0}) \rightarrow \forall u \; S_k(t, i+j, u) \leftrightarrow S_k(t, i, u)] \wedge \\
&\qquad\quad \big[\neg S_k(\underline{s}, i, \underline{0}) \rightarrow \\
&\qquad\qquad \{\neg I_k(i+j) \wedge [\forall l \, (i < l \wedge I_k(l)) \rightarrow S_k(\underline{s}, l, \underline{0})] \;\; \vee \\
&\qquad\qquad\; I_k(i+j) \wedge [\forall l \; i < l < i + j_L \rightarrow S_k(\underline{s}, l, \underline{0})] \wedge \\
&\qquad\qquad\; \bigvee_{\delta(q,r,H,r',q')} [Z_k(i+j_H) \rightarrow \neg j_H = j_R] \wedge [Z_k(i) \rightarrow \neg j_H = j_L] \wedge \\
&\qquad\qquad\qquad S_k(\underline{s}, i, q) \wedge S_k(\underline{s}, i+j_H, q') \wedge S_k(t, i, \overline{r}) \wedge S_k(t, i+j, \overline{r}')\}\big]\big\}\big).
\end{aligned}$$

Let i be any available character position in the whole string t and in $\underline{s}$, and let j be the length of a snapshot description. Then if $\sigma[i]$ is $\underline{0}$, the head is not at that position, and the corresponding character does not change during that step: $\tau[i+j] = \tau[i]$. On the contrary, if the character $\sigma[i] = q \neq \underline{0}$ does not belong to the very last snapshot, because $I_k(i+j)$ holds, then all the nearby characters of σ are $\underline{0}$, and M must be allowed to move from the current state to a new state according to some correct transition, that does not require the head to move beyond the tape limits.

The formula $\varphi_c(t, \underline{s})$ is equivalent to a disjunction where t and $\underline{s}$ assume values τ and σ that describe an $\exp(2, g(|w|))$ long computation of M, starting and ending in any configuration. If no such computation exists, then $\varphi_c(t, \underline{s})$ is equivalent in Q_+ to a contradiction—that is $Q_+ \vdash \neg\varphi_c(t, \underline{s})$. Let us agree that an empty disjunction stands for a contradiction and that $\Phi_c(\tau, \sigma)$ is an abbreviation for the above statement about τ and σ; then:

$$Q_+ \vdash \forall t\, \underline{s}\, \varphi_c(t, \underline{s}) \leftrightarrow \bigvee_{\substack{|\tau|=|\sigma|=\exp(2,k)^2 \\ \Phi_c(\tau,\sigma)}} t = \overline{\tau} \wedge \underline{s} = \overline{\sigma}.$$

- Let $\varphi_d(t, \underline{s})$ specify that $\underline{s}$ contains the final state f of M:

  $$\varphi_a(t, \underline{s}) \wedge \exists\, i\, [I_k(i) \wedge S_k(\underline{s}, i, f)].$$

  In the corresponding equivalent disjunction $\underline{s}$ assumes values σ such that, for some i, $\sigma[i] = f$:

  $$Q_+ \vdash \forall t\, \underline{s}\, \varphi_d(t, \underline{s}) \leftrightarrow \bigvee_{\substack{|\tau|=|\sigma|=\exp(2,k)^2 \\ \Phi_d(\sigma)}} t = \overline{\tau} \wedge \underline{s} = \overline{\sigma}.$$

Consider now the conjunction $\mu_k(t, \underline{s})$ given by $\varphi_b(t, \underline{s}) \wedge \varphi_c(t, \underline{s}) \wedge \varphi_d(t, \underline{s})$. This formula is equivalent in Q_+ to a disjunction where t and $\underline{s}$ assume values τ and σ that describe an $\exp(2, g(|w|)) = \exp(2, k)$ long computation of M, starting at an initial configuration on input w and ending in an accepting configuration—if any such computation exists. Otherwise $\mu_k(t, \underline{s})$ is equivalent in Q_+ to a contradiction—that is $Q_+ \vdash \neg\mu_k(t, \underline{s})$. As before, let us simply write:

$$Q_+ \vdash \forall t\, \underline{s}\, \mu_k(t, \underline{s}) \leftrightarrow \bigvee_{\substack{|\tau|=|\sigma|=\exp(2,k)^2 \\ M_w(\tau,\sigma)}} t = \overline{\tau} \wedge \underline{s} = \overline{\sigma}.$$

Take $\varphi_{M,g,w}$ to be $\exists t\, \exists \underline{s}\; \mu_k(t, \underline{s})$. If M accepts w then $Q_+ \vdash \varphi_{M,g,w}$, otherwise $Q_+ \vdash \neg\varphi_{M,g,w}$. The length of $\varphi_{M,g,w}$ is bounded by $d[|w| + g(|w|)]$, for some d. Indeed it has the form $\alpha_0\alpha_1^k\alpha_2 W^w_{|w|}(x)\alpha_3\alpha_4^k \cdots \alpha_s$, i.e. it may be simply computed by concatenating some fixed patterns, some of which are repeated $g(|w|)$ times, with the exception of $W^w_{|w|}(x)$. Since $g(|w|)$ can be computed in polynomial time, $\varphi_{M,g,w}$ can certainly be computed deterministically by a Turing machine in space $O(|w| + g(|w|))$ and time $p(|w| + g(|w|))$, for some polynomial p. This concludes the proof. ∎

The hypotheses of the preceding lemma are not the most general ones. In particular g could be any function that is computable by a binary transducer on a unary input argument in polynomial time.

For $g(n) = c \cdot n$, we have proven that there exists a d such that for each c and M there exists a polynomial-time linear-space deterministic Turing machine P_c^M such that for each input w, $P_c^M(w) = \varphi_{M,g,w}$ and $|\varphi_{M,g,w}| \leq d \cdot (c+1) \cdot |w|$. Also, if $g(n)$ is a polynomial, there exists a d such that for each M there exists a polynomial p and a polynomial-time deterministic Turing machine P_g^M such that for each input w, $P_g^M(w) = \varphi_{M,g,w}$ and $|\varphi_{M,g,w}| \leq d \cdot p(|w|)$. These facts are used in the proof of the following theorem.

Theorem*(Inseparability for Non-Deterministic Turing Machines)*

Let $T \subset \Sigma^*$ be a set of strings, in the alphabet $\Sigma \supseteq \Sigma_+$, that separates Q_+ from $\mathrm{Uns}\,\mathcal{L}_+$. Then T is $\leq_{\mathrm{p-lin}}$-hard for each class $\mathrm{NTIME}(\exp(2, c \cdot n))$ and is $\leq_{\mathrm{p}}$-hard for each class $\mathrm{NTIME}(\exp(2, n^c))$, and there exists a c_0 such that $T \notin \mathrm{NTIME}(\exp(2, c_0 \cdot n))$:

$$\begin{aligned} &\bigcup_{0<c} \mathrm{NTIME}(\exp(2, c \cdot n)) \leq_{\mathrm{p-lin}} T \quad \text{and} \\ &\bigcup_{0<c} \mathrm{NTIME}(\exp(2, n^c)) \leq_{\mathrm{p}} T \quad \text{and} \\ &\text{exists} \quad c_0 \quad \text{s.t.} \quad T \notin \mathrm{NTIME}(\exp(2, c_0 \cdot n)). \end{aligned}$$

proof Recall that $\mathrm{NTIME}(f(n))$ is the set of languages L such that there exists a non-deterministic Turing machine M_L that recognizes L in time $f(n)$, i.e. there exists a machine M_L such that for each word $w \in \Sigma^*$, if $w \notin L$ then there is no accepting computation of M_L on input w, otherwise there exists an accepting computation of length $f(|w|)$ or less. We may assume as before that if L is in $\mathrm{NTIME}(f(n))$ then there is a witnessing machine M_L that has only one final state that it never leaves once it has reached it, as in the proof of the preceding lemma.

Let $T \subset \Sigma^*$ be any set of strings such that $Q_+ \subseteq T$ and $\mathrm{Uns}\,\mathcal{L}_+ \subseteq T^c$, the complement of T. We prove that the lower bounds that are listed in the theorem statement hold for both T and T^c. In this way it is proven that they apply to any T that separates Q_+ from Uns.

We first prove that

$$\bigcup_{0<c} \mathrm{NTIME}(\exp(2, c \cdot n)) \leq_{\mathrm{p-lin}} T \quad \text{and} \quad \bigcup_{0<c} \mathrm{NTIME}(\exp(2, c \cdot n)) \leq_{\mathrm{p-lin}} T^c$$

showing that for each constant $c > 0$ and language L in $\mathrm{NTIME}(\exp(2, c \cdot n))$ there is a polynomial time linear space reduction of L to T and of L to T^c. Let L be a language in $\mathrm{NTIME}(\exp(2, c \cdot n))$, and M a witnessing machine. Let $g(n)$ be $c \cdot n$. Modify the polynomial time linear space algorithm P_c^M of the previous lemma in such a way that with the same time and space bounds on input w it outputs the sentence $\alpha_+ \wedge \varphi_{M,g,w}$—$\alpha_+$ being the conjunction of the finitely many axioms of the theory. If M accepts w then $Q_+ \vdash \varphi_{M,g,w}$ and $\alpha_+ \wedge \varphi_{M,g,w} \in Q_+ \subseteq T$, otherwise $Q_+ \vdash \neg\varphi_{M,g,w}$ and $\alpha_+ \wedge \varphi_{M,g,w} \in \mathrm{Uns}\,\mathcal{L}_+ \subseteq T^c$.

On the other hand P_c^M can be modified to output $\alpha_+ \wedge \neg\varphi_{M,g,w}$. If M accepts w then $Q_+ \vdash \varphi_{M,g,w}$ and $\alpha_+ \wedge \neg\varphi_{M,g,w} \in \mathrm{Uns}\,\mathcal{L}_+ \subseteq T^c$, otherwise $Q_+ \vdash \neg\varphi_{M,g,w}$ and $\alpha_+ \wedge \neg\varphi_{M,g,w} \in Q_+ \subseteq T$. This concludes the proof of the first statement of the theorem.

We can prove in the same way that

$$\bigcup_{0<c} \mathrm{NTIME}(\exp(2,n^c)) \leq_{\mathrm{p}} T \quad \text{and} \quad \bigcup_{0<c} \mathrm{NTIME}(\exp(2,n^c)) \leq_{\mathrm{p}} T^c$$

showing that for each constant $c > 0$ and language L in $\mathrm{NTIME}(\exp(2,n^c))$ there is a polynomial time reduction of L to T and of L to T^c. Let L be a language in $\mathrm{NTIME}(\exp(2,n^c))$, and M a witnessing machine. Let $g(n)$ be n^c and notice that the algorithm P_g^M of the previous lemma still works in polynomial time $p(g(n))$. Modify P_g^M in such a way that with the same time bound on input w it outputs the sentence $\alpha_+ \wedge \varphi_{M,g,w}$: as before this reduces $L(M)$ to T. Modifying P_g^M to output $\alpha_+ \wedge \neg\varphi_{M,g,w}$ gives a polynomial reduction of $L(M)$ to T^c.

In order to conclude the proof of the theorem, we need to modify the preceding lemma to show that the programs P_g^M, one for each machine M, could indeed be substituted by a unique program P_g, that takes the coding of a machine M and an input w to M, and outputs $\varphi_{M,g,w}$.

In general the coding of a binary Turing machine is a binary word whose length grows as a function of the cardinalities $|Q|$ and $|\delta|$ of the set of states and of the transition relation respectively. A short and natural coding could be, for example, a simple concatenation of the tuples in δ, where $\log|Q|$ bits are enough to denote each state $q \in Q$; the length of such coding would be a function of $|Q|$ and $|\delta|$ that roughly is as $\Theta(|\delta| \cdot \log|Q|)$.

In any case there are many ways of unequivocally associating a coding to each machine, and we are free to choose one that fits our needs. Let us suppose that we have defined a coding of binary Turing machines such that the length $|M|$ of the binary representation of the coding of M is for example $\Theta(|\delta|^2 \cdot |Q|^2)$, by possibly padding a more natural $\Theta(|\delta| \cdot \log|Q|)$ coding[9]. Furthermore let us choose the coding in such a way that we can append to it any binary word w to get a string $\langle M, w\rangle$ where the border between M and w is clearly marked, without using extra characters beyond 0 and 1—e.g. every second bit of the coding of M could be a 0, while a pair 11 separates M from w in $\langle M, w\rangle$.

We still need to prove that there exists a constant d such that for each linear function[10] $g(n) = c \cdot n$ there exists a polynomial-time deterministic Turing machine P_g such that for each non-deterministic Turing machine M and input w, $P_g(\langle M, w\rangle)$ is a sentence equivalent to $\varphi_{M,g,w}$ whose length is bounded by $d \cdot (|M| + |w| + g(|w|))$; the behavior of P_g on an input $\langle \alpha, w\rangle$ such that α does not correctly specify the coding of a Turing machine is irrelevant, we can for example state that P_g does not terminate on such inputs.

Looking at the given construction of $\varphi_{M,g,w}$, it should be clear that building the sentence from $\langle M, w\rangle$ in polynomial time is still not a problem. The crux is the linear bound that must hold on the output, since one can say that the length of $\varphi_{M,g,w}$ is $O(\,|w| + |\delta| \cdot \log|Q| \cdot g(|w|)\,)$ but one certainly cannot say that it is $O(|w| + |M| + g(|w|))$. The point is that the number of

[9] Indeed we could also choose some quite natural $\Theta(|\delta| \cdot \log|Q|)$ coding, as it will turn out. The point that we want to stress here is the fact that we are free to choose a coding as fat as is needed to prove our statement.

[10] Again, there is no need to limit this sentence to linear functions. But there is no point in a more general statement, for it would not imply a stronger inseparability—as can be seen at the end of the proof.

occurrences in $\varphi_{M,g,w}$ of formulas Z_k and S_k whose length grows as $\Theta(g(|w|))$ does depend on the machine M. Let us see how we can easily reduce these occurrences to a constant number in two steps.

First of all, the writing $S_k(\underline{s}, u, \underline{q})$ that appears time and again in $\mu_k(t, \underline{s})$ hides a conjunction of $l = \lceil \log(1 + |Q|) \rceil$ occurrences of S_k. Again use the fact that Q_+ is a theory with equality to reduce these occurrences to only one:

$$
\begin{aligned}
&Q_+ \vdash \forall\, u\, \underline{q}\, \underline{s} \\
&\qquad \bigwedge_{i<l} S_k(s_i, u, q_i) \;\;\leftrightarrow\;\; \forall v_0 v_1 v_2 \left[\bigvee_{i<l} v_0 = s_i \wedge v_1 = u \wedge v_2 = q_i\right] \rightarrow S_k(v_0, v_1, v_2).
\end{aligned}
$$

Then we can substitute the right-hand side of the equivalence, call it $S'_k(\underline{s}, u, \underline{q})$, for $S_k(\underline{s}, u, \underline{q})$ throughout $\varphi_{M,g,w,}$ obtaining an equivalent sentence $\varphi'_{M,g,w}$.

Still $\varphi'_c(t, \underline{s})$ grows too quickly for our purposes. Indeed, the length of the last disjunctive subformula of $\varphi'_c(t, \underline{s})$ still depends on $k = g(|w|)$ and on the number of tuples in the transition relation δ in such a way that it is $\Theta(|\delta| \cdot g(|w|) + |\delta| \cdot \log(|Q|))$. Then $\varphi''_c(t, \underline{s})$ is taken to be the logically equivalent formula obtained by reducing the occurrences of the formulas S_k and Z_k to a constant number, independent from the cardinality of the transition relation δ:

$$
\begin{aligned}
&\varphi_a(t, \underline{s}) \wedge \\
&\forall\, i\, j\, j_L\, j_S\, j_R\; \big([I_k(i) \wedge J_k(j) \wedge SSj_L = Sj = Sj_S = j_R] \rightarrow \\
&\big\{[S'_k(\underline{s}, i, \underline{0}) \rightarrow \forall u\, S_k(t, i+j, u) \leftrightarrow S_k(t, i, u)] \wedge \\
&\quad \big[\neg S'_k(\underline{s}, i, \underline{0}) \rightarrow \\
&\qquad \big\{\neg I_k(i+j) \wedge [\forall l\, (i < l \wedge I_k(l)) \rightarrow S'_k(\underline{s}, l, \underline{0})] \quad \vee \\
&\qquad I_k(i+j) \wedge [\forall l\, i < l < i + j_L \rightarrow S'_k(\underline{s}, l, \underline{0})] \wedge \\
&\qquad\; \exists\, \underline{v}\; \big([S'_k(\underline{v}_0, v_1, \underline{v}_2) \wedge S'_k(\underline{v}_3, v_4, \underline{v}_5) \wedge S_k(v_6, v_7, v_8) \wedge S_k(v_9, v_{10}, v_{11})] \wedge \\
&\qquad\quad [Z_k(v_{12}) \rightarrow \neg v_{13} = j_R] \wedge [Z_k(v_{14}) \rightarrow \neg v_{15} = j_L] \wedge \\
&\qquad\quad \bigvee_{\delta(q,r,H,r',q')} \quad \underline{v}_0 = \underline{s} \wedge v_1 = i \wedge \underline{v}_2 = q \wedge \underline{v}_3 = \underline{r}' \wedge v_4 = i + j_H \wedge \underline{v}_5 = q' \wedge \\
&\qquad\qquad\qquad v_6 = t \wedge v_7 = i \wedge v_8 = \overline{r} \wedge v_9 = t \wedge v_{10} = i + j \wedge v_{11} = \overline{s} \wedge \\
&\qquad\qquad\qquad v_{12} = i + j_H \wedge v_{13} = j_H \wedge v_{14} = i \wedge v_{15} = j_H\big\}\big)\big]\big\}\big).
\end{aligned}
$$

The corresponding sentence $\varphi''_{M,g,w}$ now has length $\Theta(|w| + g(|w|) + |\delta| \cdot \log|Q|)$, that is $O(|w| + g(|w|) + |M|)$ and we are done. In the following let us briefly use $\varphi_{M,g,w}$ to refer to this shorter but equivalent sentence.

We can now show, for both sets T and T^c, that there exists a constant e such that for each Turing machine M that recognizes the set, a sentence can be built that belongs to it and is not accepted by M in time $\exp(2, e \cdot n)$.

Let M_T be a non-deterministic Turing machine that accepts T. We do not start by fixing an appropriate constant e, but instead we are going to look at a generic linear function $g(n) = e \cdot n$ and see how we can mix the ingredients M_T and P_g in order to obtain a lower bound of $\exp(2, e \cdot n)$, for some e, on the length of any accepting computation of M_T on a certain worst input—and then we show how e can be chosen *a priori*.

We can say, in some sense, that P_g on input $\langle M, w \rangle$ outputs a sentence $\varphi_{M,g,w}$ whose meaning in Q_+ is *"There is an accepting computation of M on input w, that is shorter* *than* $\exp(2, g(|w|))$." Starting from M_T and P_g we are going to build a sentence σ whose meaning is[11] roughly *"There is* **no** *accepting computation of M_T on input* **myself**, *that is shorter than* $\exp(2, e \cdot |\sigma|)$.".

Modify the machine P_g to output $\alpha_+ \wedge \neg\varphi_{N,g,N}$ on input any machine coding N, and compose this new polynomial-time deterministic machine with the non-deterministic machine M_T; call $\overline{M}$ the resulting non-deterministic Turing machine and the corresponding coding. Then $\overline{M}$ on input N that is the coding of a Turing machine, computes $\varphi_{N,g,N}$ and starts M_T on input $\alpha_+ \wedge \neg\varphi_{N,g,N}$—$\overline{M}$ does not accept an input N that is not a correct coding.

Let σ be the sentence $\alpha_+ \wedge \neg\varphi_{\overline{M},g,\overline{M}}$ and consider the sentences σ and $\varphi_{\overline{M},g,\overline{M}}$. Either $\varphi_{\overline{M},g,\overline{M}}$ or $\neg\varphi_{\overline{M},g,\overline{M}}$ belongs to Q_+. Then $\varphi_{\overline{M},g,\overline{M}} \in Q_+$ if and only if $\sigma \in \mathrm{Uns}\mathcal{L}_+ \subseteq T^c$; $\varphi_{\overline{M},g,\overline{M}} \notin Q_+$ if and only if $\neg\varphi_{\overline{M},g,\overline{M}} \in Q_+$, if and only if $\sigma \in Q_+ \subseteq T$. This means that σ belongs to Q_+ and T if and only if there is no accepting computation of M_T on input σ within $\exp(2, g(|\overline{M}|))$ steps.

If $Q_+ \vdash \varphi_{\overline{M},g,\overline{M}}$, then $\overline{M}$ accepts $\overline{M}$ (within $\exp(2, g(|\overline{M}|))$ steps), that is M_T accepts σ, while σ belongs to $\mathrm{Uns}\mathcal{L}_+ \subseteq T^c$, a contradiction.

Then it must be the case that $Q_+ \not\vdash \varphi_{\overline{M},g,\overline{M}}$ and σ belongs to T. Hence M_T accepts σ, but it must take it a lot of time, for otherwise $\overline{M}$ would accept $\overline{M}$ within $\exp(2, g(|\overline{M}|))$ steps. The length of σ is $[q + d \cdot (2|\overline{M}| + g(|\overline{M}|))]$—where q is $|\alpha_+ \wedge \neg|$—and the time necessary to generate σ from $\overline{M}$ cannot exceed $q + p(|\overline{M}| + 2 + |\overline{M}|) = r(|\overline{M}|)$ for some polynomials p and r. Then the machine M_T cannot accept σ in less than $\exp(2, e \cdot |\sigma|)$ steps, for any e such that

$$r(|\overline{M}|) + \exp(2, e \cdot [q + d \cdot (2|\overline{M}| + g(|\overline{M}|))]) \leq \exp(2, g(|\overline{M}|)).$$

Since $\exp(2, g(n))$ grows super-polynomially and n is $O(g(n))$, we can fix *a priori* two constants e and m such that the previous inequality holds whenever $m \leq |\overline{M}|$; while building the machine $\overline{M}$ we can make sure that $|\overline{M}| > m$ by adding enough dummy states and transition tuples.

This concludes the proof of our statement. It is also clear at this point that considering a polynomial instead of a linear $g(n)$ is worthless, for in either case the above inequality contains and bounds the term $\exp(2, e \cdot |\sigma|)$.

In order to prove the same lower bound for T^c, take e and m as before, modify P_g to output $\alpha_+ \wedge \varphi_{N,g,N}$ for each input N, as before build on M_{T^c} the machine $\overline{M}$ in such a way that $|\overline{M}| > m$ and consider σ to be the sentence $\alpha_+ \wedge \varphi_{\overline{M},g,\overline{M}}$.

Again $\varphi_{\overline{M},g,\overline{M}}$ cannot belong to Q_+, otherwise σ would belong to Q_+ while M_{T^c} accepts σ. Then $\varphi_{\overline{M},g,\overline{M}} \notin Q_+$ and σ is accepted by M_{T^c}, but in time greater than $\exp(2, e \cdot |\sigma|)$. ∎

6. THE INSEPARABILITY FOR ALTERNATING TURING MACHINES

We show in this section how we are going to develop the inseparability result for alternating time and linear alternation (cf. [Ber80]). The proofs have not been written in full detail yet, but

[11] In contrast to what we claim in the introduction of [FY92], this interpretation of the following proof can be applied to the Fischer and Rabin proof as well. We realized that this is possible while verifying that the proof given in [FR74] for the lower bound on the length of derivations in complete axiom systems is indeed constructive—as pointed out to us by Egon Börger, to whom we are grateful for his observation.

we do not think that any difficulty should arise, also because they follow closely the proof in the preceding section—in the same way as the technique used to prove the lower bound in [Ber80] comes directly from [FR74].

First of all the claimed reduction of the decision problem for Presburger Arithmetic to the decision problem of any set T separating Q_+ from the unsatisfiable sentences is derived from lemma 4 of [FR75, p. 76]. A polynomial time linear space deterministic Turing machine can transform any query for S_+ into an equivalent sentence in which all quantifiers are bounded by short formulas representing in Q_+ the necessary triple exponential values. These bounded quantifier sentences are decided by Q_+, so that any query to S_+ can be transformed into a query to T, T being any separating set.

The claimed hereditary lower bound in the sense of [CH90] for any theory T extending Q_+ can be inferred as a by-product of the given non-deterministic Turing machine simulation. The formula $S_k(x, y, z)$ can be used in any model of T to obtain a monadic interpretation of the classes $\mathcal{T}_3^{2^n}$ (cf. [CH90, pp. 60–63]), which implies an ATIME$(\exp(2, cn), cn)$ hereditary lower bound on T.

Let us see in detail the construction for alternating Turing machines corresponding to the one given in the preceding section for non-deterministic Turing machines.

Let M be an alternating Turing machine, with one tape, working alphabet $\Sigma = \{0, 1\}$, set of states $Q = Q^\wedge \uplus Q^\vee$ (divided into universal and existential states) and transition relation δ. The machine starts in an initial state q_0 and computes according to δ, that is a set of five-tuples. Each tuple $\langle q, r, H, r', q'\rangle$ in δ specifies that the machine being in state q and reading the bit r, may (non-deterministically) write the bit r', enter state q' and move the head according to H. We do not restrict M to having only one final state, but we can suppose without loss of generality that the transition relation specifies that when M enters a final state, it does not change its configuration anymore. We can suppose that M does not change its configuration when it enters one for which the transition relation does not specify any possible move. In this way, for each machine configuration C and integer $h \geq 0$, the unique computation tree $\mathcal{T}(C, h)$ of root C and height h—whose nodes are machine configurations and whose edges represent one step reachability according to δ—is such that every leaf has depth h.

The nodes in $\mathcal{T}(C, h)$ can be divided into two distinct sets, the set of accepting and the set of rejecting nodes. We uniquely determine whether a node is accepting or rejecting by starting from the leaves of $\mathcal{T}(C, h)$ and climbing the tree bottom-up by induction. Each leaf is accepting if and only if it is a final configuration, that is one in which M is in a final state. If $h > 0$ then a node $q \in Q^\wedge$ of level $l - 1 \leq h - 1$ is accepting if and only if all its children, whose level is l, are accepting, and a node $q \in Q^\vee$ of level $l - 1$ is accepting if and only if it has an accepting child.

As for the number of alternations in a computation tree, for technical reasons our definition will differ from the classical one. But this gives rise to a wider class of accepted languages, and therefore the lower bound that we prove holds also for the usual measure on alternations. The number of alternations in a computation tree is usually defined (cf. [Ber80] and [BDG90]) as a function of all paths of computation trees. In [Ber80] it is the maximum number of times that in *any path* of the minimum accepting computation tree the machine moves from a state in $Q^\wedge$ to a state in $Q^\vee$ or vice versa. In [BDG90] it is the maximum number of times that in *any path* of *any* accepting computation tree the machine moves from a state in $Q^\wedge$ to a state in $Q^\vee$ or vice versa. In any case, a machine M accepts an input w in time $t(|w|)$ and with at most $a(|w|)$ alternations if and only if wherever there exists an accepting tree with root the initial configuration C_w then $\mathcal{T}(C_w, t(|w|))$ is accepting and it has at most $a(|w|)$ alternations. A language L belongs to ATIME$(t(n), a(n))$ if there exists a machine M that accepts an input w in time $t(|w|)$ and with at most $a(|w|)$ alternations.

We could give an inductive definition of the measure on alternations above. Each leaf of $\mathcal{T}(C, h)$ has alternation depth 0. If $h > 0$ then the alternation depth of each node $q \in Q^\wedge$ of level $l - 1 \leq h - 1$ is the maximum among the alternation depths of its universal children and the alternation depths increased by 1 of its existential children. In the same way the alternation depth of each node $q \in Q^\vee$ of level $l - 1$ is the maximum among the alternation depths of its existential children but with the alternation depths increased by 1 for its universal children. In this way an integer is associated to the root C that coincides with the maximum number of times that in any path of $\mathcal{T}(C, h)$ the machine moves from a state in $Q^\wedge$ to a state in $Q^\vee$ or vice versa.

In our definition the number of alternations in a computation tree depends on the accepting nodes only. The number of alternations of a subtree whose root is a rejecting node is not defined—or it can be taken to be infinite. Each accepting leaf of $\mathcal{T}(C, h)$—that is each leaf corresponding to a final configuration of M—has alternation depth 0. If $h > 0$ then the alternation depth of each accepting node $q \in Q^\wedge$ of level $l - 1 \leq h - 1$ is the maximum among the alternation depths of its universal children and the alternation depths increased by 1 of its existential children. The alternation depth of each accepting node $q \in Q^\vee$ of level $l - 1$ is the minimum among the alternation depths of its accepting existential children but with the alternation depths increased by 1 for its accepting universal children. For any accepting computation tree, the alternation depth measured in this way is less than or equal to the one measured in the classical way. Note that, since M never leaves a final state after reaching it, each final configuration has alternation depth 0 at any level of any computation tree, and if C is the root of some accepting computation tree, then all accepting $\mathcal{T}(C, h)$, $h \geq 0$, do not have necessarily the same alternation depth, but the alternation depth may at most decrease as h increases—while in the standard definition it may only increase, possibly indefinitely.[12]

[12] The reason is that we have defined the number of alternations as being sort of 'the minimum among the alternations strictly needed to accept the input (i.e. the tree root) in the given time.' An analogy may help to clearify this. Suppose for example that you want to define the space used by a non-deterministic Turing machine working in time $t(n)$ to be the minimum among the space used in each accepting path of the computation tree of depth $t(n)$, while it is usually defined to be the maximum space used in any configuration of the computation tree of depth $t(n)$. When more time is allowed, say $2t(n)$, for the same input the depth of the computation tree increases and some path that was not accepting may become accepting. If the machine uses very little space in this path, the space used in the whole non-deterministic computation may decrease. We can say that when the non-deterministic machine has more time to accept the input, it may find 'better' solutions, i.e. solutions that are not tape consuming.

As before, a machine M accepts an input w in time $t(|w|)$ and with at most $a(|w|)$ alternations if and only if in case there exists an accepting tree with root the initial configuration C_w then $\mathcal{T}(C_w, t(|w|))$ is accepting and its alternation depth is at most $a(|w|)$. Define the class $\mathrm{ATIME}'(t(n), a(n))$ as ATIME, but referring to this new definition of alternation depth: a language L belongs to $\mathrm{ATIME}'(t(n), a(n))$ if there exists a machine M that accepts an input w in time $t(|w|)$ and with at most $a(|w|)$ alternations.

If M is a witnessing machine for $L \in \mathrm{ATIME}(t(n), a(n))$ then M accepts L and the number of alternations is bounded by $a(n)$ both if we look at all computation paths or only at some of them. Then L belongs also to $\mathrm{ATIME}'(t(n), a(n))$ and each ATIME' lower bound implies the corresponding ATIME one.

In order to prove the lower bound, we have to prove for alternating machines the following:

Lemma

There exists a constant d and a polynomial $p(n)$ such that for each polynomial $g(n)$, each linear function $a(n)$ and alternating Turing machine M there exists a $p(n+g(n))$-time $O(n+g(n))$-space deterministic Turing machine P_g^M such that for each input w for M, $P_g^M(w)$ is a sentence $\psi_{M,g,a,w}$ such that $|\psi_{M,g,a,w}| \leq d \cdot [|w| + g(|w|) + a(|w|)]$ and

M accepts w within $\exp(2, g(|w|))$ steps and $a(|w|)$ alternations $\iff$
$Q_+ \vdash \psi_{M,g,a,w} \iff$
$Q_+ \not\vdash \neg\psi_{M,g,a,w}$.

proof We are going to define a sentence for each M, w, k and a that says that M accepts w within $\exp(2,k)$ steps and with at most a alternations. The length of this sentence will be linear in $k+a$. Its core is a formula $\psi_k^a[C,j]$ meaning that C is the root of an accepting computation tree $\mathcal{T}(C,j)$ ($j \leq \exp(2,k)$) with at most a alternations.

The construction of $\psi_k^a[C,j]$ is a natural consequence of the definition of acceptance in alternating Turing machines and of the construction of the formula φ_c given in the previous section for non-deterministic machines. A formula $\phi_k(i,A,B)$ similar to φ_c can be defined meaning that A and B are (appropriate snapshots of) two configurations such that B is reachable from A in exactly i steps, through a computation path that contains only states of the same type of the state of A, either universal or existential, except for the state of B, which has to be either a state of the opposite type or a final state. Another similar formula $\phi'_k(i,A,B)$ can be defined that differs from ϕ_k just because it requires that B be a non final state of the same type of A. The definition of acceptance implies that for each $a > 0$ the root of $\mathcal{T}(C,i)$ is accepting with alternation depth less than or equal to a if and only if

- the configuration C specifies that M is in a universal state q_c and for all decompositions $i_1 + i_2 = i$ of i and all configurations B, if $\phi_k(i_1, C, B)$ then $\mathcal{T}(B, i_2)$ is accepting with alternation depth less than or equal to $a-1$, and no configuration B exists such that $\phi'_k(i,C,B)$ holds;

We are not interested here in the class of languages accepted by this machines (which by the way should correspond to the usual one). We introduce this definition of 'number of alternations' just for technical reasons, and in any case we are going to establish the lower bound for the class $\mathrm{ATIME}(t(n), a(n))$ defined in the standard way.

- the configuration C specifies that M is in an existential state q_c and there exists a decomposition $i_1 + i_2 = i$ of i and a configuration B, such that $\phi_k(i_1, C, B)$ and $\mathcal{T}(B, i_2)$ is accepting with alternation depth less than or equal to $a - 1$.

Note that two negative requirements appears in the clause relative to universal states, one of which is hidden by the implication.

The construction of $\psi_k^a[C, j]$ resembles the construction of the formula $E_k^i(x, y, z, u, v, w)$ for the exponentiation function. Like E_k^i, also ψ_k^a has to be defined simultaneously for each k by induction on the second parameter. At each inductive step in the definition of E_k^i we had to refer to multiplication, but we could not use directly the formula $x \underset{k}{\otimes} y = z$ because at each step we may add at most a constant number of symbols to obtain E_k^{i+1} from E_k^i. Instead we included multiplication in the definition of E_k^0, in particular for all i $E_k^i(1, 1, 1, u, v, w)$ is equivalent to $u \underset{k}{\otimes} v = w$. At each inductive step in the definition of E_k^i we used $E_k^i(1, 1, 1, x, y, z)$ instead of $x \underset{k}{\otimes} y = z$, and finally the properties of theories with equality implied that the inductive definition of E_k^{i+1} where several occurrences of E_k^i appear can be substituted with an equivalent definition with only one occurrence of E_k^i, whose length is then linear in k.

In the same way we are going to define $\psi_k^a[C, j]$, including all needed reachability predicates in $\psi_k^a[C, j]$, and building ψ_k^{a+1} using only some occurrences of ψ_k^a, that can be reduced to just one through the technique in [CH90, pp. 15–16] and [FR79, pp. 155–157]—the one we used in the inseparability for programs. If positive and negative occurrences of ψ_k^a appeared in ψ_k^{a+1}, we would be obliged to include the $\leftrightarrow$ symbol in our first order language and in the decision problem definition, obtaining a possibly weaker lower bound[13]. We will use only positive occurrences in order to avoid adding the $\leftrightarrow$ symbol, and we include instead in the definition of ψ_k^0 the negative statements that are needed in the definition inductive step to deal with the universal states.

Using only bounded quantification, we define a formula $\psi_k^a(C, j, q, i, A, B, E, F, G)$ that means the conjunction of:

- $i, j \leq \exp(2, k)$, q is a state, C, A, B, E, G are snapshots of configuration of M of appropriate length (when simulating a machine with a single tape infinite in one direction, it is $C, A, B, E, G \leq \exp(2, k)$)
- C is the root of an accepting computation tree $\mathcal{T}(C, j)$ with at most a alternations
- $\phi_k(i, A, B)$ holds, and q is the current state in A
- $\neg\phi_k(i, E, F)$ holds
- for all snapshots H $\neg\phi_k'(i, G, H)$ holds

Suppose $\sigma_k(A)$ is the formula specifing that A is a snapshot, $\rho_k(q, A)$ a formula meaning that q is the current state in A, and $Q(q), Q^\wedge(q), Q^\vee(q), Q^f(q)$ be appropriate disjunctions meaning that q is a state, a universal one, an existential one or a final one respectively. Then the

---

[13] Even if the Cooper algorithm, expounded in [FR75] and [Opp78], works also for the extended language, we think that the two problems should be kept distinct, for there is no way of eliminating $\leftrightarrow$ from formulas without increasing substantially their length, and the problem with only $\wedge, \vee, \neg, \rightarrow$ could in principle be up to one exponential easier than the one which also includes $\leftrightarrow$. See also the remark in [CH90, p. 16].

inductive definition of ψ_k^a is as follows: the base ψ_k^0 is the formula

$$\begin{aligned}
&i, j \leq \exp(2,k) \wedge \sigma_k(C) \wedge \sigma_k(A) \wedge \sigma_k(B) \wedge \sigma_k(E) \wedge \sigma_k(G) \wedge \rho_k(q, A) \wedge \\
&\phi_k(i, A, B) \wedge \neg\phi_k(i, E, F) \wedge \forall H(\sigma_k(H) \to \neg\phi'_k(i, G, H)) \wedge \\
&\forall q\big[[\rho_k(q) \wedge Q^\vee(q)] \to \exists h \leq j \exists D[Q^f(D) \wedge \phi_k(h, C, D)]\big] \\
&\forall q\big[[\rho_k(q) \wedge Q^\wedge(q)] \to \forall h \leq j \forall D[\neg\phi'_k(j, C, D) \wedge \phi_k(h, C, D) \to Q^f(D)\big].
\end{aligned}$$

The step $\psi_k^{a+1}(C, j, q, i, A, B, E, F, G)$ is the following formula, where each $*$ is a place holder in each instance of ψ_k^a for arguments that do not influence the meaning of the given instance. It replaces either one of the available long expressions $C, j, q, i, A, B, E, F, G$ or a short one, like for example a final state or a short final configuration

$$\begin{aligned}
&\psi_k^a(*, j, q, i, A, B, E, F, G) \wedge \\
&[\psi_k^a(C, j, *, *, *, *, *, *, *) \vee \quad \exists q_c[\psi_k^a(*, *, q_c, 0, C, C, *, *, *) \wedge \\
&\quad [Q^\wedge(q_c) \to [\psi_k^a(*, *, *, j, *, *, *, *, C) \wedge \\
&\qquad \forall h_1 h_2\ (h_1 + h_2 = j \to \forall D\ \{\psi_k^a(*, *, *, h_1, *, *, C, D, *) \vee \psi_k^a(D, h_2, *, *, *, *, *, *, *)\})]] \wedge \\
&\quad [Q^\vee(q_c) \to \\
&\qquad \exists h_1 h_2 \exists D\ (h_1 + h_2 = j \wedge \psi_k^a(*, *, q_c, h_1, C, D, *, *, *) \wedge \psi_k^a(D, h_2, *, *, *, *, *, *, *))]].
\end{aligned}$$

All quantifiers are bounded. In particular in the expression $\forall D\ \{\psi_k^a(*, *, *, h_1, *, *, C, D, *) \vee \psi_k^a(D, h_2, *, *, *, *, *, *, *)\}$ D is bounded because the matrix is equivalent to

$$\psi_k^a(*, *, *, h_1, C, D, *, *, *) \to \psi_k^a(D, h_2, *, *, *, *, *, *, *).$$

The sentence $\psi_{M,g,a,w}$ is easily constructed from $\psi_k^a[C, j]$. ∎

In the same way as we proved the inseparability for non-deterministic Turing machines we can then prove the following:

<u>Theorem</u>*(Inseparability for Alternating Turing Machines)*

Let $T \subset \Sigma^*$ be a set of strings, in the alphabet $\Sigma \supseteq \Sigma_+$, that separates Q_+ from $\mathrm{Uns}\,\mathcal{L}_+$. *Then T is $\leq_{\mathrm{p-lin}}$-hard for each class $\mathrm{ATIME}(\exp(2, c \cdot n), c \cdot n)$ and is $\leq_{\mathrm{p}}$-hard for each class $\mathrm{ATIME}(\exp(2, n^c), n^c)$, and there exists a c_0 such that $T \notin \mathrm{ATIME}(\exp(2, c_0 \cdot n), c_0 \cdot n)$:*

$$\begin{aligned}
&\bigcup_{0<c} \mathrm{ATIME}(\exp(2, c \cdot n), c \cdot n) \leq_{\mathrm{p-lin}} T \quad \text{and} \\
&\bigcup_{0<c} \mathrm{ATIME}(\exp(2, n^c), n^c) \leq_{\mathrm{p}} T \quad \text{and} \\
&\text{exists} \quad c_0 \quad \text{s.t.} \quad T \notin \mathrm{ATIME}(\exp(2, c_0 \cdot n), c_0 \cdot n).
\end{aligned}$$

7. CONCLUDING REMARKS

We have shown how in the additive fragment Q_+ of the Robinson axiom system Q a sequence of formulas $x \underset{k}{\otimes} y = z$ can be defined that represents multiplication in large increasing chunks of the integers. Using this construction, we have proved a double exponential time with linear alternations inseparability result, which subsumes both the hereditary lower bound in [CH90] and the single exponential inseparability in [You85].

We have gained a substantial improvement on the known lower bounds for arithmetic theories.

The proven inseparability is stronger than the inseparability in [You85] because the lower bound is higher, the inseparability wider—since Q_+ is a proper subset of ADDAX—and the computational model more general—alternating versus non-deterministic Turing machines.

It is also strictly stronger than the hereditary lower bound in [CH90] because it implies double exponential hardness for *any set* of satisfiable sentences which *contains* Q_+ and for any theory contained in Q_+ as well.

The hereditary lower bound in [CH90] in its stronger original form is equivalent to the double exponential inseparability of the *complete* theory S_+, which contains Q_+, from the unsatisfiable sentences, while the inseparability for Q_+ implies in particular that *each* complete theory extending Q_+, S_+ in particular, is double exponentially inseparable from the unsatisfiable sentences.

The techniques used here also yield interesting insights into the techniques for proving lower bounds presented in [FR74], and gives a fully detailed application of the technique in [You85].

In particular, in order to simplify the construction of $x \underset{k}{\otimes} y = z$, we have verified that there is no need to refer to the function $g(k)$ representing the product of all prime numbers less than $\exp(2, k)$. Instead we can refer to the least common multiple $\Lambda(k) >> g(k)$ of all integers less than $\exp(2, k)$. This gives rise to a definition of $x \underset{k}{\otimes} y = z$ that is shorter than the one in [FR74] and the one suggested in [HU79, p. 371] and [MY78, pp. 200–1]. Also we have verified that a lower bound of $\Omega(\sqrt{n})$ on the number $\pi(n)$ of primes less than n is sufficient and that even the Chebychev lower bound $\Omega(n/\log n)$ is not strictly necessary.

The restriction to prime numbers in the construction given in [FR74] may seem dictated by the requirements of the hypotheses of the Chinese Remainder Theorem. This theorem would guarantee the existence of a solution of the system of $\pi(\exp(2, k))$ equations involved in the definition of the sequence of formulas $\mathrm{Pr}_k(x, y, z)$ which in [FR74] correspond to $x \underset{k}{\otimes} y = z$. We have verified in the construction of $x \underset{k}{\otimes} y = z$ that the Chinese Remainder Theorem is *not* necessary to guarantee the existence of a solution, which is immediately derived in this particular case by the construction of the system of equations—and that the same holds for the construction of $\mathrm{Pr}_k(x, y, z)$ in [FR74]. It is also obvious from the construction that two solutions of the said systems of equations must differ by at least $\Lambda(k) >> g(k) >> \exp(3, k)$. The uniqueness of the solution is guaranteed without referring to the Chinese Remainder Theorem.

Based on the construction of the $\exp(3, k)$-representation $x \underset{k}{\otimes} y = z$ we have given several proofs of substantially the same inseparability result.

On the line of [You85] we have proven in detail that the theory ADDAX is doubly exponentially inseparable from the unsatisfiable sentences when the computational model is taken to be the min-program and the complexity measure is based on the multiplication performed during the computation. We have also shown in detail how this inseparability result implies the one referring to standard non-deterministic double exponential time, as described in [You85].

We have then shown how the whole construction in [FR74] can be done with a theory that is not complete. A sequence of formulas may be used to represent a function in an incomplete theory, if we are able to prove that each element of the sequence is equivalent to the corresponding disjunction of bounded arguments and corresponding values, as is the case e.g. of $x \underset{k}{\otimes} y = z$:

$$Q_+ \vdash \forall xyz \; x \underset{k}{\otimes} y = z \quad \leftrightarrow \quad \bigvee_{n,m \leq m_{k+4}} [x = \overline{n} \wedge y = \overline{m} \wedge z = \overline{n \cdot m}].$$

A similar work can be done in the theory Q_+ for all the predicates and the functions which are needed in the standard simulation of a Turing machine.

If the theory under examination has bounded quantifier elimination, then the Turing machine simulation, which in any case involves only bounded quantifiers, provides us with a sentence $\varphi_{M,T,w}$ that is decided by the theory and is provable exactly when M accepts the input w in time $T(|w|)$. Based on this easily constructible sentence the inseparability result can be derived, in case the theory be a finitely axiomatizable one. We have also shown how the whole proof can be intepreted as the construction of a self-referencing provable sentence meaning: *"there is no short accepting computation of the given machine on input myself."*

Finally we have given a proof of double exponential alternating time with linear alternations inseparability for Q_+ that do not make any non-dischargeable use of the '$\leftrightarrow$' logical connective and applies to the problem in its most general setting.

8. ACKNOWLEDGMENTS

The first author learned the simple proof of the lower bound on the number of primes during a class given by Paul Beame at the University of Washington. Egon Börger pointed out that the proof of the lower bound on derivations in [FR74] is constructive. Lavinia Egidi wasted two days of her youth reading this report, and did not even complain for being included in the acknowledgments.

9. REFERENCES

[BDG90] José Luis Balcázar, Josep Diaz, and Joaquim Gabarró *"Structural Complexity II"* Springer-Verlag, 1990.

[Ber80] Leonard Berman "The Complexity of Logical Theories" *Theoretical Computer Science*, 11:71–77, 1980.

[BHC86] P. W. Beame, H. J. Hoover, and S. A. Cook "Log depth circuits for division and related problems" *SIAM Journal on Computing*, 15(4):994–1003, November 1986. A preliminary version of these results appeared in the 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 1–6.

[BS69] J.L. Bell and A.B. Slomson *"Models and Ultraproducts"* North-Holland, 1969.

[CH90] Kevin J. Compton and C. Ward Henson "A uniform method for proving lower bounds on the computational complexity of logical theories" *Annals of Pure and Applied Logic*, 48:1–79, 1990.

[Dek79] Mikhail I. Dekhtjar "Complexity spectra of recursive sets and approximability of initial segments of complete problems" *Elektronische Informationsverarbeitung und Kybernetik*, 15:11–32, 1979.

[Fag93] G. Faglia "*Double exponential inseparability of theories of addition*" PhD thesis, Dottorato di Ricerca in Informatica, Università di Torino e Milano, Via Comelico 39, I-20135 Milano MI, Italy, February 1993. In Italian. In preparation.

[FR74] M. J. Fischer and M. Rabin "Super-Exponential Complexity of Presburger Arithmetic" In *Complexity of Computation*, pages 27–42. SIAM-AMS, 1974.

[FR75] Jeanne Ferrante and Charles Rackoff "A decision Procedure for the First Order Theory of Real Addition with Order" *SIAM Journal of Computing*, 4:69–76, 1975.

[FR79] Jeanne Ferrante and Charles Rackoff "*The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*" Springer, 1979.

[Für82] Martin Fürer "The complexity of Presburger arithmetic with bounded quantifier alternation depth" *Theoretical Computer Science*, 18:105–111, 1982.

[FY92] G. Faglia and P. Young "On The Meaning Of Essentially Unprovable Theorems In The Presburger Theory Of Addition" In *Theoretical Computer Science – Proceedings of the 4th Italian Conference*, pages 214–228. World Scientific Publishing Co. Pte. Ltd., 1992.

[Grä88] Erich Grädel "Subclasses of Presburger arithmetic and the polynomial-time hierarchy" *Theoretical Computer Science*, 56:289–301, 1988.

[Grä89] Erich Grädel "Dominoes and the complexity of subclasses of logical theories" *Annals of Pure and Applied Logic*, 43:1–30, 1989.

[HB34] D. Hilbert and P. Bernays "*Grundlagen der Mathematik*, volume 1" Springer, 1934.

[Hei79] Joos Heintz "Definability bounds of first order theories of algebraically closed fields" In *Fundamentals of Computation Theory. Proceedings of the Conference on Algebraic, Arithmetical and Categorical Methods in Computation Theory*, volume 2 of *Math. Res.*, pages 160–166. Akademie-Verlag, Berlin, 1979.

[HU79] John E. Hopcroft and Jeffrey D. Ullman "*Introduction to Automata Theory, Languages, and Computation*" Addison Wesley, 1979.

[Men87] Elliott Mendelson "*Introduction to Mathematical Logic*" Mathematics Series. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, third edition, 1987.

[MY78] M. Machtey and P. Young "*An Introduction to the General Theory of Algorithms*" Elsevier Science Publishing Co., Inc., 1978.

[Opp78] Derek C. Oppen "A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic" *Journal of Computer and System Sciences*, 16:323–332, 1978.

[Pre29] M. Presburger "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt" In *Comptes-Rendus du ler Congrès des Mathématiciens des Pays Slavs*, 1929.

[RL78] C. R. Reddy and D. W. Loveland "Presburger arithmetic with bounded quantifier alternation" In *Tenth Annual ACM Symposium on Theory of Computing*, pages 320–325. ACM, 1978.

[Rob50] Raphael M. Robinson "An Essentially Undecidable Axiom System" In *Proceedings of the International Congress of Mathematicians. Cambridge,* **1**, pages 729–30, 1950.

[Sca84] Bruno Scarpellini "Complexity of subcases of Presburger arithmetic" *Transactions of the American Mathematical Society*, 284:203–218, 1984.

[Sch82] Arnold Schonhage "Random access machines and Presburger arithmetic" In *Logic and algorithmic*, volume 30 of *Monographies de L'Enseignement Mathematique*, pages 353–

363. Universitè de Genève, L'Enseignement Mathematique, 1982.

[TMR53] A. Tarski, A. Mostowsky, and R. Robinson "*Undecidable Theories*" North-Holland, 1953.

[Woh79] Kai Wohl "Zur Komplexität der Presburger Arithmetik und des Äquivalenzproblems einfacher Programme" In *Theoretical Computer Science (Fourth GI Conference)*, volume 67 of *Lecture Notes in Computer Science*, pages 310–318. Springer, 1979. In German.

[You85] Paul Young "Gödel Theorems, Exponential Difficulty and Undecidability of Arithmetic Theories: An Exposition" In *Proceedings of Symposia in Pure Mathematics*, pages 503–522. American Mathematical Society, 1985.

Giovanni Faglia
Dipartimento di Scienze dell'Informazione
Università di Milano
via Comelico 39
20135 Milano, Italy
e-mail: faglia@ghost.dsi.unimi.it

Paul Young
Dept. of Computer Science & Engineering
University of Washington
FR-35
Seattle, WA, 98195 USA
e-mail: young@cs.washington.edu