Instruction sequences expressing multiplication algorithms

Bergstra, J.A.; Middelburg, C.A.

Instruction Sequences Expressing Multiplication Algorithms

J.A. BERGSTRA[1], C.A. MIDDELBURG[1]

Abstract

For each function on bit strings, its restriction to bit strings of any given length can be computed by a finite instruction sequence that contains only instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction. We describe instruction sequences of this kind that compute the function on bit strings that models multiplication on natural numbers less than 2^N with respect to their binary representation by bit strings of length N, for a fixed but arbitrary $N > 0$, according to the long multiplication algorithm and the Karatsuba multiplication algorithm. One of the results obtained is that the instruction sequence expressing the former algorithm is longer than the one expressing the latter algorithm only if the length of the bit strings involved is greater than 2^8. We also go into the use of an instruction sequence with backward jump instructions for expressing the long multiplication algorithm. This leads to an instruction sequence that it is shorter than the other two if the length of the bit strings involved is greater than 2.

Keywords: bit string function, single-pass instruction sequence, backward jump instruction, long multiplication algorithm, Karatsuba multiplication algorithm, halting problem.

1 Introduction

This paper belongs to a line of research in which issues relating to various subjects from computer science, including programming language expressiveness,

---
[1]Informatics Institute, Faculty of Science, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands, email: {J.A.Bergstra,C.A.Middelburg}@uva.nl.

computability, computational complexity, algorithm efficiency, algorithmic equivalence of programs, program verification, program performance, program compactness, and program parallelization, are rigorously investigated thinking in terms of instruction sequences. An enumeration of most papers belonging to this line of research is available at [11]. The work on computational complexity presented in [4] and the work on algorithmic equivalence of programs presented in [5] were prompted by the fact that, for each function on bit strings, its restriction to bit strings of any given length can be computed by a finite instruction sequence that contains only instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction.

This fact also incited us to look for finite instruction sequences containing only the above-mentioned instructions that compute a well-known function on bit strings of a given length. Earlier, we did so taking the hash function SHA-256 from the Secure Hash Standard [16] as the well-known function on bit strings. In the current paper, we do so taking the function that models multiplication on natural numbers less than 2^N with respect to their binary representation by bit strings of length N, for a fixed but arbitrary $N > 0$, as the well-known function on bit strings.

We describe finite instruction sequences containing only the above-mentioned instructions that compute this function according to the standard multiplication algorithm, which is known as the long multiplication algorithm, and according to the Karatsuba multiplication algorithm [9, 10]. We calculate the exact size of the instruction sequence expressing the long multiplication algorithm and lower and upper estimates for the size of the instruction sequence expressing the Karatsuba multiplication algorithm. One of the results following from the calculated sizes is that the instruction sequence expressing the former algorithm is longer than the instruction sequence expressing the latter algorithm only if the length of the bit strings involved is greater than 2^8.

We also go into the use of an instruction sequence with backward jump instructions for expressing the long multiplication algorithm. We describe a finite instruction sequence containing a backward jump instruction, in addition to the above-mentioned instructions, that expresses a minor variant of the long multiplication algorithm. We calculate the exact size of this instruction sequence and find that it is shorter than the other two if the length of the bit strings involved is greater than 2. In addition, we argue that the instruction sequences expressing the long multiplication algorithm

form a hard witness of the inevitable existence of a halting problem in the practice of imperative programming.

The Karatsuba multiplication algorithm was devised by Karatsuba in 1962 to disprove the conjecture made by Kolmogorov that any algorithm to compute the function that models multiplication on natural numbers with respect to their representations in the binary number system has time complexity $\Omega(n^2)$. Shortly afterwards, this divide-and-conquer algorithm was generalized by Toom and Cook [7, 13]. Later, asymptotically faster multiplication algorithms, based on fast Fourier transforms, were devised by Schönhage and Strassen [12] and Fürer [8]. To our knowledge, except for the Schönhage-Strassen algorithm, only informal (natural language or pseudo code) descriptions of these multiplication algorithms are available. In this paper, we provide a mathematically precise alternative to the informal descriptions of the Karatsuba multiplication algorithm, using terms from an algebraic theory of single-pass instruction sequences defined in [1].

It is customary that computing practitioners phrase their explanations of issues concerning programs from an empirical perspective such as the perspective that a program is in essence an instruction sequence. An attempt to approach the semantics of programming languages from this perspective is made in [1]. The groundwork for the approach is an algebraic theory of single-pass instruction sequences, called program algebra, and an algebraic theory of mathematical objects that represent the behaviours produced by instruction sequences under execution, called basic thread algebra.[2] The line of research referred to at the beginning of this introduction originates from the above-mentioned work on an approach to programming language semantics.

The general aim of this line of research is to bring instruction sequences as a theme in computer science better into the picture. This is the general aim of the work presented in the current paper as well. However, different from usual in the work referred to above, the accent is this time mainly on a practical problem, namely the problem to devise instruction sequences that express the long multiplication algorithm and the Karatsuba multiplication algorithm. As in the work referred to above, the work presented in the current paper is carried out in the setting of program algebra.

This paper is organized as follows. First, we survey program algebra and the particular fragment and instantiation of it that is used in this paper

---

[2]In [1], basic thread algebra is introduced under the name basic polarized process algebra.

(Section 2) and sketch the Karatsuba multiplication algorithm (Section 3). Next, we describe how we deal with n-bit words by means of Boolean registers (Section 4) and how we compute the operations on n-bit words that are used in the long multiplication algorithm and/or the Karatsuba multiplication algorithm (Section 5). Then, we describe and analyze instruction sequences that express these algorithms (Section 6). After this, we go into the use of an instruction sequence with backward jump instructions for expressing the long multiplication algorithm (Sections 7) and relate the findings to the halting problem (Section 8). Finally, we make some concluding remarks (Section 9).

We rely in this paper on an intuitive understanding of what is an algorithm and when an instruction sequence expresses an algorithm. A rigorous study of these issues and related ones, carried out in the same setting as the work presented in this paper, is presented in [5].

The preliminaries to the work presented in this paper are a selection from the preliminaries to the work presented in [4]. For this reason, there is some text overlap with those papers. The preliminaries concern program algebra. We only give a brief summary of program algebra. A comprehensive introduction, including examples, can be found in [3].

2   Program Algebra

In this section, we present a brief outline of PGA (ProGram Algebra) and the particular fragment and instantiation of it that is used in the remainder of this paper. A mathematically precise treatment can be found in [4].

The starting-point of PGA is the simple and appealing perception of a sequential program as a single-pass instruction sequence, i.e. a finite or infinite sequence of instructions of which each instruction is executed at most once and can be dropped after it has been executed or jumped over.

It is assumed that a fixed but arbitrary set $\mathfrak{A}$ of *basic instructions* has been given. The intuition is that the execution of a basic instruction may modify a state and produces a reply at its completion. The possible replies are 0 and 1. The actual reply is generally state-dependent. Therefore, successive executions of the same basic instruction may produce different replies. The set $\mathfrak{A}$ is the basis for the set of instructions that may occur in the instruction sequences considered in PGA. The elements of the latter set are called *primitive instructions*. There are five kinds of primitive instructions, which are listed below:

- for each $a \in \mathfrak{A}$, a *plain basic instruction* a;

- for each $a \in \mathfrak{A}$, a *positive test instruction* $+a$;

- for each $a \in \mathfrak{A}$, a *negative test instruction* $-a$;

- for each $l \in \mathbb{N}$, a *forward jump instruction* $\#l$;

- a *termination instruction* !.

We write $\mathfrak{I}$ for the set of all primitive instructions.

On execution of an instruction sequence, these primitive instructions have the following effects:

- the effect of a positive test instruction $+a$ is that basic instruction a is executed and execution proceeds with the next primitive instruction if 1 is produced and otherwise the next primitive instruction is skipped and execution proceeds with the primitive instruction following the skipped one — if there is no primitive instruction to proceed with, inaction occurs;

- the effect of a negative test instruction $-a$ is the same as the effect of $+a$, but with the role of the value produced reversed;

- the effect of a plain basic instruction a is the same as the effect of $+a$, but execution always proceeds as if 1 is produced;

- the effect of a forward jump instruction $\#l$ is that execution proceeds with the lth next primitive instruction of the instruction sequence concerned — if l equals 0 or there is no primitive instruction to proceed with, inaction occurs;

- the effect of the termination instruction ! is that execution terminates.

To build terms, PGA has a constant for each primitive instruction and two operators. These operators are: the binary concatenation operator $;$ and the unary repetition operator $^{\omega}$. We use the notation $\overset{\bullet}{\underset{i=0}{;}}^{n} P_i$, where $P_0, \ldots, P_n$ are PGA terms, for the PGA term $P_0 ; \ldots ; P_n$. We also use the notation P^n. For each PGA term P and $n > 0$, P^n is the PGA term defined by induction on n as follows: $P^1 = P$ and $P^{n+1} = P ; P^n$.

The instruction sequences that concern us in the remainder of this paper are the finite ones, i.e. the ones that can be denoted by closed PGA terms in which the repetition operator does not occur. Moreover, the basic

instructions that concern us are instructions to set and get the content of Boolean registers. More precisely, we take the set

$$\{\mathsf{in}{:}i.\mathsf{get} \mid i \in \mathbb{N}^+\} \cup \{\mathsf{out}{:}i.\mathsf{set}{:}b \mid i \in \mathbb{N}^+ \land b \in \{0,1\}\}$$
$$\cup \{\mathsf{aux}{:}i.\mathsf{get} \mid i \in \mathbb{N}^+\} \cup \{\mathsf{aux}{:}i.\mathsf{set}{:}b \mid i \in \mathbb{N}^+ \land b \in \{0,1\}\}$$

as the set $\mathfrak{A}$ of basic instructions.

Each basic instruction consists of two parts separated by a dot. The part on the left-hand side of the dot plays the role of the name of a Boolean register and the part on the right-hand side of the dot plays the role of a command to be carried out on the named Boolean register. For each $i \in \mathbb{N}^+$:

- $\mathsf{in}{:}i$ serves as the name of the Boolean register that is used as ith input register in instruction sequences;

- $\mathsf{out}{:}i$ serves as the name of the Boolean register that is used as ith output register in instruction sequences;

- $\mathsf{aux}{:}i$ serves as the name of the Boolean register that is used as ith auxiliary register in instruction sequences.

On execution of a basic instruction, the commands have the following effects:

- the effect of get is that nothing changes and the reply is the content of the named Boolean register;

- the effect of $\mathsf{set}{:}0$ is that the content of the named Boolean register becomes 0 and the reply is 0;

- the effect of $\mathsf{set}{:}1$ is that the content of the named Boolean register becomes 1 and the reply is 1.

Let $n, m \in \mathbb{N}$, let $f \colon \{0,1\}^n \to \{0,1\}^m$, and let X be a finite instruction sequence that can be denoted by a closed PGA term in the case that $\mathfrak{A}$ is taken as specified above. Then X *computes* f if there exists a $k \in \mathbb{N}$ such that for all $b_1, \ldots, b_n \in \{0,1\}$: if X is executed in an environment with n input registers, m output registers, and k auxiliary registers, the content of the input registers with names $\mathsf{in}{:}1, \ldots, \mathsf{in}{:}n$ are $b_1, \ldots, b_n$ when execution starts, and the content of the output registers with names $\mathsf{out}{:}1, \ldots, \mathsf{out}{:}m$ are $b'_1, \ldots, b'_m$ when execution terminates, then $f(b_1, \ldots, b_n) = b'_1, \ldots, b'_m$.

3   Sketch of Karatsuba Multiplication Algorithm

Suppose that x and y are two natural numbers with a binary representation of n bits. As a first step toward multiplying x and y, split each of these representations into a left part of length $\lfloor n/2 \rfloor$ and a right part of length $\lceil n/2 \rceil$. Let us say that the left and right part of the representation of x represent natural numbers x_L and x_R and the left and right part of the representation of y represent natural numbers y_L and y_R. It is obvious that $x = 2^{\lceil n/2 \rceil} \cdot x_L + x_R$ and $y = 2^{\lceil n/2 \rceil} \cdot y_L + y_R$. From this it follows immediately that

$$x \cdot y = 2^{2 \cdot \lceil n/2 \rceil} \cdot (x_L \cdot y_L) + 2^{\lceil n/2 \rceil} \cdot (x_L \cdot y_R + x_R \cdot y_L) + x_R \cdot y_R \ .$$

In addition to this, it is known that

$$x_L \cdot y_R + x_R \cdot y_L = (x_L + x_R) \cdot (y_L + y_R) - x_L \cdot y_L - x_R \cdot y_R \ .$$

Moreover, it is easy to see that multiplications by powers of 2 are merely bit shifts on the binary representation of the natural numbers involved. All this means that, on the binary representations of x and y, the multiplication $x \cdot y$ can be replaced by three multiplications: $x_L \cdot y_L$, $x_R \cdot y_R$, and $(x_L + x_R) \cdot (y_L + y_R)$. These three multiplications concern natural numbers with binary representations of length $\lfloor n/2 \rfloor$, $\lceil n/2 \rceil$, and $\lceil n/2 \rceil + 1$, respectively. For each of these multiplications it holds that, if the binary representation length concerned is greater than 3, the multiplication can be replaced by three multiplications of natural numbers with binary representations of even shorter length.

The *Karatsuba multiplication algorithm* is the algorithm that computes the binary representation of the product of two natural numbers with binary representations of the same length by dividing the computation into the computation of the binary representations of three products as indicated above and doing so recursively until it not any more leads to further length reduction. The remaining products are usually computed according to the standard multiplication algorithm, which is known as the long multiplication algorithm.

Both the Karatsuba multiplication algorithm and the long multiplication algorithm can actually be applied to natural numbers represented in the binary number system as well as natural numbers represented in the decimal number system. The long multiplication algorithm is the multiplication algorithm that is taught in schools for computing the product of natural

numbers represented in the decimal number system. It is known that the long multiplication algorithm has uniform time complexity $\Theta(n^2)$ and the the Karatsuba multiplication algorithm has uniform time complexity $\Theta(n^{\log_2(3)}) = \Theta(n^{1,5849\ldots})$, so the Karatsuba multiplication algorithm is asymptotically faster than the long multiplication algorithm.

4 Dealing with n-Bit Words

This section is concerned with dealing with bit strings of length n by means of Boolean registers. It contains definitions which facilitate the description of instruction sequences that express the long multiplication algorithm and the Karatsuba multiplication algorithm.

Henceforth, it is assumed that a fixed but arbitrary positive natural number N has been given. The above-mentioned algorithms compute the binary representation of the product of two natural numbers represented by bit strings of the same length. In Section 6, the instruction sequences expressing these algorithms will be described for the case where this length is N.

In the sequel, bit strings of length n will mostly be called *n-bit words*. The prefix "n-bit" is left out if n is irrelevant or clear from the context.

Let $\kappa{:}i$ ($\kappa \in \{\mathsf{in}, \mathsf{out}, \mathsf{aux}\}$, $i \in \mathbb{N}^+$) be the name of a Boolean register. Then κ and i are called the *kind* and *number* of the Boolean register. Successive Boolean registers are Boolean registers of the same kind with successive numbers. Words are stored by means of Boolean registers such that the successive bits of a stored word are the contents of successive Boolean registers.

Henceforth, the name of a Boolean register will mostly be used to refer to the Boolean register in which the least significant bit of a word is stored. Let $\kappa{:}i$ and $\kappa'{:}i'$ be the names of Boolean registers and let $n \in \mathbb{N}^+$. Then we say that $\kappa{:}i$ and $\kappa'{:}i'$ *lead to partially coinciding n-bit words* if $k = k'$ and $0 < |i - i'| < n$.

The N-bit words representing the two natural numbers for which the binary representation of their product is to be computed are stored in advance of the computation in input registers, starting with the input register with number 1. It is convenient to have available the names I_1 and I_2 for the input registers in which the least significant bit of these words are stored. The $2N$-bit word representing the product is stored just before the end of the computation in output registers, starting with the output register with number 1. It is convenient to have available the name O for the

output register in which the least significant bit of this word is stored. The words representing intermediate values that arise during the computation are temporarily stored in auxiliary registers, starting with the auxiliary register with number 1.

In the case of the Karatsuba algorithm, the binary representation of the product of two natural numbers with binary representations of the same length is computed by dividing the computation into the computation of the binary representations of three products and doing so recursively until it not any more leads to further length reduction. Therefore, it is convenient to have available, for sufficiently many natural numbers i, the names I_1^i, I_2^i and O^i for the auxiliary registers in which the least significant bit of the binary representations of smaller natural numbers and their product are stored. Because at each level of recursion, except the last level, the computation of the binary representation of a product involves the computation of the binary representations of three products at the next level, it is convenient to have available, for sufficiently many natural numbers i, the names P_1^i, P_2^i and P_3^i for the auxiliary registers in which the least significant bit of these binary representations of products are stored.

It is also convenient to have available the names S_1, S_2, T_1, T_2 for the auxiliary registers in which the least significant bit of the words that represent the intermediate values that arise, other than the ones mentioned in the previous paragraph, are stored. Moreover, it is convenient to have available the name c for the auxiliary register that contains the carry/borrow bit that is repeatedly stored when computing the operations that model addition and subtraction on natural numbers with respect to their binary representation.

Therefore, we define:

$$
\begin{aligned}
I_1 &\triangleq \mathsf{in}{:}1, \\
I_2 &\triangleq \mathsf{in}{:}k \quad \text{where } k = N + 1, \\
O &\triangleq \mathsf{out}{:}1, \\
c &\triangleq \mathsf{aux}{:}1, \\
S_1 &\triangleq \mathsf{aux}{:}2, \\
S_2 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 2 \cdot N + 2, \\
T_1 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 4 \cdot N + 2, \\
T_2 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 6 \cdot N + 2,
\end{aligned}
$$

$$I_1^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 8 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),$$

$$I_2^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 9 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),$$

$$O^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 10 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),$$

$$P_1^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 12 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),$$

$$P_2^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 14 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),$$

$$P_3^i \triangleq \mathsf{aux}{:}k \ \text{ where } k = 10 \cdot N \cdot i + 16 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil).$$

Here i ranges over natural numbers in the interval with lower endpoint 0 and upper endpoint $\lceil \log_2(N-2) \rceil$. This needs some explanation.

Proposition 1 *The recursion depth of the Karatsuba multiplication algorithm applied to bit strings of length N is $\lceil \log_2(N-2) \rceil$.*

Proof: Let $n \le N$. In the Karatsuba multiplication algorithm, the binary representation of the product of two natural numbers with binary representations of length n is computed by dividing the computation into the computation of the binary representation of a product of two natural numbers with binary representations of length $\lfloor n/2 \rfloor$, the binary representation of a product of two natural numbers with binary representations of length $\lceil n/2 \rceil$, and the binary representation of a product of two natural numbers with binary representations of length $\lceil n/2 \rceil + 1$. The function f defined by $f(n) \triangleq \lceil n/2 \rceil + 1$ has the following properties: (a) $f(n) < n$ iff $n > 3$; and (b) for $n > 3$, the least m such that $f^m(n) = 3$ is $\lceil \log_2(n-2) \rceil$. This implies that the recursion depth is $\lceil \log_2(N-2) \rceil$. $\square$

Proposition 1 tells us that the maximum level of recursion that can be reached is $\lceil \log_2(N-2) \rceil$. So there are $\lceil \log_2(N-2) \rceil + 1$ possible levels of recursion, viz. $0, \ldots, \lceil \log_2(N-2) \rceil$. This means that there are sufficiently many natural numbers i for which the names I_1^i, I_2^i, O^i, P_1^i, P_2^i, and P_3^i have been introduced above. In Section 6, we will use the names I_1^i, I_2^i, O^i, P_1^i, P_2^i, and P_3^i at the level of recursion $\lceil \log_2(N-2) \rceil - i$.

5   Computing Operations on n-Bit Words

This section is concerned with computing operations on bit strings of length n. It contains definitions which facilitate the description of instruction sequences that express the long multiplication algorithm and the Karatsuba multiplication algorithm.

In this section, we will write $\beta\beta'$, where β and β' are bit strings, for the concatenation of β and β'. In other words, we will use juxtaposition for concatenation. Moreover, we will use the bit string notation b^n. For $n > 0$, the bit string b^n, where $b \in \{0, 1\}$, is defined by induction on n as follows: $b^1 = b$ and $b^{n+1} = b\,b^n$.

The basic operations on words that are relevant to the long multiplication algorithm and/or the Karatsuba multiplication algorithm are the operations that model addition, subtraction, and multiplication by 2^m, modulo 2^n, on natural numbers less than 2^n, with respect to their binary representation by n-bit words ($0 < n \leq N$, $0 < m < n$). The operation modeling multiplication by 2^m is commonly known as "shift left by m positions". For these operations, we define parameterized instruction sequences computing them in case the parameters are properly instantiated (see below):

$ADD_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq$

    $c.\mathsf{set}{:}0 \,;$

    ${\overset{\bullet}{;}}_{i=0}^{n-1}(+s_1{:}k_1{+}i.\mathsf{get}\,;\#8\,;+s_2{:}k_2{+}i.\mathsf{get}\,;\#8\,;-c.\mathsf{get}\,;\#14\,;$

        $d{:}l{+}i.\mathsf{set}{:}1\,;c.\mathsf{set}{:}0\,;\#13\,;+s_2{:}k_2{+}i.\mathsf{get}\,;\#4\,;+c.\mathsf{get}\,;\#7\,;\#7\,;$

        $+c.\mathsf{get}\,;\#5\,;d{:}l{+}i.\mathsf{set}{:}0\,;c.\mathsf{set}{:}1\,;\#3\,;+d{:}l{+}i.\mathsf{set}{:}0\,;d{:}l{+}i.\mathsf{set}{:}1)\,,$

$SUB_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq$

    $c.\mathsf{set}{:}0 \,;$

    ${\overset{\bullet}{;}}_{i=0}^{n-1}(-s_1{:}k_1{+}i.\mathsf{get}\,;\#8\,;+s_2{:}k_2{+}i.\mathsf{get}\,;\#8\,;-c.\mathsf{get}\,;\#14\,;$

        $d{:}l{+}i.\mathsf{set}{:}0\,;c.\mathsf{set}{:}0\,;\#13\,;+s_2{:}k_2{+}i.\mathsf{get}\,;\#4\,;+c.\mathsf{get}\,;\#7\,;\#7\,;$

        $+c.\mathsf{get}\,;\#5\,;d{:}l{+}i.\mathsf{set}{:}1\,;c.\mathsf{set}{:}1\,;\#3\,;-d{:}l{+}i.\mathsf{set}{:}1\,;d{:}l{+}i.\mathsf{set}{:}0)\,,$

$SHL_n^m(s{:}k, d{:}l) \triangleq$

    ${\overset{\bullet}{;}}_{i=0}^{n-1-m}(+s{:}k{+}n{-}1{-}m{-}i.\mathsf{get}\,;-d{:}l{+}n{-}1{-}i.\mathsf{set}{:}1\,;d{:}l{+}n{-}1{-}i.\mathsf{set}{:}0)\,;$

    ${\overset{\bullet}{;}}_{i=0}^{m-1}(d{:}l{+}m{-}1{-}i.\mathsf{set}{:}0)\,,$

where s, s_1, s_2 range over $\{\mathsf{in}, \mathsf{aux}\}$, d ranges over $\{\mathsf{aux}, \mathsf{out}\}$, and k, k_1, k_2, l range over $\mathbb{N}^+$. For each of these parameterized instruction sequences, all but the last parameter correspond to the operands of the operation concerned and the last parameter corresponds to the result of the operation concerned. The intended operations are computed provided that the instantiation of the last parameter and the instantiation of none of the other parameters lead to partially coinciding n-bit words. In this paper, this condition will always be satisfied.

In the case of addition and subtraction, the intended operation is computed according to the long addition algorithm and the long subtraction algorithm, respectively. There are many instruction sequences expressing these algorithms. The ones defined above are at present the shortest ones that we could devise.

From now on, if we state that a function on bit strings of length n models a function on natural numbers less than 2^n, it is implicit that it does so with respect to the binary representation of these numbers by n-bit words.

Proposition 2 *Let $n, m \in \mathbb{N}$ be such that $0 < n \leq N$ and $0 < m < n$. Then the function on bit strings of length n computed by*

1. $ADD_n(I_1, I_2, O)$; ! *models addition modulo 2^n on natural numbers less than 2^n;*

2. $SUB_n(I_1, I_2, O)$; ! *models subtraction modulo 2^n on natural numbers less than 2^n;*

3. $SHL_n^m(I_1, O)$; ! *models multiplication by 2^m modulo 2^n on natural numbers less than 2^n.*

Proof: In the case of the first and second property, we prove a stronger property that also covers the final content of the auxiliary register containing the carry/borrow bit. Each of the stronger properties is easy to prove by induction on n with case distinction on the contents of the input registers containing the most significant bits of the operands of the operation concerned and the content of the auxiliary register containing the carry/borrow bit in both the basis step and the inductive step. The third property is easy to prove by induction on n with case distinction on the content of the input register containing the most significant bit of the operand of the operation concerned in both the basis step and the inductive step.   □

Transferring n-bit words ($0 < n \leq N$) is also relevant to the multiplication algorithms. For this, we define parameterized instruction sequences as well. By one the successive bits in a constant n-bit word become the content of n successive Boolean registers and by the other the successive bits in a n-bit word that are the content of n successive Boolean registers become the

content of n other successive Boolean registers:

$$SET_n(b_0 \ldots b_{n-1}, d{:}l) \triangleq \overset{\bullet}{\underset{i=0}{\overset{n-1}{,}}}(d{:}l{+}i.\mathsf{set}{:}b_i) \ ,$$

$$MOV_n(s{:}k, d{:}l) \triangleq \overset{\bullet}{\underset{i=0}{\overset{n-1}{,}}}({+}s{:}k{+}i.\mathsf{get} \ ; -d{:}l{+}i.\mathsf{set}{:}1 \ ; d{:}l{+}i.\mathsf{set}{:}0) \ ,$$

where $b_0, \ldots, b_{n-1}$ range over $\{0, 1\}$, s ranges over $\{\mathsf{in}, \mathsf{aux}\}$, d ranges over $\{\mathsf{aux}, \mathsf{out}\}$, and k, l range over $\mathbb{N}^+$. In the case of MOV_n, the intended transfer is performed provided that the instantiation of the last parameter and the instantiation of the first parameter do not lead to partially coinciding n-bit words. In this paper, this condition will always be satisfied.

Proposition 3 *Let $n \in \mathbb{N}$ be such that $0 < n \leq N$. Then the function on bit strings of length n computed by*

1. *$SET_n(b_0 \ldots b_{n-1}, O)$; ! models the natural number constant with binary representation $b_0 \ldots b_{n-1}$;*

2. *$MOV_n(I_1, O)$; ! models the identity function on natural numbers less than 2^n.*

Proof: Each of these properties is trivial to prove by induction on n with case distinction on b_{n-1} and the content of the input register containing the most significant bits of the operand of the operation, respectively, in both the basis step and the inductive step. $\square$

For convenience's sake, we define some special cases of the parameterized instruction sequences for transferring n-bit words ($0 < m < n$):

$$ZPAD_n^m(d{:}l) \triangleq SET_{n-m}(0^{n-m}, d{:}l{+}m) \ ,$$

$$MVH_n^m(s{:}k, d{:}l) \triangleq MOV_m(s{:}k{+}(n{-}m), d{:}l) \ ,$$

$$MVL_n^m(s{:}k, d{:}l) \triangleq MOV_m(s{:}k, d{:}l) \ ,$$

where s ranges over $\{\mathsf{in}, \mathsf{aux}\}$, d ranges over $\{\mathsf{aux}, \mathsf{out}\}$, and k, l range over $\mathbb{N}^+$. $ZPAD_n^m$ is meant for turning a stored m-bit word into a stored n-bit word by zero padding. MVH_n^m and MVL_n^m are meant for transferring only the m most significant bits and the m least significant bits, respectively, of a stored n-bit word.

Because $\lceil n/2 \rceil + 1 < n$ iff $n > 3$, the Karatsuba multiplication algorithm cannot be used for modeling multiplication on natural numbers less than

2^n with respect to their binary representation by n-bit words if $n \leq 3$. Therefore, we also define a parameterized instruction sequence, in terms of the above-mentioned basic operations, that computes the operation modeling multiplication according to the long multiplication algorithm:

$$MUL_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq$$
$$\quad MOV_n(s_1{:}k_1, S_1) \,;\, ZPAD_{2n}^n(S_1) \,;\, SET_{2n}(0^{2n}, S_2) \,;$$
$$\quad {\overset{\bullet}{\underset{i=0}{\overset{n-1}{;}}}}(-s_2{:}k_2{+}i.\mathsf{get} \,;\, \#l_i \,;\, ADD_{n+i+1}(S_1, S_2, S_2) \,;\, SHL_{n+i+1}^1(S_1, S_1)) \,;$$
$$\quad MOV_{2n}(S_2, d{:}l) \,,$$
$$\quad \text{where } l_i = \mathrm{len}(ADD_{n+i+1}(S_1, S_2, S_2)) + 1 \,,$$

where s_1, s_2 range over $\{\mathsf{in}, \mathsf{aux}\}$, d ranges over $\{\mathsf{aux}, \mathsf{out}\}$, and k_1, k_2, l range over $\mathbb{N}^+$. The additions are done on the fly and the shifts are restricted to shifts by one position by shifting the result of all preceding shifts.

Proposition 4 *Let $n \in \mathbb{N}$ be such that $0 < n \leq N$. Then the function on bit strings of length n computed by $MUL_n(I_1, I_2, O) \,;\, !$ models multiplication on natural numbers less than 2^n.*

Proof: We prove a stronger property that also covers the final contents of the $2n$ successive auxiliary registers starting with the one named S_1 and the $2n$ successive auxiliary registers starting with the one named S_2. This stronger property is easy to prove, using Propositions 2 and 3, by induction on n with case distinction on the content of the input register containing the most significant bit of the second operand of the operation concerned in both the basis step and the inductive step.   □

The calculation of the lengths of the parameterized instruction sequences defined above is a matter of simple additions and multiplications. The lengths of these instruction sequences are as follows:

$$\mathrm{len}(SHL_n^m(s{:}k, d{:}l)) = 3 \cdot n - 2 \cdot m \,,$$
$$\mathrm{len}(ADD_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 21 \cdot n + 1 \,,$$
$$\mathrm{len}(SUB_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 21 \cdot n + 1 \,,$$
$$\mathrm{len}(SET_n(b_0 \ldots b_{n-1}, d{:}l)) = n \,,$$
$$\mathrm{len}(MOV_n(s{:}k, d{:}l)) = 3 \cdot n \,,$$

$$\operatorname{len}(ZPAD_n^m(d{:}l)) = n - m \;,$$
$$\operatorname{len}(MVH_n^m(s{:}k, d{:}l)) = 3 \cdot m \;,$$
$$\operatorname{len}(MVL_n^m(s{:}k, d{:}l)) = 3 \cdot m \;,$$
$$\operatorname{len}(MUL_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 36 \cdot n^2 + 24 \cdot n + 1 \;.$$

The instruction sequences defined in this section do compute the intended operations in case of fully coinciding n-bit words.

6   Long Multiplication and Karatsuba Multiplication

In this section, we describe and analyze instruction sequences that express the long multiplication algorithm and the Karatsuba multiplication algorithm, using the definitions given in Sections 4 and 5. The latter algorithm is applicable only if $N \geq 3$.

$LMUL_N$ is the instruction sequence described by

$$MUL_N(I_1, I_2, O)\,;\,! \;.$$

We know by Proposition 4 that $LMUL_N$ computes the function on bit strings that models multiplication on natural numbers less than 2^N. It does so according to the long multiplication algorithm.

Proposition 5 $\operatorname{len}(LMUL_N) = 36 \cdot N^2 + 24 \cdot N + 2.$

Proof: This is trivial because $\operatorname{len}(LMUL_N) = \operatorname{len}(MUL_N(I_1, I_2, O)) + 1.$ $\square$

$KMUL_N$, where $N \geq 3$, is the instruction sequence described by

$$MOV_N(I_1, I_1^{\lceil \log_2(N-2) \rceil})\,;\, MOV_N(I_2, I_2^{\lceil \log_2(N-2) \rceil})\,;$$
$$KMA_N\,;\, MOV_{2N}(O^{\lceil \log_2(N-2) \rceil}, O)\,;\,! \;,$$

where KMA_n is inductively defined in Table 1. $KMUL_N$ computes the function on bit strings of length N that models multiplication on natural numbers less than 2^N according to the Karatsuba multiplication algorithm.

In order to compute the binary representation of the product of two natural numbers with binary representations of length n by dividing the computation into the computations of the binary representations of three

Table 1: Definition of KMA_n $(1 \leq n \leq N)$

if $n \leq 3$ then:

$$KMA_n = MUL_n(I_1^{\ell(n)}, I_2^{\ell(n)}, O^{\ell(n)}) \,,$$

if $n > 3$ then:

$KMA_n =$

$\quad MVH_n^{\lfloor n/2 \rfloor}(I_1^{\ell(n)}, I_1^{\ell(\lfloor n/2 \rfloor)}) \,;\, MVH_n^{\lfloor n/2 \rfloor}(I_2^{\ell(n)}, I_2^{\ell(\lfloor n/2 \rfloor)}) \,;$

$\quad KMA_{\lfloor n/2 \rfloor} \,;\, MOV_{2\lfloor n/2 \rfloor}(O^{\ell(\lfloor n/2 \rfloor)}, P_1^{\ell(n)}) \,;$

$\quad MVL_n^{\lceil n/2 \rceil}(I_1^{\ell(n)}, I_1^{\ell(\lceil n/2 \rceil)}) \,;\, MVL_n^{\lceil n/2 \rceil}(I_2^{\ell(n)}, I_2^{\ell(\lceil n/2 \rceil)}) \,;$

$\quad KMA_{\lceil n/2 \rceil} \,;\, MOV_{2\lceil n/2 \rceil}(O^{\ell(\lceil n/2 \rceil)}, P_2^{\ell(n)}) \,;$

$\quad MVH_n^{\lfloor n/2 \rfloor}(I_1^{\ell(n)}, T_1) \,;\, ZPAD_{\lceil n/2 \rceil+1}^{\lfloor n/2 \rfloor}(T_1) \,;$

$\quad MVL_n^{\lceil n/2 \rceil}(I_1^{\ell(n)}, T_2) \,;\, ZPAD_{\lceil n/2 \rceil+1}^{\lceil n/2 \rceil}(T_2) \,;\, ADD_{\lceil n/2 \rceil+1}(T_1, T_2, I_1^{\ell(\lceil n/2 \rceil+1)}) \,;$

$\quad MVH_n^{\lfloor n/2 \rfloor}(I_2^{\ell(n)}, T_1) \,;\, ZPAD_{\lceil n/2 \rceil+1}^{\lfloor n/2 \rfloor}(T_1) \,;$

$\quad MVL_n^{\lceil n/2 \rceil}(I_2^{\ell(n)}, T_2) \,;\, ZPAD_{\lceil n/2 \rceil+1}^{\lceil n/2 \rceil}(T_2) \,;\, ADD_{\lceil n/2 \rceil+1}(T_1, T_2, I_2^{\ell(\lceil n/2 \rceil+1)}) \,;$

$\quad KMA_{\lceil n/2 \rceil+1} \,;\, MOV_{2(\lceil n/2 \rceil+1)}(O^{\ell(\lceil n/2 \rceil+1)}, P_3^{\ell(n)}) \,;$

$\quad ZPAD_{2(\lceil n/2 \rceil+1)}^{2\lfloor n/2 \rfloor}(P_1^{\ell(n)}) \,;\, ZPAD_{2(\lceil n/2 \rceil+1)}^{2\lceil n/2 \rceil}(P_2^{\ell(n)}) \,;$

$\quad SUB_{2(\lceil n/2 \rceil+1)}(P_3^{\ell(n)}, P_1^{\ell(n)}, T_1) \,;\, SUB_{2(\lceil n/2 \rceil+1)}(T_1, P_2^{\ell(n)}, T_1) \,;$

$\quad ZPAD_{2n}^{2(\lceil n/2 \rceil+1)}(P_1^{\ell(n)}) \,;\, ZPAD_{2n}^{2(\lceil n/2 \rceil+1)}(P_2^{\ell(n)}) \,;\, ZPAD_{2n}^{2(\lceil n/2 \rceil+1)}(T_1) \,;$

$\quad SHL_{2n}^{2\lceil n/2 \rceil}(P_1^{\ell(n)}, T_2) \,;\, SHL_{2n}^{\lceil n/2 \rceil}(T_1, T_1) \,;$

$\quad ADD_{2n}(T_2, T_1, T_1) \,;\, ADD_{2n}(T_1, P_2^{\ell(n)}, O^{\ell(n)}) \,,$

where $\ell(m) = \lceil \log_2(m-2) \rceil$.

products as required by the Karatsuba multiplication algorithm, the instruction sequence KMA_n contains the instruction sequences $KMA_{\lfloor n/2 \rfloor}$, $KMA_{\lceil n/2 \rceil}$, and $KMA_{\lceil n/2 \rceil+1}$. Each of these three instruction sequences is immediately preceded by an instruction sequence that transfers the binary representations of the two natural numbers of which it has to compute

the binary representation of their product into the appropriate Boolean
registers for the instruction sequence concerned. Moreover, each of these
three instruction sequences is immediately followed by an instruction se-
quence that transfers the binary representation of the product that it has
computed into the appropriate Boolean registers for KMA_n. The tail end of
KMA_n completes the computation by performing some operations on the
three binary representations of products computed before as required by the
Karatsuba multiplication algorithm. For the rest, instruction sequences for
zero padding are scattered over KMA_n where necessary to obtain the locally
right length of binary representations of natural numbers.

Proposition 6 *If $N \geq 3$, then the function on bit strings of length N
computed by $KMUL_N$ models multiplication on natural numbers less than 2^N.*

Proof: It is straightforward to prove this by induction on N, using the
equations from Section 3 that form the basis of the Karatsuba multiplication
algorithm and Propositions 2, 3, and 4.   □

The following proposition gives a lower estimate and an upper estimate
for the length of $KMUL_N$.

Proposition 7 *If $N \geq 3$, then:*

$$\text{len}(KMUL_N) \geq 1184 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 716 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} + 12 \cdot N - 70 \, ,$$
$$\text{len}(KMUL_N) \leq 1005 \cdot 3^{\lceil \log_2(N-2) \rceil} - 358 \cdot 2^{\lceil \log_2(N-2) \rceil} + 12 \cdot N - 249 \, .$$

Proof: Because $\text{len}(KMUL_N) = \text{len}(KMA_N) + 12 \cdot N + 1$, we have to
prove that

$$\text{len}(KMA_N) \geq 1184 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 716 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} - 71 \, ,$$
$$\text{len}(KMA_N) \leq 1005 \cdot 3^{\lceil \log_2(N-2) \rceil} - 358 \cdot 2^{\lceil \log_2(N-2) \rceil} - 250 \, .$$

Let $c_1 = \text{len}(MUL_1)$, $c_2 = \text{len}(MUL_2)$, $c_3 = \text{len}(MUL_3)$, and for each $n > 3$,
$c_n = \text{len}(KMA_n) - \text{len}(KMA_{\lfloor n/2 \rfloor}) - \text{len}(KMA_{\lceil n/2 \rceil}) - \text{len}(KMA_{\lceil n/2 \rceil + 1})$. Us-
ing the already calculated lengths of the parameterized instruction sequences
defined in Section 5, we obtain by simple calculations that $c_1 = 61$, $c_2 = 193$,
$c_3 = 397$, and for each $n > 3$, $c_n = 126 \cdot \lceil n/2 \rceil + 116 \cdot n + 142$. Let $c'_0 = c_3$,
$c''_0 = c_3$, and for each $m > 0$, $c'_m = c_{2m+2}$ and $c''_m = c_{2m+1}$. In other
words, $c'_0 = 397$, $c''_0 = 397$, and for each $m > 0$, $c'_m = 358 \cdot 2^{m-1} + 500$
and $c''_m = 358 \cdot 2^m + 142$. Because $\lfloor x \rfloor = k$ iff $k \leq x < k + 1$, $\lceil x \rceil = k$

iff $k - 1 < x \leq k$, and $\log_2(x) = y$ iff $x = 2^y$, it is clear that $c_n \leq c'_m$ if $m = \lceil \log_2(n - 2) \rceil$ and $c_n \geq c''_m$ if $m = \lfloor \log_2(n) \rfloor - 1$.

Let $M = \lceil \log_2(N - 2) \rceil$, and let $m \leq M$. It follows directly from the proof of the proposition at the end of Section 4 that, for all n such that $m = \lceil \log_2(n - 2) \rceil$, the deepest level of recursion at which KMA_n occurs is $M - m$. Moreover, it follows directly from the definition of KMA_n that, for all $n > 0$, KMA_n occurs at this level only if n is less than or equal to the greatest n' such that $m = \lceil \log_2(n' - 2) \rceil$. We also have that $c_n \leq c_{n'}$ if $n \leq n'$, and $c_{n'} \leq c'_m$ if $m = \lceil \log_2(n' - 2) \rceil$. All this means that $\text{len}(KMA_N) \leq \sum_{i=0}^{M}(c'_i \cdot 3^{M-i})$. In other words, $\text{len}(KMA_N) \leq 397 \cdot 3^M + \sum_{i=1}^{M}((358 \cdot 2^{i-1} + 500) \cdot 3^{M-i})$. Using elementary properties of sums and the property that $\sum_{i=0}^{k} x^i = (1 - x^{k+1})/(1 - x)$, we obtain $397 \cdot 3^M + \sum_{i=1}^{M}((358 \cdot 2^{i-1} + 500) \cdot 3^{M-i}) = 397 \cdot 3^M + 358 \cdot (3^M - 2^M) + 500 \cdot ((3^M - 1)/2) = 1005 \cdot 3^M - 358 \cdot 2^M - 250$. Hence, because $M = \lceil \log_2(N - 2) \rceil$, $\text{len}(KMA_N) \leq 1005 \cdot 3^{\lceil \log_2(N-2) \rceil} - 358 \cdot 2^{\lceil \log_2(N-2) \rceil} - 250$.

Let $M' = \lfloor \log_2(N) \rfloor - 1$, and let $m \leq M'$. We can show similarly to above that, for all n such that $m = \lfloor \log_2(n) \rfloor - 1$, the least deep level of recursion at which KMA_n occurs is $M' - m$. Moreover, it follows directly from the definition of KMA_n that, for all $n > 0$, KMA_n occurs at this level only if n is greater than or equal to the least n' such that $m = \lfloor \log_2(n') \rfloor - 1$. We also have that $c_n \geq c_{n'}$ if $n \geq n'$, and $c_{n'} \geq c''_m$ if $m = \lfloor \log_2(n') \rfloor - 1$. All this means that $\text{len}(KMA_N) \geq \sum_{i=0}^{M'}(c''_i \cdot 3^{M'-i})$. In other words, $\text{len}(KMA_N) \geq 397 \cdot 3^{M'} + \sum_{i=1}^{M'}((358 \cdot 2^i + 142) \cdot 3^{M'-i})$. Using the same properties of sums as before, we obtain $397 \cdot 3^{M'} + \sum_{i=1}^{M'}((358 \cdot 2^i + 142) \cdot 3^{M'-i}) = 397 \cdot 3^{M'} + 358 \cdot (2 \cdot (3^{M'} - 2^{M'})) + 142 \cdot ((3^{M'} - 1)/2) = 1184 \cdot 3^{M'} - 716 \cdot 2^{M'} - 71$. Hence, because $M' = \lfloor \log_2(N) \rfloor - 1$, $\text{len}(KMA_N) \geq 1184 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 716 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} - 71$. $\qquad \square$

It is unclear to us whether it is practically possible to improve the lower estimate and upper estimate for the length of $KMUL_N$ considerably.

The following is a corollary of Propositions 5 and 7.

Corollary 1 $\text{len}(LMUL_N) = \Theta(N^2)$ and $\text{len}(KMUL_N) = \Theta(N^{\log_2(3)}) = \Theta(N^{1,5849\ldots})$.

This corollary can be paraphrased as follows: the length of the instruction sequences $LMUL_N$ and $KMUL_N$, which express the long multiplication algorithm and the Karatsuba multiplication algorithm, are asymptotically bounded, up to a constant factor, both above and below by N^2 and $N^{\log_2(3)}$,

respectively. It is striking because these algorithms are known to compute the function that models multiplication on natural numbers less than 2^N with respect to their binary representation by N-bit words also in time asymptotically bounded, up to a constant factor, both above and below by N^2 and $N^{\log_2(3)}$, respectively. This suggests, like some results from [4], that instruction sequence size and computation time are polynomially related measures.

Using Propositions 5 and 7, it is easy to check that (a) $LMUL_N$ is longer than $KMUL_N$ only if $N > 264$ and (b) $LMUL_N$ is longer than $KMUL_N$ if $N > 6666$. On that account, the following is another corollary of Propositions 5 and 7.

Corollary 2 $N > 2^8$ *if* $\text{len}(LMUL_N) > \text{len}(KMUL_N)$ *and* $\text{len}(LMUL_N) > \text{len}(KMUL_N)$ *if* $N > 2^{13}$.

In the area of algorithm efficiency, like in the area of computational complexity, the focus is mainly on asymptotic properties of algorithms, like Corollary 1. To our knowledge, there is virtually no attention in this area to properties related to crossover points between algorithms, like Corollary 2. We think that properties of the latter kind are frequently more relevant to practice than properties of the former kind. However, existing knowledge about crossover points between algorithms is mainly based on experimental data which are highly dependent on the computer, operating system, programming language and compiler used in the experiment. Moreover, if this kind of knowledge is referred to at all, it is often turned into the form of a rule of thumb. For example, the following statement and minor variants of it can be found at many places (webpages, articles, and books) without further justification: "As a rule of thumb, Karatsuba is usually faster when the multiplicands are longer than 320-640 bits" (see e.g. [15]).

It is obvious that $LMUL_N$ and $KMUL_N$ need the same number of input registers and the same number of output registers. However, the number of auxiliary registers used by $KMUL_N$ is always greater than the number of auxiliary registers used by $LMUL_N$. The number of auxiliary registers used by $KMUL_N$ is $10 \cdot N \cdot \lceil \log_2(N-2) \rceil + 18 \cdot N + 1$ and the number of auxiliary registers used by $LMUL_N$ is only $4 \cdot N + 1$. In the instance that $N = 2^8$, these numbers correspond to $\pm 3K$ bytes and ± 128 bytes, respectively; and in the instance that $N = 2^{13}$, these numbers correspond to $\pm 148K$ bytes and $\pm 4K$ bytes, respectively.

In this paper, we do not answer the question whether there exist instruction sequences shorter than $LMUL_N$ and $KMUL_N$ that express the long

multiplication algorithm and Karatsuba multiplication algorithm, respectively. The practical problem with proving or disproving the existence of shorter instruction sequences is that it needs basically an extremely extensive case distinction. We expect that, if the length of $LMUL_N$ and/or $KMUL_N$ can be reduced, it cannot be reduced much. The reason for this is that we have striven in Section 5 for instruction sequences without unreachable subsequences, different suffixes with the same behaviour on execution, and jump instruction that can be eliminated without introducing different suffixes with the same behaviour on execution.

7   Long Multiplication and
Backward Jump Instructions

In this section, a minor variant of the long multiplication algorithm is expressed by an instruction sequence that contains a backward jump instruction in addition to instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction.

We use the fragment without repetition operator of an extension of PGA with, for each $l \in \mathbb{N}$, a *backward jump instruction* $\backslash\#l$ as additional primitive instruction. On execution of an instruction sequence, the effect of a backward jump instruction $\backslash\#l$ is that execution proceeds with the lth previous primitive instruction of the instruction sequence concerned — if l equals 0 or there is no primitive instruction to proceed with, inaction occurs. We write PGA_{bj} for the above-mentioned extension of PGA. For a mathematically precise treatment of PGA_{bj} without repetition operator, we refer to the treatment of C, which is a variant of PGA, in [6]. The fragment of PGA_{bj} without the repetition operator coincides with the fragment of C without backward instructions other than backward jump instructions.

The additional basic operations on words that are relevant in this section are the operations that model Euclidean division by 2^m, decrement by 1, and nonzero test on natural numbers less than 2^n, with respect to their representation by n-bit words ($0 < n \le N$, $0 < m < n$). The operation modeling Euclidean division by 2^m is commonly known as "shift right by m positions". For these operations, we define parameterized instruction sequences computing them in case the parameters are properly instantiated (see below):

$SHR_n^m(s{:}k, d{:}l) \triangleq$

$\quad \overset{\bullet}{;}{}_{i=0}^{n-1-m}(+s{:}k{+}m{+}i.\mathsf{get}\,;\, -d{:}l{+}i.\mathsf{set}{:}1\,;\, d{:}l{+}i.\mathsf{set}{:}0)\,;$

$\quad \overset{\bullet}{;}{}_{i=0}^{m-1}(d{:}l{+}n{-}m{+}i.\mathsf{set}{:}0)\,,$

$DEC_n(s{:}k, d{:}l) \triangleq$

$\quad \overset{\bullet}{;}{}_{i=0}^{n-1}(-s{:}k{+}i.\mathsf{get}\,;\,\#3\,;\, d{:}l{+}i.\mathsf{set}{:}0\,;\,\#5\,;\, d{:}l{+}i.\mathsf{set}{:}1)\,;\,\#1\,;\,\#1\,;\,\#1\,,$

$ISNZ_n(s{:}k) \triangleq$

$\quad \overset{\bullet}{;}{}_{i=0}^{n-1}(+s{:}k{+}i.\mathsf{get}\,;\,\#2)\,;\,\#2\,,$

where s ranges over $\{\mathsf{in}, \mathsf{aux}\}$, d ranges over $\{\mathsf{aux}, \mathsf{out}\}$, and k, l range over $\mathbb{N}^+$. For each of the first two parameterized instruction sequences, the first parameter correspond to the operand of the operation concerned and the second parameter corresponds to the result of the operation concerned. The intended operations are computed provided that the instantiation of the first parameter and the instantiation of the second parameters do not lead to partially coinciding n-bit words. In this section, this condition will always be satisfied. No result is stored on execution of $ISNZ_n$. Instead, the first primitive instruction following $ISNZ_n$ is skipped if the nonzero test fails.

Proposition 8 *Let $n, m \in \mathbb{N}$ be such that $0 < n \le N$ and $0 < m < n$. Then the function on bit strings of length n computed by*

1. *$SHR_n^m(I_1, O)\,;\,!$ models Euclidean division by 2^m modulo 2^n on natural numbers less than 2^n;*

2. *$DEC_n(I_1, O)\,;\,!$ models subtraction by 1 modulo 2^n on natural numbers less than 2^n;*

3. *$ISNZ_n(I_1)\,;\,+O.\mathsf{set}{:}1\,;\,O.\mathsf{set}{:}0\,;\,!$ models the function isnz from natural numbers less than 2^n to natural numbers less than 2^1 defined by $isnz(0) = 0$ and $isnz(k+1) = 1$ with respect to their binary representation by n-bit words and 1-bit words, respectively.*

Proof: Each of these properties is easy to prove by induction on n with case distinction on the content of the input register containing the most significant bit of the operand of the operation concerned in both the basis step and the inductive step. $\qquad\square$

The lengths of the parameterized instruction sequences defined above are as follows:

$$\text{len}(SHR_n^m(s{:}k, d{:}l)) = 3 \cdot n - 2 \cdot m \ ,$$
$$\text{len}(DEC_n(s{:}k, d{:}l)) = 5 \cdot n + 3 \ ,$$
$$\text{len}(ISNZ_n(s{:}k)) = 2 \cdot n + 1 \ .$$

For each bit of the representation of the multiplier, $LMUL_N$ contains a different instruction sequence. This seems to exclude the use of backward jump instructions to obtain an instruction sequence of significantly shorter length, unless provision is made for some form of indirect addressing for Boolean registers. However, there exists a minor variant of the long multiplication algorithm that makes it possible to have the same instruction sequence for each bit of the representation of the multiplier. From the least significant bit of the representation of the multiplier onwards, the algorithm concerned shifts the representation of the multiplier by one position to the right after it has dealt with a bit. In this way, the next bit remains the least significant one throughout.

We proceed with describing an instruction sequence without backward jump instructions that expresses this minor variant of the long multiplication algorithm.

$LMUL_N'$ is the instruction sequence described by

$$MOV_N(I_1, S_1)\,;\, ZPAD_{2N}^N(S_1)\,;\, SET_{2N}(0^{2N}, S_2)\,;\, MOV_N(I_2, T_1)\,;$$
$$(-T_1.\mathsf{get}\,;\, \#l\,;\, ADD_{2N}(S_1, S_2, S_2)\,;\, SHL_{2N}^1(S_1, S_1)\,;\, SHR_N^1(T_1, T_1))^N\,;$$
$$MOV_{2N}(S_2, O)\,;\, !\ ,$$

where

$$l = \text{len}(ADD_{2N}(S_1, S_2, S_2)) + 1 = 42 \cdot N + 2 \ .$$

Proposition 9 *The function on bit strings of length N computed by $LMUL_N'$ models multiplication on natural numbers less than 2^N.*

Proof: We prove a stronger property that also covers the final contents of the $2N$ successive auxiliary registers starting with the one named S_1, the $2N$ successive auxiliary registers starting with the one named S_2, and the N successive auxiliary registers starting with the one named T_1. This stronger property is straightforward to prove, using Propositions 2, 3, and 8, by induction on N with case distinction on the content of the input register containing the most significant bit of the second operand of the operation concerned in both the basis step and the inductive step.   □

Proposition 10 $\operatorname{len}(LMUL'_N) = 51 \cdot N^2 + 14 \cdot N + 1$.

Proof: This is a matter of simple additions, subtractions, and multiplications. $\qquad\square$

The following is a corollary of Propositions 5 and 10.

Corollary 3 $\operatorname{len}(LMUL'_N) > \operatorname{len}(LMUL_N)$.

For each bit of the representation of the multiplier, $LMUL'_N$ contains the same instruction sequence. That is, it contains N duplicates of the same instruction sequence. This duplication can be eliminated by implementing a loop by means of a backward jump instruction.

We proceed with describing an instruction sequence with a backward jump instruction that expresses the minor variant of the long multiplication algorithm. We write $\underline{N}$ for the shortest representation of the natural number N in the binary number system.

$LMUL''_N$ is the instruction sequence described by

$$MOV_N(I_1, S_1)\,;\, ZPAD_{2N}^N(S_1)\,;\, SET_{2N}(0^{2N}, S_2)\,;\, MOV_N(I_2, T_1)\,;$$
$$SET_{\lfloor \log_2(N) \rfloor + 1}(\underline{N}, T_2)\,;$$
$$-T_1.\mathsf{get}\,;\, \#l_1\,;\, ADD_{2N}(S_1, S_2, S_2)\,;\, SHL_{2N}^1(S_1, S_1)\,;\, SHR_N^1(T_1, T_1)\,;$$
$$DEC_{\lfloor \log_2(N) \rfloor + 1}(T_2, T_2)\,;\, ISNZ_{\lfloor \log_2(N) \rfloor + 1}(T_2)\,;\, \backslash \#l_2\,;$$
$$MOV_{2N}(S_2, O)\,;\, !\,,$$

where

$$l_1 = \operatorname{len}(ADD_{2N}(S_1, S_2, S_2)) + 1 = 42 \cdot N + 2\,,$$
$$l_2 = \operatorname{len}(-T_1.\mathsf{get}\,;\, \ldots\,;\, ISNZ_{\lfloor \log_2(N) \rfloor + 1}(T_2)) = 51 \cdot N + 7 \cdot \lfloor \log_2(N) \rfloor + 10\,.$$

Proposition 11 *The function on bit strings of length N computed by $LMUL''_N$ models multiplication on natural numbers less than 2^N.*

Proof: We prove a stronger property that also covers the final contents of the $2N$ successive auxiliary registers starting with the one named S_1, the $2N$ successive auxiliary registers starting with the one named S_2, the N successive auxiliary registers starting with the one named T_1, and the $\lfloor \log_2(N) \rfloor + 1$ successive auxiliary registers starting with the one named T_2. This stronger property is straightforward to prove, using Propositions 2, 3, and 8, by

induction on N with case distinction on the content of the input register containing the most significant bit of the second operand of the operation concerned in both the basis step and the inductive step. $\square$

Proposition 12 $\mathrm{len}(LMUL''_N) = 66 \cdot N + 8 \cdot \lfloor \log_2(N) \rfloor + 13$.

Proof: This is a matter of simple additions, subtractions, and multiplications. $\square$

The following is a corollary of Propositions 5, 10, and 12.

Corollary 4 $\mathrm{len}(LMUL''_N) = \Theta(N)$ *while both* $\mathrm{len}(LMUL_N) = \Theta(N^2)$, *and* $\mathrm{len}(LMUL'_N) = \Theta(N^2)$.

Hence, $LMUL''_N$ is asymptotically shorter than both $LMUL_N$ and $LMUL'_N$. By Corollary 1, we know that $LMUL''_N$ is asymptotically shorter than $KMUL_N$ too.

The following is a corollary of Propositions 5, 7, 10, and 12.

Corollary 5 *Both* $\mathrm{len}(LMUL''_N) < \mathrm{len}(LMUL_N)$ *and* $\mathrm{len}(LMUL''_N) < \mathrm{len}(LMUL'_N)$ *if* $N > 1$, *and what is more,* $\mathrm{len}(LMUL''_N) < \mathrm{len}(KMUL_N)$ *if* $N > 2$.

Hence, $LMUL''_N$ is already shorter than $LMUL_N$, $LMUL'_N$, and $KMUL_N$ if N is still very small. In fact, long multiplication is non-trivial only if $N > 1$ and Karatsuba multiplication is applicable only if $N > 2$.

8   Long Multiplication and the Halting Problem

In this section, we argue that the instruction sequences $LMUL'_N$ and $LMUL''_N$ from Section 7 form a hard witness of the inevitable existence of a halting problem in the practice of imperative programming.

Turing's result regarding the undecidability of the halting problem (see e.g. [14]) is a result about Turing machines. In [2], we consider it as a result about programs rather than machines, taking instruction sequences as programs. The instruction sequences concerned are essentially the finite instruction sequences that can be denoted by closed $\mathrm{PGA_{bj}}$ terms. Unlike in the current paper, the basic instructions are not fixed, but their effects are restricted to the manipulation of something that can be understood as the content of the tape of a Turing machine with a specific tape alphabet,

together with the position of the tape head. Different choices of basic instructions give rise to different halting problem instances and one of these instances is essentially the same as the halting problem for Turing machines. Because of their orientation to Turing machines, we consider all instances treated in [2] theoretical halting problem instances.

All halting problem instances would evaporate if the instruction sequences concerned would be restricted to the ones without backward jump instructions. This is irrespective of whether the effects of the basic instructions have anything to do with the manipulation of a Turing machine tape. In the case that we have basic instructions to set and get the content of Boolean registers, instruction sequences without backward jump instructions are sufficient to compute all functions $f : \{0,1\}^n \to \{0,1\}^m$ ($n, m \in \mathbb{N}$). This raises the question whether there exists a good reason for not abandoning backward jump instructions altogether in such cases. The function that models multiplication on natural numbers less than 2^N with respect to their binary representation by N-bit words offers a good reason: the length of the instruction sequence that computes it according to the long multiplication algorithm can be reduced significantly by the use of backward jump instructions. The length of the instruction sequence that computes this function can be reduced even more by the use of backward jump instructions than by going over to one of the multiplication algorithms that are known to yield shorter instruction sequences without backward jump instructions than the long multiplication algorithm such as for example the Karatsuba multiplication algorithm.

Thus, the instruction sequences $LMUL'_N$ and $LMUL''_N$ form a hard witness of the inevitable existence of a halting problem in the practice of imperative programming, where programs must have manageable size. Because of its orientation to actual programming, we consider the halting problem for the instruction sequences with forward and backward jump instructions, and with only basic instructions to set and get the content of Boolean registers, a practical halting problem. It is unknown to us whether there is a connection between the solvability or unsolvability of the halting problem for these instruction sequences and some form of diagonal argument. It is easy to prove that this halting problem is both NP-hard and coNP-hard. We do not know whether stronger lower bounds for its complexity can be found in the literature. An extensive search for such lower bounds and other results concerning this halting problem or a similar halting problem has been unsuccessful.

9   Concluding Remarks

We have described finite instruction sequences, containing only instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction, that compute the function that models multiplication on natural numbers less than 2^N with respect to their binary representation by N-bit words according to the long multiplication algorithm and the Karatsuba multiplication algorithm. We have described those instruction sequences by means of terms of PGA, an algebraic theory of single-pass instruction sequences.

Thus, we have provided mathematically precise alternatives to the natural language and pseudo code descriptions of these multiplication algorithms found in mathematics and computer science literature on multiplication algorithms. Moreover, we have calculated the exact size of the instruction sequence $LMUL_N$ expressing the long multiplication algorithm and lower and upper estimates for the size of the instruction sequence $KMUL_N$ expressing the Karatsuba multiplication algorithm. The results following from the calculated sizes include: (a) $\mathrm{len}(LMUL_N) = \Theta(N^2)$ and $\mathrm{len}(KMUL_N) = \Theta(N^{\log_2(3)})$; (b) $N > 2^8$ if $\mathrm{len}(LMUL_N) > \mathrm{len}(KMUL_N)$, and $\mathrm{len}(LMUL_N) > \mathrm{len}(KMUL_N)$ if $N > 2^{13}$. It is suggested by (a) that instruction sequence size and computation time are polynomially related measures. It is still an open question whether this is the case.

As a bonus, we have found that the number of auxiliary registers used by $LMUL_N$ is $4 \cdot N + 1$ and the number of auxiliary registers used by $KMUL_N$ is $10 \cdot N \cdot \lceil \log_2(N-2) \rceil + 18 \cdot N + 1$. It is also an open question whether the number of auxiliary registers that are used by an instruction sequence and computation space are related measures.

We have also gone into the use of an instruction sequence with backward jump instructions for expressing the long multiplication algorithm. We have described a finite instruction sequence $LMUL_N''$ containing a backward jump instruction, in addition to the instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction, that expresses a minor variant of the long multiplication algorithm. We have calculated the exact size of this instruction sequence and have found that: (a) $\mathrm{len}(LMUL_N'') = \Theta(N)$; (b) $\mathrm{len}(LMUL_N'') < \mathrm{len}(LMUL_N)$ if $N > 1$, and (c) $\mathrm{len}(LMUL_N'') < \mathrm{len}(KMUL_N)$ if $N > 2$. Furthermore, we have related these findings to the halting problem.

Acknowledgements

References

[1] J. A. Bergstra and M. E. Loots. Program Algebra for Sequential Code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002. `doi:10.1016/S1567-8326(02)00018-8`.

[2] J. A. Bergstra and C. A. Middelburg. Instruction Sequence Processing Operators. *Acta Informatica*, 49(3):139–172, 2012. `doi:10.1007/s00236-012-0154-2`.

[3] J. A. Bergstra and C. A. Middelburg. *Instruction Sequences for Computer Science*, volume 2 of *Atlantis Studies in Computing*. Atlantis Press, Amsterdam, 2012. `doi:10.2991/978-94-91216-65-7_2`.

[4] J. A. Bergstra and C. A. Middelburg. Instruction Sequence Based Non-uniform Complexity Classes. *Scientific Annals of Computer Science*, 24(1):47–89, 2014. `doi:10.7561/SACS.2014.1.47`.

[5] J. A. Bergstra and C. A. Middelburg. On Algorithmic Equivalence of Instruction Sequences for Computing Bit String Functions. *Fundamenta Informaticae*, 138(4):411–434, 2015. `doi:10.3233/FI-2015-1219`.

[6] J. A. Bergstra and A. Ponse. An Instruction Sequence Semigroup with Involutive Anti-Automorphisms. *Scientific Annals of Computer Science*, 19:57–92, 2009.

[7] S. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, Cambridge, MA, 1966.

[8] M. Fürer. Faster Integer Multiplication. *SIAM Journal of Computing*, 39(3):979–1005, 2009. `doi:10.1137/070711761`.

[9] A. A. Karatsuba. The Complexity of Computations. *Proceedings of the Steklov Institute of Mathematics*, 211:169–183, 1995.

[10] A. A. Karatsuba and Y. P. Ofman. Multiplication of Multidigit Numbers on Automata. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962. in Russian.

[11] C. A. Middelburg. Instruction Sequences as a Theme in Computer Science. https://instructionsequence.wordpress.com

[12] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3–4):281–292, 1971. doi:10.1007/BF02242355.

[13] A. A. Toom. The Complexity of a Scheme of Functional Elements Simulating the Multiplication of Integers. *Doklady Akademii Nauk SSSR*, 150(2):496–498, 1963. in Russian.

[14] A. M. Turing. On Computable Numbers, With an Application to the Entscheidungs Problem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1937. doi:10.1112/plms/s2-42.1.230. Correction: *ibid*, 43:544–546, 1937. doi:10.1112/plms/s2-43.6.544.

[15] Karatsuba Algorithm. In Wikipedia, 2018. Retrieved on July 1, 2018, from http://en.wikipedia.org/wiki/Karatsuba_algorithm.

[16] Secure Hash Standard. National Institute of Standards and Technology, FIPS PUB 180-4, March 2012.