Instruction sequence expressions for the Karatsuba multiplication algorithm

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

Publication date
2013
Document Version
Submitted manuscript

Instruction Sequence Expressions for the Karatsuba Multiplication Algorithm

J.A. Bergstra and C.A. Middelburg

Informatics Institute, Faculty of Science, University of Amsterdam,
Science Park 904, 1098 XH Amsterdam, the Netherlands
J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl

Abstract. The Karatsuba multiplication algorithm is an algorithm for computing the product of two natural numbers represented in the binary number system. This means that the algorithm actually computes a function on bit strings. The restriction of this function to bit strings of any given length can be computed according to the Karatsuba multiplication algorithm 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 the instruction sequences concerned for the restrictions to bit strings of the different lengths by uniform terms from an algebraic theory.

Keywords: single-pass instruction sequence, bit string, Karatsuba multiplication algorithm, long multiplication algorithm.

1998 ACM Computing Classification: F.1.1, F.2.1.

1 Introduction

The Karatsuba multiplication algorithm [12,13] was devised by Karatsuba in 1962 to disprove the conjecture made by Kolmogorov that any algorithm to compute the product of two natural numbers represented in the binary number system has time complexity $\Omega(n^2)$. Shortly afterwards, this divide-and-conquer algorithm was generalized by Toom and Cook [9,15]. Later, asymptotically faster multiplication algorithms, based on fast Fourier transforms, were devised by Schönhage and Strassen [14] and Fürer [10]. 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 introduced 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.[1] As a continuation of this work on an approach to programming language semantics, (a) the notion of an instruction sequence was subjected to systematic and precise analysis using the groundwork laid earlier and (b) selected issues relating to well-known subjects from the theory of computation and the area of computer architecture were rigorously investigated thinking in terms of instruction sequences (see e.g. [2,3,5,8]).

The general aim of all the work referred to above 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. Different from usual in the work referred to above, but as in the recent work presented in [7], the accent is this time on a practical problem. The practical problem is to devise instruction sequences that compute the product of two natural numbers represented in the binary number system according to 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 (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 basic and derived operations on n-bit words that are used in the Karatsuba multiplication algorithm (Section 5). Then, we give the description of instruction sequences that compute the product of two natural numbers represented in the binary number system according to the Karatsuba multiplication algorithm (Section 6). Finally, we make some concluding remarks (Section 7).

The preliminaries to the work presented in this paper are the same as the preliminaries to the work presented in [7], which are in turn a selection from the preliminaries to the work presented in [6]. 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 among other things be found in [4].

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 [6].

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.

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

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 $\mathop{;}_{i=0}^{n} P_i$, where $P_0, \ldots, P_n$ are PGA terms, for the PGA term $P_0 ; \ldots ; P_n$.

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

$$\begin{aligned}&\{\mathsf{in}{:}i.\mathsf{get} \mid i \in \mathbb{N}^+\} \cup \{\mathsf{out}{:}i.\mathsf{set}{:}b \mid i \in \mathbb{N}^+ \wedge b \in \{0,1\}\}\\ &\quad \cup \{\mathsf{aux}{:}i.\mathsf{get} \mid i \in \mathbb{N}^+\} \cup \{\mathsf{aux}{:}i.\mathsf{set}{:}b \mid i \in \mathbb{N}^+ \wedge b \in \{0,1\}\}\end{aligned}$$

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 $f : \{0,1\}^n \to \{0,1\}^m$, 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 the 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 product of two natural numbers represented in the binary number system by dividing the computation into the computation of three products as indicated above and doing so recursively until it not any more leads to products of natural numbers with binary representations of further reduced length. 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 $\Theta(n^{\log_2(3)}) = \Theta(n^{1,5849\cdots})$, 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 compute the product of two natural numbers represented in the binary number system according to the Karatsuba multiplication algorithm. 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.

It is assumed that a fixed but arbitrary positive natural number N has been given. The Karatsuba multiplication algorithm actually computes a function on bit strings. In Section 6, we will give a description of an instruction sequence that computes the restriction of the function concerned to bit strings of a fixed but arbitrary length according to this algorithm. N is taken as this fixed but arbitrary length.

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 content 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 $|i - i'| < n$.

The words that represent the two natural numbers whose product is to be computed are stored in advance of the whole 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 word that represents the product is stored before the end of the whole 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 that represent the intermediate values that are computed are temporarily stored during the whole computation in auxiliary registers, starting with the auxiliary register with number 1.

Because the product of two natural numbers will be computed by recursively computing products of smaller natural numbers, 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 representations of smaller natural numbers and their product are stored. Because at each level of recursion, except the last level, the computation of a product involves the computation 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 the representation of these products are stored.

It is also convenient to have available the names t_1, t_2, T_1, T_2 for the auxiliary registers in which the least significant bit of words that represent the intermediate values that are computed, other than the ones mentioned in the previous paragraph, are stored.[2] Moreover, it is convenient to have available the name c for the auxiliary register that contains the carry bit that is repeatedly stored when computing the addition operation.

Therefore, we define:

$$
\begin{aligned}
I_1 &\triangleq \mathsf{in}{:}1, \\
I_2 &\triangleq \mathsf{in}{:}k \quad \text{where } k = N + 1, \\
O &\triangleq \mathsf{out}{:}1, \\
c &\triangleq \mathsf{aux}{:}1, \\
t_1 &\triangleq \mathsf{aux}{:}2, \\
t_2 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 2 \cdot N + 2, \\
T_1 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 4 \cdot N + 2, \\
T_2 &\triangleq \mathsf{aux}{:}k \quad \text{where } k = 6 \cdot N + 2,
\end{aligned}
$$

[2] The auxiliary registers with names t_1 and t_2 are reserved for the least significant bit of intermediate values that arise when computing one of the derived operations on bit strings introduced in Section 5.

$$
\begin{aligned}
I_1^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 8 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),\\
I_2^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 9 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),\\
O^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 10 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),\\
P_1^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 12 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),\\
P_2^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 14 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil),\\
P_3^i &\triangleq \mathsf{aux}{:}k \text{ where } k = 10 \cdot N \cdot i + 16 \cdot N + 2 \quad (0 \le i \le \lceil \log_2(N-2) \rceil).
\end{aligned}
$$

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. *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 computation of the product of two natural numbers with binary representations of length n is divided into the computation of a product of two natural numbers with binary representations of length $\lfloor n/2 \rfloor$, a product of two natural numbers with binary representations of length $\lceil n/2 \rceil$, and 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 $\lceil \log_2(N-2) \rceil$. □

The proposition above 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, ..., $\lceil \log_2(N-2) \rceil$). 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 that compute the product of two natural numbers represented in the binary number system according to 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 Karatsuba multiplication algorithm are bitwise negation, shift left m positions ($0 < m < n$) and addition on n-bit words ($0 < n \le N$). For these operations, we define parameterized instruction sequences computing them in case the parameters are properly

instantiated (see below):

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

$$SHL_n^m(s{:}k, d{:}l) \triangleq$$
$$\quad {}^{\bullet}_{;}{}^{n-1-m}_{i=0}(+s{:}k{+}n{-}1{-}m{-}i.\mathsf{get}\ ;\ \#2\ ;\ +d{:}l{+}n{-}1{-}i.\mathsf{set}{:}0\ ;\ d{:}l{+}n{-}1{-}i.\mathsf{set}{:}1)\ ;$$
$$\quad {}^{\bullet}_{;}{}^{m-1}_{i=0}(d{:}l{+}m{-}1{-}i.\mathsf{set}{:}0)\ ,$$

$$ADD_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq$$
$$\quad c.\mathsf{set}{:}0\ ;$$
$$\quad {}^{\bullet}_{;}{}^{n-1}_{i=0}(+s_1{:}k_1{+}i.\mathsf{get}\ ;\ \#4\ ;\ +s_2{:}k_2{+}i.\mathsf{get}\ ;\ \#7\ ;\ \#9\ ;\ +s_2{:}k_2{+}i.\mathsf{get}\ ;\ \#10\ ;$$
$$\qquad +c.\mathsf{get}\ ;\ \#10\ ;\ \#16\ ;\ +c.\mathsf{get}\ ;\ \#7\ ;\ \#13\ ;\ +c.\mathsf{get}\ ;\ \#11\ ;\ \#9\ ;\ +c.\mathsf{get}\ ;\ \#4\ ;$$
$$\qquad d{:}l{+}i.\mathsf{set}{:}0\ ;\ c.\mathsf{set}{:}1\ ;\ \#6\ ;\ d{:}l{+}i.\mathsf{set}{:}1\ ;\ c.\mathsf{set}{:}1\ ;\ \#3\ ;$$
$$\qquad +d{:}l{+}i.\mathsf{set}{:}0\ ;\ d{:}l{+}i.\mathsf{set}{:}1)\ ,$$

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.

Transferring n-bit words ($0 < n \leq N$) is also relevant to the Karatsuba multiplication algorithm. 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:

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

$$MOV_n(s{:}k, d{:}l) \triangleq {}^{\bullet}_{;}{}^{n-1}_{i=0}(+s{:}k{+}i.\mathsf{get}\ ;\ \#2\ ;\ +d{:}l{+}i.\mathsf{set}{:}0\ ;\ d{:}l{+}i.\mathsf{set}{:}1)\ ,$$

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.

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.

The Karatsuba multiplication algorithm as usually described involves subtraction and multiplication on n-bit words ($0 < n \leq N$ for subtraction and $0 < n \leq 3$ for multiplication). For these operations, which can be defined in terms of the above-mentioned basic operations, we also define parameterized instruction sequences computing them:

$$
\begin{aligned}
&SUB_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq \\
&\quad SET_n(1\,0^{n-1}, t_1)\,;\, NOT_n(s_2{:}k_2, t_2)\,;\, ADD_n(t_1, t_2, t_2)\,;\, ADD_n(s_1{:}k_1, t_2, d{:}l)\,,
\end{aligned}
$$

$$
\begin{aligned}
&MUL_n(s_1{:}k_1, s_2{:}k_2, d{:}l) \triangleq \\
&\quad MOV_n(s_1{:}k_1, t_1)\,;\, ZPAD_{2n}^n(t_1)\,;\, SET_{2n}(0^{2n}, t_2)\,; \\
&\quad \bullet_{i=0}^{n-1}(-s_2{:}k_2{+}i.\mathsf{get}\,;\, \#l_i\,;\, ADD_{n+i+1}(t_1, t_2, t_2)\,;\, SHL_{n+i+1}^1(t_1, t_1))\,; \\
&\quad MOV_{2n}(t_2, d{:}l)\,, \\
&\text{where } l_i = \mathrm{len}(ADD_{n+i+1}(t_1, t_2, t_2)) + 1\,,
\end{aligned}
$$

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 $\mathbb{N}^+$. In the case of MUL_n, the product is computed according to the long multiplication algorithm. The additions are done on the fly and the shifts are restricted to one position by shifting the result of all preceding shifts.

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:

$$
\begin{aligned}
&\mathrm{len}(NOT_n(s{:}k, d{:}l)) = 4 \cdot n\,, \\
&\mathrm{len}(SHL_n^m(s{:}k, d{:}l)) = 4 \cdot n - 3 \cdot m\,, \\
&\mathrm{len}(ADD_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 26 \cdot n + 1\,, \\
&\mathrm{len}(SET_n(b_0 \ldots b_{n-1}, d{:}l)) = n\,, \\
&\mathrm{len}(MOV_n(s{:}k, d{:}l)) = 4 \cdot n\,, \\
&\\
&\mathrm{len}(SUB_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 57 \cdot n + 2\,, \\
&\mathrm{len}(MUL_n(s_1{:}k_1, s_2{:}k_2, d{:}l)) = 45 \cdot n^2 + 30 \cdot n\,, \\
&\mathrm{len}(ZPAD_n^m(d{:}l)) = n - m\,, \\
&\mathrm{len}(MVH_n^m(s{:}k, d{:}l)) = 4 \cdot m\,, \\
&\mathrm{len}(MVL_n^m(s{:}k, d{:}l)) = 4 \cdot m\,.
\end{aligned}
$$

Note that the instruction sequences defined in this section do compute the intended operations in case of fully coinciding n-bit words. Slightly shorter instruction sequences are defined for bitwise negation, addition, and transfer of a stored word in [7], but those instruction sequences do not compute the intended operations in case of fully coinciding n-bit words.

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:

$KMA_n =$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 The Karatsuba Multiplication Algorithm

In this section, we give the description of instruction sequences that compute the product of two natural numbers represented in the binary number system according to the Karatsuba multiplication algorithm using the definitions given in Sections 4 and 5.

For $N \geq 3$, an instruction sequence $KMUL_N$ is uniformly 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.

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 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 sequence 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.

Claim. *Assume that $N \geq 3$. Then the instruction sequence $KMUL_N$ computes the function on bit strings of length N that models the multiplication of two natural numbers less than 2^N on their representations in the binary number system.*

We do not formally prove this claim. If we assume that all parameterized instruction sequences defined in Section 5 compute their intended operations correctly, it is straightforward to prove the claim because the Karatsuba multiplication algorithm is followed precisely in the description of $KMUL_N$. Making occasionally use of elementary knowledge about the modelling of arithmetic operations on the representations of natural numbers in the binary number system, it is straightforward, but tedious, to prove that all parameterized instruction sequences defined in Section 5 compute their intended operations correctly.

Fact. *Assume that $N \geq 3$. Then:*

$$\begin{aligned}
&\mathrm{len}(KMUL_N) \geq 1790 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 1146 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} + 16 \cdot N - 148 \,, \\
&\mathrm{len}(KMUL_N) < 1504 \cdot 3^{\lceil \log_2(N-2) \rceil} - 573 \cdot 2^{\lceil \log_2(N-2) \rceil} + 16 \cdot N - 434 \,.
\end{aligned}$$

Proof. Because $\mathrm{len}(KMUL_N) = \mathrm{len}(KMA_N) + 16 \cdot N + 1$, we have to prove that

$$\begin{aligned}
&\mathrm{len}(KMA_N) \geq 1790 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 1146 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} - 149 \,, \\
&\mathrm{len}(KMA_N) < 1504 \cdot 3^{\lceil \log_2(N-2) \rceil} - 573 \cdot 2^{\lceil \log_2(N-2) \rceil} - 435 \,.
\end{aligned}$$

Let $c_1 = \mathrm{len}(MUL_1)$, $c_2 = \mathrm{len}(MUL_2)$, $c_3 = \mathrm{len}(MUL_3)$, and for each $n > 3$, $c_n = \mathrm{len}(KMA_n) - \mathrm{len}(KMA_{\lfloor n/2 \rfloor}) - \mathrm{len}(KMA_{\lceil n/2 \rceil}) - \mathrm{len}(KMA_{\lceil n/2 \rceil+1})$. Using the already calculated lengths of the parameterized instruction sequences defined in Section 5, we obtain by simple calculations that $c_1 = 75$, $c_2 = 240$, $c_3 = 495$, and for each $n > 3$, $c_n = 281 \cdot \lceil n/2 \rceil + 146 \cdot n + 298$. Let $c'_0 = c_3$, $c''_0 = c_3$, and for each $m > 0$, $c'_m = c_{2^m+2}$ and $c''_m = c_{2^{m+1}}$. In other words, $c'_0 = 495$, $c''_0 = 495$,

and for each $m > 0$, $c'_m = 573 \cdot 2^{m-1} + 871$ and $c''_m = 573 \cdot 2^m + 298$. 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 $\mathrm{len}(KMA_N) \leq \sum_{i=0}^{M}(c'_i \cdot 3^{M-i})$. In other words, $\mathrm{len}(KMA_N) \leq 495 \cdot 3^M + \sum_{i=1}^{M}((573 \cdot 2^{i-1} + 871) \cdot 3^{M-i})$. Using elementary properties of sums and the property that $\sum_{i=0}^{k} x^i = (1-x^{k+1})/(1-x)$, we obtain $495 \cdot 3^M + \sum_{i=1}^{M}((573 \cdot 2^{i-1} + 871) \cdot 3^{M-i}) = 495 \cdot 3^M + 573 \cdot (3^M - 2^M) + 871 \cdot ((3^M - 1)/2) < 1504 \cdot 3^M - 573 \cdot 2^M - 435$. Hence, because $M = \lceil \log_2(N-2) \rceil$, $\mathrm{len}(KMA_N) < 1504 \cdot 3^{\lceil \log_2(N-2) \rceil} - 573 \cdot 2^{\lceil \log_2(N-2) \rceil} - 435$.

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 $\mathrm{len}(KMA_N) \geq \sum_{i=0}^{M'}(c''_i \cdot 3^{M'-i})$. In other words, $\mathrm{len}(KMA_N) \geq 495 \cdot 3^{M'} + \sum_{i=1}^{M'}((573 \cdot 2^i + 298) \cdot 3^{M'-i})$. Using the same properties of sums as before, we obtain $495 \cdot 3^{M'} + \sum_{i=1}^{M'}((573 \cdot 2^i + 298) \cdot 3^{M'-i}) = 495 \cdot 3^{M'} + 573 \cdot (2 \cdot (3^{M'} - 2^{M'})) + 298 \cdot ((3^{M'} - 1)/2) = 1790 \cdot 3^{M'} - 1146 \cdot 2^{M'} - 149$. Hence, because $M' = \lfloor \log_2(N) \rfloor - 1$, $\mathrm{len}(KMA_N) \geq 1790 \cdot 3^{\lfloor \log_2(N) \rfloor - 1} - 1146 \cdot 2^{\lfloor \log_2(N) \rfloor - 1} - 149$. □

For $N \geq 1$, the instruction sequence $LMUL_N$ described by $MUL_N(I_1, I_2, O)\,;!$ computes according to the long multiplication algorithm the function on bit strings of length N that models the multiplication of two natural numbers less than 2^N on their representations in the binary number system. We have that $\mathrm{len}(LMUL_N) = 45 \cdot N^2 + 30 \cdot N + 1$. Using the lower estimate and upper estimate for the length of $KMUL_N$ given above, it is easy to check that $KMUL_N$ becomes more efficient than $LMUL_N$ for some N between 2^8 and 2^{13}. More accurate bounds are 290 and 7293. However, the latter bounds are not very useful, because it is very possible that the length of both $LMUL_N$ and $KMUL_N$ can be reduced somewhat.[3]

It is obvious that $KMUL_N$ and $LMUL_N$ 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 $10 \cdot N \cdot \lceil \log_2(N-2) \rceil + 18 \cdot N + 1$ and the

[3] A statement from [16], which is cited on many other webpages, is the following: "Karatsuba is usually faster when the multiplicands are longer than 320-640 bits". To our knowledge, this statement is not justified anywhere.

number of auxiliary registers used by $LMUL_N$ is only $4 \cdot N + 1$. In the instance that $N = 2^{13}$, these numbers are 1212417 (which corresponds to ±148K bytes) and 32769 (which corresponds to ±4K bytes), respectively.

7 Concluding Remarks

By means of terms from the algebraic theory of single-pass instruction sequences known as PGA, we have uniformly described, for $N \geq 3$, an instruction sequence $KMUL_N$ that computes according to the Karatsuba multiplication algorithm the product of two natural numbers less than 2^N represented in the binary number system. Thus, we have provided a mathematically precise alternative to the natural language and pseudo code descriptions of the Karatsuba multiplication algorithm found in mathematics and computer science literature on multiplication algorithms.

We have given a lower estimate and an upper estimate for the length of $KMUL_N$. From this it follows that, for all $N \geq 3$, $\mathrm{len}(KMUL_N) = \Theta(3^{\log_2(N)}) = \Theta(N^{\log_2(3)})$. This can be paraphrased as "the non-uniform time complexity of the Karatsuba multiplication algorithm is of the same order as its uniform time complexity", because it is shown in [6] that, in the case of instruction sequences of the kind that we have dealt with in this paper, instruction sequence length is a computational complexity measure that coincides with the classical non-uniform time complexity measure based on Turing machines that take advice.

We expect that the number of auxiliary registers used by instruction sequence is a computational complexity measure closely related to non-uniform space complexity. An option for future work is investigating the possible role of this complexity measure in devising new multiplication algorithms. The quarter square multiplication algorithm has been devised to reduce the non-uniform space complexity of multiplication algorithms based on table look-up for obtaining the product of two natural numbers represented in the binary number system (see e.g. [11]). However, to our knowledge, no multiplication algorithm has been devised to reduce the space complexity of multiplication algorithms other than those based on table look-up.

Acknowledgements

We thank Dimitri Hendriks from the VU University Amsterdam for carefully reading a draft of this paper and for pointing out an error in it.

References

1. Bergstra, J.A., Loots, M.E.: Program algebra for sequential code. Journal of Logic and Algebraic Programming 51(2), 125–156 (2002)
2. Bergstra, J.A., Middelburg, C.A.: Indirect jumps improve instruction sequence performance. Scientific Annals of Computer Science 22(2), 253–265 (2012)

3. Bergstra, J.A., Middelburg, C.A.: Instruction sequence processing operators. Acta Informatica 49(3), 139–172 (2012)
4. Bergstra, J.A., Middelburg, C.A.: Instruction Sequences for Computer Science, Atlantis Studies in Computing, vol. 2. Atlantis Press, Amsterdam (2012)
5. Bergstra, J.A., Middelburg, C.A.: On the expressiveness of single-pass instruction sequences. Theory of Computing Systems 50(2), 313–328 (2012)
6. Bergstra, J.A., Middelburg, C.A.: Instruction sequence based non-uniform complexity classes. `arXiv:1301.3297v2 [cs.CC]` (January 2013)
7. Bergstra, J.A., Middelburg, C.A.: Instruction sequence expressions for the secure hash algorithm SHA-256. `arXiv:1308.0219v5 [cs.PL]` (August 2013)
8. Bergstra, J.A., Ponse, A.: An instruction sequence semigroup with involutive anti-automorphisms. Scientific Annals of Computer Science 19, 57–92 (2009)
9. Cook, S.A.: On the Minimum Computation Time of Functions. Ph.D. thesis, Harvard University, Cambridge, MA (1966)
10. Fürer, M.: Faster integer multiplication. SIAM Journal of Computing 39(3), 979–1005 (2009)
11. Johnson, E.L.: A digital quarter square multiplier. IEEE Transactions on Computers C-29(3), 258–261 (1980)
12. Karatsuba, A.A.: The complexity of computations. Proceedings of the Steklov Institute of Mathematics 211, 169–183 (1995)
13. Karatsuba, A.A., Ofman, Y.P.: Multiplication of multidigit numbers on automata. Doklady Akademii Nauk SSSR 145(2), 293–294 (1962), in Russian
14. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing 7(3–4), 281–292 (1971)
15. Toom, A.A.: The complexity of a scheme of functional elements simulating the multiplication of integers. Doklady Akademii Nauk SSSR 150(2), 496–498 (1963), in Russian
16. Karatsuba algorithm. In Wikipedia (2013), retrieved on November 11, 2013, from `http://en.wikipedia.org/wiki/Karatsuba_algorithm`