JOURNAL OF COMPUTER AND SYSTEM SCIENCES 45, 285–295 (1992)

Minimum-Complexity Pairing Functions*

KENNETH W. REGAN

Department of Computer Science,
State University of New York at Buffalo, 226 Bell Hall, Buffalo, New York 14260

Received September 28, 1988; revised August 28, 1990

Pairing functions are bijections from $\mathbb{N}\times\mathbb{N}$ to $\mathbb{N}$, and they are important in logic, computing, and mathematics on the whole. We exhibit the first known pairing function $\langle\cdot,\cdot\rangle$ which is computable in linear time and constant space. In fact, both $\langle\cdot,\cdot\rangle$ and its inverse are computable by finite-state transducers which run in real time. By contrast, the familiar examples of pairing functions in the literature are computable in linear time if and only if integer multiplication can be accomplished in linear time, which is considered doubtful by many. We also present two kinds of monotone pairing functions which are computable on-line in linear time and log space; the first is also computable off-line in zero space. We conjecture that every monotone pairing function requires log space to compute on-line.

1. INTRODUCTION

What is the easiest way to compute a bijection $\langle\cdot,\cdot\rangle$ from $\mathbb{N}\times\mathbb{N}$ to $\mathbb{N}$? Pictorially put, what is the most efficient rule for numbering the cells of an $\omega\times\omega$ chessboard? Tracing with pencil and paper, one might choose one of the following patterns:

x \ y	1	2	3	4
1	1	2	4	7
2	3	5	8	·
3	6	9	·	·
4	10	·	·	·

x \ y	1	2	3	4
1	1	2	5	10
2	4	3	6	11
3	9	8	7	·
4	·	·	·	·

x \ y	1	2	3	4
1	1	2	5	10
2	3	4	7	·
3	6	8	9	·
4	11	·	·	·

What matters, however, is the ease or difficulty of computing $\langle x, y\rangle$ given x and y. Each of the above rules has the possible drawback of being *hard for* the problem of multiplying two given integers, because (i) this reduces to squaring in linear time via the identity $a\cdot b=\frac{1}{2}[(a+b)^2-a^2-b^2]$, and (ii) given z, one can compute z^2 via calls to $\langle z, 1\rangle$, $\langle z, z\rangle$, and/or $\langle z-1, 1\rangle$. It is a major open question whether multiplication can be accomplished in linear time on the Turing machine model, at

* Part of this research was conducted at Cornell University, supported by the U.S. Army Research Office through the Cornell Mathematical Sciences Institute, Contract DAAG 29-85-C-0018.

0022-0000/92 $5.00

least when x and y are given their standard representations as binary strings. The familiar grade-school algorithm takes time $O(n^2)$, while the best known upper bound is $O(n \cdot \log n \cdot \log\log n)$ (see [ScSt71]). Every pairing function we have seen in the literature, spanning [Rog67, BJ74, MY78, Cu80, LP81, and Rose84], is integer-multiplication-hard in this sense. Subject to a negative answer to this question, a linear-time computable pairing function $\langle \cdot, \cdot \rangle$ must distribute not only the squares, but every set $S \subseteq \mathbb{N}$ which forms a linear-time oracle for multiplication so irregularly that the appropriate arguments x, y for $\langle \cdot, \cdot \rangle$ cannot be found from a, b in linear time.

Nevertheless, here is a linear-time pairing function which ought to be considered "folklore," though we know of no reference for it: Think of a natural number $n > 0$ as the string $\mathrm{str}(n) \in \Sigma^*$, where $\Sigma := \{0, 1\}$, obtained by writing n in base-two notation and deleting the leading "1." E.g., $\mathrm{str}(1) = \lambda$, the *empty string*. Given $x, y \in \Sigma^*$, write xy for the concatenation of x and y (not multiplication), and $|x|$ for the *length* of x in bits (not absolute value). Then define

$$H(x, y) := wxy, \qquad \text{where} \quad w := \mathrm{str}(2\,|x| + |y| - 1). \tag{1.1}$$

The numerical graph of $H(\cdot, \cdot)$ is intuitively an "exponential stretching" of the first pattern above, proceeding along diagonal lines of constant $|x| + |y|$ rather than constant $x + y$. Patching around the three undefined values for $x = \lambda$ and $|y| \leq 1$ then yields a linear-time computable pairing function. Defining $H'(x, y) := xyw$ instead and doing the same patch yields the pairing function called $\langle \cdot, \cdot \rangle_l$ in Section 4.

However, these functions are still not the most efficient. For instance, no TM T computing $H(x, y)$ can start writing the output value wxy until it has read virtually all of the input x, y. If T runs *on-line* (meaning: without a reversal of the input head), then T needs linear space to store x and y. By contrast, H' (and hence $\langle \cdot, \cdot \rangle_l$) is computable in log space on-line. However, H' then requires linear space on-line to *invert*. (We leave the interested reader to verify these two assertions, which are not used later.)

It would be best for T to operate in *real time*, so that a bit is read and some bits are output at each step, with *zero* storage requirement. Seeking this amounts to asking: can a pairing function be computed by a finite-state transducer? The main point of this paper is that the answer is *yes*. Moreover, and unlike $\langle \cdot, \cdot \rangle_l$, the pairing function $\langle \cdot, \cdot \rangle_0$ we construct can be inverted as easily as it can be computed. Except for quibbles, $\langle \cdot, \cdot \rangle_0$ and $\langle \cdot, \cdot \rangle_0^{-1}$ use the absolute minimum in each of time, space, and reversals.

However, $\langle \cdot, \cdot \rangle_0$ fails to be *monotone* in its second argument and increases rather rapidly (quadratically) in its first argument. We also construct a related linear-time/zero-space pairing function $\langle \cdot, \cdot \rangle_m$ which is monotone in both arguments, but computing $\langle \cdot, \cdot \rangle_m$ requires one input tape head reversal. We conjecture that any TM which computes a monotone pairing function while running *on-line* must use

logarithmic space. Any refutation of this conjecture would be a mathematically interesting construction.

Section 2 presents background information, especially on finite-state transducers. Section 3 constructs $\langle \cdot, \cdot \rangle_0$ and verifies the properties claimed for it. Section 4 shows that several other pairing functions cannot be computed as easily as $\langle \cdot, \cdot \rangle_0$, and gives a few open problems. Section 5 raises some possible applications for this work.

2. Preliminaries

From now on we consider natural numbers to be strings over the alphabet $\Sigma := \{0, 1\}$ under the correspondence *str* defined in Section 1. This induces the sequence λ, 0, 1, 00, 01, 10, 11, 000, ..., which is often called the *standard ordering* of Σ^*.

A *pairing function* $\langle \cdot, \cdot \rangle$ is a bijection from $\Sigma^* \times \Sigma^*$ to Σ^*. We also consider $\langle \cdot, \cdot \rangle$ to be a bijection from $\Sigma^* \# \Sigma^*$ to Σ^*, where "#" is a special *separator symbol.* The associated *projection functions* π_1 and π_2 are defined for all $x, y \in \Sigma^*$ by: $\pi_1(\langle x, y \rangle) = x$, $\pi_2(\langle x, y \rangle) = y$. The function is *monotone* (with respect to the standard ordering of Σ^*, and in both arguments) if $x_1 \leqslant x_2 \wedge y_1 \leqslant y_2 \Rightarrow \langle x_1, y_1 \rangle \leqslant \langle x_2, y_2 \rangle$ for all $x_1, x_2, y_1, y_2 \in \Sigma^*$.

Our model of a (multitape) deterministic Turing machine T is the standard one of [HU79], whereby T has a single read-only input tape, a single write-only one-way output tape, and an arbitrary finite number of worktapes. We consider TMs which have input alphabet $\Sigma_I := \{0, 1, \#\}$, output alphabet $\Sigma_O := \{0, 1\}$, and a worktape alphabet which contains Σ_I and the *blank* $\not\Delta$, but is otherwise arbitrary. All tapes are semi-infinite, extending to the right. To represent the arguments x, y of $\langle \cdot, \cdot \rangle$ it might be more realistic to have two input tapes, but we shall not be hurt by abiding with the single-tape constraint and regarding "$x \# y$" as the input to T. It would also be realistic to suppose that the first cells of the input and work tapes have a special endmarker symbol which is detectable by the machine. However, we allow the common convention that if a TM attempts to move a head past the left end of a tape, the head stays where it is, and we remark on this following Theorem 4.1.

We consider machines T which obey a second convention from [HU79], namely, that a TM computation may only halt in a special halting state q_h with the input head scanning the blank cell to the right of the input. Note that any halting computation of T on an input x takes at least $|x| + 1$ transitions. It is easy to place a given TM into this form.

The *time* used in a computation of T equals the total number of transitions. The *space* used is the total number of *worktape* cells written to; that is, it includes neither the input tape nor the output tape cells. The *reversal count* is the number of times that a worktape or input tape head moves left when its previous *move* has been right, or right when its previous move has been left. If its input head never

moves left in any computation, then T is said to run *on-line*. For $k \geqslant 1$, a TM T runs in *delay* k if it runs on-line, and in any k consecutive transitions in any computation there is one in which the input head moves right. T runs in *real time* iff T runs in delay 1. Without loss of generality we may suppose that (the finite control of) an on-line TM has no transitions calling for the input head to move left, and that a real-time TM has only transitions calling for it to move right.

For any TM S which runs in constant space, we can find an equivalent TM T which never moves or writes with its worktape head(s) at all, by coding the finitely-many different possible worktape configurations of S into the finite control of T. The following specification for T is more general than various formalizations of a "finite-state transducer" which have appeared in the literature. We distinguish between the blank $\not\lambda$ as an input character and the empty string λ as output, and read "L," "R," and "S" respectively as left, right, or stationary.

DEFINITION 2.1. A *zero-space transducer* (ZST) T consists of an input tape, an output tape, and state transitions of the form $(q, a; w, d, r)$, where

$q \in Q \setminus \{q_h\}$	is the current state
$a \in \Sigma_I \cup \{\not\lambda\}$	is an input character
$w \in \Sigma_O \cup \{\lambda\}$	is an output character or the empty string
$d \in \{\text{L}, \text{R}, \text{S}\}$	denotes the movement of the input head, and
$r \in Q$	is the next state.

T is a *generalized* zero-space transducer (GZST) if we allow w to be any string in Σ_O^*.

We restrict attention to GZSTs which are deterministic, and write $T(x) = y$ iff the computation of T on input x reaches state q_h with y on the output tape.

DEFINITION 2.2. (a) A GZST T is *prompt* if every tuple $(q, \not\lambda; w, d, r)$ has $r = q_h$ and $w = \lambda$, and *nonerasing* if for every tuple $(q, a; w, d, r)$, $a \neq \not\lambda \Rightarrow w \neq \lambda$.

(b) The *lag* of an on-line TM M is the maximum number of symbols which M can print when started in any state on blank input.

The extra powers of a GZST T over a ZST are that T can move its input head left or keep it stationary, and that T can produce output and continue operation after reading the blank cell which marks the end of the input. Two more-familiar notions of finite transductions (see [HU79, DDQ78]) come from the following restrictions: A *Mealy machine* equals a realtime, prompt, nonerasing ZST (and so produces output of length equal to that of its input). A *deterministic generalized sequential transducer* (GST) equals a realtime, prompt GZST.

The ability of a GZST to output any string in one step allows us to use a "normal form" for *on-line* GZSTs T, in which all tuples $(q, \not\lambda; w, d, r)$ either have $r = q_h$

and $d =$ "R," or lead to a special infinite-looping state. This normal form is computed by looking ahead to all transitions on $\not\lambda$ which follow any tuple of the form $(q, \not\lambda; v, d, r)$, $v \in \Sigma_O \cup \{\lambda\}$. Then the lag of T equals $\max\{|w| \mid (q, \not\lambda; w, \mathrm{R}, q_h)$ is a tuple of T in this normal form$\}$, and T is prompt iff T has lag 0. Furthermore, if a GZST T runs on-line and computes a total function, then T runs in delay k, where $k \leqslant |Q|$. These observations are the gist of the following elementary result, which ties together the various models for our purposes and justifies our claim to have a real-time computable pairing function.

PROPOSITION 2.1. *The following are equivalent for any total function* $f: \Sigma_I^* \to \Sigma_O^*$:

(a) *f is computed by an on-line GZST T.*

(b) *f is computed by an on-line ZST T'.*

(c) *f is computed by a real-time GZST T''.*

Moreover, the simulations among T, T', and T'' preserve promptness and nonerasure. In particular, any total function f which is computable on-line by a prompt ZST is computable by a GST.

The proof is left to the reader. We remark that the equivalence of on-line and off-line finite-state acceptors (see [HU79]) does not extend to prompt (G)ZSTs: consider a machine which on any input x notices the first occurrence of "00" if any, advances to the next occurrence of "00," and copies the intervening string to the output while moving left to the first "00." We may now state the main theorem in full detail.

MAIN THEOREM 2.2. *There is a bijection* $\langle \cdot, \cdot \rangle_0: \Sigma^* \times \Sigma^* \to \Sigma^*$ *which, when represented as a function from* $\Sigma^* \# \Sigma^*$ *to* Σ^*, *is computable by a prompt, real-time GZST which outputs at most two bits per step. In addition, its inverse* $\langle \cdot, \cdot \rangle_0^{-1}$: $\Sigma^* \to \Sigma^* \# \Sigma^*$ *is computable by a ZST in real time. In particular,* π_1 *is computed by a real-time ZST which is prompt, while* π_2 *is computed by a real-time ZST with lag* 1.

COROLLARY 2.3. $\langle \cdot, \cdot \rangle_0$ *is computable by a prompt ZST which runs in delay* 2.

The automata for $\langle \cdot, \cdot \rangle_0$ can be made to run with equal efficiency when x and y are placed on separate tapes (and then become non-erasing), but no better. Hence our result for the "$x \# y$" representation is a strength, not a restriction. We give the proof, plus additional technical observations, in the next section.

3. A PAIRING FUNCTION $\langle \cdot, \cdot \rangle_0$ OF MINIMUM COMPLEXITY

The basic idea is to pair strings x and y by doubling each bit of x and appending y. The difficulty is that y may itself begin with a doubled bit. To mark the dividing

point, we insert an "opposite bit" either before or after the first bit of y (if $y \neq \lambda$); e.g., taking $x :=$ "101" and $y :=$ "000" gives "1100111000" under the former and "1100110100" under the latter. Neither rule gives a bijection, however; e.g., "1100110" is not in the range of either. The two strings $y =$ "0" and $y =$ "1" cause the problem. We "patch around them" by taking z to be the second predecessor of y (in *some* ordering of Σ^*) when $|y| \geq 2$ and apply the insertion to z instead. First we define inductively:

DEFINITION 3.1. Component functions for pairing functions and their inverses:

$d(x)$: $d(\lambda) := \lambda$, $d(0x) := 00d(x)$, $d(1x) := 11d(x)$.

$i_1[x]$: $i_1[0x] := 10x$, $i_1[1x] := 01x$; $i_1[\lambda]$ is undefined.

$i_2[x]$: $i_2[0x] := 01x$, $i_2[1x] := 10x$; $i_2[\lambda]$ is undefined.

$j_1[x]$: $j_1[0x] := x$, $j_1[1x] := x$; $j_1[\lambda]$ is undefined.

$j_2[x]$: $j_2[00x] := 0x$, $j_2[01x] := 0x$, $j_2[10x] := 1x$, $j_2[11x] := 1x$; $j_2[y]$ is undefined for $y \in \{\lambda, 0, 1\}$.

Descriptive names are respectively "double bits," "insert in the first place," "insert in the second place," "jettison the first bit," and "jettison the second bit." The last function we need is a bijection $b(\cdot)$ from strings of length ≥ 2 to strings of length ≥ 1.

LEMMA 3.1. *Let $b(\cdot)$ be a bijection from $\Sigma^* \setminus \{\lambda, 0, 1\}$ to $\Sigma^* \setminus \{\lambda\}$, and let $i[\cdot]$ be either of the functions i_1, i_2 above. Then the mapping*

$$P(x, y) := \begin{cases} d(x)\, y & \text{if } y \in \{\lambda, 0, 1\} \\ d(x)\, i[b(y)] & \text{otherwise} \end{cases} \tag{3.1}$$

defines a bijection from $\Sigma^* \times \Sigma^*$ to Σ^*.

Proof. Clearly $P(x, y)$ is well defined for all $x, y \in \Sigma^*$. If $i = i_1$ then let $j[\cdot]$ be the function j_1 defined above; else if $i = i_2$ then $j := j_2$. Define the functions $\pi_1, \Pi_2 : \Sigma^* \to \Sigma^*$ for all $z \in \Sigma^*$ by:

$$\pi_1(z) := \text{the longest } x \text{ such that for some } w,\ z = d(x)w; \tag{3.2}$$

$$\Pi_2(z) := \begin{cases} w & \text{if } w \in \{\lambda, 0, 1\} \\ b^{-1}(j[w]) & \text{otherwise.} \end{cases} \tag{3.3}$$

(Note. Here w in (3.3) comes from (3.2), and capital letters denote quantities which depend on the choice of $b(\cdot)$ and $i[\cdot]$.) For any z, $\pi_1(z)$ is computed by removing all adjacent pairs of like bits from the left side of z, and then w is well defined as the string left over. Then $\Sigma^* \setminus \{\lambda, 0, 1\} \subseteq \mathrm{Dom}(j)$ (even when $j := j_2$); and because

$\mathrm{Ran}(j) = \Sigma^* \setminus \{\lambda\} = \mathrm{Dom}(b^{-1})$, $\Pi_2(z)$ is well defined. Now the reader may verify that $P(\cdot, \cdot)$ is both injective and surjective, with inverse (π_1, Π_2). ▮

It is readily apparent that $d(\cdot)$, its quasi-inverse $\pi_1(\cdot)$, and the simple bit-inserting and deleting operations $i[\cdot]$ and $j[\cdot]$ are all computable by zero-space transducers in delay 2. Hence it remains to optimize $b(\cdot)$ and make sure that its composition with $i[\cdot]$ loses no efficiency. Simply taking $b(y)$ to be $y-2$ (that is, $\mathrm{str}(\mathrm{str}^{-1}(y)-2)$) yields the pairing function $\langle \cdot, \cdot \rangle_m$ in the next section, but we show that this requires a head reversal to compute in zero space, intuitively because the "carries" proceed in the wrong direction.

To eliminate the reversal, let $\phi: \Sigma^* \to \Sigma^*$ be the function which reverses the string given as argument; e.g., $\phi[10110] = 01101$. Then define "backward subtraction by 2" for all $x \in \Sigma^*$ by $b_{-2}(x) := \phi[\phi[x]-2]$; undefined for $x = \lambda$ or $x =$ "0." This clearly yields a bijection from $\Sigma^* \setminus \{\lambda, 0, 1\}$ to $\Sigma^* \setminus \{\lambda\}$, and its inverse b_{+2} is defined for all x by $b_{+2}(x) := \phi[\phi[x]+2]$.

DEFINITION 3.2. For all $x, y, z \in \Sigma^*$ define

$$\langle x, y \rangle_0 := \begin{cases} d(x)\, y & \text{if } y \in \{\lambda, 0, 1\}, \\ d(x)\, i_2[b_{-2}(y)] & \text{otherwise,} \end{cases} \tag{3.4}$$

$$\pi_1(z) := \max\{x \in \Sigma^* \mid (\exists w \in \Sigma^*)[z = d(x)w]\}, \tag{3.5}$$

$$\pi_2(z) := \begin{cases} w & \text{if } w \in \{\lambda, 0, 1\} \\ b_{+2}(j_2[w]) & \text{otherwise.} \end{cases} \tag{3.6}$$

Here again, w in (3.6) is the unique string such that $z = \pi_1(z)w$, and the max in (3.5) is taken with respect to length. For some examples: $\langle x, \lambda \rangle = d(x)$ for all $x \in \Sigma^*$; in particular, $\langle \lambda, \lambda \rangle_0 = \lambda$. $\langle \lambda, 111 \rangle_0 = i_2[b_{-2}(111)] = i_2[101] = 1001$, and $\langle 101, 1000 \rangle_0 = 110011 \cdot i_2[111] = 1100111011$. For the inverses, $\pi_1(01) = \lambda$, $\pi_2(01) = b_{+2}(\text{"0"}) = 00$, and $\pi_2(0000111010101) = b_{+2}(j_2[1010101]) = b_{+2}(110101) = 101101$.

Proof of the Main Theorem 2.2. By Lemma 3.1, the function $\langle \cdot, \cdot \rangle_0$ is a bijection from $\Sigma^* \times \Sigma^*$ to Σ^*, and π_1, π_2 are the associated projection functions. We can program a real-time GZST T to operate as follows on input $x \# y$: First print each bit of x twice until reading the "#" symbol. If y is empty then halt; else print and remember the first bit of y. If there is no second bit of y then halt; else print the opposite bit to the one remembered—*and*—if the current bit of y is a "1," then also print "0" to accomplish the subtraction, move right, and enter a "copy" state which prints the remaining bits of y until the end of y is reached. Else, if the current bit is "0," then move right and enter a "carry" state. In the carry state, T halts if it reads the blank at the end of y, prints "10" and enters the copy state if the current bit of y is "1," and prints "1" and remains in the carry state if the bit is "0."

From this description, T moves its input head right at each step, and is prompt.

The constructions of a prompt real-time ZST computing π_1, and a real-time ZST computing π_2 with lag 1, are immediate from the above descriptions. ∎

Some Technical Notes. The lag of 1 for π_2 is best possible, since $\pi_2(00) = \lambda$ while $\pi_2(0) \neq \lambda$. We do not know of a real-time, lag-free pairing function, both of whose inverses are likewise lag-free. The state used to remember that the first bit of y was "0" is actually equivalent to the "carry" state, and so T requires only five states. Both T and the equivalent delay-2 ZST T' produced by Proposition 2.1 also output at least one bit every two steps. In fact, T and T' only output λ when reading the "#" symbol; this justifies a remark following Corollary 2.3.

To set up the motivation for the next section we list some additional properties of the pairing function $\langle \cdot, \cdot \rangle_0$. The proof is by inspection.

PROPOSITION 3.2. *For all* $x, y, z \in \Sigma^*$,

(a) $|x| + |y| \leqslant |\langle x, y \rangle_0|$, *with equality iff* $x = \lambda$ *and* $y \in \{\lambda, 0, 1\}$.

(b) $2|x| + |y| \leqslant |\langle x, y \rangle_0| \leqslant 2|x| + |y| + 1$.

(c) *If* $x \leqslant z$ *(in the standard ordering of* Σ^**), then* $\langle x, y \rangle_0 \leqslant \langle z, y \rangle_0$.

4. Monotone Pairing Functions and Log-Space Requirements

We use the tools in Section 3 to construct several more linear-time pairing functions, which have both advantages and disadvantages relative to $\langle \cdot, \cdot \rangle_0$. Given $x, y \in \Sigma^*$, define $w(x, y) := \mathrm{str}(2|x| + |y| - 1)$ as in (1.1). It is interesting that $w(\lambda, y)$ is undefined for the same three cases $y \in \{\lambda, 0, 1\}$ that require the subtract-2 patch in the definition of $\langle \cdot, \cdot \rangle_0$.

DEFINITION 4.1. For all $x, y \in \Sigma^*$ define:

$$\langle x, y \rangle_m := \begin{cases} d(x)\, y & \text{if } y \in \{\lambda, 0, 1\}, \\ d(x)\, i_2[y - 2] & \text{otherwise,} \end{cases} \tag{4.1}$$

$$\langle x, y \rangle_l := \begin{cases} y & \text{if } x = \lambda \text{ and } y \in \{\lambda, 0, 1\}, \\ b_{-2}(x \cdot y \cdot w(x, y)) & \text{otherwise,} \end{cases} \tag{4.2}$$

$$\langle x, y \rangle_r := \begin{cases} d(x)\, y & \text{if } y \in \{\lambda, 0, 1\}, \\ d(x)\, i_1[b_{-2}(y)] & \text{otherwise.} \end{cases} \tag{4.3}$$

THEOREM 4.1. (a) $\langle \cdot, \cdot \rangle_m$ *is a monotone pairing function which is computable by a zero-space transducer in linear time, with one input head reversal. However, any TM which computes* $\langle \cdot, \cdot \rangle_m$ *on-line requires log space.*

(b) $\langle \cdot, \cdot \rangle_l$ *is monotone and computable on-line in linear time and log space. However, any TM which computes* $\langle \cdot, \cdot \rangle_l$ *on-line does require log space and runs with unbounded lag.*

(c) $\langle \cdot, \cdot \rangle_r$ *is monotone with respect to the reverse standard ordering of* Σ^*, *and enjoys the same minimum complexity as* $\langle \cdot, \cdot \rangle_0$.

Proof. The bijectivity of these functions and the upper bounds on their complexity follows from the results in Section 3. The on-line log-space lower bounds are obtainable by straightforwardly extending arguments of [HLS65, HU69] so that they work for transducers. (The modification is to show that a function $f: \Sigma_I^* \to \Sigma_O^*$ can be computed on-line in $o(\log n)$ space only if there is some $k \geq 0$ such that for all strings z and prefixes y of z, deleting the last k bits of $f(y)$ yields a prefix of $f(z)$. Then one can show that neither $\langle \cdot, \cdot \rangle_m$ nor $\langle \cdot, \cdot \rangle_l$ has this property.) The function $\langle \cdot, \cdot \rangle_r$ is obtained merely by substituting i_1 for i_2 in the definition of $\langle \cdot, \cdot \rangle_0$. ∎

Some Technical Remarks. Here, the ability to compute the $i_2[y-2]$ component of $\langle \cdot, \cdot \rangle_m$ off-line in zero space depends critically on the presence of the "#" sign on the input tape as an endmarker. A finite-state transducer T' with separate input tapes for x and y cannot compute $\langle \cdot, \cdot \rangle_m$ unless one replaces the convention adopted in Section 2 by one allowing T' to detect the left end of a tape. The projection functions for $\langle \cdot, \cdot \rangle_m$ and $\langle \cdot, \cdot \rangle_r$ are just as easy to compute as the corresponding pairing function, but those for $\langle \cdot, \cdot \rangle_l$ appear to require *linear* space to compute on-line, as asserted in Section 1. $\langle \cdot, \cdot \rangle_0$ itself is not monotone under the reverse ordering of Σ^*. Substituting i_1 for i_2 in $\langle \cdot, \cdot \rangle_m$ rather than $\langle \cdot, \cdot \rangle_0$ yields the pairing function originally given in [Reg86a].

Taken together, Theorem 4.1(a)–(c) say that the right-to-left bias in the standard ordering of Σ^* and the left-to-right movement of Turing machines are to blame for the apparent lack of a monotone real-time pairing function. J. Case and J. Royer [CaRo86] have constructed a monotone GZST-computable pairing function using a "bit-interlacing" scheme, but like $\langle \cdot, \cdot \rangle_l$ it is not computable on-line in less than log space. The right–left conflict comes down to whether the least or most significant bits of a natural number should be written first. It is natural to ask whether this conflict can be overcome.

Open Question. With reference to the standard ordering of Σ^*, is it possible to compute a monotone pairing function on-line in zero space?

We consider Theorem 4.1 to be evidence that the answer is "no," but have been unable to extend the arguments on TM computations based on [HLS65, HU69] to prove so general a statement. On the other hand, a construction giving a "yes" answer would be interesting even for purely numerical reasons.

Taking $m := |x| + |y|$, Proposition 3.2(b) gives the bound $|\langle x, y \rangle_0| \leq 2m+1$ for all $x, y \in \Sigma^*$, and the same applies to $\langle \cdot, \cdot \rangle_m$ and $\langle \cdot, \cdot \rangle_r$. On the other hand, for any pairing function $\langle \cdot, \cdot \rangle$ and function $f: \mathbb{N} \to \mathbb{N}$ such that $|\langle x, y \rangle| \leq m + f(m)$ for all $x, y \in \Sigma^*$, elementary counting arguments show that f must be bounded below a.e. by $1 + \log_2 m$. This lower bound is achieved by $\langle \cdot, \cdot \rangle_l$, which intuitively packs in strings of the least possible lengths over each bound $n > 0$ for $|x| + |y|$. This suggests the

Open Question. Is there a zero-space computable pairing function $\langle \cdot, \cdot \rangle$ which satisfies $|\langle x, y \rangle| \leq m + O(\log m)$ for all $x, y \in \Sigma^*$, where $m := |x| + |y|$?

5. Conclusion and Prospects

Many papers in the literature of complexity theory contain a line to the effect of, "Let $\langle \cdot, \cdot \rangle$ be a pairing function which is polynomial-time computable." This can now be changed to "which is *linear*-time computable." Some applications for the savings in complexity are indicated in [Reg86a, Reg86b]. In parallel complexity one may be able to use the fact that these pairing functions are $\mathscr{AC}^0$-computable.

The schemes used to define the pairing functions in this paper have enough freedom to construct others, and some of these may yield interesting results. Some may serve as tools for proving that certain functions $f: \mathbb{N} \to \mathbb{N}$ are integer-multiplication-hard to compute, via showing that the set of values $\{\langle n, f(n) \rangle_m \mid n \in \mathbb{N}\}$ is an oracle for multiplication. They may also yield better schemes for allocating and accessing storage of two (and higher) dimensional arrays, drawing on criteria for functions from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ which have been formulated by A. Rosenberg and others (see [Ros77, RoSt77]).

Pairing functions have several applications in logic, notably in defining single *Gödel numbers* for sequences of objects in a formal system. One might expect that the lower the complexity of the pairing function, the better one may use it to analyze *weak* formal systems, such as those obtained by restricting induction in *Peano arithmetic*. Pending, however, is the development of closer connections between classes of functions defined machine-theoretically and classes of functions defined from low-level formal systems. Chapter 5 of [Rose84] presents some examples and pertinent open problems.

Last, say that a *discretely-ordered ring* $\mathbf{R} = (S, +, \cdot, \leqslant)$ is *linear-time computable* if the addition, multiplication, and ordering relation are linear-time computable. (We do not hold it necessary that S itself be accepted in linear time as a subset of Σ^*.) Not only is it open whether $(\mathbb{Z}, +, \cdot, \leq)$ is linear-time computable, it is not known whether any such ring $\mathbf{R}$ exists. Even though the translation from $\mathbb{Z}$ to $\mathbf{R}$ would likely take more than linear time to compute, there might still be savings in some long runs of numerical calculations because it would only need to be applied at the beginning and end. We inquire whether certain restrictions of these pairing functions to some set S can be combined into an efficient substitute for integer multiplication.

Acknowledgments

I especially thank the anonymous referee, for suggestions on greatly tightening the paper, and Professors John Case, Arnold Rosenberg, and James Royer, for information on their own work involving pairing functions and suggestions for further research.

Note added in proof. For any $k > 1$, the bound $|\langle x, y\rangle_0| \leqslant 2|x| + |y| + 1$ in Proposition 3.2(b) can be improved to $|\langle x, y\rangle_k| \leqslant (1 + 1/k)|x| + |y| + k$: Write $x = x_1 x_2 \cdots x_m x'$ where each x_i has length k and $|x'| < k$. Then one can construct a function $h_k(x', y)$ such that $\langle x, y\rangle_k := 0x_1 0x_2 \cdots 0x_m \cdot h_k(x', y)$ is a delay-2 prompt ZST pairing function which achieves the bound. The author thanks Yenjo Han of Rochester for a suggestion which prompted this realization, and for other comments on the paper. An on-line GZST is essentially the same as a *subsequential machine*; for this and related models, see C. Reutenauer and M. P. Schutzenberger, Minimization of rational word functions, *SIAM J. Comput.* **20** (1991), 669–685.

REFERENCES

[BJ74] G. BOOLOS AND R. JEFFREY, "Computability and Logic," Cambridge University Press, Cambridge, 1974.

[CaRo86] J. CASE AND J. ROYER, "Progressions of Relatively Succinct Programs in Subrecursive Hierarchies," Technical Report 86-007, Computer Science Department, The University of Chicago, 1986.

[Cu80] N. CUTLAND, "Computability," Cambridge University Press, Cambridge, 1980.

[DDQ78] P. DENNING, J. DENNIS, AND J. QUALITZ, "Machines, Languages, and Computation," Prentice-Hall, Englewood Cliffs, NJ, 1978.

[HLS65] J. HARTMANIS, P. LEWIS, AND R. STEARNS, Hierarchies of memory-limited computation, *in* "Proceedings, IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965" (now FOCS), pp. 179–190.

[HU69] J. HOPCROFT AND J. ULLMAN, Some results on tape-bounded Turing machines, *J. Assoc. Comput. Mach.* **16** (1969), 168–177.

[HU79] J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.

[LP81] H. LEWIS AND C. PAPADIMITRIOU, "Elements of the Theory of Computation," Prentice-Hall, New York, 1981.

[MY78] M. MACHTEY AND P. YOUNG, "An Introduction to the General Theory of Algorithms," North-Holland, New York, 1978.

[Reg86a] K. REGAN, The topology of provability in complexity theory, *in* "Proceedings, 2nd Annu. Conf. on Structure in Complexity Theory, Ithaca, NY, June 1986," Lect. Notes in Comput. Sci., Vol. 223, pp. 291–310, Springer-Verlag, Berlin, 1986.

[Reg86b] K. REGAN, "On the Separation of Complexity Classes," Doctoral dissertation, Oxford University, 1986.

[Rog67] H. ROGERS, "Theory of Recursive Functions and Effective Computability," McGraw-Hill, New York, 1967.

[Rose84] H. ROSE, "Subrecursion: Functions and Hierarchies," Oxford Logic Guides, No. 9, Clarendon Press, Oxford, 1984.

[Ros67] A. ROSENBERG, Real time definable languages, *J. Assoc. Comput. Mach.* **14**, No. 4 (1967), 645–662.

[Ros77] A. ROSENBERG, On storing concatenable arrays, *J. Comput. System Sci.* **14**, No. 2 (1977), 157–174.

[RoSt77] A. ROSENBERG AND L. STOCKMEYER, Storage schemes for boundedly extensible arrays, *Acta Inform.* **7** (1977), 289–303.

[ScSt71] A. SCHÖNHAGE AND V. STRASSEN, Schnelle Multiplikation grosser Zahlen, *Computing* (Arch. Elektron. Rechnen) **7** (1971), 281–292.