INFORMATIQUE THÉORIQUE ET APPLICATIONS

JEAN-CAMILLE BIRGET

Two-way automaton computations

Informatique théorique et applications, tome 24, nº 1 (1990), p. 47-66

<http://www.numdam.org/item?id=ITA_1990__24_1_47_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 24, n° 1, 1990, p. 47 à 66)

TWO-WAY AUTOMATON COMPUTATIONS (*)

by Jean-Camille BIRGET [1]

Communicated by J.-E. PIN

Abstract. – *Computations of a two-way automaton on an input tape are studied using the algebraic notion of trace of a two-way computation (due to J. P. Pécuchet), and certain "reductions" of traces.*

This paper only deals with two-way computations which begin and finish at one or the other of the two ends of the input tape. The traces of such computations are characterized, and formulas are given which tell how traces are combined when the corresponding inputs are concatenated.

Another tool for studying a two-way automaton (also due to Pécuchet) is the language of its control unit (considered over a "double alphabet"). This "control language" determines the entire two-way automaton and this leads to formulas relating the control language and the language accepted by the two-way automaton itself.

Résumé. – *Nous étudions les calculs d'un automate boustrophedon en employant la notion de trace d'un calcul (due à J. P. Pécuchet) et certaines « réductions » de traces.*

Cet article ne considère que les calculs boustrophedons qui commencent et finissent à l'un ou l'autre bout de la bande d'entrée. Nous décrivons les traces des calculs de ce type et donnons des formules qui indiquent comment se combinent les traces lorsque les entrées correspondantes sont concaténées.

Un autre instrument dans l'étude des automates boustrophedons (dû aussi à J. P. Pécuchet) est le langage de l'unité de contrôle (exprimé sur un « double alphabet »). Ce « langage de contrôle » détermine l'automate boustrophedon et ceci conduit à des formules reliant le langage de contrôle et le langage accepté par l'automate boustrophedon.

(*) Received June 1987, revised March 1988.

This research was supported in part by U.S. Army Grant DAAG-29-85-K-0099 and U.S. Air Force Grant AFOSR-85-0186 through the Center for Mathematical System Theory, University of Florida, Gainesville, FL 32611.

[1] Computer Science Department, University of Nebraska, Lincoln, NE 68588, U.S.A.

0988-3754 90/01 47 20/$4.00/© AFCET Gauthier-Villars

1. INTRODUCTION AND DEFINITIONS

This paper is a continuation of my paper [2] and of J. P. Pécuchet's work [7], but can be read independently. The main concern in [2] was the effect of entire input words on the states, and this was described by the global state transition maps $[\rightarrow u \rightarrow]$, $[\leftrightarrows]$, $[u \leftrightarrows]$, $[\leftarrow u \leftarrow]$ associated to any input u.

In this paper I study two-way automata from the point of view of their *computations* (rather than just the effect of the computations on the states). In order to do that, a slightly different model of a two-way automaton is introduced (similar to the one in [7]). In this paper I only study computations that begin and finish at one or the other end of the input tape (and not somewhere within the input). For such two-way automaton computations I give formulas that show how computations are combined as their corresponding inputs are concatenated. Interestingly there is a connection between two-way automaton computations and semigroup regularity (especially, the regular-semigroup construction of [3] and [4]).

Finally, I give a new presentation of Pécuchet's theorem about the relation between the language recognized by the two-way automaton and the language recognized by the control unit of the two-way automaton.

DEFINITIONS: In this paper the following model of a *two-way automaton* will be used: it is a structure $(\vec{Q}, Q, \Sigma, \Sigma \cup \bar{\Sigma}, \bullet)$ where Σ is the input alphabet (used on the tape of the two-way automaton); $Q = \vec{Q} \cup \overleftarrow{Q}$ is the set of states, where $\vec{Q}$ is the set of *right-moving states* and $\overleftarrow{Q}$ is the set of *left-moving states*. (The current state determines the possible direction(s) that the reading head will take in the next move.) In practice the number of states will be finite, but this assumption is usually not necessary in the constructions.

A two-way automaton can be viewed as a tape (containing an input string $\in \Sigma^*$) on which a reading head can move to the left and to the right, together with a *(finite) control unit*. This control unit is a sequential (one-way) automaton $(Q, \Sigma \cup \bar{\Sigma}, \bullet)$ where $Q = \vec{Q} \cup \overleftarrow{Q}$ (as above), $\Sigma \cup \bar{\Sigma}$ is the input alphabet of the control unit (where $\bar{\Sigma}$ is a copy of Σ, disjoint from Σ), and "$\bullet$" is the next-state operation. Intuitively the control unit works as follows: when the reading-head reads a letter $a \in \Sigma$ on the input tape, the reading head will transmit the letter $a \in \Sigma$ to the control unit, provided the current state belongs to $\vec{Q}$; if the current state belongs to $\overleftarrow{Q}$ then the letter $\bar{a} \in \bar{\Sigma}$ (the "barred" copy of $a \in \Sigma$) will be transmitted to the control unit. The next-state operation $\bullet$ is a relation $(q, c) \in Q \times (\Sigma \cup \bar{\Sigma}) \rightarrow q \bullet c \subseteq Q$. By the previous sentence, $q \bullet c$ is not defined ($q \bullet c = \varnothing$) if $q \notin \vec{Q}$ but $c \in \Sigma$, or if $q \notin \overleftarrow{Q}$ but $c \in \bar{\Sigma}$.

A current *configuration* of the two-way automaton is a word $a_1 \ldots a_i q a_{i+1} \ldots a_n \in \Sigma^* Q \Sigma^*$, where $a_1 \ldots a_i a_{i+1} \ldots a_n$ is the input string on the tape, and q is the current state, with the reading head positioned *between* cells i and $i+1$ of the tape.

The *next configuration* of the two-way automaton is obtained from the current configuration $a_1 \ldots a_i q a_{i+1} \ldots a_n$ as follows:

If the current state q belongs to $\vec{Q}$ then the reading head moves *right* and the input letter a_{i+1} is read and transmitted to the control unit. The new state q' is any element of $q \bullet a_{i+1}$ (or $q' = q \bullet a_{i+1}$ if $\bullet$ is a function, in the deterministic case). The reading-head will position itself between cells $i+1$ and $i+2$. So the next configuration will be $a_1 \ldots a_i a_{i+1} q' a_{i+2} \ldots a_n$.

If the current state q belongs to $\tilde{Q}$ then the reading-head moves *left*, reading the input letter a_i and transmitting the letter $\bar{a}_i \in \bar{\Sigma}$ to the control unit. The new state q'' is any element of $q \bullet a_i$ (or $q'' = q \bullet a_i$ if $\bullet$ is a function), and the reading-head will position itself between cells $i-1$ and i. So the next configuration will be $a_1 \ldots a_{i-1} q'' a_i \ldots a_n$.

A *two-way computation* on a given input word is a finite sequence of configurations of the two-way automaton in which each one is obtained from the previous one by application of the next-state relation.

The *trace* (Pécuchet [7]) of a two-way computation is the sequence of letters in the "double alphabet" $\Sigma \cup \bar{\Sigma}$ received by the control unit during that two-way computation. The trace of a computation is the "input" as seen by the control unit. Although the reading-head executes a two-way movement on the tape (with input $\in \Sigma^*$) there is a one-way flow of information from the reading head to the control unit; the information consists precisely of the trace ($\in (\Sigma \cup \bar{\Sigma})^*$) of the two-way computation.

Non-determinism arises when the next-state relation $\bullet$ is not a function, and also when $\vec{Q} \cap \tilde{Q} \neq \varnothing$.

Remark : The model of a two-way automaton used here is slightly different from the one used in most of the literature (e. g. [6], [8]). It is similar to the model that I used in [2], but in addition it incorporates Pécuchet's "alphabet doubling" idea (*see* his paper [7] and also Eilenberg [5], p. 285.) This change does not give the two-way automaton any increased power, but it will give us increased means for describing the behavior of two-way automata.

Traces of two-way computations belong to $(\Sigma \cup \bar{\Sigma})^+$. The subsemigroup Σ^+ describes right movements, while $\bar{\Sigma}^+$ describes left movements.

It is convenient to apply the bar ($-$) not only to letters of Σ (which yields $\bar{\Sigma}$) but also to words of Σ^+, and even to words of $(\Sigma \cup \bar{\Sigma})^+$, using the following definition: if $w = x_1 x_2 \ldots x_n \in (\Sigma \cup \bar{\Sigma})^+$, where each $x_i \in \Sigma \cup \bar{\Sigma}$ is a letter, then $\bar{w} = \bar{x}_n \ldots \bar{x}_2 \bar{x}_1$. We need the additional conventions that $\bar{x} = a \in \Sigma$ if $x = \bar{a} \in \bar{\Sigma}$, and $\bar{x} = \bar{a} \in \bar{\Sigma}$ if $x = a \in \Sigma$. Henceforth we can replace each sequence of barred letters ($\in \bar{\Sigma}^+$) by a single barred word. Technically this amounts to identifying the free semigroup $(\Sigma \cup \bar{\Sigma})^+$ with the free product $\Sigma^+ \circledast \bar{\Sigma}^+$ (of Σ^+ and $\bar{\Sigma}^+$). The two semigroups $(\Sigma \cup \bar{\Sigma})^+$ and $\Sigma^+ \circledast \bar{\Sigma}^+$ are isomorphic. Every element of $(\Sigma \cup \bar{\Sigma})^+$ is of the form $u_1 \bar{v}_1 u_2 \bar{v}_2 \ldots u_n \bar{v}_n$ where $n \geqq 1$, and $u_i, v_i \in \Sigma^+$ $(1 \leqq i \leqq n)$ -- except that we also allow u_1 or v_n to be the empty word (in which case we simply drop u_1 respectively v_n from the expression).

When we use automata as *acceptors* we have to fix start state $q_0 \in Q$ and a set of accept states $F \subseteq Q$, and a "direction of acceptance" (see section 4). This way we obtain two formal languages from any two-way automaton:

1. A language $L^{(1)} \subseteq (\Sigma \cup \bar{\Sigma})^+$ (over the double alphabet $\Sigma \cup \bar{\Sigma}$) recognized by the control unit; it will be called the "*control language*". (This language was introduced by Pécuchet [7]).

2. A language $L^{(2)} \subseteq \Sigma^+$ (over the input alphabet Σ) recognized by the two-way automaton; this language will be called the "*two-way language*". See section 4 for definitions of two-way acceptance.

An interesting question, due to Pécuchet, is: How are $L^{(2)}$ and $L^{(1)}$ related?

Both languages are finite-state when Q is finite: for $L^{(1)}$ it is obvious; for $L^{(2)}$ that is precisely the Rabin-Shepherdson theorem [8] about the equivalence between one-way and two-way finite automata.

2. TRACES OF TWO-WAY COMPUTATIONS ON A GIVEN TAPE INPUT

In order to relate a trace of a two-way computation to the input (on tape) on which the computation is carried out, we use the following *reduction* operation on words in $(\Sigma \cup \bar{\Sigma})^+$.

DEFINITION: Consider all the rewrite rules $u \bar{u} u \to u$, $\bar{u} u \bar{u} \to \bar{u}$ where u ranges over Σ^+. For $w \in (\Sigma \cup \bar{\Sigma})^*$ define $red(w)$ to be the word in $(\Sigma \cup \bar{\Sigma})^*$ obtained by applying the above rules repeatedly "as often as possible", until a word is obtained to which no such rule can be applied anymore.

Fact (2.1) (Pécuchet [7]): The result $red(w)$ obtained from $w \in (\Sigma \cup \bar{\Sigma})^*$ by applying the above rewrite rules repeatedly as much as possible is *unique* (independently of the order and places in which rules are applied). So the operation $w \to red(w)$ is a well-defined function on $(\Sigma \cup \bar{\Sigma})^*$.

The *proof* follows from the "diamond (Church-Rosser) property" of the above rewrite rules: If $w_1 \leftarrow w \rightarrow w_2$ then there exists $w' \in (\Sigma \cup \bar{\Sigma})^*$ such that $w_1 \overset{*}{\rightarrow} w' \overset{*}{\leftarrow} w_2$. (Notation: Let $x, y \in (\Sigma \cup \bar{\Sigma})^*$; then $x \rightarrow y$ iff y is obtained from x by application of one rewrite rule of the above type; $x \overset{*}{\rightarrow} y$ iff $x = y$ or if y is obtained from x by repeated applications of rewrite rules of the above type.) See [7] for details, or [1] pp. 36-40 for a similar result and a similar proof. ■

We have the following basic properties:

Fact (2.2):

(*a*) *For all* $w \in (\Sigma \cup \bar{\Sigma})^*$: $red(\bar{w}) = \overline{red(w)}$.

(*b*) *For all* $w_1, w_2 \in (\Sigma \cup \bar{\Sigma})^*$: $red(w_1 w_2) = red(red(w_1)\, red(w_2))$.

[Proof outline. For (*a*): w and $\bar{w}$ are reduced in a symmetric way, applying $u \bar{u} u \rightarrow u$ instead of $\bar{u} u \bar{u} \rightarrow \bar{u}$, and vice versa. For (*b*): The element $red(w_1 w_2)$ is unique, and independent of the order and place of applications of the rewrite rules.] ■

DEFINITION: A two-way automaton computation is said to be *left-to-right* iff at the beginning of the computation the reading-head is positioned at the left end of the input (on tape) while the state is right-moving ($\in \vec{Q}$), and at the end of the computation the reading-head is placed at the right end of the input while the state is right-moving ($\in \vec{Q}$) or left moving ($\in \tilde{Q}$). See *fig.* 1.

A two-way computation is said to be *left-to-left* iff it begins at the left end of the input in a right-moving state and finishes at the left end of the input (in a left-moving or right moving state).

A *strict left-to-left* computation is a left-to-left two-way computation during which the entire input is visited. See *fig.* 2.

By symmetry one defines *right-to-left*, and *right-to-right* computations.

Remark. – Contrary to [2] I do not require in the above definition that at the end of a left-to-right computation the state be right-moving (and similarly for the other computations). The results of sections (2) and (3) would also be true with the conventions of [2]. However the proofs in section 4 are easier with the present conventions.

Also, in the above definition of a computation (see also the definition in section 1) we are not saying that the automaton has to halt at the end of a computation; a computation is just a sequence of configurations (a segment of a maximal computation).

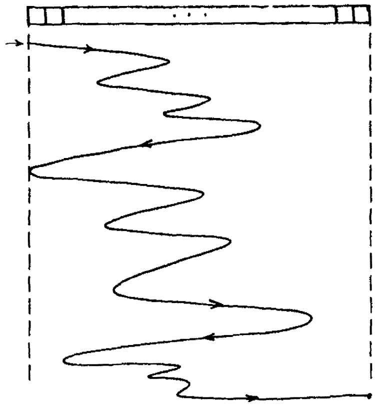

Fig. 1

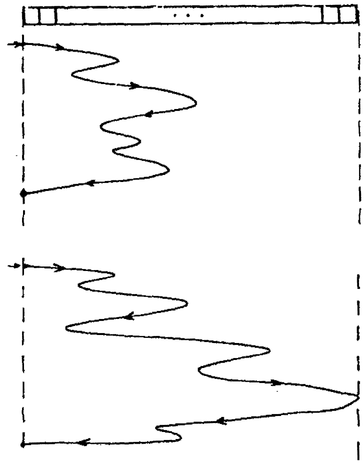

Fig. 2

Figure 1. — A left-to-right computation.
Figure 2. — Two left-to-left computations, the second being strict.

LEMMA(2.3) (*Relation between input and trace*): *Let* $w \in (\Sigma \cup \bar{\Sigma})^*$ *and* $u \in \Sigma^+$. *Then we have:*

(1) *w is the trace of some* left-to-right *computation* (*for some two-way automaton*) *on input u iff* $red(w) = u$.

(2) *w is the trace of some* left-to-left *computation* (*for some two-way automaton*) *on input u iff* $red(w) = p\bar{p}$, *where* $p \in \Sigma^+$ *and p is a prefix of u* (*i.e.*: $p \in \Sigma^+$ *and* $u \in p\Sigma^*$).

The computation is strict *left-to-left iff the trace w satisfies* $red(w) = u\bar{u}$.

(3) *w is the trace of some* right-to-right *computation iff* $red(w) = \bar{s}s$, *where* $s \in \Sigma^+$ *and s is a suffix of u* (*i.e.*: $s \in \Sigma^+$ *and* $w \in \Sigma^* s$).

In the strict *case* $red(w) = \bar{u}u$.

(4) *w is the trace of a* right-to-left *computation iff* $red(w) = \bar{u}$.

Proof: The four relations are proved in a similar way, and I will consider only the first one.

($\Rightarrow$) Suppose *w* is the trace of a left-to-right computation on input *u*. We must show that $red(w) = u$.

The trace of a left-to-right two-way computation is of the form $w = u_1 \bar{v}_1 u_2 \bar{v}_2 \ldots u_n \bar{v}_n u_{n+1}$, where $u_i, v_j \in \Sigma^+$ ($1 \leqq i \leqq n+1$, $1 \leqq j \leqq n$). The proof goes by induction on the number *n* of barred segments $\bar{v}_j$. We prove that *red*(w) is equal to *red*(w'), where w' is also a trace of a left-to-right computation on input u, but w' contains only $n-1$ barred subsegments. The word w' is obtained from w by removing one zigzag ($\underset{\rightarrow}{\rightleftarrows}$ or $\underset{\leftarrow}{\leftrightarrows}$) through application of a rewrite rule $v\bar{v}v \to v$ or $\bar{u}u\bar{u} \to \bar{u}$: consider the shortest among the segments $\bar{v}_1, u_2, \bar{v}_2, \ldots, u_n, \bar{v}_n$, of w (excluding u_1 and u_{n+1}), and remove a zigzag around that shortest segment.

($\Leftarrow$) Conversely, suppose *red*$(w)=u$. We must show that there exists a left-to-right computation of some two-way automaton on input u, whose trace is w.

If *red*$(w)=u$ then there exists a sequence $w_{n+1}\,(=w), w_n, \ldots, w_1, w_0\,(=u)$ of words in $(\Sigma \cup \bar{\Sigma})^*$ such that (for $0 \leqq i \leqq n$): w_i is obtained from w_{i+1} by application of a rewrite rule of the form $x\bar{x}x \to x$ or $\bar{x}x\bar{x} \to \bar{x}$ (with $x \in \Sigma^+$). We prove by induction on increasing i that all the words w_i $(0 \leqq i \leqq n+1)$ are traces of left-to-right computations on input u. This is certainly true for $u=w_0$ (case $i=0$): just take a one-way computation.

Inductive step $(i \to i+1)$: w_{i+1} is of the form $m x \bar{x} x r$ (or $m \bar{x} x \bar{x} r$) where $w_i = m x r$ (respectively $w_i = m \bar{x} r$), $x \in \Sigma^+$, and $m, r \in (\Sigma \cup \bar{\Sigma})^*$. Let us consider the case where $w_{i+1} = m x \bar{x} x r$ (the other case is similar). If w_i is the trace of a left-to-right computation on input u, then w_{i+1} will also be a trace of a left-to-right computation on input u: w_{i+1} is obtained by first executing the initial part mx of the computation w_i, second, carrying out a back-and-forth movement on x (so at this point the trace is $m x \bar{x} x$), third, executing the remainder r of w_i. ■

Another way to state lemma (2.3) is as follows:

Fact (2.4): *Let $u \in \Sigma^+$ be a given input. The set of traces of all left-to-right, respectively left-to-left, resp. right-to-right, resp. right-to-left computations (for all possible two-way automata) on input u are:*

(1) *red*$^{-1}(u)\,(=\{w / red(w)=u\})$ *for* left-to-right *traces.*

We will denote this set by $\{\to u \to\}$.

(2) $\displaystyle\bigcup_{\substack{p \in \Sigma^+ \\ p \text{ prefix of } u}} red^{-1}(p\bar{p})$ *for* left-to-left *traces.*

We denote this set by $\{\rightleftarrows u\}$.

(3) $\displaystyle\bigcup_{\substack{s \in \Sigma^+ \\ s \text{ suffix of } u}} red^{-1}(\bar{s}s)$ *for* right-to-right *traces.*

We denote this set by $\{u \rightleftarrows\}$.

(4) *$red^{-1}(\bar{u})$ for* right-to-left *traces*.

We denote this set by $\{\leftarrow u \leftarrow\}$.

(General definition: $red^{-1}(y) = \{x \in (\Sigma \cup \bar{\Sigma})^* / red(x) = y\}$.)

The next section will give formulas relating the traces on uv (concatenation of inputs) to the traces on u respectively v. It will follow from those formulas that the above subsets of $(\Sigma \cup \bar{\Sigma})^+$ are rational languages.

Remark: If w is any word in $(\Sigma \cup \bar{\Sigma})^+$ then $red^{-1}(w)$ will be the empty set unless w is indeed a reduced word [clearly, if an element w is not in the range of the function *red* then its inverse image $red^{-1}(w)$ is empty]. When $red^{-1}(w) \neq \varnothing$ then every element of $red^{-1}(w)$ can be obtained by starting with w and repeatedly applying rewrite rules of the form $u \to u\bar{u}u$, $\bar{u} \to \bar{u}u\bar{u}$ (where u ranges over Σ^+).

One easily checks that for each single letter $a \in \Sigma$ we have:

$$\{\to a \to\} = (a\bar{a})^* a, \qquad \{\rightleftarrows a\} = (a\bar{a})^+,$$
$$\{a \rightleftarrows\} = (\bar{a}a)^+, \qquad \{\leftarrow a \leftarrow\} = (\bar{a}a)^* a.$$

3. TRACES, AND THE CONCATENATION OF INPUTS

THEOREM (3.1) (traces on concatenated inputs): *Let $u, v \in \Sigma^+$. Then:*

$$\{\to uv \to\} = \{\to u \to\} . (\{\rightleftarrows v\} . \{u \rightleftarrows\})^* . \{\to v \to\}$$
$$\{\rightleftarrows uv\} = \{\rightleftarrows u\} \cup \{\to u \to\} . (\{\rightleftarrows v\} . \{u \rightleftarrows\})^* . \{\rightleftarrows v\} . \{\leftarrow u \leftarrow\}$$
$$\{uv \rightleftarrows\} = \{v \rightleftarrows\} \cup \{\leftarrow v \leftarrow\} . (\{u \rightleftarrows\} . \{\rightleftarrows v\})^* . \{u \rightleftarrows\} . \{\to v \to\}$$
$$\{\leftarrow uv \leftarrow\} = \{\leftarrow v \leftarrow\} . (\{u \rightleftarrows\} . \{\rightleftarrows v\})^* . \{\leftarrow u \leftarrow\}.$$

NOTATION: Here "." denotes the concatenation of languages (*i.e.* $L_1 . L_2 = \{xy / x \in L_1, y \in L_2\}$), and "*" is the Kleene star (*i.e.* $L^* = \bigcup_{n \geq 0} L^n$, where $L^0 = \{$"empty word"$\}$, and $L^{n+1} = L^n . L$).

Proof: Since the four formulas have very similar proofs I will only consider the first one.

($\subseteq$) We have to show that $red(w) = uv$ implies

$$w \in \{\to u \to\} . (\{\rightleftarrows v\} . \{u \rightleftarrows\})^* . \{\to v \to\}.$$

By fact 2.3, if $red(w)=uv$ then there exists a left-to-right computation (of some two-way automaton) with input uv and trace w. Such a computation has the form given in figure 3.

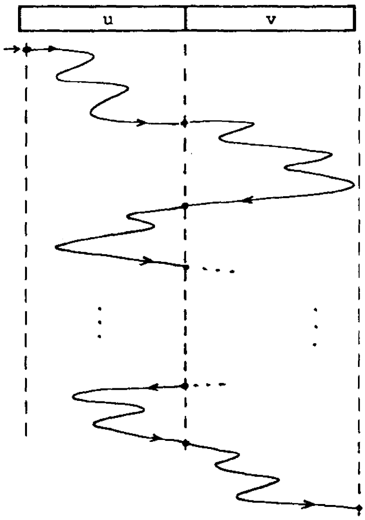

Figure 3. — A left-to-right computation on input uv.

During that computation the reading-head of the two-way automaton starts at the left end of u, and eventually leaves the string u (on the right end of u) and enters into v. The piece of computation done so far has a trace in $\{\rightarrow u \rightarrow\}$.

Next, either the reading-head eventually finishes on the *right* end of v (in that case the trace w of the computation belongs to $\{\rightarrow u \rightarrow\}.\{\rightarrow v \rightarrow\} \subseteqq \{\rightarrow u \rightarrow\}.(\{\rightleftarrows v\}.\{u \rightleftarrows\})^*.\{\rightarrow v \rightarrow\}$), *or* the reading-head eventually exists from v on the *left* end of v and reenters into u. The trace of the piece of computation done so far belongs to $\{\rightarrow u \rightarrow\}.\{\rightleftarrows v\}$.

This back-and-forth movement goes on a finite number of times. Let $k (\geqq 0)$ be the number of times that the reading-head crosses the u-v boundary in the backwards (right-to-left) direction. Then the trace w belongs to

$$\{\rightarrow u \rightarrow\}.(\{\rightleftarrows v\}.\{u \rightleftarrows\})^k.\{\rightarrow v \rightarrow\},$$

and hence to $\{\rightarrow u \rightarrow\}.(\{\rightleftarrows v\}.\{u \rightleftarrows\})^*.\{\rightarrow v \rightarrow\}$.

($\supseteq$) Next we prove that

$$red^{-1}(uv)=\{\to uv\to\}\supseteq\{\to u\to\}(\{\rightleftarrows v\}\{u\rightleftarrows\})^*\{\to v\to\},$$

or equivalently: $red\,(\{\to u\to\}(\{\rightleftarrows v\}\{u\rightleftarrows\})^*\{\to v\to\})\subseteq\{uv\}$, which in turn is equivalent to $red\,(\{\to u\to\}(\{\rightleftarrows v\}\{u\rightleftarrows\})^*\{\to v\to\})=uv$.
By fact 2.2 (*b*), and by the definition of $\{\to u\to\}$, $\{\rightleftarrows v\}$, $\{u\rightleftarrows\}$, $\{\to v\to\}$, we have:

$$\begin{aligned}&red(\{\to u\to\}(\{\rightleftarrows v\}\{u\rightleftarrows\})^*\{\to v\to\})\\ &\quad=red(red\{\to u\to\}.(red\{\rightleftarrows v\}.red\{u\rightleftarrows\})^*.red\{\to v\to\})\\ &\quad=red\,(u.(\{y\,\bar{y}/y\in\Sigma^+,\ y\text{ prefix of }v\}\\ &\qquad\qquad.\{\bar{x}\,x/x\in\Sigma^+,x\text{ suffix of }u\})^*.v)\end{aligned}$$

Any element of

$$u(\{y\bar{y}/y\in\Sigma^+,y\text{ prefix of }v\}.\{\bar{x}x/x\in\Sigma^+,x\text{ suffix of }u\})^*\,v$$

is of the form $w_n=u\,y_1\,\bar{y}_1\,\bar{x}_1\,x_1\,y_2\,\bar{y}_2\,\bar{x}_2\,x_2\,\ldots\ldots\,y_n\,\bar{y}_n\,\bar{x}_n\,x_n\,v$, where $n\geq 0$ and y_i is a prefix of v, x_i is a suffix of u (for $1\leq i\leq n$). Since $\bar{y}_i\,\bar{x}_i=\overline{x_i\,y_i}$ we also have

$$w_n=u\,y_1\,\overline{x_1\,y_1}\,x_1\,y_2\,\overline{x_2\,y_2}\,x_2\,y_3\,\ldots\ldots\,x_{n-1}\,y_n\,\overline{x_n\,y_n}\,x_n\,v.$$

So this expression of w_n contains n subsegments in $\bar{\Sigma}^+$ ("barred" subsegments).

We must show that $red(w_n)=uv$. In order to do that we show that if $n>0$, then from w_n one can obtain a word w_{n-1} (by applying a rewrite rule of the form $z\bar{z}z\to z$ or $\bar{z}z\bar{z}\to\bar{z}$, with $z\in\Sigma^+$, hence $red(w_n)=red(w_{n-1})$), such that w_{n-1} contains only $n-1$ subsegments in $\bar{\Sigma}^+$.

Let us consider the shortest of all the segments $x_i\,y_i\,(1\leq i\leq n)$, $x_i\,y_{i+1}$ $(1\leq i\leq n-1)$.

Case 1: The shortest such segment is of the form $x_i\,y_i$. Since x_{i-1} and x_i are both suffixes of u, it follows that either x_{i-1} is a suffix of x_i or x_i is a suffix of x_{i-1}. Therefore $x_{i-1}\,y_i$ is a suffix of $x_i\,y_i$ or vice versa; but since $x_i\,y_i$ was chosen of minimum length, $x_i\,y_i$ must be a suffix of $x_{i-1}\,y_i$. Similarly $x_i\,y_i$ must be a prefix of $x_i\,y_{i+1}$. (Indeed, y_i and y_{i+1} are both prefixes of v, so y_i is a prefix of y_{i+1} or vice-versa, hence, $x_i\,y_i$ is a prefix of $x_i\,y_{i+1}$ or vice-versa, but $x_i\,y_i$ is of minimum length.) So there exist s_{i-1}, $s_i\in\Sigma^*$ with $x_{i-1}\,y_i=s_{i-1}\,x_i\,y_i$, and $x_i\,y_{i+1}=x_i\,y_i\,s_i$.

Now

$$w_n = u y_1 \ldots x_{i-1} y_i \overline{x_i y_i} x_i y_{i+1} \ldots x_n v = u y_1 \ldots s_{i-1} x_i y_i \overline{x_i y_i} x_i y_i s_i \ldots x_n v,$$

from which one obtains the word

$$w_{n-1} = u y_1 \ldots s_{i-1} x_i y_i s_i \ldots x_n v$$

by application of the rewrite rule $x_i y_i \overline{x_i y_i} x_i y_i \to x_i y_i$. This word w_{n-1} has only $n-1$ segments in $\bar{\Sigma}^+$ [and of course $red(w_n) = red(w_{n-1})$].

Case 2: The shortest segment is of the form $x_i y_{i+1}$. Then $x_i y_{i+1}$ must be a prefix of $x_i y_i$ and a suffix of $x_{i+1} y_{i+1}$ (by the same reasoning as in case 1). Thus there exist t_{i+1}, $t_i \in \Sigma^*$ with

$$x_i y_i = x_i y_{i+1} t_{i+1} \quad \text{and} \quad x_{i+1} y_{i+1} = t_i . x_i y_{i+1}.$$

Then (after a calculation similar to case 1), by application of the rewrite rule

$$\overline{x_i y_{i+1}} x_i y_{i+1} \overline{x_i y_{i+1}} \to \overline{x_i y_{i+1}},$$

we obtain a word w_{n-1} with only $n-1$ segments in $\bar{\Sigma}^+$ and $red(w_n) = red(w_{n-1})$.

This way w_n is successively replaced by words w_j with $red(w_n) = red(w_j)$ $(n \geqq j \geqq 0)$, and w_j has only j subsegments in $\bar{\Sigma}^+$. Finally $w_0 = uv \in \Sigma^+$ (no "barred" subsegments), so $red(w_n) = red(w_0) = uv$. ■

Remark: The formula for $\{\leftarrow uv \to\}$ is equivalent to the formula for $\{\to uv \to\}$, by the fact that $\{\leftarrow uv \leftarrow\} = \{\to \overline{uv} \to\}$, if one defines $\{\to \bar{w} \to\} = red^{-1}(\bar{w})$. Similarly the formulas for $\{\leftrightarrows uv\}$ and $\{uv \leftrightarrows\}$ are equivalent.

IMPORTANT COROLLARY (3.2): *For every $u \in \Sigma^+$ the sets $\{\to u \to\}$, $\{\leftrightarrows u\}$, $\{u \leftrightarrows\}$, $\{\leftarrow u \leftarrow\}$, are finite-state languages.*

Proof: The corollary follows easily from the formulas, by induction on the length of u (since only rational operations appear in the formulas). ■

After this corollary one wonders wether *red* and red^{-1} preserve finite-stateness in general. This is not the case however.

Fact (3.3) (Pécuchet) : *There exist languages $L \subseteqq (\Sigma \cup \bar{\Sigma})^+$ that are finite-state but such that $red(L)$ is not finite-state (and not even context-free).*

There exist languages $L \subseteq \Sigma^+$ that are finite-state, but such that $red^{-1}(L)$ is not finite-state (and not even context-free). In fact $red^{-1}(\Sigma^+)$ is not context-free (for any alphabet Σ). The *proofs* use the Pumping Lemma (Ogden's version); *see* [7]. ■

Recall (fact 2.3) that $red^{-1}(\Sigma^+)$ is the set of all traces of left-to-right computations of two-way automata. The fact that $red^{-1}(\Sigma^+)$ is not finite-state then implies the following: *The finite control unit of a two-way finite automaton cannot always know whether the "input" $\in (\Sigma \cup \bar{\Sigma})^+$ it receives really arizes from a left-to-right two-way computation.* In other words: it would be possible for the reading-head to deceive the control unit, by sending to it certain words of $(\Sigma \cup \bar{\Sigma})^+$ that are not traces of any two-way movement on a tape. The control unit could not always notice that.

Although $red(.)$ and $red^{-1}(.)$ do not preserve finite-stateness, we will see some operations that do preserve it, namely the operations $L \to red(L) \cap \Sigma^+$, due to Pécuchet, and $L \to clos(L) =$ the set of *all* words obtainable from words in L by repeatedly applying rewrite rules of the form $x\bar{x}x \to x$, $\bar{x}x\bar{x} \to \bar{x}$ with $x \in \Sigma^+$.

Closure under the rewrite rules:

$$\{u\bar{u}u \to u,\ \bar{u} \to \bar{u}u\bar{u} / u \in \Sigma^+\}.$$

Definition: If L is a subset of $(\Sigma \cup \bar{\Sigma})^*$ then $clos(L)$ is the subset of $(\Sigma \cup \bar{\Sigma})^+$ obtained by *closing L under the rewrite rules $u\bar{u}u \to u$, $\bar{u}u\bar{u} \to \bar{u}$* (as u ranges over Σ^+). More precisely, a word $w \in (\Sigma \cup \bar{\Sigma})^*$ belongs to $clos(L)$ iff w is the result of applying any finite number (including zero) of rules of the form $u\bar{u}u \to u$ or $\bar{u}u\bar{u} \to \bar{u}$ (as u ranges over all of Σ^+), in some order, to a word of L. Notice that for $w \in (\Sigma \cup \bar{\Sigma})^*$, $clos(w)$ is a set, while $red(w)$ is a single word [and $red(w) \in clos(w)$].

Fact (3.4): *If $L \subseteq (\Sigma \cup \bar{\Sigma})^*$ is a finite-state language, then $clos(L)$ is also a finite-state language.*

Proof: Let $A_1 = (Q, \Sigma \cup \bar{\Sigma}, \bullet, q_0, F)$ be a one-way deterministic finite automaton recognizing the language $L \subseteq (\Sigma \cup \bar{\Sigma})^*$.

From A_1 we shall construct a two-way non-deterministic finite automaton A_2 which accepts $clos(L)$. The idea for A_2 is as follows. When A_2 processes a word w of $clos(L)$ it has the ability to read w in the left-to-right direction, but it can also "choose" to make zig-zag movements $\underset{\rightarrow}{(\rightleftarrows)}$ on any subsegment of w (provided that this subsegment is of the form $u \in \Sigma^+$ or $\bar{u} \in \bar{\Sigma}^+$). During these zig-zag movements A_2 continues simulating the states-transitions of A_1. The zig-zags on u, respectively $\bar{u}$, correspond to applying rewrite rules of the form $u \bar{u} u \to u$, resp. $\bar{u} u \bar{u} \to \bar{u}$ (with $u \in \Sigma^+$). Finally, the input w will be accepted iff w can be obtained from a word in L (accepted by A_1) by applying rewrite rules of the form $u \bar{u} u \to u$, $\bar{u} u \bar{u} \to \bar{u}$ with $u \in \Sigma^+$ (*i.e.* by performing zig-zags on subsegments u or $\bar{u}$ of w). Actually such zig-zags can occur within each other.

A precise description of A_2 follows now. Here $\Sigma \cup \bar{\Sigma}$ is the actual tape-alphabet of the two-way automaton A_2, and not the "double alphabet" of the control unit; $\bar{\Sigma}$ does *not* indicate backwards movements of A_2.

$$A_2 = (\vec{Q}_2, \tilde{Q}_2, \Sigma \cup \bar{\Sigma}, \ldots, \oplus, \{q_0', q_0''\}, F_2),$$

where

$$\vec{Q}_2 = \{\to\} \times \{+, -\} \times Q, \qquad \tilde{Q}_2 = \{\leftarrow\} \times \{+, -\} \times Q,$$
$$F_2 = \{\to\} \times \{+, -\} \times F \subseteq \vec{Q}_2,$$

and

$$q_0' = (\to, +, q_0), \qquad q_0'' = (\to, -, q_0)$$

(*i.e.* here I use two start-states). The alphabet of the control unit is ignored here.

Words are accepted by the two-way automaton A_2 if they give rise to some left-to-right computation which starts on the left in a start-state, and ends on the right in an accept state.

The next-state relation $\oplus : (\vec{Q}_2 \cup \tilde{Q}_2) \times (\Sigma \cup \bar{\Sigma}) \to \vec{Q}_2 \cup \tilde{Q}_2$ is defined as follows:

for states in $\vec{Q}_2$ and

for $a \in \Sigma$: $(\to, \pm, q) \oplus a = \{(\to, +, q \bullet a), (\leftarrow, +, q \bullet a)\}$,

for $\bar{a} \in \bar{\Sigma}$: $(\to, \pm, q) \oplus \bar{a} = \{(\to, -, q \bullet \bar{a}), (\leftarrow, -, q \bullet \bar{a})\}$;

for states in $\tilde{Q}_2$ and

for $a \in \Sigma$:

$$(\leftarrow, +, q) \oplus a = \{(\leftarrow, +, q \bullet \bar{a}), (\to, +, q \bullet \bar{a})\},$$
$$(\leftarrow, -, q) \oplus a = \varnothing \text{ (undefined)};$$

for $\bar{a} \in \bar{\Sigma}$:

$$(\leftarrow, +, q) \oplus \bar{a} = \varnothing \text{ (undefined)},$$
$$(\leftarrow, -, q) \oplus \bar{a} = \{(\leftarrow, -, q \bullet a), (\rightarrow, -, q \bullet a)\}.$$

From this definition the three coordinates of the states (namely $\{\rightarrow, \leftarrow\}$, $\{+, -\}$, and Q) receive the following interpretation: the first coordinate (in $\{\rightarrow, \leftarrow\}$) indicates the direction of the movement of the two-way automaton A_2. The second coordinate (in $\{+, -\}$) indicates *whether in the last right-ward movement the letter read belonged to* Σ *or to* $\bar{\Sigma}$; the rules $(\leftarrow, -, q) \oplus a = \varnothing = (\leftarrow, +, q) \oplus \bar{a}$ guarantee then that during a left-ward movement the reading head never crosses over from segments in Σ^+ to segments in $\bar{\Sigma}^+$ or vice versa. The third coordinate (in Q) simulates the state of A_1 (accepting L).

Let us finally prove that A_2 recognizes *clos* (L).

(1) *Proof that if a word belongs to clos* (L) *then it is accepted by* A_2: induction on the *number of applications of rewrite rules* (of the form $u\bar{u}u \rightarrow u$ or $\bar{u}u\bar{u} \rightarrow \bar{u}$ where $u \in \Sigma^+$) to a word in L, in order to reach a certain word of *clos* (L).

(Step 0): Then every word of L is accepted by A_2 (when only states in $\vec{Q}_2$ are used).

(Inductive step): If a word of the form $xu\bar{u}uy$, or $x\bar{u}u\bar{u}y$, (where $u \in \Sigma^+$) is accepted by A_2 then the word xuy (respectively $x\bar{u}y$) is also accepted by A_2: in the case of $xu\bar{u}uy$ one replaces each piece of computation on the subsegment $\bar{u}$ by the corresponding left-moving computation on the subsegment u of xuy (obtained by putting a bar $-$ on the trace); in the case of $x\bar{u}u\bar{u}y$ one replaces each piece of computation on the subsegment u by the corresponding left-moving computation on the subsegment $\bar{u}$ of $x\bar{u}y$.

(2) *Proof that if a word is accepted by* A_2 *then it belongs to* clos (L): induction on the number of right-to-left turns (or "reversals" $\rightleftarrows$) of accepting computations of A_2.

(Step 0): If a word is accepted in a computation of A_2 without any turns (*i. e.* one-way movement), then the word must belong to L. [And, of course, $L \subseteq clos(L)$.]

(Inductive step): Suppose the accepting computation of A_2 on input $w \in (\Sigma \cup \bar{\Sigma})^*$ involves a left-to-right turn. Then w must be of the form $w = xuy$, where u is a maximally long subsegment of w belonging to Σ^+ such that a turn occurs on u (or w is of the form $w = x\bar{u}y$, where $\bar{u}$ is a maximally long subsegment of w belonging to $\bar{\Sigma}^+$ such that a turn occurs on $\bar{u}$). By the construction of A_2, the reading head never moves off u to the left side of u (*see* the definition of $\oplus:(\leftarrow, +, q)\oplus\bar{a}=\varnothing$); similarly in the case $x\bar{u}y$ the reading head does not move off $\bar{u}$ to the left side of $\bar{u}$. We conclude that the two-way computation of A_2 on w consists of a succession of three left-to-right computations: first one on x, second one on u (respectively $\bar{u}$), third one on y. The computation on u has a trace $t\in\{\rightarrow u\rightarrow\}$ (respectively $t\in\{\rightarrow\bar{u}\rightarrow\}=\{\leftarrow u\leftarrow\}$). Now replace $w=xuy$ (resp. $w=x\bar{u}y$) by xty. Clearly w can be obtained from xty by applying rewrite rules of the form $s\bar{s}s\rightarrow s$, $\bar{s}s\bar{s}\rightarrow\bar{s}$ with $s\in\Sigma^+$, since $t\in\{\rightarrow u\rightarrow\}$ (resp. $t\in\{\leftarrow u\leftarrow\}$). Therefore, if $xty\in clos(L)$ then $w\in clos(L)$. But xty can be accepted by A_2 using a computation involving fewer turns than the accepting computation on w. Thus, by inductive hypothesis, xty does indeed belong to $clos(L)$; from there it follows that w belongs to $clos(L)$. ■

Remark 1: If instead of the above automaton A_2 one uses the two-way automaton $A'_2=(\vec{Q}'_2=\{\rightarrow\}\times Q,\ \overleftarrow{Q}'_2=\{\leftarrow\}\times Q,\ \Sigma\cup\bar{\Sigma},\ \ldots, \circ, (\rightarrow, q_0), \{\rightarrow\}\times F)$, accepting by left-to-right computations, with next-state relation $\circ$ defined by:

$$(\rightarrow, q)\circ a=\{(\rightarrow, q\bullet a), (\leftarrow, q\bullet a)\},$$

$$(\rightarrow, q)\circ \bar{a}=\{(\rightarrow, q\bullet \bar{a}), (\leftarrow, q\bullet \bar{a})\},$$

$$(\leftarrow, q)\circ a=\{(\rightarrow, q\bullet \bar{a}), (\leftarrow, q\bullet \bar{a})\},$$

$$(\leftarrow, q)\circ \bar{a}=\{(\rightarrow, q\bullet a), (\leftarrow, q\bullet a)\},$$

then the language recognized is the closure of L under the rewrite rules $w\bar{w}w\rightarrow w$, where w ranges over *all* of $(\Sigma\cup\bar{\Sigma})^+$. Hence this closure also preserves finite-stateness. Again, in this automaton we ignore the alphabet of the control unit.

Remark 2: The closure under the rewrite rules $u\rightarrow u\bar{u}u$, $\bar{u}\rightarrow\bar{u}u\bar{u}$ (as u ranges over Σ^+) does *not* preserve finite-stateness. For example, this closure of $a^+=\{a^n/n>0\}$ is equal to $red^{-1}(a^+)$, which is not even context-free (see Fact 3.3). So the operation $clos(.)$ is not a "rational transduction" (it preserves finite-stateness, but its inverse does not; see e. g. [1] for a definition of rational transduction).

Two-way automata thus lead to a class of finite-stateness preserving transformations (like $clos(.)$, and Pécuchet's $\Sigma^+\cap red(.)$, etc.; see the next section and also Pécuchet [7]), which are not rational transductions.

4. THE CONTROL LANGUAGE $L^{(1)}$ AND ITS RELATION TO THE TWO-WAY LANGUAGE

For the remainder of the paper we consider a fixed two-way automaton $(\vec{Q}, \tilde{Q}, \Sigma, \Sigma \cup \bar{\Sigma}, \bullet, q_0, F)$, where q_0 is a chosen start state, and F is a chosen set of accept states.

The control unit of this two-way automaton is the one-way automaton $(\vec{Q} \cup \tilde{Q}, \Sigma \cup \bar{\Sigma}, \bullet, q_0, F)$. It accepts a language $L^{(1)} \subseteq (\Sigma \cup \bar{\Sigma})^*$, called the *control language*.

We will consider four different modes of acceptance of a *two-way language* $\subseteq \Sigma^*$ by a two-way automaton.

1. *Left-to-right* acceptance: there we assume $q_0 \in \vec{Q}$, and $F \subseteq \vec{Q} \cup \tilde{Q}$. A word $u \in \Sigma^*$ is accepted iff there exists a left-to-right computation of the given two-way automaton on input u, starting at the left end of u in state q_0 and ending at the right end of u in a state of F.

2. *Left-to-left* acceptance: Assume $q_0 \in \vec{Q}$ and $F \subseteq \tilde{Q} \cup \vec{Q}$. A word $u \in \Sigma^*$ is accepted iff there exists a left-to-left computation of the given two-way automaton on input u, starting at the left end of u in state q_0 and ending at the left end of u in a state of F. We do not require here that all of u is actually visited during this accepting computation.

3. & 4. In a symmetric way one defines right-to-left and right-to-right acceptance.

Remark: Contrary to [2], we do not assume that $F \subseteq \vec{Q}$ for left-to-right acceptance, nor that $F \subseteq \tilde{Q}$ for left-to-left acceptance, etc. The reason of this change is that by using this mode of acceptance Pécuchet's theorem can be proved (Theorem 4.2) more easily. The model of [2] is then a special case of this more general convention.

These four languages are called *two-way languages* and denoted $L^{(2)}$; the context will tell which of the four modes of acceptance is referred to.

If A is an alphabet and $L \subseteq A^*$, $w \in A^*$, then the *left-quotient* $w^{-1} L$ is $\{x \in A^* / wx \in L\}$.

The left quotients $w^{-1} L^{(1)}$ [as w ranges over $(\Sigma \cup \bar{\Sigma})^*$] are the states of the minimum automaton of $L^{(1)}$ (*see* e. g. [6]). The complete description of the *minimum automaton of* $L^{(1)}$ is:

states: $Q_m = \{ w^{-1} L^{(1)} / w \in (\Sigma \cup \bar{\Sigma})^*, w^{-1} L^{(1)} \neq \varnothing \}$,

input alphabet $= \Sigma \cup \bar{\Sigma}$,

start state $= L^{(1)}$,

accept states

$$F_m = \{ w^{-1} L^{(1)} / w \in L^{(1)} \} = \{ w^{-1} L^{(1)} / \text{"empty word"} \in w^{-1} L^{(1)} \};$$

the next-state function $\circledcirc : Q_m \times (\Sigma \cup \bar{\Sigma})^* \to Q_m$ is defined by:

$$(x^{-1} L^{(1)}) \circledcirc y = y^{-1} x^{-1} L^{(1)} (= (xy)^{-1} L^{(1)}), \qquad \text{for} \quad x, y \in (\Sigma \cup \bar{\Sigma})^*.$$

[Remark: The empty set $\varnothing$ is not used as a state. If $\varnothing = w^{-1} L^{(1)}$ for some w, then the minimum automaton will be a partial automaton: the next-state function is not defined when the next state would be $\varnothing$.]

The importance of the control language is expressed in the following:

Fact (4.0): *Let $A_2 = (\vec{Q}, \tilde{Q}, \Sigma, \Sigma \cup \bar{\Sigma}, \bullet, q_0, F)$ be a two-way finite automaton whose control language is $L^{(1)}$, and whose two-way language is $L^{(2)} \subseteq \Sigma^*$ (defined with respect to any one of the four modes of two-way acceptance). Then the language $L^{(1)}$ by itself determines a two-way finite automaton accepting $L^{(2)}$ (if we take the same mode of two way acceptance as for A_2), whose control language is also $L^{(1)}$.*

Proof: The control language $L^{(1)}$ determines its own minimum (one-way) automaton, described above (whose state set is $Q_m = \{ w^{-1} L^{(1)} / w \in (\Sigma \cup \bar{\Sigma})^* \}$, etc.). In order *to obtain a two-way automaton from $L^{(1)}$, we define:*

$$\vec{Q}_m = \{ w^{-1} L^{(1)} \in Q_m / w^{-1} L^{(1)} \cap \Sigma (\Sigma \cup \bar{\Sigma})^* \neq \varnothing \},$$
$$\tilde{Q}_m = \{ w^{-1} L^{(1)} \in Q_m / w^{-1} L^{(1)} \cap \bar{\Sigma} (\Sigma \cup \bar{\Sigma})^* \neq \varnothing \}.$$

Equivalently, a state $w^{-1} L^{(1)}$ is right-moving iff there exists at least one letter $a \in \Sigma$ such that $(w^{-1} L^{(1)}) \circledcirc a = a^{-1} w^{-1} L^{(1)} \neq \varnothing$ (it is easy to check that for any alphabet A and any $L \subseteq A^*$, and $a \in A$ we have: $a^{-1} L \neq \varnothing$ iff $L \cap a A^* \neq \varnothing$). So $\vec{Q}_m$ is the set of those states in Q_m on which the action of some letter in Σ is defined. Similarly, $\tilde{Q}_m$ is the set of those states in Q_m on which the action of some letter in $\bar{\Sigma}$ is defined.

Continuing the description of the two-way automaton determined by $L^{(1)}$: the start state, accept states and next-state function are exactly the same as for the minimum automaton of $L^{(1)}$.

It is easy to check that this two-way automaton $(\vec{Q}_m, \tilde{Q}_m, \Sigma, \Sigma \cup \bar{\Sigma}, \circledcirc, \text{etc.})$, constructed form $L^{(1)}$, accepts $L^{(2)}$ (according to the same mode of two-way acceptance as the initial two-way automaton A_2), and has $L^{(1)}$ as its control language. ■

The rest of this section contains formulas which relate the control language $L^{(1)}$ to the various two-way languages.

LEMMA (4.1): *Let $u \in \Sigma^+$, and let $L^{(1)}$ be the control language of a given two-way automaton (with two-way language $L^{(2)}$).*

Then the set of traces of accepting left-to-right *computations on input u is exactly*

$$\{\rightarrow u \rightarrow\} \cap L^{(1)}.$$

Moreover:

$$u \in L^{(2)} \qquad \textit{iff} \quad \{\rightarrow u \rightarrow\} \cap L^{(1)} \neq \varnothing.$$

Similar formulas hold for the other modes of acceptance (left-to-left, etc.).

Proof: ($\Rightarrow$) If $u \in L^{(2)}$ then let w be the trace of an accepting two-way computation of u. Then (by definition of acceptance) $w \in \{\rightarrow u \rightarrow\}$ and $w \in L^{(1)}$.

($\Leftarrow$) If $\{\rightarrow u \rightarrow\} \cap L^{(1)}$ contains some element w, consider w as the trace of a two-way computation on input u (on tape), starting in state $L^{(1)}$ (which is the start state of the automaton for $L^{(2)}$ constructed from $L^{(1)}$ in Fact 4.0), and ending in state $w^{-1} L^{(1)}$, which satisfies $w \in L^{(1)}$, *i.e.* $w^{-1} L^{(1)}$ is an accept state. Hence $u \in L^{(2)}$. ■

THEOREM (4.2) (Pécuchet): (*a*) *Let $L^{(1)}$ be the control language of a two-way automaton, and let $L^{(2)}$ be the two-language relative to left-to-right acceptance. Then:*

$$L^{(2)} = \Sigma^* \cap red(L^{(1)}).$$

(*b*) *If L is any finite-state language $(\Sigma \cup \bar{\Sigma})^*$ then the language*

$$\Sigma^* \cap red(L) \quad \textit{in } \Sigma^* \textit{ is also finite-state.}$$

Similar regularity preserving operations can be obtained for the other modes of two-way acceptance.

Proof: We only consider the case of left-to-right acceptance (Pécuchet's theorem – see [7] for his original proof).

(*a*) Let $u \in L^{(2)}$; then by Lemma 4.1 there exists $w \in \{\rightarrow u \rightarrow\} \cap L^{(1)} (\neq \varnothing)$. Hence $u = red(w)$, and so $u \in \Sigma^* \cap red(L^{(1)})$.

Conversely, if $u \in \Sigma^* \cap red(L^{(1)})$, there exists $w \in L^{(1)}$ such that $u = red(w) \in \Sigma^*$. Hence $w \in \{\rightarrow u \rightarrow\}$. Now $w \in \{\rightarrow u \rightarrow\} \cap L^{(1)}$, hence $\{\rightarrow u \rightarrow\} \cap L^{(1)} \neq \varnothing$, so (by Lemma 4.1) $u \in L^{(2)}$.

(*b*) If $L \subseteq (\Sigma \cup \bar{\Sigma})^*$ is a finite-state language, we can construct (as in the proof of Fact 4.0) a two-way automaton whose control language is L. By part (*a*) of the present fact, the left-to-right two-way language of this automaton is equal to $\Sigma^* \cap red(L)$. Moreover, by the Rabin-Shepherdson theorem, this language $\Sigma^* \cap red(L)$ is finite-state (being recognized by a two-way finite automaton). ■

Remark: The converse of Pécuchet's formula is not true (as we saw already in Fact 3.3), *i.e. if* $\Sigma^* \cap red(L)$ is finite-state, that does not imply that L is finite-state.

5. QUESTIONS AND RESEARCH PROBLEMS

(*a*) Study formal power series (e.g. with coefficients in $\mathbb{N}$) accepted by two-way finite automata.

For one-way finite automata the formal-power-series approach is related to solving systems of left (or right) linear equations. What kind of "*equations*" (or other representations) should one use to find the power series accepted by a two-way automaton?

(*b*) A question which is related to power series is *ambiguity*.

It would be interesting to study *unambiguous* two-way finite automata (where every word has at most one accepting computation). Relate unambigous two-way automata to "*two-way bimachines*".

(*c*) Do the (non-rational) *transductions* $L \to \Sigma^+ \cap red(L)$ and $L \to clos(L)$ have interesting "*non-linear matrix representations*" (as studied by Pin and Sakarovitch [9])?

(*d*) Another problem, not necessarily related to two-way automata:

Find an automaton model (recognizing only finite-state languages) which corresponds to the languages accepted via non-linear matrix representations (see Theorem 2.4 in the English version of [9]).

(*e*) Find new examples of finite-state languages (and of transductions which preserve finite-stateness) for which finite-stateness (resp. preservation of finite-stateness) is most naturally proved by using two-way automata.

Examples are $L \to \Sigma^+ \cap red(L)$, $L \to clos(L)$, and

$$L \to \frac{1}{2}L = \{x/\exists y : xy \in L, |x| = |y|\} \quad \text{(see [6] 1979, p. 73).}$$

ACKNOWLEDGMENTS

This paper owes a lot to the paper [7] of Jean-Pierre Pécuchet. I would like to thank Prof. John Rhodes, Berkeley, for his encouragement, and Doug Albert for his comments. I would also like to thank Prof. Rudolf E. Kalman for giving me an opportunity to work at the Center for Mathematical System Theory, University of Florida, where a large part of this paper was written.

REFERENCES

1. J. Berstel, *Transductions and Context-Free Languages*, Teubner, Stuttgart, 1979.
2. J. C. Birget, Concatenation of Inputs in a Two-Way Automaton, Theoret. Comp. Sci., Vol. 63, 1989, pp. 141-156.
3. J. C. Birget, *Machines and expansions of a semigroup, and applications*, Ph. D. thesis, U. of California, Berkeley, May 1983.
4. J. C. Birget, *Arbitrary Versus Regular Semigroups*, J. Pure and Appl. Algebra, Vol. 34, 1984, pp. 56-115.
5. S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press, 1974.
6. J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, 1969, and *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.
7. J. P. Pécuchet, *Automates boustrophedon, semigroupe de Birget et monoïde inversif libre*, R.A.I.R.O. (Revue française d'automatique, d'informatique et de rech. opérat.), Informatique théorique, Vol. 19.1, 1985, pp. 71-100.
8. J. C. Shepherdson, *The Reduction of Two-Way to One-Way Automata, I.B.M.* J. Res. and Dev., Vol. 3.2, 1959, pp. 198-200, and in E. F. Moore (Ed.), *Sequential Machines: Selected Papers*, Addison-Wesley, 1964.
9. J. E. Pin and J. Sakarovitch, *Some Operations and Transductions which Preserve Rationality*, 6th G.I. (= Gesellschaft für Informatik) Conference, Lecture Notes in Comp. Sci. (Springer Verlag) 145, pp. 277-288 and: *Une application de la représentation matricielle des transductions*, Theoretical Computer Science, 35, 1985, pp. 271-293.
10. J. C. Birget, *Proof of a Conjecture of R. Kannan*, Proc. 21st A.C.M. Symp. on Theory of Computing, 1989, pp. 445-453.