ELSEVIER

Theoretical Computer Science 267 (2001) 83–104

Theoretical Computer Science

www.elsevier.com/locate/tcs

Using acceptors as transducers ☆

Matti Nykänen

Department of Computer Science, University of Helsinki, P.O. Box 26, Teollisuuskatu 23, FIN-00014, Helsinki, Finland

Abstract

We wish to use a given nondeterministic two-way multi-tape acceptor as a transducer by supplying the contents for only some of its input tapes, and asking it to generate the missing contents for the other tapes. We provide here an algorithm for assuring beforehand that this transduction always results in a finite set of answers. We also develop an algorithm for evaluating these answers whenever the previous algorithm indicated their finiteness. Furthermore, our algorithms can be used for speeding up the simulation of these acceptors even when not used as transducers.

Keywords: Multi-tape automata; Transducers; Finiteness dependencies

1. Introduction

In this paper we study the following problem: assume that we are given a nondeterministic two-way multi-tape acceptor $\mathscr{A}$ and a subset X of its tapes. We would like to use $\mathscr{A}$ no longer as an acceptor which receives input on all its tapes, but instead as a kind of transducer [15, Chapter 2.7] which receives input on tapes X only and generates as output the set of missing inputs onto the other tapes. We then face the following two problems:

Problem 1. *Can it be guaranteed that given any choice of input strings for tapes X the set of corresponding outputs of $\mathscr{A}$ will always remain finite*?

Problem 2. *In those cases where Problem* 1 *can be solved positively, how can the actual set of outputs corresponding to a given choice of input strings be computed*?

☆ Supported by the Academy of Finland, grant number 42977.

E-mail address: matti.nykanen@cs.helsinki.fi (M. Nykänen).

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00297-8

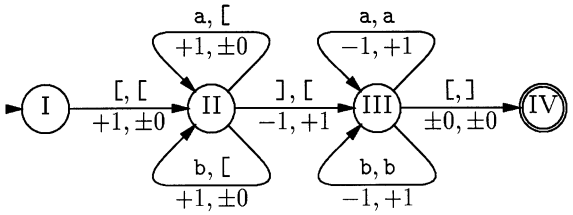

Fig. 1. A 2-FSA for recognizing strings and their reversals.

The motivation for studying these two problems came from *string databases* [3, 7, 11] which manipulate strings instead of indivisible atomic entities. Such databases are of interest, for example, in bioinformatics, because they allow the direct representation and manipulation of the stored nucleotide (DNA or RNA) sequences. While one can base a query language for these databases on a fixed set of sequence comparison predicates, such as in, for example, the PROXIMAL system [5], it would be more flexible to allow user-defined string processing predicates as well.

If we assume an SQL-like notation [1, Chapter 7.1] for the query language, then one possible query for such a string database might be stated as follows:

SELECT x_2
FROM x_1 IN R
WHERE $\phi_{\mathrm{rev}}(x_1, x_2)$

Here $\phi_{\mathrm{rev}}(x_1, x_2)$ is some user-defined expression which compares the strings w_1 and w_2 denoted by the variables x_1 and x_2, say "w_2 is the reversal of w_1". Then this query requests every string w_2 that is the reversal of some string w_1 currently stored in the database table R. Note, in particular, that these strings w_2 need (and in general can) not be stored anywhere in the database; the query evaluation machinery must generate them instead as needed.

We have developed elsewhere [10, 11, 17] a logical framework for such a query language. This framework accommodates expressions like $\phi_{\mathrm{rev}}(x_1, x_2)$ via a multidimensional extension of the modal extended temporal logic suggested by Wolper [29]. The multi-tape acceptors studied here are exactly the computational counterparts to these logical expressions.

A given query to a database is considered to be "safe" for execution if there is a way to evaluate its answer finitely [1, pp. 75–77]. One safe plan for evaluating the aforementioned query would be as follows, where $L(\mathscr{A}_{\mathrm{rev}})$ is the string relation accepted by $\mathscr{A}_{\mathrm{rev}}$, a multi-tape acceptor corresponding to the expression $\phi_{\mathrm{rev}}(x_1, x_2)$. (One such acceptor is shown as Fig. 1 below.)

for all strings w_1 in table R **do**
  $V \leftarrow \{w_2 : \langle w_1, w_2 \rangle \in L(\mathscr{A}_{\mathrm{rev}})\}$;
  output every string in V
end for

Our two problems stem from these safe evaluation plans. Problem 1 is "How could we infer from $\phi_{\text{rev}}(x_1, x_2)$ that the set V is always going to be finite for every string w_1 that could possibly appear in R?" Problem 2 is in turn "Once the finiteness of every possible V has been ensured, how can we simulate this $\mathscr{A}_{\text{rev}}$ (efficiently) for each w_1 to generate the V corresponding to this particular w_1?"

We have studied elsewhere [9, 17, Chapter 4.4] how solutions for Problem 1 can be used to guide the selection of appropriate safe execution plans. To this end, Section 1.2 presents the problem in not only automata but also database theoretic terms.

One possible solution would have been to restrict beforehand the language for the string handling expressions such as $\phi_{\text{rev}}(x_1, x_2)$ into one which ensures this finiteness by definition, say by fixing x_1 to be the input variable, which is mapped into the output variable x_2 as a kind of transduction [3, 7]. However, in logic-based data models [1], the use of transducers seems less natural than acceptors, because the concept of information flow from input to output is alien to the logical level, and of interest only in the query evaluation level. But we must eventually also evaluate our string database queries, and then we must infer which of our acceptors can be used as transducers, and how to perform these inferred transductions, and thus we face the aforementioned problems.

The rest of this paper is organized as follows. Section 1.1 presents the acceptors we wish to use as transducers, while Section 1.2 formalizes Problem 1. Section 2 first reviews what is already known about its decidability, and then presents our algorithms, which give sufficient conditions for answering Problem 1 in the affirmative. Section 3 presents then an explicit evaluation method for those acceptors that these algorithms accept, answering Problem 2 in turn. Finally, Section 4 concludes our presentation.

1.1. The automaton model

Let the alphabet Σ be a *finite* set of characters fixed in advance, let Σ^* denote the set of all finite sequences of these characters, let $\varepsilon \in \Sigma^*$ denote the empty sequence of this kind, and let w^t denote concatenating $t \in \mathbb{N}$ copies of $w \in \Sigma^*$, as usual. We shall study relations on these sequences, or *strings* drawn from Σ^* in what follows.

On the other hand, database theory often studies sequence database models where Σ is taken to be conceptually *infinite* instead, as in for example [7, 19, 22, 24, 25]. Then the emphasis is on data given as *lists* of data items provided by the user. Conversely, our emphasis is on data given as strings in an alphabet fixed beforehand by the database designer. In other words, our approach fixes an appropriate alphabet Σ as part of the database schema, while the list approach considers Σ as part of the data instead. However, our approach has been employed even for managing list data [2].

We, furthermore, assume *left and right tape end-markers* '[' *and* ']' not in Σ. Then we define the nth character of a given string $w \in \Sigma^*$ with length $|w| = m$ as

$$\underbrace{c_1 \ldots c_m}_{w}[n] = \begin{cases} [, & n = 0, \\ ], & n = m + 1, \\ c_n, & 1 \leqslant n \leqslant m. \end{cases}$$

Intuitively, our automaton model is a "two-way multi-tape nondeterministic finite state automaton with end-markers"; similar devices have been studied by, for example, Harrison and Ibarra [14] and Rajlich [20]. Formally, a *k-tape finite state automaton* (*k-FSA*) [11, Section 3; 17, Chapter 3.1] is a tuple $\mathscr{A} = \langle \Sigma, k, Q_{\mathscr{A}}, s_{\mathscr{A}}, F_{\mathscr{A}}, T_{\mathscr{A}} \rangle$ with six elements where

(1) Σ is the finite alphabet as explained above;
(2) $k \in \mathbb{N}$ is the number of tapes;
(3) $Q_{\mathscr{A}}$ is a finite set of *states*;
(4) $s_{\mathscr{A}} \in Q_{\mathscr{A}}$ is a distinguished *start* state;
(5) $F_{\mathscr{A}} \subseteq Q_{\mathscr{A}}$ is the set of *final* states; and
(6) $T_{\mathscr{A}}$ is a set of *transitions* of the form $p \xrightarrow[d_1,\ldots,d_k]{c_1,\ldots,c_k} q$, where $p, q \in Q_{\mathscr{A}}$, each $c_i \in \Sigma \cup \{[,]\}$, and each $d_i \in \{-1, 0, +1\}$.

We, moreover, require that $d_i = -1$ implies $c_i \neq [$ and $d_i = +1$ implies $c_i \neq]$; this ensures that the heads do indeed stay within the tape area limited by these end-markers.

A *configuration* of $\mathscr{A}$ on *input* $\boldsymbol{w} = \langle w_1, \ldots, w_k \rangle \in (\Sigma^*)^k$ is of the form $C = \langle p, n_1, \ldots, n_k \rangle$, where $p \in Q_{\mathscr{A}}$ and $0 \leqslant n_i \leqslant |w_i| + 1$ for all $1 \leqslant i \leqslant k$. This C corresponds intuitively to the situation, where $\mathscr{A}$ is in state p, and each head $i = 1, \ldots, k$ is scanning square number n_i of the tape containing string w_i. Hence, we say that $\langle q, n_1 + d_1, \ldots, n_k + d_k \rangle$ is a possible *next* configuration of C if and only if $p \xrightarrow[d_1,\ldots,d_k]{w_1[n_1],\ldots,w_k[n_k]} q \in T_{\mathscr{A}}$. Now $+1$ can be interpreted as "read forward", while -1 means "rewind the tape to read the preceding square again", and 0 "stand still". We call tape i of $\mathscr{A}$ *unidirectional* if no transition in $T_{\mathscr{A}}$ specifies direction -1 for it; otherwise tape i is called *bidirectional* instead.

A *computation* of $\mathscr{A}$ on input $\boldsymbol{w}$ is a sequence $\mathscr{C} = C_0 C_1 C_2 \ldots$ of these configurations, which starts with the *initial* configuration $C_0 = \langle s_{\mathscr{A}}, 0, \ldots, 0 \rangle$, and each C_{j+1} is a possible next configuration of the preceding configuration C_j. This computation $\mathscr{C}$ is *accepting* if and only if it is finite, its last configuration C_f has no possible next configurations, and the state of this C_f belongs to $F_{\mathscr{A}}$. The *language* $L(\mathscr{A})$ *accepted by* $\mathscr{A}$ consists of those inputs $\boldsymbol{w}$, for which there exists an accepting computation $\mathscr{C}$. Note that this language is a k-fold relation on strings in the general case.

Because $\mathscr{A}$ is nondeterministic, we can without loss of generality assume that no transitions leave the final states $F_{\mathscr{A}}$. We can, for example, introduce a new state $f_{\mathscr{A}}$ into $Q_{\mathscr{A}}$, and set $F_{\mathscr{A}} = \{f_{\mathscr{A}}\}$. Then for every state p previously in $F_{\mathscr{A}}$, and every character combination $c_1, \ldots, c_k \in \Sigma \cup \{[,]\}$, on which there is no transition leaving p, we add the transition $p \xrightarrow[0,\ldots,0]{c_1,\ldots,c_k} f_{\mathscr{A}}$. In this way, whenever a computation of $\mathscr{A}$ would halt in state p, it performs instead one extra transition into the (now unique) new final state $f_{\mathscr{A}}$, and halts there instead.

We often view $\mathscr{A}$ as a *transition graph* $G_{\mathscr{A}}$ with nodes $Q_{\mathscr{A}}$ and edges $T_{\mathscr{A}}$. In particular, a computation of $\mathscr{A}$ can be considered to trace a path $\mathscr{P}$ within $G_{\mathscr{A}}$ starting from node $s_{\mathscr{A}}$. It is furthermore expedient to restrict attention to *nonredundant* $\mathscr{A}$ where each state is either $s_{\mathscr{A}}$ itself or on some path $\mathscr{P}$ from it into some state in $F_{\mathscr{A}}$. Fig. 1

presents a 2-FSA $\mathscr{A}_{\mathrm{rev}}$ in this form where $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. The language $L(\mathscr{A}_{\mathrm{rev}})$ accepted by it consists of the pairs $\langle u, v\rangle$, where string v is the reversal of string u: looping in state II finds the right end of the bidirectional tape 1 without moving the unidirectional tape 2, while looping in state III compares the contents of these two tapes in opposite directions.

Another often useful simplification is the following way to detect mutually incompatible transition pairs.

Definition 1. Tape i of k-FSA $\mathscr{A}$ is *locally consistent* if and only if every consecutive pair

$$p \xrightarrow[d_1,\ldots,d_k]{c_1,\ldots,c_k} q \xrightarrow[d'_1,\ldots,d'_k]{c'_1,\ldots,c'_k} r$$

of transitions in $T_{\mathscr{A}}$ satisfies the condition

$$(d_i = 0 \Rightarrow c'_i = c_i) \wedge (d_i = +1 \Rightarrow c'_i \neq [\,]) \wedge (d_i = -1 \Rightarrow c'_i \neq]). \tag{1}$$

This ensures that there are configurations, in which this pair can indeed be taken; whether these configurations do ever occur in any computation is quite another matter. For example, both tapes in Fig. 1 are locally consistent. If, in particular, tape i is both unidirectional and locally consistent, then given any path

$$\mathscr{P} = s_{\mathscr{A}} \xrightarrow[d_{(1,1)},\ldots,d_{(k,1)}]{c_{(1,1)},\ldots,c_{(k,1)}} p_1 \xrightarrow[d_{(1,2)},\ldots,d_{(k,2)}]{c_{(1,2)},\ldots,c_{(k,2)}} p_2 \xrightarrow[d_{(1,3)},\ldots,d_{(k,3)}]{c_{(1,3)},\ldots,c_{(k,3)}} p_3 \cdots p_m$$

in $G_{\mathscr{A}}$ we can construct an input string

$$w_i = c'_1 c'_2 c'_3 \ldots c'_m$$

for tape i, which allows $\mathscr{P}$ to be followed, if we choose

$$c'_j = \begin{cases} c_{(i,j)} & \text{if } (d_{(i,j)} = +1 \vee j = m) \wedge (c_{(i,j)} \neq [\,), \text{ and} \\ \varepsilon & \text{otherwise.} \end{cases}$$

For example in Fig. 1 the w_2 from Eq. (2) spells out the sequence of transitions taken when looping in state III. Harrison and Ibarra provide a related construction for deleting unidirectional input tapes from multi-tape pushdown automata [14, Theorem 2.2], while Rajlich [20, Definition 1.1] allows the reading head to scan two adjacent squares at the same time for similar purposes.

Again the nondeterminism of $\mathscr{A}$ allows us to enforce Definition 1 for tape i at the cost of expanding the size of $\mathscr{A}$ by a factor of $(|\Sigma| + 2)$: construct a k-FSA $\mathscr{B}$ with state space $Q_{\mathscr{B}} = Q_{\mathscr{A}} \times (\Sigma \cup \{[,]\})$, which remembers the character under tape head i. Add for each transition $p \xrightarrow[d_1,\ldots,d_k]{c_1,\ldots,c_k} q$ the transitions $\langle p, c_i\rangle \xrightarrow[d_1,\ldots,d_k]{c_1,\ldots,c_k} \langle q, c'_i\rangle$ into $T_{\mathscr{B}}$ satisfying Eq. (1). Set finally $s_{\mathscr{B}} = \langle s_{\mathscr{A}}, [\rangle$ and $F_{\mathscr{B}} = F_{\mathscr{A}} \times (\Sigma \cup \{[,]\})$ to complete our construction.

1.2. The limitation problem

This section introduces our *limitation problem* [11, Definition 3.1; 17, Definition 3.3] concerning the automata defined in Section 1.1.

Definition 2. Given a $(k + l)$-FSA $\mathscr{A}$, determine if there exists a *limitation function* $\mathscr{W}: \mathbb{N}^k \to \mathbb{N}$ with the following property: if $\langle u_1, \ldots, u_k, v_1, \ldots, v_l \rangle \in L(\mathscr{A})$, then $\max\{|v_j| : 1 \leqslant j \leqslant l\} \leqslant \mathscr{W}(|u_1|, \ldots, |u_k|)$.

If this is the case, then we say that $\mathscr{A}$ satisfies the *finiteness dependency* [21] $\Gamma = \{1, \ldots, k\} \rightsquigarrow \{k+1, \ldots, k+l\}$. These dependencies are a special case of *functional* dependencies in database theory [1, Section 8.2]. Intuitively, $\mathscr{A}$ is a finite representation of the conceptually infinite database table $L(\mathscr{A}) \subseteq (\Sigma^*)^{k+l}$, while Γ assures that if we select rows from this table by supplying values for the columns $1, \ldots, k$, we do always receive a finite answer. In this way $\mathscr{A}$ can be used both safely and declaratively as a string processing tool within our string database model. Thus our goal is to treat the (user-defined) string processing operation $\mathscr{A}$ as just another relation as far as the database query language is concerned; such transparency is, in fact, being advocated for the forthcoming object/relational database proposal [4, pp. 49–55]. We discuss elsewhere [12] how this overall string processing mechanism of ours relates to this proposal and how it could be incorporated into such database management systems.

In terms of automata theory we require that for any *input* $\langle u_1, \ldots, u_k \rangle \in (\Sigma^*)^k$ the possible *outputs* $\langle v_1, \ldots, v_l \rangle \in (\Sigma^*)^l$ must remain a finite set. This is what is meant by "using acceptors as transducers": we supply strings for only some tapes (here $1, \ldots, k$) of the acceptor $\mathscr{A}$, and ask it to produce us all those contents for the missing tapes (here $k+1, \ldots, k+l$) it would have accepted given the known tape contents. The limitation problem is then to determine beforehand whether this computation will always return a finite result or not. Weber [27, 28] has studied a related question whether the set of *all* possible outputs on any inputs of a given transducer remains finite, and if so, what is the maximal output length.

2. Solving the limitation problem

The hardness of the limitation problem has been shown to depend crucially on the amount of bidirectional tapes in $\mathscr{A}$. The problem has been shown elsewhere to be undecidable for FSAs with *two* bidirectional tapes [11, Theorem 5.1; 17, Chapter 4.1]: given a Turing machine [15, Chapter 7] M one can write a corresponding 3-FSA $\mathscr{A}_M$ with two bidirectional tapes, which accepts exactly the tuples $\langle u, v, w \rangle$, where v and w together encode a sequence of computation steps taken by M on input u. Here v and w must be read twice, requiring bidirectionality. Then asking whether $\mathscr{A}_M$ satisfies $\{1\} \rightsquigarrow \{2, 3\}$ amounts to asking whether M is total. This read-twice construction is reminiscent of representing the valid computations of a given Turing machine as an intersection of two Context-Free Languages [15, Chapter 8.6], and shows that it is

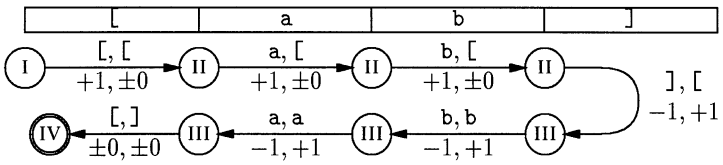

Fig. 2. A crossing behavior of the 2-FSA in Fig. 1.

also undecidable to determine whether a given finiteness dependency is satisfied by the intersection of the relations denoted by two given FSAs, even when these FSAs have no bidirectional tapes at all [17, Corollary 6.1].

On the other hand, the limitation problem becomes decidable if we restrict attention to those FSAs with at most *one* bidirectional tape [11, Theorem 5.2; 17, Chapter 4.2]. Intuitively, all the unidirectional tapes are first made locally consistent, after which Eq. (2) allows us to construct their contents at will, so that we can concentrate on the sole bidirectional tape. This tape can in turn be studied by using an extension of the well-known *crossing sequence construction* [15, Chapter 2.6] for converting two-way finite automata into classical one-way finite automata. This method is clearly impractical, however. Therefore, this paper presents in Section 2.1 a practical partial solution, which furthermore applies even in some cases involving multiple bidirectional tapes. Section 2.2 then develops this solution further to yield yet more explicit limitation information.

Example 3. The 2-FSA $\mathscr{A}_{\text{rev}}$ in Fig. 1 satisfies both $\{1\} \rightsquigarrow \{2\}$ and $\{2\} \rightsquigarrow \{1\}$ with the same limitation function $\mathscr{W}(n) = n$, because the reversal of a string is no longer than the string itself. This is moreover decidable, because only tape 1 is bidirectional in $\mathscr{A}_{\text{rev}}$. To see how limitation inference proceeds consider Fig. 2, which exhibits the crossing behavior of $\mathscr{A}_{\text{rev}}$ when tape 1 contains the string `ab`. For example, determining $\{2\} \rightsquigarrow \{1\}$ involves checking that every character written onto the bidirectional output tape 1 is "paid for" by reading something from the unidirectional input tape 2 as well, although this payment may occur much later during the computation; here it occurs when tape 1 is reread in reverse. This can in turn be seen from the automaton $\mathscr{B}$ produced by the crossing sequence construction by noting that the loops of $\mathscr{B}$ around the repeating crossing sequence indicated in Fig. 2 consume tape 2 as well.

The 2-FSA $\mathscr{A}_{\text{rev}}$ is also considered to satisfy the trivial finiteness dependency $\{1,2\} \rightsquigarrow \emptyset$ by definition. On the other hand, $\mathscr{A}_{\text{rev}}$ does not satisfy $\emptyset \rightsquigarrow \{1,2\}$, because $L(\mathscr{A}_{\text{rev}})$ is not finite.

2.1. An algorithm for determining limitation

Our technique for solving the limitation problem given in Definition 2 is based on the following two observations. Let $\mathscr{A}$ be the $(k+l)$-FSA and $\Gamma = \{1,\ldots,k\} \rightsquigarrow \{k+1,\ldots,k+l\}$ the finiteness dependency in question.

```
function halting( G:transition graph G_A of a k-FSA A;
                  X:subset of {1,...,k}):𝔹;
 1: b ← 1;
 2: for all i ∈ {1,...,k} \ X do
 3:    H ← G without transitions that specify ']' for tape i;
 4:    b ← b ∧ (H contains no path from s_A into any state in F_A)
 5: end for
 6: return b
end halting;
```

Fig. 3. An algorithm for testing Observation 1.

Observation 1. *If $\mathscr{A}$ accepts some input $\langle w_1,\ldots,w_{k+l}\rangle$ with some computation $\mathscr{C}$, where some head $k \leqslant j \leqslant k+l$ never visits the corresponding right end-marker* ']', *then $\mathscr{A}$ also accepts all the suffixed inputs*

$$\{\langle w_1,\ldots,w_{j-1},w_j v,w_{j+1},\ldots,w_{k+l}\rangle\colon\ v\in\Sigma^*\}$$

with the same $\mathscr{C}$. Hence, $\mathscr{A}$ cannot satisfy Γ in this case.

Observation 2. *If, on the other hand, every accepting computation of $\mathscr{A}$ visits the right end-marker* ']' *on all output tapes, then the only way $\mathscr{A}$ can violate Γ is by looping while generating output onto some output tape but without* "*consuming*" *any of the inputs at the same time – that is, by returning again and again to read the same squares of the input tapes.*

However, the (un)decidability results mentioned in the beginning of this section indicate that reasoning about actual computations is infeasible. Thus, we reason instead about the structure of the transition graph $G_{\mathscr{A}}$. Therefore, instead of Observation 1, the algorithm in Fig. 3 merely tests that there is no path $\mathscr{P}$ from the start state $s_{\mathscr{A}}$ into a final state, which never requires ']' to appear on some output tape, whereas it would have sufficed to show that no such $\mathscr{P}$ is ever traversed during any accepting computation. ($\mathbb{B}$ denotes the Boolean type with values **0** as 'false' and **1** as 'true'.)

Similarly, the algorithm in Fig. 4 enforces a more stringent condition than Observation 2: every cycle $\mathscr{L}$ in $G_{\mathscr{A}}$, during which some output tape is advanced to direction $+1$, must also move some input tape i into direction ± 1 but not back into the opposite direction ∓ 1. Then this tape i acts as a *clock*, which eventually terminates the potentially dangerous repetitive traversals of $\mathscr{L}$. Again, if $\mathscr{A}$ violates Γ, then some $\mathscr{L}'$ failing this condition must exist in $G_{\mathscr{A}}$, but the converse need not hold, because repetitions of $\mathscr{L}'$ need not necessarily occur during any accepting computation of $\mathscr{A}$; Fig. 6 presents a loop which seems at first glance to generate arbitrary many copies of character 'a' onto its output tape 2, because it seems to move back and forth on its input tape 1. However, closer scrutiny reveals that this behavior is, in fact, impossible because the same square on tape 1 must first contain character 'a' in order to get into state q, but later character 'b' in order to get back into state p.

```
function looping( G:subgraph of G_A for a k-FSA A;
                  X:subset of {1,...,k}):𝔹;
 1: b ← 1;
 2: H_1,...,H_m ← the maximal strongly connected components of G;
 3: Delete from G all transitions between different components (a.);
 4: for all i ← 1,...,m do
 5:   if some tape j ∈ X winds to direction ±1 in H_i but not to ∓1 then
 6:     Delete from H_i all transitions that wind this tape j (b.);
 7:     d ← looping(H_i, X)
 8:   else
 9:     d ← no tape in {1,...,k} \ X winds into direction +1 in H_i
10:   end if
11:   b ← b ∧ d
12: end for
13: return b
end looping;
```

Fig. 4. An algorithm for testing Observation 2.

Making the tapes locally consistent as in Definition 1 will catch some of these impossible transition sequences, including all cases that arise due to the demands on the contents of the unidirectional tapes. On the other hand, Fig. 6 presents a bidirectional tape 1 which is already locally consistent but still impossible. If there is just one bidirectional tape altogether, then the crossing sequence construction alluded to above in Example 3 can be seen as a method for detecting these impossibilities and eliminating them from further consideration. Unfortunately, we have no method of this kind for the general case.

The more stringent condition given above is enforced by repeatedly deleting those transitions, which can justifiably be argued not to take part in any loops of the kind mentioned in Observation 2. This technique is related to analyzing the input–output behavior of logic programs [16, 26], which analyze the call graph of the given program component by component. However, our technique remains simpler, because our automata are more restricted than general logic programs.

More precisely, the edge deletions made by the algorithm in Fig. 4 can be justified as follows. Consider the first call made by the main algorithm in Fig. 5. Every loop mentioned in Observation 2 must clearly be contained in some component H_i of $G = G_{\mathscr{A}}$, the entire transition graph of the k-FSA $\mathscr{A}$:

(a) A transition between two different strongly connected components cannot then surely belong to any loop of this kind. The deletions in step 3 are therefore warranted.

(b) Any transition τ that winds the clock tape j selected for the current component H_i cannot belong to any loop of this kind either, because τ cannot be traversed indefinitely often. These traversals will namely wind the input tape j eventually onto either end-marker, because tape j is not wound into the opposite direction by any other transition τ' in H_i. The deletions in step 6 are therefore warranted as well.

```
function limited( A:k-FSA;
                  X:subset of {1,...,k}):𝔹;
 1: return halting(G_A, X) ∧ looping(G_A, X)
end limited;
```

Fig. 5. An algorithm for determining limitation.

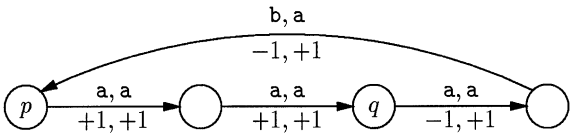

Fig. 6. A loop that cannot be traversed repeatedly.

This reasoning can then be applied in the subsequent recursive calls on the reduced components H_i as well, because we can then assume inductively that the loops broken during the earlier calls could not have been ones mentioned in Observation 2

Formalizing this reasoning shows that the algorithm in Fig. 5 is indeed correct as follows.

Theorem 4. *Let $\mathscr{A}$ be a $(p+q+r)$-FSA with tapes $1,\ldots,p$ unidirectional, and let the algorithm in Fig. 5 return* **1** *on $\mathscr{A}$ and $\{1,\ldots,p+q\}$. Then $\mathscr{A}$ satisfies $\{1,\ldots,p+q\} \rightsquigarrow \{p+q+1,\ldots,p+q+r\}$ with the function*

$$\mathscr{W}(m_1,\ldots,m_p,n_1,\ldots,n_q) = g_{\mathscr{A}}(m_1,\ldots,m_p,n_1,\ldots,n_q) - 1$$

where

$$g_{\mathscr{A}}(m_1,\ldots,m_p,n_1,\ldots,n_q) = |Q_{\mathscr{A}}| \left(\max(p,1) + \sum_{i=1}^{p} m_i\right) \left(\prod_{j=1}^{q} (n_j+2)\right).$$

Proof. Let us assume that $\mathscr{C}$ is an arbitrary computation of $\mathscr{A}$ on some input $\boldsymbol{z} = \langle u_1,\ldots,u_p,v_1,\ldots,v_q,w_1,\ldots,w_r\rangle$. We begin by proving the following two claims about this $\mathscr{C}$ which correspond to Observations 1 and 2.

Claim 5. *If $\mathscr{C}$ is accepting, then for every tape $p+q+1 \leqslant j \leqslant p+q+r$ the computation $\mathscr{C}$ takes some transition which requires '$]$' on tape j.*

Let otherwise h be a tape which violates this Claim 5. $\mathscr{C}$ traces a path through $G_{\mathscr{A}}$ from $s_{\mathscr{A}}$ into some state in $F_{\mathscr{A}}$. Then step 4 of the algorithm in Fig. 3 sets $b = \mathbf{0}$ when testing $i = h$ which violates our assumption that the algorithm in Fig. 5 returns **1**, thus proving this Claim 5.

Claim 6. *No head* $p+q+h$ *moves to direction* $+1$ *more than*

$$l = \mathscr{W}(|u_1|,\ldots,|u_p|,|v_1|,\ldots,|v_q|)+1$$

times during the computation $\mathscr{C}$.

Assume to the contrary that some h violates this Claim 6. Post a *fence* between two adjacent configurations C_g and C_{g+1} in $\mathscr{C}$ whenever tape $p+q+h$ moves to direction $+1$. By our contrary assumption, at least $l+1$ of these fences are posted. Consider, on the other hand, two configurations C_x and C_y of $\mathscr{C}$ to have the same *color* if and only if they share the same state and the same head positions for tapes $1,\ldots,p+q$. At most l of these colors are available, recalling the assumption that tapes $1,\ldots,p$ are unidirectional. Therefore, $\mathscr{C}$ must contain two configurations C_x and C_y which have the same color but are separated by an intervening fence. Consider then the sequence of transitions which transform C_x into C_y, as a path $\mathscr{L}$ within $G_{\mathscr{A}}$. This $\mathscr{L}$ forms a closed cycle, and tape heads $1,\ldots,p+q$ are on the same squares both before and after traversing $\mathscr{L}$, because C_x and C_y shared a common color. Let us then see which of the steps 3 or 6 of the algorithm in Fig. 4 will first delete some transition that belongs to $\mathscr{L}$. It cannot be step 3, because all of $\mathscr{L}$ belongs initially to the same maximal strongly connected component. But it cannot be step 6 either, because if $\mathscr{L}$ ever moves a tape $j\in X$ into some direction ± 1, it must also move tape j into the opposite direction ∓ 1 as well, in order to return its head onto the same square both before and after $\mathscr{L}$. Hence $\mathscr{L}$ persists untouched to the very end of the recursion on step 9, and there the presence of the transition of $\mathscr{L}$ that crosses the fence between C_x and C_y yields $d=\mathbf{0}$, which subsequently violates our assumption that the algorithm in Fig. 5 returns **1**, thus proving this Claim 6. □

Claims 5 and 6 are combined into a proof of the theorem as follows. Assume that $z\in L(\mathscr{A})$; that is, $\mathscr{A}$ has some accepting computation $\mathscr{C}$ on input z. It suffices to show that $|w_h|\leqslant l-1$ for every $1\leqslant h\leqslant r$. Tape head $p+q+h$ must cross every border between two adjacent tape squares from left to right, because otherwise $\mathscr{C}$ would not meet Claim 5. Claim 6 states in turn that $\mathscr{C}$ performs at most l crossings of this kind. This means that tape $p+q+h$ contains at most $l+1$ squares, of which the first and the last are reserved for the end-markers, leaving at most $l-1$ squares for the characters of the input string w_h. □

Example 7. Consider the 2-FSA $\mathscr{A}_{\mathrm{rev}}$ in Fig. 1. The algorithm in Fig. 5 can detect that it satisfies $\{1\}\rightsquigarrow\{2\}$ as follows. The algorithm in Fig. 3 returns **1**, because every path into the final state IV must contain III $\xrightarrow[0,0]{[,]}$ IV.

Evaluating the algorithm in Fig. 4 proceeds in turn as follows. First, all transitions from one state into another are deleted in step 3, leaving only the loops around states II and III. This is depicted in Fig. 7, where the components themselves are dotted, and the transitions between them (and thus deleted in step 3) are dashed. These loops are in turn deleted in step 6 when processing the corresponding components, and therefore this

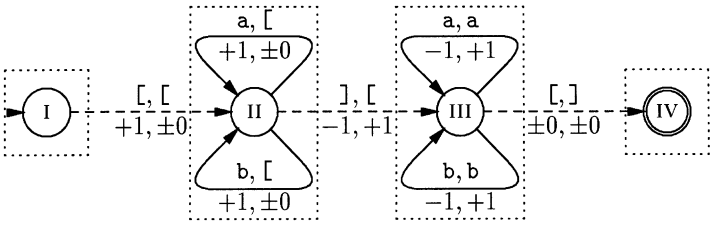

Fig. 7. The division of the 2-FSA in Fig. 1 into components.

function eventually returns **1** as well. However, Theorem 4 provides a rather imprecise limitation function $\mathscr{W}(n) = 4n + 7$ compared to the one given in Example 3.

On the other hand, the algorithm in Fig. 5 fails to detect $\{2\} \rightsquigarrow \{1\}$, which was detected in Example 3: looping in state II advances tape 1 without moving tape 2, and seems therefore dangerous to the algorithm in Fig. 4. Intuitively, $\mathscr{A}_{\text{rev}}$ first guesses nondeterministically some string, and only later verifies its guess against the input. In Example 3, crossing sequences were examined to see that this later checking in state III indeed reduces the acceptable outputs into only finitely many (here just one).

Essentially the same limitation function as in Theorem 4 suffices whenever all of the output tapes $p + q + 1, \ldots, p + q + r$ to be limited are unidirectional, even if $\{1, \ldots, p + q\} \rightsquigarrow \{p + q + 1, \ldots, p + q + r\}$ cannot be verified by the algorithm in Fig. 5 [9, Theorem 2.1]. This is natural, because the algorithm in Fig. 5 ignored the effects of moving any output tape $p + q + 1, \ldots, p + q + r$ into direction -1.

Theorem 8. *Let $p+1, \ldots, p+q$ be all the bidirectional tapes in the $(p+q+r)$-FSA $\mathscr{A}$, and let $\mathscr{A}$ satisfy $\Gamma = \{1, \ldots, p + q\} \rightsquigarrow \{p + q + 1, \ldots, p + q + r\}$. Then*

$$\mathscr{W}'(m_1, \ldots, m_p, n_1, \ldots, n_q) = (|\Sigma| + 2)^r g_{\mathscr{A}}(m_1, \ldots, m_p, n_1, \ldots, n_q) - 1$$

is a corresponding limitation function, where $g_{\mathscr{A}}$ is as in Theorem 4.

Proof. Consider the proof of Theorem 4, and assume further in Claim 6 that all the output tapes $p+q+1, \ldots, p+q+r$ are made locally consistent as in Definition 1; this new assumption introduces the factor $(|\Sigma| + 2)^r$ into $\mathscr{W}'$. With this modification, the original fencing-coloring construction shows that if some accepting computation $\mathscr{C}$ on input z advances some output tape $p+q+h$ more than $\mathscr{W}'(|u_1|, \ldots, |u_p|, |v_1|, \ldots, |v_q|)+1$ times, then the path of transitions taken by this $\mathscr{C}$ can be partitioned into three subpaths $\mathscr{K}\mathscr{L}\mathscr{M}$ where $\mathscr{L}$ begins in C_x and ends in C_y which share the same color and contains a transition τ that crosses some fence between C_x and C_y. However, now $\mathscr{A}$ must also accept all the pumped inputs

$$\langle u_1, \ldots, u_p, v_1, \ldots, v_q, w_{(1,\mathscr{K})} w_{(1,\mathscr{L})}^t w_{(1,\mathscr{M})}, \ldots, w_{(r,\mathscr{K})} w_{(r,\mathscr{L})}^t w_{(r,\mathscr{M})} \rangle,$$

where $t \in \mathbb{N}$, and each $w_{(k,\mathscr{J})}$ denotes the string of characters in those squares of output tape $p + q + k$, onto which the head lands during $\mathscr{J}$ (']' excluded). This is in effect

an application of Eq. (1) to the output tapes $p+q+1,\ldots,p+q+r$. The presence of τ within $\mathscr{L}$ shows that $w_{(h,\mathscr{L})} \neq \varepsilon$, and hence Γ fails by observation 2, thereby proving this modified Claim 6.

Claim 5 continues to hold, as reasoned in Observation 1, and the theorem follows again as before. □

Turning now to assessing when the algorithm in Fig. 5 does detect finiteness dependencies we see that it is successful at least when all tapes are unidirectional.

Theorem 9. *Let $\mathscr{A}$ be a nonredundant $(k+l)$-FSA with all tapes unidirectional and locally consistent, and let the algorithm in Fig. 5 return* **0** *on $\mathscr{A}$ and $\{1,\ldots,k\}$. Then $\mathscr{A}$ does not satisfy $\Gamma = \{1,\ldots,k\} \rightsquigarrow \{k+1,\ldots,k+l\}$.*

Proof. The nonredundancy of $\mathscr{A}$ and the unidirectionality and local consistency of all its tapes imply by Eq. (2) that for every path $\mathscr{P}$ in $G_{\mathscr{A}}$ we can always find an accepting computation $\mathscr{C}$ on some input $\boldsymbol{w}$ traversing $\mathscr{P}$. Letting $\mathscr{P}$ then be any subgraph of $G_{\mathscr{A}}$ which caused the algorithm in Fig. 5 to return **0** yields some $\mathscr{C}$ whose existence violates Γ along the lines of Theorem 8. □

2.2. Two variants of the limitation algorithm

This section explores two possible directions into which the algorithm given in Section 2.1 could be developed further. They both alter the nonrecursive step 9 of the algorithm in Fig. 4, which tests some strongly connected component H_i in the transition graph $G_{\mathscr{A}}$ of $\mathscr{A}$. Moreover, this H_i is known not to be shrinkable further by the algorithm.

The first direction *enlarges* the set of FSAs $\mathscr{A}$ and finiteness dependencies Γ that can be verified to hold by relaxing this test as follows. Suppose H_i contained some output tape $j \in \{1,\ldots,k\}\backslash X$ that is wound into the reverse direction -1 but not into the forward direction $+1$. Then this tape j can again be used as a clock for shrinking H_i further, similarly to steps 5–7, because the head on tape j cannot travel backwards forever, but must stop at least once the left end-marker '[' is reached.

Algorithmically, this direction leads to replacing steps 5–10 of the algorithm in Fig. 4 with the steps given in Fig. 8.

The other direction into which the algorithm given in Section 2.1 can be developed is to *constrain* further the set of FSAs $\mathscr{A}$ and finiteness dependencies Γ that can be verified to hold by restricting the test performed on step 9 of the algorithm in Fig. 4 to require that the component H_i being tested must not have any transitions left. Call the correspondingly modified limitation algorithm of Section 2.1 *fastidious*; it thus requires that *all* the transitions of $\mathscr{A}$ are deleted in order to verify Γ.

Example 10. The calculations in Example 7 show that the 2-FSA $\mathscr{A}_{\text{rev}}$ in Fig. 1 does actually satisfy the finiteness dependency $\{1\} \rightsquigarrow \{2\}$ even fastidiously.

if some tape $j \in X$ winds to direction ± 1 in H_i but not in direction ∓ 1
then
  Delete from H_i all transitions that wind this tape j (b.);
  $d \leftarrow \mathsf{looping}(H_i, X)$
else if some tape $j \in \{1,\ldots,k\} \setminus X$ winds to direction -1 in H_i but not in direction $+1$ **then**
  Delete from H_i all transitions that wind this tape j;
  $d \leftarrow \mathsf{looping}(H_i, X \cup \{j\})$
else
  $d \leftarrow$ no tape in $\{1,\ldots,k\} \setminus X$ winds into direction $+1$ in H_i
end if

Fig. 8. The enlarging additions to the algorithm in Fig. 4.

One advantage of fastidious verification is more efficient simulation of $\mathscr{A}$, as will be explained in Section 3.1. The rest of this section explains another advantage, namely that it enables straightforward construction of better explicit limitation functions than the generic one provided by Theorem 4. This construction is similar in spirit to van Gelder's analysis of logic programs with systems of equations [26], except that recurrences are used instead.

Let therefore $\mathscr{A}$ be a $(k+l)$-FSA satisfying the finiteness dependency $\Gamma = \{1,\ldots,k\} \rightsquigarrow \{k+1,\ldots,k+l\}$ fastidiously. A suitable limitation function would evidently be

$$\mathscr{W}''(m_1,\ldots,m_k) = \max\{f_j^{s_{\mathscr{A}}}(m_1,\ldots,m_k;0,\ldots,0;0):\ 1 \leqslant j \leqslant l\} - 1 \tag{2}$$

if each auxiliary function $f_j^p(m_1,\ldots,m_k;n_1,\ldots,n_k;h)$ provided some upper limit to the character position on output tape $k+j$ where the right end-marker '⌋' is encountered. Here $\mathscr{A}$ is assumed to be in state $p \in Q_{\mathscr{A}}$, each of its input tapes $1 \leqslant i \leqslant k$ on character $0 \leqslant n_i \leqslant m_i + 1$ of an unspecified input string with length $m_i \in \mathbb{N}$ and its designated output tape $k+j$ on character h of some unspecified output string.

These auxiliary functions can in turn be obtained by labeling the transition graph $G_{\mathscr{A}}$ of $\mathscr{A}$ with suitable expressions as follows:

- The expression for a state $p \in Q_{\mathscr{A}}$ is the maximum of the expressions for those transitions $\tau \in T_{\mathscr{A}}$ that leave from p:

$$f_j^p(m_1,\ldots,m_k;n_1,\ldots,n_k;h) = \max\{f_j^\tau(m_1,\ldots,m_k;n_1,\ldots,n_k;h):\ \tau \in T_{\mathscr{A}} \text{ leaves from } p\}. \tag{3}$$

  Graphically speaking, one can consider each node p of graph $G_{\mathscr{A}}$ to become labeled with the operator 'max' applied to the arrows that exit from p.
- The expression for a transition $\tau \in T_{\mathscr{A}}$ is the expression for the state that τ enters, adjusted with the effects of τ on the tapes in question:

$$f_j^{p \xrightarrow[b_1,\ldots,b_k,d_1,\ldots,d_l]{c_1,\ldots,c_k,e_1,\ldots,e_l} q}(m_1,\ldots,m_k;n_1,\ldots,n_k;h) = \begin{cases} \psi & \text{if } \bigwedge\limits_{1 \leqslant i \leqslant k} \phi_i, \\ 1 & \text{otherwise,} \end{cases} \tag{4}$$

where

$$\psi = \begin{cases} h & \text{if } e_j =], \\ f_j^q(m_1,\ldots,m_k;n_1+b_1,\ldots,n_k+b_k;h+d_j) & \text{otherwise, and,} \end{cases}$$

$$\phi_i = \begin{cases} (n_i = 0) & \text{if } c_i = [, \\ (n_i = m_i + 1) & \text{if } c_i =], \\ (1 \leqslant n_i \leqslant m_i) & \text{otherwise.} \end{cases}$$

Here the "otherwise" branch of Eq. (4) denotes the case when the transition cannot apply by virtue of some input tape head position n_i violating its corresponding restriction ϕ_i. Then value 1 is warranted, because it is the earliest possible position in which '$]$' can possibly appear.

Graphically speaking, this shows how to construct the expressions for the arrows maximized by node p of graph $G_{\mathscr{A}}$ in Eq. (3) above.

Fastidiousness guarantees that this labeling does indeed yield a function: otherwise the expression for some $f_j^p(m_1,\ldots,m_k;n_1,\ldots,n_k;h)$ refers (indirectly) back to itself with no change to its arguments $n_1,\ldots,n_k$. This, however, implies a cycle C in $G_{\mathscr{A}}$ from p back to itself for which none of the input tapes $1,\ldots,k$ moves in exactly one direction. This in turn means that C could not be deleted by the fastidious limitation algorithm after all.

Accordingly, if a fastidious version of the algorithm in Fig. 8 is used instead, the labeling must then include all of $h_1,\ldots,h_l$ instead of just h, because then also the output tapes $k+1,\ldots,k+l$ may have been used in deleting transitions from the $(k+l)$-FSA $\mathscr{A}$ in question, and thus these output tape head positions might be the ones that cannot repeat as arguments for some expression.

On the other hand, dropping the fastidiousness restriction does not altogether invalidate this approach either, it just makes it more difficult to provide an explicit counterpart for Eq. (3). It is namely now possible that the expression for some $f_j^p(m_1,\ldots,m_k;n_1,\ldots,n_k;h)$ does indeed refer back to itself without changing its arguments $n_1,\ldots,n_k$: follow the expressions f_j^τ of Eq. (4) for the transitions τ that still remain in the component containing state p even after the successful execution of the algorithm in Fig. 4. However, these expressions f_j^τ are also guaranteed not to increase h, for otherwise the execution of the algorithm in Fig. 4 would have been unsuccessful. Thus a function for f_j^p does still exist, even though giving an explicit expression for it is difficult.

A similar labeling technique suffices even for the crossing sequence construction mentioned in Example 3 instead of a fastidious algorithm from Section 2.1, provided that the labeling is performed relative to the resulting crossing sequence automaton instead of the original [17, Chapter 5.2].

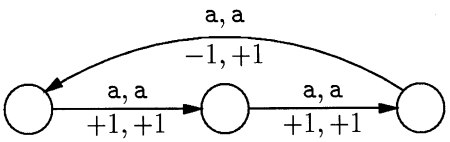

Fig. 9. A bidirectional loop that eventually ends.

Example 11. The labeling to generate Eq. (3) for Example 10 proceeds as follows. Applying Eq. (5) to the transitions leaving state III leads to

$$f_1^{\mathrm{III}\ \xrightarrow[\pm 0,\pm 0]{[.]}\ \mathrm{IV}}(m;n;h)=\begin{cases} h & \text{if } n=0,\\ 1 & \text{otherwise,}\end{cases}$$

$$f_1^{\mathrm{III}\ \xrightarrow[-1,+1]{a,a}\ \mathrm{III}}(m;n;h)=f_1^{\mathrm{III}\ \xrightarrow[-1,+1]{b,b}\ \mathrm{III}}(m;n;h),$$

$$=\begin{cases} f_1^{\mathrm{III}}(m;n-1;h+1) & \text{if } 1\leqslant n\leqslant m,\\ 1 & \text{otherwise}\end{cases}$$

which appear in Eq. (4) for state III, namely

$$f_1^{\mathrm{III}}(m;n;h)=\max\left\{\begin{array}{l} f_1^{\mathrm{III}\ \xrightarrow[\pm 0,\pm 0]{[.]}\ \mathrm{IV}}(m;n;h), f_1^{\mathrm{III}\ \xrightarrow[-1,+1]{a,a}\ \mathrm{III}}(m;n;h),\\ f_1^{\mathrm{III}\ \xrightarrow[-1,+1]{b,b}\ \mathrm{III}}(m;n;h)\end{array}\right\},$$

$$=\begin{cases} h & \text{if } n=0,\\ f_1^{\mathrm{III}}(m;n-1;h+1) & \text{if } 1\leqslant n\leqslant m,\\ 1 & \text{otherwise,}\end{cases}$$

$$=\begin{cases} h+n & \text{if } 0\leqslant n\leqslant m,\\ 1 & \text{otherwise,}\end{cases}$$

where the last simplification solves the recurrence found above for f_1^{III}. Continuing similarly for state II leads eventually to the tight limitation function $\mathscr{W}''(m)=m$ mentioned in Example 3.

A different possibility for improving on the limitation algorithm given in Section 2.1 would be to take into account not only the directions but also the total amount of tape movement. For instance, the current algorithms will not break a loop like in Fig. 9, because the input tape 1 in question moves in both directions, even though the *overall net effect* of these movements is $+1$, or to move one square forward, and therefore the loop cannot execute indefinitely. Calculating such net effects have

been recently studied in [18], but the resulting algorithms differ significantly from the approach presented here.

3. Evaluation of the limited answers

After inferring that the given $(k+l)$-FSA $\mathscr{A}$ does indeed satisfy the given finiteness dependency $\Gamma = \{1,\ldots,k\} \rightsquigarrow \{k+1,\ldots,k+l\}$ we then want to generate the (finite) set of outputs $\boldsymbol{v} = \langle v_1 \ldots, v_l \rangle$ for a given input $\boldsymbol{u} = \langle u_1,\ldots,u_k \rangle$, or to solve Problem 2. This problem is known to be difficult in the general case: let $\mathscr{B}$ be a 2-FSA with an unidirectional input tape 1 and a bidirectional output tape 2, and ask if a given input u can produce *any* output v. This problem is equivalent to whether $\mathscr{B}$, considered as a *checking stack automaton*, accepts u [20, Theorem 5.1] which is known to be either PSPACE- or NP-complete, depending on whether $\mathscr{B}'$ is a part of the instance or not [6, Problem AL5]. However, the additional information Γ provides certain optimization possibilities.

A straightforward way to obtain an evaluation algorithm is to convert the output tapes from *read-only* into *write-once*, and perform these writing operations concurrently with the simulation of the nondeterministic control. Fig. 10 shows the resulting algorithm where the simulations of all the possible computations are performed in a depth-first order using a stack S. The algorithm maintains for each $1 \leqslant j \leqslant l$ an extensible character array $W_j[0 \ldots L_j]$ which holds the contents for the tape squares the output head $k+j$ has already examined during the computation $\mathscr{C}$ of $\mathscr{A}$ currently being simulated. Fig. 11 shows the indices during one simulation of the 2-FSA $\mathscr{A}_{\text{rev}}$ from Fig. 1, where the input tape 1 contains the string `ab` whose reversal is being generated onto the output tape 2.

In Fig. 10, $T_{\mathscr{A}}$ is enumerated as $\tau_1,\ldots,\tau_{|T_{\mathscr{A}}|}$, and τ_0 is a new starting pseudo-transition $\cdots \xrightarrow[0,\ldots,0]{\cdots} s_{\mathscr{A}}$. We also assume as in Section 1.1 that no final state in $F_{\mathscr{A}}$ has any outgoing transitions.

Note that Γ alone does not guarantee that the algorithm in Fig. 10 halts, it just guarantees that only finitely many different outputs are ever generated. Consider namely the situation in Fig. 12, where the 2-FSA $\mathscr{A}_{\text{rev}}$ in Fig. 1 is being used as a transducer in the opposite direction to Fig. 11: input is read from tape 2 and written onto tape 1. As explained in Example 7, $\mathscr{A}_{\text{rev}}$ is now guessing nondeterministically a possible output for later verification against the input. But how long guesses should $\mathscr{A}_{\text{rev}}$ be allowed to make?

This question can be answered by adding the extra conditions

$$W_j[L_j] = \text{`]'} \vee L_j \leqslant \mathscr{W}(|u_1|,\ldots,|u_k|) \tag{5}$$

for each $1 \leqslant j \leqslant l$ into branch 15 of the algorithm in Fig. 10 where $\mathscr{W}$ is a limitation function corresponding to Γ. This addition is warranted, because if during the currently simulated computation $\mathscr{C}$ some j violates Eq. (5) then more than $\mathscr{W}(|u_1|,\ldots,|u_k|)$

procedure simulate($\mathcal{A}$:$(k+l)$-FSA;
$u_1, \ldots, u_k$:Σ^*);
1: Set $m_i \leftarrow 0$ for all $1 \leq i \leq k$;
2: Set $n_j \leftarrow 0$, $L_j \leftarrow 0$ and $W_j[L_j] \leftarrow$ [for all $1 \leq j \leq l$;
3: Set $t \leftarrow 1$;
4: Initialize stack S to contain $\langle 0; L_1, \ldots, L_l \rangle$ only;
5: **while** S is nonempty **do**
6: Let $\langle t'; L'_1, \ldots, L'_l \rangle$ be top element in S, and let $\tau_{t'} = p \xrightarrow[d'_1,\ldots,d'_k,e'_1,\ldots,e'_l]{\cdots} q$;
7: **if** $q \in F_{\mathcal{A}}$ **then**
8: **output**$(v_1, \ldots, v_l)$ where each $v_j = W_j[1] \ldots W_j[L_j - 1]$;
9: Set $t \leftarrow |T_{\mathcal{A}}| + 1$
10: **else if** $t > |T_{\mathcal{A}}|$ **then**
11: Set $m_i \leftarrow m_i - d'_j$ for all $1 \leq i \leq k$;
12: Set $n_j \leftarrow n_j - e'_j$ and $L_j \leftarrow L'_j$ for all $1 \leq j \leq l$;
13: Pop off the top element from S;
14: Set $t \leftarrow t' + 1$;
15: **else if** $\tau_t = q \xrightarrow[d_1,\ldots,d_k,e_1,\ldots,e_l]{u_1[m_1],\ldots,u_k[m_k],c_1,\ldots,c_l} r$ satisfies $(n_j \leq L_j) \Rightarrow (W_j[n_j] = c_j)$ for all $1 \leq j \leq l$ **then**
16: Push $\langle t; L_1, \ldots, L_l \rangle$ into S;
17: Set $m_i \leftarrow m_i + d_i$ for all $1 \leq i \leq k$;
18: Set $L_j \leftarrow L_j + 1$ and $W_j[L_j] \leftarrow c_j$ for all those $1 \leq j \leq l$ on which $n_j > L_j$ holds;
19: Set $n_j \leftarrow n_j + e_j$ for all $1 \leq j \leq l$;
20: Set $t \leftarrow 1$
21: **else**
22: Set $t \leftarrow t + 1$
23: **end if**
24: **end while**
end simulate;

Fig. 10. An algorithm for using acceptors as transducers.

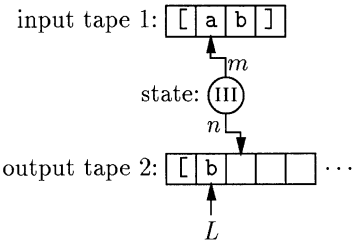

Fig. 11. Simulating the 2-FSA in Fig. 1 as a transducer.

characters from Σ have been output onto tape $k+j$. In this case $\mathscr{C}$ must be eventually rejecting and can hence be discarded at once without further ado.

Now that the L_j have been bounded by $\mathscr{W}$ the stack S will always contain only finitely many different configurations C of the transducer being simulated on input $\boldsymbol{u}$. (These transducer configurations C can be defined in a straightforward way by

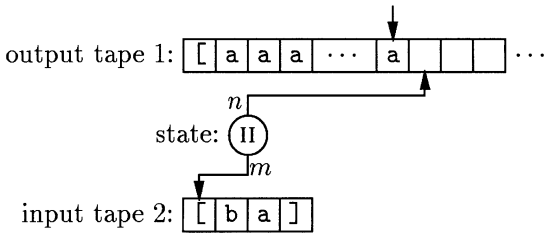

Fig. 12. Generating an indefinitely long output with the 2-FSA in Fig. 1.

extending the acceptor configurations defined in Section 1.1 with write-once output tapes.) Although stack S represents these configurations C only implicitly, they can be reconstructed as in branch 10 of the algorithm in Fig. 10. However, some of these configurations C can still repeat, because the transducer being simulated can also loop on the already known parts of its tapes without generating new output. Fortunately, this looping can be detected and eliminated simply by testing in branch 15 of the algorithm in Fig. 10 that the new configuration C_{new} being pushed into stack S does not yet occur in stack S. This is a standard way to avoid repetition during a depth-first search [23, Chapter 3.6]. We have also experimented with comparing C_{new} against *all* the configurations C encountered so far in the entire search conducted by the algorithm in Fig. 10 on the current input $\boldsymbol{u}$, but this proved to be extremely inefficient in practice.

Now, we have solved Problem 2 by developing a halting variant of the evaluation algorithm in Fig. 10. However, this solution suffers from two drawbacks.

Drawback 1. *The value of a limitation function is needed in Eq.* (5) *to estimate – and hopefully tightly – the depth at which ultimately rejecting output-generating computations can be pruned.*

This could be termed the "compile-time" drawback: the limitation function must be formed when the acceptor is proposed as a possible transducer, while its value is required before each invocation of the simulation algorithm in Fig. 10.

Drawback 2. *The whole stack S must be scanned against repeating configurations C when pushing each new configuration* C_{new} *in the algorithm in Fig.* 10.

This could in turn be termed the "run-time" drawback, because it adds loop checking overhead the execution of the simulation algorithm in Fig. 10.

Fortunately, both of these drawbacks can be alleviated by considering *how* Γ was inferred to hold, as discussed in the remainder of this section.

3.1. When the limitation algorithm succeeds

Consider first the case where Γ was inferred to hold by having the algorithm in Fig. 5 return **1** on $\mathscr{A}$ and Γ. Claim 6 in the proof of Theorem 4 shows that every

computation $\mathscr{C}$ of $\mathscr{A}$ is "self-limiting" in the sense that no L_j can grow indefinitely. Thus Eq. (5) is *not* needed after all, thereby alleviating Drawback 1.

Claim 6 alleviates also Drawback 2. Two occurrences C_x and C_y of the same configuration during $\mathscr{C}$ have the same color by definition. The proof of the claim shows that C_x and C_y can only arise by traversing a closed loop $\mathscr{L}$, which is not deleted during the algorithm in Fig. 4. We therefore modify this algorithm to *mark* in $\mathscr{A}$ the transitions it considers deleted. Then the algorithm in Fig. 10 can stop scanning its stack S as soon as the most recent marked transition is seen. This holds even when the marking has been performed by the enlarged algorithm in Fig. 8.

This reasoning shows another benefit of the fastidious variant of the algorithm in Fig. 4: then every transition gets marked, and therefore scanning the stack S is no longer required at all. That is, the algorithm in Fig. 10 suffices *unmodified* in this case, and all run-time loop checking overhead has been eliminated.

Example 12. Applying this modified marking algorithm in Fig. 4 into the 2-FSA $\mathscr{A}_{\mathrm{rev}}$ in Fig. 1 and $\Gamma = \{1\} \rightsquigarrow \{2\}$ succeeds even fastidiously by Example 10. Then the algorithm in Fig. 10 and $\mathscr{A}_{\mathrm{rev}}$ can generate the reversal of any given string in linear time with respect to its length. In other words, choosing this evaluation strategy leads into an optimal way to perform string reversals.

Note finally that this marking technique can also speed up the simulation of those m-FSAs $\mathscr{B}$ which are still used as acceptors and not transducers: just compute the marking given by the modified algorithm in Fig. 4 with $\mathscr{B}$ and $\{1,\ldots,m\}$ (which yields $\mathbf{1}$), and use the resulting stack scanning optimization strategy during the simulation of $\mathscr{B}$ on any given input $\langle u_1,\ldots,u_m\rangle$. Again the marks identify transitions under which it is not necessary to look when scanning the stack for repeating configurations during the simulation.

3.2. When all the outputs are unidirectional

Another strategy related to the one developed in Section 3.1 works when all the output tapes of $\mathscr{A}$ are unidirectional and the finiteness dependency Γ still holds but this fact can no longer be inferred by the algorithm developed in Section 2.1.

In this case the proof of Theorem 8 shows that a halting but still correct variant of the simulation algorithm in Fig. 10 can be obtained by adding into its branch 15 the extra condition that configurations C_x and C_y of the same color – in the sense of that proof – may not repeat within any computation $\mathscr{C}$: if the path $\mathscr{L}$ of transitions from C_x into C_y advances any of the unidirectional output tapes $p+q+1,\ldots,p+q+r$, then this $\mathscr{C}$ must be rejecting because it would violate Γ via Observation 2, while otherwise taking $\mathscr{L}$ during $\mathscr{C}$ was unnecessary because then C_x and C_y are the same configuration. This reasoning provides the loop checking discipline which guarantees the halting of the simulation algorithm in this case.

Furthermore, this simulation is also amenable to the stack scanning optimization technique developed in Section 3.1: a variant of the algorithm in Fig. 4 which merely attempts to mark every transition it possibly can – instead of trying to test for Γ and fail – identifies some of those cycles $\mathscr{L}$ that can cause some color to repeat. These marks can then again be used for limiting stack scanning during simulation.

4. Conclusions

We studied the problem of using a given nondeterministic two-way multi-tape acceptor as a transducer by supplying inputs onto only some of its tapes, and asking it to generate the rest. We developed a family of algorithms for ensuring that this transduction does always yield finite answers, and another family of algorithms for actually computing these answers when they are guaranteed to exist. In addition, these two families of algorithms provided a way to execute and optimize the simulation of nondeterministic two-way multi-tape acceptors by restricting the amount of work that must be performed during run-time loop checking.

These algorithms have been implemented in the prototype string database management system being developed at the Department of Computer Science in the University of Helsinki [8, 12, 13].

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.
[2] N. Balkir, E. Sukan, G. Ozsoyoglu, Z. Ozsoyoglu, VISUAL – a graphical icon-based query language, IEEE Internat. Conf. on Data Engineering (1996) 524–533.
[3] A. Bonner, G. Mecca, Sequences, datalog and transducers, J. Comput. System Sci. 57(3) (1998) 234–259.
[4] C. Date, H. Darwen, Foundation for Object/Relational Databases – The Third Manifesto, Addison-Wesley, Reading, MA, 1998.
[5] S. Ganguly, M. Noordewier, PROXIMAL: a database system for the efficient retrieval of genetic information, Comput. Biol. Med. 26(3) (1996) 199–207.
[6] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[7] S. Ginsburg, X. Wang, Regular sequence operations and their use in database queries, J. Comput. System Sci. 56(1) (1998) 1–26.
[8] G. Grahne, R. Hakli, M. Nykänen, E. Ukkonen, AQL: an alignment based query language for querying string databases, in: C. Prabhu (Ed.), Databases for the Millennium 2000: Proc. 9th Internat. Conf. on Management of Data (COMAD'98), Tata McGraw-Hill Publishing Company Limited, New Delhi, 1998, pp. 235–251.
[9] G. Grahne, M. Nykänen, Safety, translation and evaluation of alignment calculus, in Advances in Databases and Information Systems, Electronic Workshops in Computing, British Computer Society, 1997, pp. 295–304. <http://www.ewic.org.uk/ewic/workshop/view.cfm/ADBIS-97>.
[10] G. Grahne, M. Nykänen, E. Ukkonen, Reasoning about strings in databases, ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Database Systems, 1994, pp. 303–312.
[11] G. Grahne, M. Nykänen, E. Ukkonen, Reasoning about strings in databases, J. Comput. System Sci. 59 (1999) 116–162.
[12] R. Hakli, M. Nykänen, H. Tamm, A declarative programming system for manipulating strings, 6th Fenno-Ugric Symp. on Software Technology, 1999, pp. 29–40.

[13] R. Hakli, M. Nykänen, H. Tamm, E. Ukkonen, Implementing a declarative string query language with string restructuring, in: G. Gupta (Ed.), 1st Internat. Workshop on Practical Aspects of Declarative Languages, PADL'99, January 1999, Lecture Notes in Computer Science, vol. 1551, Springer, Berlin, pp. 179–195.
[14] M. Harrison, O. Ibarra, Multi-tape and multi-head pushdown automata, Inform. and Control 13 (1968) 433–470.
[15] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[16] R. Krishnamurthy, R. Ramakrishnan, O. Shmueli, A framework for testing safety and effective computability, J. Comput. System Sci. 52 (1996) 100–124.
[17] M. Nykänen, Querying string databases with modal logic, Ph.D. Thesis, Department of Computer Science, University of Helsinki, Finland, 1997.
[18] M. Nykänen, E. Ukkonen, Finding paths with the right cost, in: C. Meinel, S. Tison (Eds.), Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1563, 1999, pp. 345–355.
[19] P. Pistor, R. Traunmüller, A database language for sets, lists and tables, Inform. Systems 11 (4) (1986) 323–336.
[20] V. Rajlich, Absolutely parallel grammars and two-way finite-state transducers, J. Comput. System Sci. 6 (1972) 324–342.
[21] R. Ramakhrisnan, F. Bancilhon, A. Silberschatz, Safety of recursive Horn clauses with infinite relations (extended abstract), in: ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1987, pp. 328–339.
[22] J. Richardson, Supporting lists in a data model (a timely approach), Very Large Data Bases Conf., 1992, pp. 127–138.
[23] S. Russell, P. Norvig, Artificial Intelligence: a Modern Approach, Prentice-Hall, Englewood cliffs, NJ, 1995.
[24] P. Seshadri, M. Livny, R. Ramakrishnan, SEQ: a model for sequence databases, IEEE Internat. Conf. on Data Engineering, 1995, pp. 232–239.
[25] B. Subramanian, T. Leung, S. Vandenberg, S. Zdonik, The AQUA approach to querying lists and trees in object-oriented databases, IEEE Internat. Conf. on Data Engineering, 1995, pp. 80–89.
[26] A. van Gelder, Deriving constraints among argument sizes in logic programs, Tech. Report UCSC-CRL-89-41, University of California, Santa Cruz, 1989.
[27] A. Weber, On the valuedness of finite transducers, Acta Inform. 27 (1990) 749–780.
[28] A. Weber, On the lengths of values in a finite transducer, Acta Inform. 29 (1992) 663–687.
[29] P. Wolper, Temporal logic can be more expressive, Inform. and Control 56 (1983) 72–99.