Learning Rational Functions

Adrien Boiret, Aurélien Lemay, Joachim Niehren

Learning Rational Functions

Adrien Boiret[1,3], Aurélien Lemay[2,3], and Joachim Niehren[1,3]

[1] Inria, Lille
[2] University of Lille
[3] Mostrare project of Inria & Lifl (Cnrs Umr 8022)

Abstract. Rational functions are transformations from words to words that can be defined by string transducers. Rational functions are also captured by deterministic string transducers with lookahead. We show for the first time that the class of rational functions can be learned in the limit with polynomial time and data, when represented by string transducers with lookahead in the diagonal-minimal normal form that we introduce.

1 Introduction

Learning algorithms for regular languages of words or trees are usually based on the Myhill-Nerode theorem, that is on an algebraic characterization of the unique minimal automaton recognizing the target language [14,2,6,13]. The learning problem is then to identify this unique automaton in the limit from finite samples of positive and negative examples that characterize the language. For various classes of automata, this can be done in polynomial time in the size of the sample, while there exist characteristic samples of polynomial cardinality in the size of the target automaton. This approach has been established for finite deterministic automata (Dfas) [12,16], for deterministic tree automata [17], and for deterministic stepwise tree automata for unranked trees [3].

Learning algorithms for classes of transformation on words or trees can be obtained in an analoguous manner, if they can be defined by an appropriate class of deterministic transducers that enjoys a Myhill-Nerode type theorem. The classical example is the class of deterministic (subsequential) string transducers (Dts) [5,18]. It characterizes the unique minimal Dt for the target transformation, that is compatible with the domain and earliest in output production. Such transducers can be learned by the Ostia algorithm from finite samples of input-output pairs, under the assumption that a Dfa defining the domain is given [19]. More recently, this result could be extended to the class of deterministic top-down tree transducers with domain inspection [10,15]. Furthermore, a unique minimization result – that can be based on a Myhill-Nerode theorem – was obtained for deterministic bottom-up tree transducers [11].

The motivation of the present article is to extend these results to classes of transducers with look-ahead. The natural starting point is the class of deterministic string transducers with lookahead ($\textsc{Dt}_\ell$), which capture the class of rational functions (see e.g. [1]), i.e. they have the same expressiveness as functional string transducers [8]. Based on another Myhill-Nerode type theorem, Reutenaurer and Schützenberger showed in [20] that there exists a unique minimal look-ahead automaton compatible with the domain that can be used to define some $\textsc{Dt}_\ell$. The underlying Dt itself can be made earliest and minimal. This yields a unique two-phase minimal normal form for rational functions.

The learning problem – that remained open for many years – is whether one can learn rational functions from finite samples of input-output examples and a Dfa for the domain. In this paper, we contribute a positive answer in Gold's learning model from polynomial time and data, under the assumption that rational functions are represented by diagonal-minimal normal form. This is a new class of normal forms that we introduce concommitantly with a new learning algorithm based on diagonalization. The main problem was to overcome the difficulty to identify a two-phase minimal normal form from examples.

Outline. We first recall traditional results on rational and subsequential functions (Section 2) and then the result of Reutenauer and Schützenberger on two-phase $\textsc{Dt}_\ell$ normalization (Section 3). In section 4, we indicate how to build a look-ahead from a basic test over suffixes. In Section 6, we indicate how this test can be done from a finite sample which leads to section 5 where we present the complete learning algorithm.

2 Rational Functions

We assume an input alphabet Σ and an output alphabet Δ, both of which are finite sets. Input words in Σ^* are ranged over by u and v and output words in Δ^* by w. We are interested in partial functions $\tau \subseteq \Sigma^* \times \Delta^*$. We denote the domain of a partial function by $dom(\tau)$ and freely write $\tau(u) = w$ instead of $(u, w) \in \tau$.

A string transducer is a tuple $M = \langle \Sigma, \Delta, Q, init, rul, fin \rangle$ where Σ and Δ are finite alphabet for input and output words, Q is a finite set of states, $init \subseteq Q$ is a set of initial states, $fin \subseteq Q \times \Delta^*$ the set of final states equipped with output words, and $rul \subseteq (Q \times \Sigma) \times (\Delta^* \times Q)$ is a finite set of transitions. We say that $q \xrightarrow{a/w} q'$ is a rule of M if $(q, a, w, q') \in rul$, and that $q \xrightarrow{w}$ is a final output if $(q, w) \in fin$. This arrow notion is also used in graphical representations of string transducers.

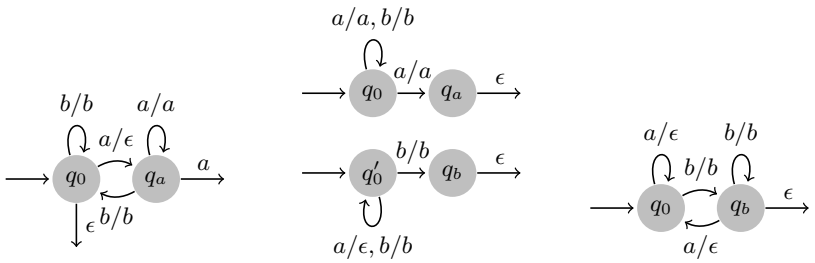


Fig. 1. (a) A DT for τ_1. (b) A string transducer for τ_2. (c) A DT for τ_3.

We denote by $[\![M]\!] \subseteq \Sigma^* \times \Delta^*$ the set of pairs (u, w) such that w is an output word that can be produced from input word u by M. More formally, a pair (u, w) belongs to $[\![M]\!]$ if there exists an index n, decompositions $u = a_1 \cdot \ldots \cdot a_n$ and $w = w_1 \cdot \ldots \cdot w_n \cdot w_f$, and a sequence of states $q_0 \cdot \ldots \cdot q_n$ such that $q_0 \in \mathit{init}$, $q_{i-1} \xrightarrow{a_i/w_i} q_i$ is a rule of M for all $1 \leq i \leq n$, and $q_n \xrightarrow{w_f}$ is a final output. A partial function is called rational if it is equal to $[\![M]\!]$ for some string transducer M, which is then called a functional transducer.

A string transducer is called deterministic or a DT (or subsequential) if it has at most one initial state and if *rul* and *fin* are partial functions. Clearly, every DT defines a rational function. Such functions are called subsequential, a notion going back to Schützenberger.

Example 1. The total function τ_1 on words with alphabet $\{a, b\}$ that erases all a's immediately followed by b is subsequential. See Fig. 1 for a DT defining it. Notice that the final output is needed, for instance for transducing the word aa correctly to itself.

The function τ_2 that deletes all a's in words whose last letter is b while performing the identity otherwise is rational, but not subsequential since the last letter cannot be predicted deterministically.

But if one restricts the domain of τ_2 to words ending by b, we obtain a partial function τ_3 which is subsequential, as illustrated in Fig. 1.

We denote by M_q the transducer equal to M except that q is the only initial state. A word $u \in \Sigma^*$ reaches a state q if there is a sequence of letters $a_1 \ldots a_n = u$ and of states $q_0 \ldots q_n$ such that $q_0 \in \mathit{init}$, $q_n = q$ and $q_{i-1} \xrightarrow{a_i/w_i} q_i$ is a rule of M for all $1 \leq i \leq n$ for some w_i. We call a DT M earliest if for all states q of M except the initial one, either the domain of $[\![M_q]\!]$ is the empty set or the least common prefix of all words in the range of $[\![M_q]\!]$ is the empty word.

Theorem 1 (Choffrut (1979) [4,5]). *Any subsequential function can be defined by some earliest* DT*. The earliest* DT *with a minimal number of states for a subsequential function is unique modulo state renaming.*

The DTs in Fig. 1 (a) and (c) are both earliest and minimal. Note that a smaller single state DT would be sufficient for defining τ_3 if the domain could be checked externally, which is not the case in this model.

Oncina and Varo [19] used the Myhill-Nerode behind Theorem 1 as a theoretical ground for a learning algorithm for subsequential functions τ from a finite sample $S \subseteq \tau$ and a DFA D recognizing the domain of τ.

Theorem 2 (Oncina and Varo (1996)). *For any* DFA D *there exists a learning algorithm* $\textsc{Ostia}_D$ *that identifies subsequential functions whose domain is recognized by* D *from polynomial time and data.*

That is: for any DT M defining a subsequential function τ whose domain is recognized by D there exists a finite sample $S \subseteq \tau$ called characteristic for τ, whose size is polynomial in the size of M, such that from any sample $S' \subseteq \tau$ that contains S, $\textsc{Ostia}_D(S')$ computes a DT defining τ in polynomial time in the size of S'.

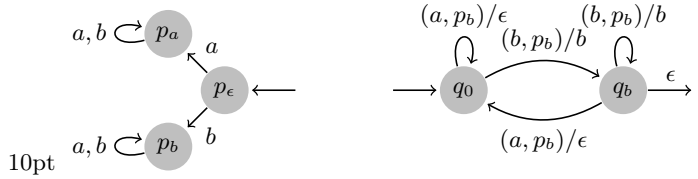
3 Transducers with Look-Ahead

As stated before, rational functions are captured by deterministic transducers with look-ahead. The look-ahead can be performed by some DFA that annotates the letters of the input word by states from right to left in a preprocessing step. The string transducer then processes the annotated word from left to right. More formally, we can identify a DFA A with alphabet Σ and state set P with a string transducer that reads the word right to left, while always outputing the pair of the current letter and the current state: an automaton rule $q \xrightarrow{a} q'$ of A is considered as a transducer rule $q \xrightarrow{a/(a,q')} q'$. This way, the rational function $[\![A]\!]$ maps a word $u \in \Sigma^*$ to the identical word but annotated with look-ahead states $[\![A]\!](u) \in (\Sigma \times P)^*$. Furthermore, the DFA used as a lookahead must be complete, so that it defines a total function.

A deterministic string transducer with look-ahead (DT_ℓ) is a pair $N = \langle A, M \rangle$ such that A is a DFA with alphabet Σ and state set P called the look-ahead, and M is a DT with signature $\Sigma \times P$ with state set Q. A DT_ℓ $N = \langle A, M \rangle$ defines the rational function $[\![N]\!] = [\![M]\!] \circ [\![A]\!]$: an input word $u \in \Sigma^*$ is first annotated with states of the look-ahead A from right to left, and then transformed by DT M from left to right. The following theorem is known as the decomposition theorem of Elgot and Mezei [8].

Theorem 3 (Elgot and Mezei (1965)). *A partial function* τ *is rational if and only if it is defined by some* DT_ℓ.

Fig. 2. The look-ahead for τ_2 and a matching DT.



10pt

Fig. 3. A look-ahead for τ_3, and a matching DT, both compatible with their domains.

Given a string transducer M that defines a partial function, the idea is to use a look-ahead automaton to annotate positions by the set P of those states of M by which a final state can be reached at the end of the word. One can then define a DT_ℓ N which simulates M except that it always selects an arbitrary transition leading to some state of P. Which of these transition is selected does not matter since M is functional

Example 2. A DT_ℓ for τ_2 is given in Fig. 2. Note that 3 look-ahead states are needed in order to distinguish suffixes ending with b or not.

We next study the question of whether there exists a unique minimal lookahead automaton for any rational function. We obtain a positive result by reformulating a Myhill-Nerode style theorem for bi-machines from Reutenauer and Schützenberger [20].

A relation $\sim$ over $\Sigma^* \times \Sigma^*$ is called a left-congruence if $v_1 \sim v_2$ implies $u \cdot v_1 \sim u \cdot v_2$ for all input words v_1, v_2, u. Every look-ahead automaton A defines a left-congruence $\sim_A$ such that $v_1 \sim_A v_2$ if and only if v_1 and v_2 are evaluated to the same state by A (from the right to the left). Conversely, for any left-congruence $\sim$ with a finite number of equivalence classes, we can define a look-ahead automaton $A(\sim)$ such that $\sim$ is equal to $\sim_A$. The states of A are the equivalence classes $[u]_\sim$ of input words u, the unique initial state is the equivalence class of the empty word, and the transition rules have the form $[a \cdot u]_\sim \xleftarrow{a} [u]_\sim$ for all $u \in \Sigma^*$ and $a \in \Sigma$. Final states are irrelevant for look-ahead automata.

Domains of partial functions τ need to be treated carefully for look-ahead minimization. Let the left residual of its domain be $dom(\tau)v^{-1} = \{u \mid u \cdot v \in dom(\tau)\}$. The domain induces a left-congruence on suffixes that we call compatibility with the domain: v_1 and v_2 are compatible with the

$dom(\tau)$ if $dom(\tau)v_1^{-1} = dom(\tau)v_2^{-1}$. A relation $\sim$ is said compatible with $dom(\tau)$ if it is a refinement of the compatibility relation, i.e., if $v_1 \sim v_2$ implies that v_1 and v_2 are compatible with $dom(\tau)$. Similarly, a look-ahead automaton A is compatible with a domain if $\sim_A$ is.

Let τ be a rational function. The difference between two output words is $\mathit{diff}(w \cdot w_1, w \cdot w_2) = (w_1, w_2)$ such that the common prefix of w_1 and w_2 is empty. The difference between two input words modulo τ is defined by $\mathit{diff}_\tau(v_1, v_2) = \{\mathit{diff}(\tau(u \cdot v_1), \tau(u \cdot v_2)) \mid u \cdot v_1,\ u \cdot v_2 \in dom(\tau)\}$. This allows to define a left-congruence $\sim_\tau$ that is compatible with $dom(\tau)$:

Definition 1. *$v_1 \sim_\tau v_2$ if and only if v_1 and v_2 are compatible with $dom(\tau)$ and $\#\mathit{diff}_\tau(v_1, v_2) < \infty$.*

Example 3. The equivalence τ_1 has a single class since $\mathit{diff}_\tau(v_1, v_2)$ is finite for every $v_1, v_2 \in \Sigma^*$. Function τ_2 has two equivalence classes, since $v_1 \sim_{\tau_2} v_2$ if either both end with b or none. Indeed, $A(\sim_{\tau_2})$ is the look-ahead automaton in Fig. 2. Let $u_n = a^n \cdot b^n$. Then we have $\tau_2(u_n \cdot v_1) = u_n \cdot v_1$ while $\tau_2(u_n \cdot v_2) = b^n \cdot \tau_2(v_2)$. So $\mathit{diff}_{\tau_2}(v_1, v_2)$ contains the pairs $(a^n \cdot b^n \cdot v_1, b^n \cdot \tau_1(v_2))$ for all n, which as an infinite cardinality. Subsequential function τ_3 has 3 equivalence classes: a single state look-ahead automaton for τ_3 would not be compatible with the domain as for instance $dom(\tau_3)a^{-1} \neq dom(\tau_3)b^{-1}$. The DT_ℓ with minimal look-ahead for τ_3 that is compatible with the domain has three states and is also the look-ahead given in Fig. 3. Note that neither the look-ahead nor the DT are size minimal. Fig. 1 shows that there is no need for a look-ahead and Fig. 2 shows that for this look-ahead, τ_3 only needs a one-state DT. □

We say that a left congruence $\sim$ partitions $\sim_\tau$ if $\sim$ is a subset of $\sim_\tau$. For every partial function τ and an equivalence relation $\sim$ on Σ^*, we can define a unique partial function σ with minimal domain such that $\tau = \sigma \circ [\![A(\sim)]\!]$. This function σ, that we denote by $\sigma(\tau, \sim)$, can be applied only to annotated words in the image of $[\![A(\sim)]\!]$; it ignores annotations and applies τ. The following result was originally stated for bimachines.

Theorem 4 (Reutenauer & Schützenberger [20]). *For any rational function τ the left-congruence $\sim_\tau$ has a finite number of equivalence classes. Furthermore, for any other left-congruence $\sim$ partitionning $\sim_\tau$ into finitely many classes, the function $\sigma(\tau, \sim)$ is subsequential.*

As a result, any look-ahead for τ compatible with the domain of τ has the form $A(\sim)$ for some left-congruence $\sim$ that partitions $\sim_\tau$. Also, $\sigma(\tau, \sim)$ being subsequential, Theorem 1 shows that it can be defined by a unique minimal DT, that we denote by $M_\tau(\sim)$. The unique 'right-minimal' DT_ℓ of τ then is the DT_ℓ $N_\tau(\sim)$ equal to $\langle A(\sim), M_\tau(\sim)\rangle$.

4 Building the Look-Ahead Automaton

Our next objective is to find a suitable look-ahead automaton for the unknown target function τ, of which we only know the domain and a finite sample of input-output pairs. One might want to identify the minimal look-ahead automaton $A(\sim_\tau)$, but we cannot hope to decide whether $v_1 \sim_\tau v_2$ for any two words v_1 and v_2, since we would have to check whether $\mathit{diff}_\tau(v_1, v_2)$ is finite or infinite. This is difficult to archieve from a finite set of examples. We will work around this problem based on the following lemma which provides a bound on the cardinality of $\mathit{diff}_\tau(v_1, v_2)$.

Lemma 1. *Let $\tau \subseteq \Sigma^* \times \Delta^*$ be a rational function, $\sim$ a left congruence that partitions $\sim_\tau$ and m be the number of states of $M_\tau(\sim)$. If $v_1 \sim v_2$ then $\#\mathit{diff}_\tau(v_1, v_2) \leq m$.*

Proof. With $N = N_\tau(\sim)$, $v_1 \sim v_2$ implies $v_1 \sim_\tau v_2$, so that $\mathit{dom}(\tau)v_1^{-1} = \mathit{dom}(\tau)v_2^{-1}$. We denote by $[\![N]\!]^u(v)$ (resp. $[\![N]\!]_v(u)$) the output of v (resp. u) when reading $u \cdot v$. Then for any prefix $u \in \mathit{dom}(\tau)v_1^{-1}$, $\tau(u \cdot v_i) = [\![N]\!]_{v_i}(u) \cdot [\![N]\!]^u(v_i)$. By construction, $[\![N]\!]_{v_1}(u) = [\![N]\!]_{v_2}(u)$, so $\mathit{diff}(\tau(u \cdot v_1), \tau(u \cdot v_2)) = \mathit{diff}([\![N]\!]^u(v_1), [\![N]\!]^u(v_2))$. As $[\![N]\!]^u(v_i)$ only depends on the state reached by u in $A(\sim)$, the number of values of $([\![N]\!]^u(v_1), [\![N]\!]^u(v_2))$ for varying u is bounded by the number of states of $M_\tau(\sim)$, i.e. $\#\mathit{diff}_\tau(v_1, v_2) \leq m$. □

Given a natural number m we define the binary relation C_τ^m on input words such that $(v_1, v_2) \in C_\tau^m$ if $\#\mathit{diff}_\tau(v_1, v_2) \leq m$. In this case, we say that v_1 is m-close to v_2. As we will show in Section 6 for any m, we can characterize relation C_τ^m by finite samples of input-output pairs for τ.

Let m_τ be the number of states in $M_\tau(\sim_\tau)$. By Lemma 1 we know that $\sim_\tau = C_\tau^{m_\tau}$. So if we knew this bound m_τ and if we could construct a look-ahead automaton from $C_\tau^{m_\tau}$, then we were done. We first consider how to construct a look-ahead automaton from C_τ^m under the assumption that $m \geq m_\tau$.

Our algorithm La given in Fig. 4 receives as inputs a binary relation R on input words and a natural number l, and returns as output a minimal deterministic finite automata, or raises an exception. Algorithm La is motivated by the Myhill-Nerode theorem for deterministic finite automata, in that for l greater than the index of $\sim_\tau$ and $R = C_\tau^{m_\tau} = \sim_\tau$ it constructs the minimal deterministic automaton $A(\sim_\tau)$. We will also apply it, however, in cases where R is even not an equivalence relation. In particular, relation $R = C_\tau^m$ may fail to be transitive for $m < m_\tau$. In this case we may have to force our algorithm to terminate. We do so by bounding the number of states that is to be generated by l.

fun LA(R, l) % where $R \subseteq \Sigma^* \times \Sigma^*$, $l \in \mathbb{N}$ **in**
1: **let** $Q = \text{SET}.new(\{\varepsilon\})$, $Agenda = \text{QUEUE}.new([\varepsilon])$
2: **while** Agenda.isnonempty() **do**
3: $v := Agenda.pop()$
4: **for** $a \in \Sigma$ such that $a \cdot v$ increases **do**
5: **if** $\nexists v' \in Q$ such that $(a \cdot v, v') \in R$
6: **then** $Agenda.push(a \cdot v)$, $Q.add(a \cdot v)$ **else skip**
7: **if** $Q.card() > l$ **then exception** "too many states" **else skip**
8: **let** $rul = \{v \xrightarrow{a} v' \mid v, v' \in Q, (a \cdot v, v') \in R\}$ **in**
9: **return** $\langle \Sigma, Q, \{\varepsilon\}, \emptyset, rul \rangle$

Fig. 4. Construction of look-ahead automata.

Algorithm LA proceeds as follows. It fixes some total ordering on words, such that shorter words preceed on longer words. It then behaves as if R were a left congruences while searching for the least word in each equivalence class of R. These least words will be the states of the output automaton that LA constructs. The algorithm raises an exception if the number of such states is greater then l. It adds the transitions $v \xrightarrow{a} v'$ for any two states v, v' that it discovered under the condition that $(a \cdot v, v') \in R$ (if several v' fits, we pick the first in our order). We observe the following: if R is a left congruence of finite index smaller than l then LA(R, l) terminates without exception and returns the minimal deterministic automata whose left-congruence is R. In particular for $m \geq m_\tau$ and $R = C_\tau^m$ (so that $R = \sim_\tau$), the algorithm returns $A(\sim_\tau)$. However, if $m < m_\tau$, the only property that we can assume about relation C_τ^m is that it is contained in $\sim_\tau$. The following lemma shows a little surprisingly that successful result are always appropriate nevertheless.

Lemma 2. *Let τ be a rational function and R a relation contained in $\sim_\tau$. Either* LA(R, l) *raises an exception or it returns a look-ahead valid for τ.*

If v_1 and v_2 are actually tested by the algorithm, then for v_1 and v_2 to be in the same state, we need $v_1 \, R \, v_2$, and thus $v_1 \sim_\tau v_2$. Then, given that $\sim_\tau$ is a left-congruence, we can prove by recursion that if two words v_1 and v_2 reach the same state of LA(R, l), then $v_1 \sim_\tau v_2$. Hence, R partitions $\sim_\tau$ so this LA(R, l) is a valid look-ahead for τ by Theorem 4.

5 The Learning Algorithm

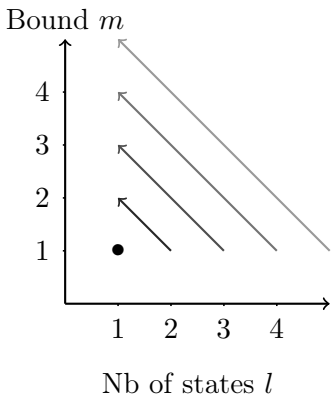
We next present an algorithm for learning a rational function τ from a domain automata D with $L(D) = dom(\tau)$ and a finite sample $S \subseteq \tau$ of input-output pairs. Furthermore, our learning algorithm assumes

```
fun LEARN_D(S)
 1:(m, l) := (1, 1)
 2:repeat
 3:    try let A = LA(C^m_{S,D}, l)  in
 4:        let S' = {([[A]](u), v) | (u, v) ∈ S)}  in
 5:        let D' be a DFA that represents words of D annotated by A  in
 6:        return  ⟨A, OSTIA_{D'}(S')⟩  and exit
 7:    catch "too many states"  then
 8:        (m, l) :=  successor of (m, l) in diagonal order
```

Fig. 5. Learning algorithm for rational functions with domain $L(D)$.

that there exists an oracle $C^m_{S,D}$ that can decide whether a pair of input words belongs to C^m_τ. Given such an oracle, the learning algorithm can simulate calls of algorithm $\text{LA}(C^m_\tau, l)$. How such an oracle can be obtained for sufficiently rich samples S is shown in the next section.

Two unknowns remain to be fixed: a bound m for which LA eventually finds a valid look-ahead and the number l of states of this valid look-ahead. The idea of learning algorithm LEARN_D in Fig. 5 is that to try out all pairs (m, l) in diagonally increasing order $(1,1) < (1,2) < (2,1) < (1,3) < \ldots$. For any such pair (m, l) it then calls $\text{LA}(C^m_{S,D}, l)$, until this algorithm succeeds to return an automaton. By Lemma 1, any such automaton is a valid look-ahead for τ. By Proposition 1, this procedure must be successful no later than for (m_τ, l_τ). Finally, the algorithm decorates the examples of S by applying the newly obtained look-ahead automaton, and learns the corresponding subsequential transducer by using the OSTIA algorithm.



It should be noticed that the target of this algorithm is *not* the DT_ℓ for τ with minimal look-ahead $A(\sim_\tau)$. The look-ahead obtained is simply the first automaton obtained in the diagonal order such that $\text{LA}(C^m_{S,D}, l)$ terminates successfully. We call the DT_ℓ obtained in this way the 'diagonal' DT_ℓ of τ. Note that the diagonal DT_ℓ of τ may be smaller that the corresponding right-minimal DT_ℓ with minimal look-ahead. In any case, it may not be much bigger as stated by the following lemma.

Lemma 3. *Let τ be a partial rational function with right-minimal* DT_ℓ *$\langle A(\sim_\tau), M(\sim_\tau)\rangle$, let m be the number of states of $M(\sim_\tau)$, and $\sim$ be a finite left-congruence that partitions $\sim_\tau$ of index n. The number of states*

fun $C^m_{S,D}(v_1, v_2)$
1: **if** $L(D)v_1^{-1} \neq L(D)v_2^{-1}$ **then return** false
2: **else if** $\#\{\mathit{diff}(w_1, w_2) \mid (u \cdot v_1, w_1), (u \cdot v_2, w_2) \in S\} \leq m$
3: **then return** true **else return** false

Fig. 6. Implemention of the oracle.

of the look-ahead of $\langle A(\sim), M(\sim)\rangle$ *has then at most* mn *states and is of global size* $O(mn^2)$.

Indeed, to obtain the Dt $M(\sim)$, one can pick $M(\sim_\tau)$ and change its transition to take into account states of $A(\sim)$ instead of those of $A(\sim_\tau)$. This transducer has m states and at worse mn transitions. However, it does not have the right domain (words annotated by states of $M(\sim)$): this requires a product with the Dfa of the correct domain, which has m states. The actual Dt $M(\sim)$ being minimal, it has at most this size.

6 Characteristic Samples

It remains to show that there exists an oracle $C^m_{S,D}$ that decides membership to C^m_τ for all suffuciently rich finite samples $S \subseteq \tau$, and that the size of such samples is polynomial in the size of the target diagonal transducer with look-ahead. We use the function defined in Fig. 6 which when applied to a pair of words (v_1, v_2) verifies that they have equal residuals for the domain, and computes their difference on S instead of τ. In order to see that the former can be done in polynomial time, we only need to check that there are deterministic automata recognizing $L(D)v_1^{-1}$ and $L(D)v_2^{-1}$ of polynomial size.

The next question is what examples a sample S needs to contain so that this test becomes truly equivalent to m-closeness. In order to be usable in La, note that $C^m_{S,D}(v_1, v_2)$ has to behave like $C^m_\tau(v_1, v_2)$ only on pairs of suffixes considered there. We define $s_{m,l}(\tau)$ as the words creating new states in $\textsc{La}(C^m_\tau(v_1, v_2), l)$ (there is at most l of them). As the algorithm La also observes successors of $s_{m,l}$, we need to define the set $k_{m,l}(\tau) = s_{m,l}(\tau) \cup \{a \cdot v \mid v \in s_{m,l}(\tau), a \in \Sigma\}$. We call a sample S ℓ-characteristic for τ with respect to m and l if every element of $k_{m,l}$ appears as the suffix of an input word in S and if S allows the correct evaluation of C^m_τ on those elements, i.e.:

- for every $v \in s_{m,l}(\tau)$, $\exists u \in \Sigma^*$, $w \in \Delta^*$ such that $(u \cdot v, w) \in S$,
- for $v_1 \in s_{m,l}$, $v_2 \in k_{m,l}$ with $(v_1, v_2) \notin C^m_\tau$ and $\mathit{dom}(\tau)v_1^{-1} = \mathit{dom}(\tau)v_2^{-1}$, $\#\{\mathit{diff}(w_1, w_2) \mid (u \cdot v_1, w_1), (u \cdot v_2, w_2) \in S\} > m$.

Lemma 4. *For a partial rational function τ, a* DFA *D recognizing $dom(\tau)$, and two positive integers m and l, let $v_1 \in s_{m,l}(\tau)$, $v_2 \in k_{m,l}(\tau)$, if S is a ℓ-characteristic sample for τ with respect to m and l, then the test $C^m_{S,D}(v_1, v_2)$ returns true if and only if $(v_1, v_2) \in C^m_\tau$.*

One thing that has to be checked is that there exists an ℓ-characteristic samples of reasonable size for any m, l. This is obvious for the cardinality. In order to show that the length of words can also be guaranteed to be short, one can use the following method: for any non-equivalent suffixes v_1 and v_2 of different domain, one pick any set of words that allow to obtain enough element in $\mathit{diff}_\tau(v_1, v_2)$, and reduce them to a reasonable length (of size $\mathcal{O}(|N|^2)$) where N is any transducer recognizing τ) using pumping arguments.

Lemma 5. *For a partial rational function τ, a* DFA *D recognizing $dom(\tau)$, two integers m and l, and a sample S ℓ-characteristic for τ with respect to m and l:* $\textsc{La}(C^m_{S,D}, l) = \textsc{La}(C^m_\tau, l)$.

In particular, if $\textsc{La}(C^m_{S,D}, l)$ raises an exception if and only if $\textsc{La}(C^m_\tau, l)$ does. Note that we need a sample that is (globally) ℓ-characteristic, for all pairs $\langle m, l\rangle$ encountered during the run, i.e. all the $\langle m, l\rangle$ smaller than the values for the diagonal $\textsc{Dt}_\ell$. Once the look-ahead is learned, we can apply the OSTIA algorithm, which requires a sample labelled by the look-ahead, and not on $\Sigma^* \times \Delta^*$. We deal with this by labelling all the input words in S when the look-ahead $A(\sim)$ is found. For S to be enough to learn the subsequential transducer $M_\tau(\sim)$, its labelling must contain a characteristic sample for the OSTIA algorithm as defined in [19]. In other words, S is called DT-characteristic for τ and $\sim$ if it contains a characteristic sample for $M_\tau(\sim)$ in OSTIA, minus the labelling by $\sim$.

Finally, for the algorithm $\textsc{Learn}_D$ to produce the diagonal $\textsc{Dt}_\ell$, the input sample needs to be ℓ-characteristic. Also, it has to be DT-characteristic for τ and the look-ahead $\sim$ it found. A sample S is then said to be characteristic for a rational function τ if it fulfils all those conditions. This gives the following result:

Theorem 5. *For any* DFA *D the learning algorithm $\textsc{Learn}_D$ identifies rational functions with domain $L(D)$ represented by their diagonal $\textsc{Dt}_\ell$ from polynomial time and data.*

That is: for any $\textsc{Dt}_\ell$ N in diagonal form defining a rational function τ whose domain $L(D)$, there exists a finite sample $S \subseteq \tau$ called characteristic for τ whose size is polynomial in the size of N, such that from any sample $S' \subseteq \tau$ that contains S, $\textsc{Learn}_D(S')$ computes a $\textsc{Dt}_\ell$ in diagonal-minimal normal form defining τ in polynomial time in the size of S'.

Conclusion and Future Work. Our learning algorithm for DT_ℓs answers the long standing open learning question for rational functions, for the case where diagonal-minimal DT_ℓ normal forms are used for their representation. Whether other representations lead to negative results is left open. More importantly, we would like to extend our result to deterministic top-down tree transducers with look-ahead, which have the same expressiveness than functional top-down tree transducers [9].

References

1. J. Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.
2. J. Berstel, L. Boasson, O. Carton, and I. Fagnot. Minimization of automata. *Computing Research Repository*, abs/1010.5318, 2010.
3. J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67, 2007.
4. C. Choffrut. A generalization of Ginsburg and Rose's characterisation of g-s-m mappings. In *ICALP*, volume 71 of LNCS, 88–103. 1979
5. C. Choffrut. Minimizing subsequential transducers: a survey. *TCS*, 292(1):131–143, 2003.
6. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, M. Tommasi. Tree automata techniques and applications. 2007.
7. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–137, 1997.
8. C. C. Elgot and G. Mezei. On relations defined by generalized finite automata. *IBM Journ. of Research and Development*, 9:88–101, 1965.
9. J. Engelfriet. Top-down tree transducers with regular look-ahead. *Math. Syst. Theory*, 10:198–231, 1977.
10. J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *JCSS*, 75(5):271–286, 2009.
11. S. Friese, H. Seidl, and S. Maneth. Minimization of deterministic bottom-up tree transducers. *DLT*, vol. 6224 of *LNCS*, 185–196. 2010.
12. E. M. Gold. Complexity of automaton identification from given data. *Infor. and Cont.*, 37:302–320, 1978.
13. J. Högberg, A. Maletti, and J. May. Backward and forward bisimulation minimization of tree automata. *TCS*, 410(37):3539–3552, 2009.
14. J. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. *TMC*, 189–196, 1971.
15. A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for Top-Down XML transf. *PODS*, 285–296, 2010.
16. J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. *Patt. Recog. and Image Anal.*, 49–61, 1992.
17. J. Oncina and P. García. Inference of recognizable tree sets. Tech. report, Univ. de Alicante, 1993.
18. J. Oncina, P. Garcia, E. Vidal. Learning subsequential transducers for pattern recognition and interpretation tasks. *Patt Anal & Mach Intell*, 15:448–458, 1993.
19. J. Oncina and M. A. Varo. Using domain information during the learning of a subsequential transducer. *ICGI*, vol. 1147 in LNAI, 313–325, 1996.
20. C. Reutenauer and M. P. Schützenberger. Minimalization of rational word functions. *SIAM Journal on Computing*, 20:669–685, 1991.