From Two-Way Transducers to Regular Function Expressions

Nicolas Baudru, Pierre-Alain Reynier

From Two-Way Transducers to Regular Function Expressions

Nicolas Baudru[1] and Pierre-Alain Reynier[1]

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{nicolas.baudru,pierre-alain.reynier}@lis-lab.fr

Abstract. Transducers constitute a fundamental extension of automata. The class of regular word functions has recently emerged as an important class of word-to-word functions, characterized by means of (functional, or unambiguous, or deterministic) two-way transducers, copyless streaming string transducers, and MSO-definable graph transformations. A fundamental result in language theory is Kleene's Theorem, relating finite state automata and regular expressions. In [3], the authors introduced a set of regular function expressions and proved a similar result for regular word functions, by showing the equivalence with copyless streaming string transducers. In this paper, we propose a direct, simplified and effective translation from unambiguous two-way transducers to regular function expressions extending the Brzozowski and McCluskey algorithm. In addition, we identify a subset of regular function expressions characterizing the (strict) subclass of functional sweeping transducers.

1 Introduction

The theory of regular languages has been extended in numerous directions, including finite and infinite trees. Another natural extension is moving from languages to transductions. One of the strengths of the class of regular languages is their equivalent presentation by means of automata, logic, algebra and regular expressions. Regular expressions are of particular interest for specification purposes in a declarative manner. We are interested in this paper in regular expressions for specifying word-to-word functions.

While finite state automata are very robust under modifications in the model, the situation is different for transducers. It is well known that non-determinism and two-wayness increase the expressive power, even when one only considers functional transductions. The class of functions realized by non-deterministic two-way transducers, so-called *regular functions*, has attracted recently a strong interest [1–4, 13, 6, 14, 8]. It is very expressive and allows to express natural transformations that are not definable by one-way transducers (*e.g.* duplicate the input word, or produce its mirror image). This class also enjoys a logical characterization using Monadic Second-Order graph transductions interpreted on strings [10], and can also be defined using the model of copyless streaming string transducers (SST) [1].

A natural line of research concerns the identification of adequate regular expressions to specify different classes of word functions, as investigated in [9, 15].

It is well known that rational relations (realized by non-deterministic one-way transducers) can be described using standard regular expressions on pairs of input and output words, as a special case of Schützenberger theorem for weighted automata [17]. Imposing on these regular expressions the restriction of being unambiguous on their input part yields a presentation of rational functions [5]. More recently, a set of regular combinators has been introduced in order to characterize the class of regular functions [3]. The equivalence goes through the model of copyless SST, and relies on a very involved proof.

In this paper, we propose a new construction showing that any regular function can be expressed using the regular functions expressions of [3]. Unlike [3], we take as input unambiguous two-way finite state transducers. Intuitively, our construction lifts the standard state-elimination algorithm proposed by Brzozowski and McCluskey in [7] for one-way finite state automata to unambiguous two-way finite state transducers. The difficulty lies in two aspects: going from a one-way to a two-way model, and going from automata to transducers. In order to address these issues, we use the construction of the Shepherdson automaton [18] and the notion of traversals of two-way automata, that allow to describe the computation flow of a two-way automaton over some input word. In addition, we label the edges of these flow graphs with regular function expressions.

More precisely, the state-elimination algorithm requires to be able to compute the union, concatenation and Kleene star of transitions of the automaton. We thus have to be able to perform these operations on flows. Unlike union and concatenation, the operation of Kleene iteration may yield complex behaviors that are difficult to describe by means of regular function expressions. A key contribution of our work is a deep investigation of these flows, and the identification of an important characteristic: the number of crossing edges. Using this parameter, we first identify a subclass of flows for which the representation of the Kleene iteration by means of regular function expressions is rather simple, and then present a construction allowing to reduce the Kleene iteration of arbitrary flows to that of the subclass. This results in a very simple proof, which we believe will be useful for further extensions. A side-result of our work is the exhibition of a set of operators characterizing the class of sweeping transducers (this result could also be deduced from [15]). Sweeping transducers induce a third class of functions in-between rational and regular functions.

The very recent work [11] adopts an approach similar to ours in order to lift the results of [3] to infinite words. However, to deal with Kleene iteration, they resort to an unambiguous version of Simon's factorization forest theorem.

It is worth mentioning that our results are presented for word transducers but they could be easily adapted to the setting of transducers producing output in some monoid, as in [3].

We introduce the model of transducers and the regular function expressions in Section 2, and our labelled flow graphs in Section 3. In Section 4, we present the algorithm, and the main results concerning the representation of the Kleene star of flow graphs. The case of sweeping transducers is dealt with in Section 4.5.

The second author is funded by the DeLTA project (ANR-16-CE40-0007).

2 Definitions

2.1 Words, Languages and Transducers

Given a finite alphabet A, we denote by A^* the set of finite words over A, and by ϵ the empty word. The length of a word $u \in A^*$ is its number of symbols, denoted by $|u|$. For all $i \in \{1, \dots, |u|\}$, we denote by $u[i]$ the i-th letter of u. Given $n > 0$, we denote by $[n]$ the set $\{0, 1, \dots, n-1\}$. A *language* over A is a set $L \subseteq A^*$. Given two languages L, L' over A, the concatenation of L and L', denoted LL', is defined as $\{uv \mid u \in L, v \in L'\}$. We say that L and L' are *unambiguously concatenable* whenever for every word $w \in LL'$, there exist unique words $u \in L$ and $v \in L'$ such that $w = uv$. Given a language L over A, the Kleene star of L, denoted by L^*, is defined as $\{u_1 \cdots u_n \mid n \geqslant 0 \text{ and } \forall i, u_i \in L\}$. The Kleene plus of L, denoted by L^+, is defined as LL^*. When $\epsilon \notin L$, we say that L is *unambiguously iterable* if for every word $w \in L^*$, there exist unique words $u_1, \dots, u_n \in L$ such that $w = u_1 \cdots u_n$.

A *monoid* is a set M equipped with an associative internal law and a neutral element. Given an alphabet A and a monoid M, a *transduction* from A to M is a relation $R \subseteq A^* \times M$. A transduction R is *functional* if it is a function. The transducers we will introduce will define transductions. We will say that two transducers T, T' are *equivalent* whenever they define the same transduction.

When dealing with two-way machines, we assume the alphabet A to be extended into $\overline{A}$ by adding two special symbols $\vdash, \dashv$, and we consider input words with left and right markers. The automaton then reads the input word $\vdash u \dashv$, and we set $u[0] = \vdash$ and $u[|u| + 1] = \dashv$. We also define $\mathbb{D} = \{\leftarrow, \rightarrow\}$.

Automata. A *two-way finite state automaton* (2NFA) over a finite alphabet A is a tuple $\mathcal{A} = (Q, q_0, F, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and $\delta \subseteq Q \times \overline{A} \times Q \times \mathbb{D}$ is the transition relation. We describe the behaviour of $\mathcal{A}$ on some input word $\vdash u \dashv$. Informally, a 2NFA has a reading head pointing between symbols (and possibly on the left of $\vdash$ and on the right of $\dashv$). A *configuration* of $\mathcal{A}$ is a triple $(q, i, d) \in Q \times \mathbb{N} \times \mathbb{D}$, where $0 \leqslant i \leqslant |u| + 2$ is the position of the reading head on the input tape. The direction d indicates whether the next input letter read is on the left or on the right of the reading head. The configurations (q, i, d) and (q', i', d') are *consecutive* if we have $(q, u[i + m_d], q', d') \in \delta$, where $m_\rightarrow = 0$ and $m_\leftarrow = -1$, and $i' = i + 1$ if $d = d' = \rightarrow$, $i' = i - 1$ if $d = d' = \leftarrow$, and $i' = i$ otherwise. A *run* is a finite sequence of consecutive configurations. It is *accepting* if the first configuration is $(q_0, 0, \rightarrow)$, and the last configuration is $(q, |u| + 2, \rightarrow)$ with $q \in F$. Note that this latter configuration does not allow additional transitions. An exemple of run is depicted in Figure 2. The *language* of $\mathcal{A}$ is the set of words $u \in A^*$ such that there exists an accepting run of $\mathcal{A}$ on $\vdash u \dashv$.

A two-way finite state automaton is:

- *deterministic* if we may write δ as a function from $Q \times \overline{A}$ to $Q \times \{\leftarrow, \rightarrow\}$.
- *unambiguous* if for any $u \in A^*$, there is at most one accepting run on $\vdash u \dashv$.
- *one-way* it it does not have transitions of the form $(q, a, q', \leftarrow)$.
- *sweeping* if the head can change direction only at the extremities of the input.

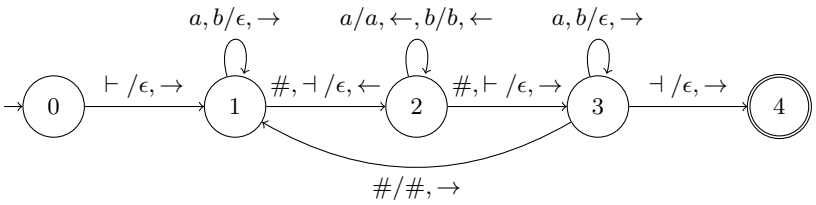

Fig. 1. A 2NFT realizing the function f of Example 2.

Transducers. Given two finite alphabets A and B, *two-way finite state transducers* (2NFT) from A to B extend 2NFA over A with a one-way left-to-right output tape containing elements of B^*. They are defined as 2NFA except that the transition relation δ is extended with outputs: $\delta \subseteq Q \times \overline{A} \times B^* \times Q \times \{\leftarrow, \rightarrow\}$. When a transition (q, a, w, q', d) is fired, the word w is appended to the right of the output tape. The transduction defined by a 2NFT $\mathcal{T}$ is the relation $R(\mathcal{T})$ defined as the set of pairs $(u, v) \in A^* \times B^*$ such that v is the output of an accepting run on the word $\vdash u \dashv$.

We say that a 2NFT $\mathcal{T}$ is *functional* if the relation $R(\mathcal{T})$ is a function. We say that a 2NFT $\mathcal{T}$ is *deterministic* (resp. *unambiguous*, *one-way*, *sweeping*) if its underlying 2NFA is. Observe that if $\mathcal{T}$ is deterministic or unambiguous, there is at most one accepting run for each input word, and thus $R(\mathcal{T})$ is a function.

Theorem 1. *[12] The classes of functional, deterministic and unambiguous* 2NFT *are equivalent, and strictly more expressive than that of functional sweeping transducers, which in turn is strictly more expressive than the class of functional one-way transducers.*

We call *rational functions* (resp. *sweeping functions*, *regular functions*) the ones that are definable by one-way transducers (resp. sweeping transducers, two-way transducers).

Example 2. Given a word u, we denote by $\mathsf{mirror}(u)$ its mirror image. Given $A = \{a, b, \#\}$, we consider the function mirror^* mapping an input word u, whose decomposition according to $\#$'s is $u = u_1 \# u_2 \# \dots \# u_n$, to the word $v = v_1 \# v_2 \# \dots \# v_n$, with $v_i = \mathsf{mirror}(u_i)$ for all i. This function is realized by the deterministic 2NFT depicted in Figure 1. An execution of this transducer on the input word $u = ab\#bba$ is depicted in Figure 2.

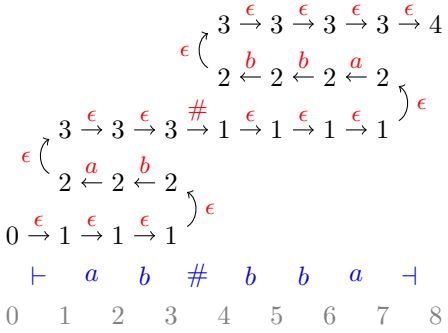

Fig. 2. A run of the 2NFT of Example 2.

2.2 Shepherdson automaton construction

A *crossing sequence* is a sequence of states encountered by a run at a given position in the input word. For instance, if we consider the run depicted in

Figure 2, the crossing sequence at position 0 (resp. position 8) is the tuple (0) (resp. (4)), while all the other crossing sequences are equal to the triple $(1, 2, 3)$. It is well-known that the crossing sequences of accepting runs of unambiguous 2NFA have size bounded by $2|Q|$, as a state cannot appear twice in a crossing sequence at two indices of same parity (otherwise a loop is entered).

Unlike one-way automata, partial runs of a two-way automaton on some factor of an input word do not all go through the input word from left to right. Flows are graphs allowing to describe these partial runs between two crossing sequences. More precisely, a *flow of size* (n, m), $n, m > 0$, is a particular directed graph F with set of vertices $[n] \times \{L\} \uplus [m] \times \{R\}$. Vertices in $[n] \times \{L\}$ (resp. $[m] \times \{R\}$) are called left (resp. right) vertices, satisfying additional conditions that intuitively imply that it connects two crossing sequences (see [4]). An edge is a *crossing edge* if it is between two vertices on different sides, it is a *return edge* otherwise. From now on, we fix an integer M and only consider flows of size (n, m) with $n, m \leq M$. We denote by $\mathbb{F}$ this finite set of flows. It is well-known that $\mathbb{F}$ can be equipped with an operation of composition (denoted by $F \circ F'$), yielding a monoid. This operation can be understood as the identification of paths in the concatenation of the two graphs. We state a simple property of flows:

Lemma 3. *Let F, F' be two flows with k and l crossing edges respectively. Then $F \circ F'$ has at most* $\min(k, l)$ *crossing edges.*

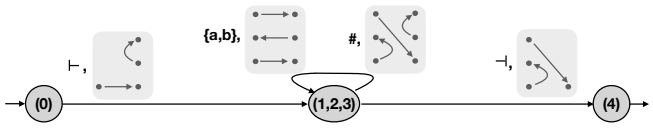

Fig. 3. The Shepherdson automaton of the underlying 2NFA of Example 2. Flows associated with transitions are indicated for clarity.

The *Shepherdson automaton* of an unambiguous two-way automaton $\mathcal{A}$ is an equivalent one-way automaton obtained as follows: states are crossing sequences, and there is a transition between two states on some letter a if there exists an adequate flow on this letter. This automaton may be of exponential size. W.l.o.g., we assume that the Shepherdson automaton is trimmed, meaning that every state appears in some accepting run. Note that as $\mathcal{A}$ is unambiguous, then so is its Shepherdson automaton. More precisely, for every transition, there is a single flow that can be associated with it. An example is depicted in Figure 3.

2.3 Function Expressions

We recall the function expressions introduced in [3] to specify word-to-word functions from A^* to B^*. First observe that in [3], these operators are mappings from elements of A^* to elements of some monoid M. We have chosen to present our results in the setting of transducers but they could easily be extended to an arbitrary monoid as output. Observe also that we do not follow the terminology used in [3], and rather adopt a terminology coming from formal power series (see

for instance [9, 15]). It is however trivial to verify that the operators we introduce correspond to the ones of [3].

Constant functions. Given a language $L \subseteq A^*$ and $v \in B^*$, the *constant function* L/v is such that $\mathrm{dom}(L/v) = L$ and for all $u \in L$, $L/v(u) = v$.

Sum. Given two functions f, g such that $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$, the *sum* $f \oplus g$ is such that $\mathrm{dom}(f \oplus g) = \mathrm{dom}(f) \uplus \mathrm{dom}(g)$ and for all $u \in \mathrm{dom}(f \oplus g)$, $(f \oplus g)(u) = f(u)$ if $u \in \mathrm{dom}(f)$; $(f \oplus g)(u) = g(u)$ if $u \in \mathrm{dom}(g)$.

Hadamard product. Given two functions f, g, the *Hadamard product* $f \otimes g$ is such that $\mathrm{dom}(f \otimes g) = \mathrm{dom}(f) \cap \mathrm{dom}(g)$ and for all $u \in \mathrm{dom}(f \otimes g)$, $(f \otimes g)(u) = f(u)g(u)$.

Cauchy products. Given two functions f, g such that $\mathrm{dom}(f)$ and $\mathrm{dom}(g)$ are unambiguously concatenable, the *Cauchy product* $f \bullet g$ and the *left Cauchy product* $f \overset{\leftarrow}{\bullet} g$ are such that $\mathrm{dom}(f \bullet g) = \mathrm{dom}(f \overset{\leftarrow}{\bullet} g) = \mathrm{dom}(f)\mathrm{dom}(g)$ and:

$$\forall u = u_1u_2 \text{ with } u_1 \in \mathrm{dom}(f), u_2 \in \mathrm{dom}(g), \quad \begin{cases} (f \bullet g)(u) &= f(u_1)g(u_2) \\ (f \overset{\leftarrow}{\bullet} g)(u) &= g(u_2)f(u_1) \end{cases}$$

Kleene stars. Given f such that $\mathrm{dom}(f)$ is unambiguously iterable, the *Kleene star* f^* and the *left Kleene star* $f^{\overset{\leftarrow}{*}}$ are s. t. $\mathrm{dom}(f^*) = \mathrm{dom}(f^{\overset{\leftarrow}{*}}) = L^*$ and:

$$\forall u = u_1u_2 \dots u_n \text{ with } \forall i, u_i \in \mathrm{dom}(f), \quad \begin{cases} f^*(u) &= f(u_1)f(u_2) \dots f(u_n) \\ f^{\overset{\leftarrow}{*}}(u) &= f(u_n)f(u_{n-1}) \dots f(u_1) \end{cases}$$

Chained stars. Given a function f and a language L such that $L^2 \subseteq \mathrm{dom}(f)$ and L is unambiguously iterable, the *chained star* $\langle f, L\rangle^{\circledast}$, and the *left chained star* $\langle f, L\rangle^{\overset{\leftarrow}{\circledast}}$, are such that $\mathrm{dom}(\langle f, L\rangle^{\circledast}) = \mathrm{dom}(\langle f, L\rangle^{\overset{\leftarrow}{\circledast}}) = L^{\geqslant 2}$, and:

$$\forall u = u_1u_2 \dots u_n \text{ with } \forall i, u_i \in L, \quad \begin{cases} \langle f, L\rangle^{\circledast}(u) = f(u_1u_2)f(u_2u_3) \dots f(u_{n-1}u_n) \\ \langle f, L\rangle^{\overset{\leftarrow}{\circledast}}(u) = f(u_{n-1}u_n) \dots f(u_2u_3)f(u_1u_2) \end{cases}$$

We consider the following grammars: (L is a regular language over A, $v \in B^*$)

$$\begin{aligned} \mathit{Reg} \ni f, g &::= L/v \mid f \oplus g \mid f \otimes g \mid f \bullet g \mid f \overset{\leftarrow}{\bullet} g \mid \langle f, L\rangle^{\circledast} \mid \langle f, L\rangle^{\overset{\leftarrow}{\circledast}} \\ \mathit{Rat} \ni f, g &::= L/v \mid f \oplus g \mid f \bullet g \mid f^* \end{aligned}$$

A function expression obtained from some grammar G is called a G-expression. *Reg*-expressions are called *regular function expressions*.

Example 4. We give examples to illustrate these operators:

- the identity function mapping on A^* can be defined as $\mathsf{id}_{A^*} = \left(\bigoplus_{a \in A} \{a\}/a\right)^*$,
- the mirror image on A^* can be defined as $\mathsf{mirror}_{A^*} = \left(\bigoplus_{a \in A} \{a\}/a\right)^{\overset{\leftarrow}{*}}$,
- the function last mapping word $ua \in A^*$ to $a^{|ua|}$, for every $a \in A$, can be defined as $\mathsf{last} = \bigoplus_{a \in A} \left((A/a)^* \bullet \{a\}/a\right)$.
- the function mirror^* of Example 2 can be defined as $(\mathsf{mirror}_{\{a,b\}^*} \bullet \{\#\}/\#)^* \bullet \mathsf{mirror}_{\{a,b\}^*}$.

Theorem 5. *The following equivalences hold:*

- *Rational functions are equivalent to Rat-expressions [5].*
- *Regular functions are equivalent to Reg-expressions [3].*

3 Function Expression Flow Automata

Intuitively, we refine the Shepherdson automaton construction by labelling transitions with flows, in which each edge is labelled by a function expression. From now on, we fix an input alphabet A and an output alphabet B.

Definition 6. *A* word flow W *is a flow of* $\mathbb{F}$ *whose edges are labelled by words of* B^**. The set of word flows is denoted by* $\mathbb{W}$*.*

Composition of flows can be lifted to word flows, by labelling new edges with the concatenation of words labelling edges along the path. We denote it by $W \circ W'$.

Definition 7. *A* function expression flow (FEF) E *over domain* L *is a flow of* $\mathbb{F}$ *whose edges are labelled by function expressions from* A^* *to* B^* *of domain* L*.*

We denote by $\mathbb{E}$ the set of all function expression flows, by $\mathrm{dom}(E)$ the domain of a FEF E and by $\mathrm{flow}(E)$ its underlying flow ($\mathrm{flow}(E) \in \mathbb{F}$).

A function expression flow $E \in \mathbb{E}$ of size (n, m) defines a functional transduction from A^* to $\mathbb{W}$: for all $u \in \mathrm{dom}(E)$, $E(u)$ is the word flow W of size (n, m) such that $(x, f(u), y)$ is an edge of W iff (x, f, y) is an edge of E.

We say that two FEFs $E, E' \in \mathbb{E}$ *are disjoint* if $\mathrm{dom}(E) \cap \mathrm{dom}(E') = \emptyset$.

Definition 8. *A* label *of size* (n, m)*, with* $n, m > 0$*, is a non-empty set of FEFs of size* (n, m) *that are pairwise disjoint.*

We denote by $\mathbb{L}$ the set of labels and define the domain of a label as the union of the domains of the FEFs it contains: $\mathrm{dom}(\mathcal{L}) = \biguplus_{E \in \mathcal{L}} \mathrm{dom}(E)$. A label also defines a functional transduction from A^* to $\mathbb{W}$: for any $u \in \mathrm{dom}(\mathcal{L})$, $\mathcal{L}(u) = E(u)$ for the unique FEF E of $\mathcal{L}$ such that $u \in \mathrm{dom}(E)$.

Definition 9. *A* function expression flow automaton *(FEFA for short) is a tuple* $\mathcal{A} = (Q, q_0, q_f, \delta)$ *where* Q *is a finite set of states,* q_0 *(resp.* q_f*) is the initial (resp. final) state, and* $\delta \subseteq Q \times \mathbb{L} \times Q$ *is the finite transition relation. We require that there is no incoming (resp. outgoing) transition to the initial state (resp. from the final state), and that there exists a mapping* $\mathsf{size} : Q \to \mathbb{N}$ *such that* $\mathsf{size}(q_0) = \mathsf{size}(q_f) = 1$ *and, for every* $(q, \mathcal{L}, q') \in \delta$*,* $\mathcal{L}$ *is of size* $(\mathsf{size}(q), \mathsf{size}(q'))$*.*

Given a word $u \in A^*$, an *execution* on u of a FEFA $\mathcal{A} = (Q, q_0, q_f, \delta)$ is a decomposition $u_1 \cdots u_k$ of u and a sequence $(q_i, \mathcal{L}_i, q_{i+1})_{1 \leqslant i \leqslant k}$ of consecutive transitions such that $u_i \in \mathrm{dom}(\mathcal{L}_i)$ for all i. The execution is *accepting* if $q_1 = q_0$ and $q_{k+1} = q_f$. We say that a FEFA is *unambiguous* if for every word u, there is at most one accepting execution on u. When this holds, following above notations, the word flow associated with such an execution is $\mathrm{out}(u) = \mathcal{L}_1(u_1) \circ \mathcal{L}_2(u_2) \circ \ldots \circ \mathcal{L}_k(u_k)$. The properties of the mapping size ensure that this composition is well defined.

The semantics of a FEFA $\mathcal{A}$ is a functional transduction from A^* to B^*. Indeed, given a word u with some accepting execution which produces the word flow $\mathrm{out}(u)$, the properties of the mapping size in the definition of FEFA ensure

that $\mathrm{out}(u)$ is of size $(1,1)$, hence reduced to a single left-to-right crossing edge, labelled by some word $v \in B^*$. We thus consider that $\mathcal{A}$ maps u to v.

The following lemma follows from Shepherdson automaton construction:

Lemma 10. *Given an unambiguous* 2NFT*, one can build an equivalent unambiguous FEFA.*

4 A Brzozowski and McCluskey-like Algorithm

4.1 Presentation of the algorithm

The standard Brzozowski and McCluskey (BMC for short) algorithm takes as input a one-way finite-state automaton $\mathcal{A}$, with distinguished initial and final states, and proceeds as follows: it removes all the states of the automaton one by one, except the initial and final states. Each time a state is removed, the remaining transitions are modified in order to obtain an equivalent automaton. The transitions are now labelled by regular expressions. More precisely, consider the removal of some state q_2. Then, for all states q_1 and q_3 of $\mathcal{A}$ we have to replace the transitions $q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_2 \xrightarrow{e_3} q_3$ and $q_1 \xrightarrow{e_4} q_3$ by a unique transition $q_1 \xrightarrow{e} q_3$. In order to obtain an equivalent automaton, one defines $e = e_1e_3 + e_1e_2^+e_3 + e_4$. At the end of the algorithm, one ends up with a single transition between the initial and the final state. The regular expression e labelling this transition describes exactly the whole behaviour of the automaton.

In order to adapt this algorithm to unambiguous FEFA, one needs to define the operations of sum, concatenation and Kleene plus (*i.e.* Kleene star with a positive number of iterations) on labels. It is worth observing that unambiguity of the FEFA ensures that sum (resp. concatenation, Kleene plus) involves labels with disjoint domains (resp. unambiguously concatenable domains, unambiguously iterable domains). As we will see in the subsequent sections, while sum and concatenation are easy to deal with, Kleene plus is more involved. We first present in Subsection 4.3 how to compute the Kleene plus for a restricted class of labels, called simple labels, and as a corollary we obtain a particular instance of BMC algorithm valid for FEFA for which BMC always eliminates self-loops with simple labels. Last, we show in Subsection 4.4 how the Kleene plus of arbitrary labels can be obtained using this particular instance of BMC algorithm for FEFA.

Putting everything together, we obtain:

Theorem 11. *Given an unambiguous* 2NFT*, our Brzozowski and McCluskey-like algorithm returns an equivalent Reg-expression.*

4.2 Sum, concatenation and Kleene plus of labels

Sum. Let $\mathcal{L}_1, \mathcal{L}_2$ be two labels of $\mathbb{L}$ having the same size and such that $\mathrm{dom}(\mathcal{L}_1)$ and $\mathrm{dom}(\mathcal{L}_2)$ are disjoint. It is possible to define a new label, called the *sum of* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ and denoted by $\mathcal{L}_1 \oplus \mathcal{L}_2$, whose domain is $\mathrm{dom}(\mathcal{L}_1) \uplus \mathrm{dom}(\mathcal{L}_2)$, and such that for all $u \in \mathrm{dom}(\mathcal{L}_1) \uplus \mathrm{dom}(\mathcal{L}_2)$, $\mathcal{L}_1 \oplus \mathcal{L}_2(u) = \mathcal{L}_1(u)$ if $u \in \mathrm{dom}(\mathcal{L}_1)$,

and $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathcal{L}_2(u)$ if $u \in \mathrm{dom}(\mathcal{L}_2)$. This label is simply obtained by taking the union of FEFs of $\mathcal{L}_1$ and $\mathcal{L}_2$.

Concatenation. Let $\mathcal{L}_1, \mathcal{L}_2$ be two labels of size (n, m) and (m, k), and such that $\mathrm{dom}(\mathcal{L}_1)$ and $\mathrm{dom}(\mathcal{L}_2)$ are unambiguously concatenable. It is possible to define a new label, called the *concatenation of* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ and denoted by $\mathcal{L}_1 \bullet \mathcal{L}_2$, whose domain is $\mathrm{dom}(\mathcal{L}_1)\mathrm{dom}(\mathcal{L}_2)$, and such that for all $u = u_1u_2$, with $u_i \in \mathrm{dom}(\mathcal{L}_i)$ for $i \in \{1, 2\}$, $(\mathcal{L}_1 \bullet \mathcal{L}_2)(u) = \mathcal{L}_1(u_1) \circ \mathcal{L}_2(u_2)$.

We first explain how to define the concatenation of two FEFs E_1 and E_2, denoted by $E_1 \bullet E_2$. Its underlying flow is $F = \mathrm{flow}(E_1) \circ \mathrm{flow}(E_2)$. As the sizes of flows are bounded, it is not difficult to observe that the function expressions labelling the edges of F can be obtained from the ones of E_1 and E_2 using the Hadamard and Cauchy product operators.

The concatenation of labels is then obtained by cartesian product as:

$$\mathcal{L}_1 \bullet \mathcal{L}_2 = \{E_1 \bullet E_2 \mid E_1 \in \mathcal{L}_1, E_2 \in \mathcal{L}_2\}.$$

Kleene plus. Given a label $\mathcal{L}$ of size (m, m) such that $\mathrm{dom}(\mathcal{L})$ is unambiguously iterable, we will explain how to define the *Kleene plus of* $\mathcal{L}$, denoted by $\mathcal{L}^+$, whose domain is $\mathrm{dom}(\mathcal{L})^+$, and which is such that

$$\forall u = u_1u_2 \dots u_n \text{ with } \forall i, u_i \in \mathrm{dom}(\mathcal{L}), \quad \mathcal{L}^+(u) = \mathcal{L}(u_1) \circ \mathcal{L}(u_2) \circ \dots \circ \mathcal{L}(u_n).$$

Constructing the label $\mathcal{L}^+$ is difficult and is addressed in the next two subsections.

4.3 Kleene plus over simple labels

Definition 12. *An FEF $E \in \mathbb{E}$ is* simple *if its flow contains no return edge.*
A label $\mathcal{L} \in \mathbb{L}$ is simple *if for all $E, E' \in \mathcal{L}$,* $\mathrm{flow}(E \bullet E') = \mathrm{flow}(E)$.

As a first step, given a simple FEF E and a language L that is unambiguously iterable and such that $L^2 \subseteq \mathrm{dom}(E)$, we define the *chained star of E w.r.t. L*, denoted by $\langle E, L\rangle^{\circledast}$, as follows. It is a FEF whose domain is $L^{\geqslant 2}$ and such that for any word $u = u_1u_2 \dots u_n \in L^{\geqslant 2}$, we have $\langle E, L\rangle^{\circledast}(u) = E(u_1u_2) \circ E(u_2u_3) \circ \dots \circ E(u_{n-1}u_n)$. This FEF is obtained from E by applying the chained star operator (or its left version) on each (crossing) edge of E.

We turn to the construction of the label $\mathcal{L}^+$ when $\mathcal{L}$ is a simple label. The definition of simple labels implies that for all $n > 0$ and $E, E_1, \dots, E_n \in \mathcal{L}$, $\mathrm{flow}(E) = \mathrm{flow}(E \bullet E_1 \bullet E_2 \bullet \dots \bullet E_n)$. In consequence, the flows that appear in $\mathcal{L}^{\geq 2}$ are the flows of $\mathcal{L}$. Let $L = \mathrm{dom}(\mathcal{L})$. Given some $E \in \mathcal{L}$, we claim we can build a FEF $\tilde{E}$ with domain $\mathrm{dom}(E)L^{\geq 1}$, size (m, m) and $\mathrm{flow}(\tilde{E}) = \mathrm{flow}(E)$, that "simulates" all sequences of FEFs of $\mathcal{L}$ beginning by E. More precisely, its semantics is as follows: for all $E_1, \dots E_n \in \mathcal{L}$, if $u = u_0u_1 \dots u_n$ with $u_0 \in \mathrm{dom}(E)$ and $u_i \in \mathrm{dom}(E_i)$ for all i, then

$$\tilde{E}(u) = E(u_0) \circ E_1(u_1) \circ \dots \circ E_n(u_n). \tag{1}$$

We can then define the label $\mathcal{L}^+$ as $\mathcal{L} \oplus \{\tilde{E} \mid E \in \mathcal{L}\}$.

We detail now the construction of $\tilde{E}$ for a FEF $E \in \mathcal{L}$, depicted in Figure 4. We start with a property of simple labels, obtained using Lemma 3:

Lemma 13. *All FEFs of a simple label have the same number of crossing edges.*

We first decompose E into two FEFs $\bar{E}$ and $\hat{E}$ such that for every word $u \in \mathrm{dom}(E)$, we have $E(u) = \hat{E}(u) \circ \bar{E}(u)$. Formally, given a FEF E, $\bar{E}$ is the FEF obtained by deleting all return edges on the left (and the corresponding nodes), and $\hat{E}$ is the FEF obtained by deleting all return edges on the right (and the corresponding nodes) and moreover by replacing all function expressions of crossing edges of E with $\mathrm{dom}(E)/\varepsilon$ (dashed edges in Figure 4). By Lemma 13, there exists c such that each $E \in \mathcal{L}$ has c crossing edges. Let F be the flow consisting of no return edges and c crossing edges, then for all $E_1, E_2 \in \mathcal{L}$, we have $\mathrm{flow}(\bar{E}_1 \bullet \hat{E}_2) = F$. As a consequence, we can define [1] the FEF $E_{\clubsuit} = \bigoplus_{E_1,E_2 \in \mathcal{L}} \bar{E}_1 \bullet \hat{E}_2$. Observe that $E_{\clubsuit}$ is a simple FEF. We can thus use the chained star $E_{\circledast} = \langle E_{\clubsuit}, L \rangle^{\circledast}$ of domain $L^{\geq 2}$ to rewrite Eq. (1) as:

$$\tilde{E}(u) = \hat{E}(u_0) \circ \underbrace{(\bar{E} \bullet \hat{E}_1)(u_0u_1) \circ (\bar{E}_1 \bullet \hat{E}_2)(u_1u_2) \circ \ldots \circ (\bar{E}_{n-1} \bullet \hat{E}_n)(u_{n-1}u_n)}_{E_{\circledast}(u_0 \ldots u_n)} \circ \bar{E}_n(u_n)$$

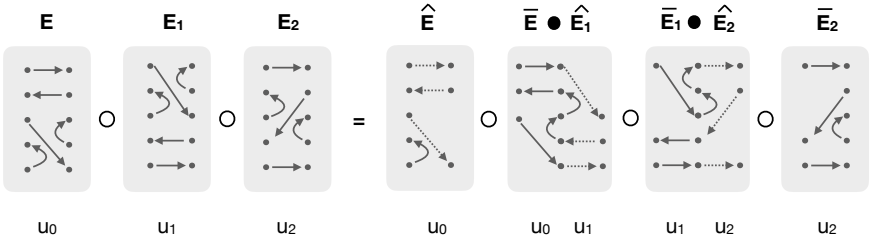

Fig. 4. Illustration of the decomposition used for simple labels.

Remark 14. We have thus proven that we are able to compute the Kleene plus for simple labels. This implies that we can apply our BMC algorithm for a given FEFA if each time a state is removed, its self-loop is labelled with a simple label. We call this the *simple-BMC algorithm*.

4.4 Kleene plus over arbitrary labels

We present now how to construct the label $\mathcal{L}^+$ when $\mathcal{L}$ is an arbitrary label. For simplicity, we construct a label $\mathcal{L}^{\geq 2}$, and then define $\mathcal{L}^+ = \mathcal{L} \oplus \mathcal{L}^{\geq 2}$. The intuitive idea is to use the previously described simple-BMC algorithm. Therefore, we consider an automaton $\mathcal{A}$ that consists only of three transitions $\iota \xrightarrow{\mathcal{L}} \alpha \xrightarrow{\mathcal{L}} \alpha \xrightarrow{\mathcal{L}} \beta$ with initial state ι and final state β. We will exhibit a finite unfolding $\mathcal{A}_{\mathrm{Unf}} = (Q_{\mathrm{Unf}}, \iota, \beta, \longrightarrow_{\mathrm{Unf}})$ of $\mathcal{A}$, equivalent to $\mathcal{A}$, which has a property ensuring that the simple-BMC algorithm can be applied.

Let $2N + 1$ be the greatest number of crossing edges among the FEFs of $\mathcal{L}$. Let S_i be the sets of underlying flows with $2i + 1$ crossing edges finitely generated

[1] Given two FEFs E, E' with same underlying flow F, $E \oplus E'$ is the FEF with flow F and set of edges $\{(x, f \oplus f', y) \mid (x, f, y) \in E, (x, f', y) \in E'\}$.

by $\mathcal{L}$ with concatenation of FEFs. By Lemma 3 there are at most N such sets S_i. Then, we set $Q_{\mathrm{Unf}} = \{\iota, \beta\} \cup \prod_{i\in 0,\dots,N}(S_i \cup \{\top\})$. The alphabet of $\mathcal{A}_{\mathrm{Unf}}$ consists of subsets of $\mathcal{L}$. Intuitively, the state $(s_i)_{0\leq i\leq N}$ reached in $\mathcal{A}_{\mathrm{Unf}}$ after reading a sequence of labels $\mathcal{L}_1, \dots, \mathcal{L}_n$ is a decomposition of $\mathrm{flow}(\mathcal{L}_1 \bullet \dots \bullet \mathcal{L}_n)$. Formally the transition function $\longrightarrow_{\mathrm{Unf}}$ is defined as follows: ($\top$ is neutral for composition)

- for each $F \in \{\mathrm{flow}(E) \mid E \in \mathcal{L}\}$, $\iota \xrightarrow{\{E\in\mathcal{L} \mid \mathrm{flow}(E)=F\}}_{\mathrm{Unf}} (s_i)_{0\leq i\leq N}$ with $s_i = F \in S_i$ and for all $j \neq i$, $s_j = \top$;
- $(s_0, \dots, s_N) \xrightarrow{\mathcal{L}}_{\mathrm{Unf}} \beta$ for all $(s_0, \dots, s_N) \in Q_{\mathrm{Unf}}$;
- let $q = (s_0, \dots, s_N) \in Q_{\mathrm{Unf}}$, $0 \leq j \leq N$ and $F \in \mathbb{F}$. We define $\mathcal{L}_{q,j,F}$ as the set of FEFs $E \in \mathcal{L}$ such that $F = s_j \circ s_{j+1} \circ \dots \circ s_N \circ \mathrm{flow}(E) \in S_j$ and for all $i > j$, $s_i \circ s_{i+1} \circ \dots \circ s_N \circ \mathrm{flow}(E) \notin S_i$. If $\mathcal{L}_{q,j,F} \neq \emptyset$, then $q \xrightarrow{\mathcal{L}_{q,j,F}}_{\mathrm{Unf}} (s'_i)_{0\leq i\leq N}$, with $s'_j = F$, $s'_i = s_i$ if $i < j$, and $s'_i = \top$ if $i > j$.

For $j \leq N$, we define Q_j as the set of states $(s_0, \dots, s_j, \top, \dots, \top)$ with $s_j \neq \top$. We write $q_0 \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q_n$ if there is an execution $(q_i \xrightarrow{\mathcal{L}_i}_{\mathrm{Unf}} q_{i+1})_{0\leq i<n}$ of $\mathcal{A}_{\mathrm{Unf}}$ with $\mathcal{L}_\sigma = \mathcal{L}_0 \bullet \dots \bullet \mathcal{L}_{n-1}$, and such that for every $i \in \{1, \dots, n-1\}$, $q_i \in \bigcup_{k\geqslant j} Q_k$.

Lemma 15. *If $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q'$ with $q = (s_i)_{0\leq i\leq N}$ and $q' = (s'_i)_{0\leq i\leq N}$ in Q_j, then: (1) for all $E \in \mathcal{L}_\sigma$, $s_j \circ \mathrm{flow}(E) = s'_j$ and, (2) $\mathrm{flow}(E)$ has $2j+1$ crossing edges.*

A careful analysis of the different types of edges in flows of $\mathcal{L}_\sigma$ allows to deduce from Lemma 15 that $\mathcal{L}_\sigma$ is simple whenever $q' = q$. More generally, using similar techniques, we can show:

Lemma 16. *Let $j \geq 0$ and $q \in Q_j$. If $\{\mathcal{L}_1, \dots, \mathcal{L}_k\}$ is a set of labels such that $q \overset{\mathcal{L}_i}{\rightsquigarrow}_{\geqslant j} q$ for all i, then $\oplus_{i\in\{1,\dots,k\}}\mathcal{L}_i$ is a simple label.*

We now consider the application of BMC algorithm that successively eliminates states in Q_j, for $j = N, N-1, \dots, 0$. We can prove by induction on the number of steps performed by the algorithm that for every remaining state q, the label $\mathcal{L}_q$ of the self-loop around q is simple. Indeed, as we observed in Subsection 4.3, the flows associated with the Kleene plus of a simple label are exactly those of that label. This allows to show that the flows appearing in $\mathcal{L}_q$ are obtained by finitely many paths around q in $\mathcal{A}_{\mathrm{Unf}}$ that only go through states in $\bigcup_{k\geqslant j} Q_k$, where j is such that $q \in Q_j$. Hence $\mathcal{L}_q$ is simple by Lemma 16.

4.5 The case of sweeping transducers

We consider now the case of sweeping transducers. We introduce the two following grammars for function expressions: (L is a regular language over A, $v \in B^*$)

$$\overleftarrow{Rat} \ni f, g ::= L/v \mid f \oplus g \mid f \overset{\leftarrow}{\bullet} g \mid f^{\overset{\leftarrow}{*}}$$
$$SW \ni s, s' ::= s \oplus s' \mid f \mid f \otimes g \quad \text{where } f, g \in Rat \cup \overleftarrow{Rat}$$

Theorem 17. *Sweeping functions are equivalent to SW-expressions.*

Proof. One observes that for sweeping transducers, the Kleene plus only concerns simple FEFs, for which the Kleene star operator of expressions is sufficient. □

5 Conclusion

In this paper, we have extended the standard state elimination algorithm due to Brzozowski and McCluskey to unambiguous two-way finite-state transducers. This yields a simple, direct and effective translation from these transducers to regular function expressions. We have also identified a subclass of expressions characterizing sweeping functions.

This work opens the way to numerous applications and extensions: deciding one-wayness [13, 4], studying infinite words [2] and identifying star-free expressions for first-order definable transformations [16, 14, 8] for instance.

References

1. Alur, R., Černý, P.: Expressiveness of streaming string transducers. In: FSTTCS, LIPIcs., vol. 8, pp. 1–12. Schloss Dagstuhl. Leibniz-Zent. Inform. (2010)
2. Alur, R., Filiot, E., Trivedi, A.: Regular transformations of infinite strings. In: LICS. pp. 65–74 (2012)
3. Alur, R., Freilich, A., Raghothaman, M.: Regular combinators for string transformations. In: CSL-LICS '14. pp. 9:1–9:10. ACM (2014)
4. Baschenis, F., Gauwin, O., Muscholl, A., Puppis, G.: Untwisting two-way transducers in elementary time. In: LICS'17. pp. 1–12. IEEE Computer Society (2017)
5. Berstel, J.: Transductions and context-free languages, Teubner Studienbücher : Informatik, vol. 38. Teubner (1979)
6. Bojanczyk, M.: Transducers with origin information. In: ICALP 2014. LNCS, vol. 8573, pp. 26–37. Springer (2014)
7. Brzozowski, J.A., McCluskey, E.J.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Trans. Electronic Computers 12(2), 67–76 (1963)
8. Carton, O., Dartois, L.: Aperiodic two-way transducers and FO-transductions. In: CSL. LIPIcs, vol. 41, pp. 160–174. Schloss Dagstuhl. Leibniz-Zent. Inform. (2015)
9. Choffrut, C., Guillon, B.: An algebraic characterization of unary two-way transducers. In: MFCS. LNCS, vol. 8634, pp. 196–207. Springer (2014)
10. Courcelle, B.: Monadic second-order definable graph transductions: a survey. Theoret. Comput. Sci. 126(1), 53–75 (1994)
11. Dave, V., Gastin, P., S, K.: Regular transducer expressions for regular transformations over infinite words. In: LICS'18. IEEE Computer Society (2018), to appear
12. Engelfriet, J., Hoogeboom, H.J.: MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Log. 2(2), 216–254 (2001)
13. Filiot, E., Gauwin, O., Reynier, P.A., Servais, F.: From two-way to one-way finite state transducers. In: LICS. pp. 468–477. IEEE Computer Society (2013)
14. Filiot, E., Krishna, S.N., Trivedi, A.: First-order definable string transformations. In: FSTTCS. LIPIcs, vol. 29, pp. 147–159. Schloss Dagstuhl. Leibniz-Zent. Inf. (2014)
15. Lombardy, S.: Two-way representations and weighted automata. RAIRO - Theor. Inf. and Applic. 50(4), 331–350 (2016)
16. McNaughton, R., Papert, S.: Counter-free automata. The M.I.T. Press, Cambridge, Mass.-London (1971)
17. Schützenberger, M.P.: On the definition of a family of automata. Information and Control 4(2-3), 245–270 (1961)
18. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM Journal of Research and Development 3(2), 198–200 (1959)