International Journal of Foundations of Computer Science

From Two-Way Transducers to Regular Function Expressions

Nicolas Baudru
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
nicolas.baudru@lis-lab.fr

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
pierre-alain.reynier@lis-lab.fr

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

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. Recently, a set of regular function expressions has been introduced and used to prove a similar result for regular word functions, by showing its 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, our approach allows us to derive 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, 2, 3, 4, 14, 7, 15, 9]. It is very expressive and allows to express natural trans-

formations 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 [11], 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 formal power series, as investigated in [10, 17]. 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 [20]. Imposing on these regular expressions the restriction of being unambiguous on their input part yields a presentation of rational functions [6]. More recently, a set of *regular combinators* has been introduced allowing to characterize the class of regular functions [3].

In this paper, we propose a new construction showing that any regular function can be expressed using the regular combinators introduced in [3]. Our approach differs from the one of [3] w.r.t. two aspects: first, we take as input unambiguous two-way finite state transducers while [3] starts with copyless streaming string transducers; second, our construction lifts the standard state-elimination algorithm proposed by Brzozowski and McCluskey [8] while [3] follows the approach of McNaughton and Yamada [19]. In order to propose a state elimination algorithm for 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 crossing sequence automaton [16] 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 behaviours 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 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 [17]). Sweeping transducers induce a third class of functions in-between rational and regular functions.

In parallel to our work published in [5], the authors of [12] also studied transformations from two-way transducers to the class of expressions introduced in [3], in order to lift these results to infinite words. They also consider the monoid of flows as starting point of their study. As explained before, the main difficulty lies in the Kleene iteration, which becomes rather easy if the flow is idempotent. This is where our approaches differ. The approach followed in [12] resorts to strong algebraic tools in order to end up directly to the setting of idempotent flows, using an unambiguous version of the forest factorization theorem of Simon [21]. This approach is then extended to infinite words. Our objective is to propose an algorithmic approach to computing an equivalent expression, based on the adaptation of the famous state-elimination algorithm. By nature, this algorithm gathers different behaviours of the automaton and thus requires to deal with (finite) sets of flows. Our notion of simple set of flows allows us to solve the challenge of computing the Kleene iteration of a finite set of flows.

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 Sections 4 and 5, we define the operations of union, concatenation and Kleene star on flows labelled with regular function expressions. In Section 6, 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 7.

The present work is an extension of our previous work [5]. Compared with this conference version, it constitutes a significant improvement including omitted definitions, proofs, algorithms and some additional examples.

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, we define L^k as $\{u_1 \cdots u_k \mid \forall i, u_i \in L\}$. The Kleene star of L, denoted by L^*, is defined as $\bigcup_{k \geqslant 0} L^k$. The Kleene plus of L, denoted by L^+, is defined as LL^*. We write $L^{\geqslant k}$ to refer to the language $L^k L^*$. 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 partial function. A *word-to-word function* is a functional transduction to a free monoid. 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 non-deterministic state automaton* (2NFA) over a finite alphabet A is a tuple $\mathcal{A} = (Q, q_0, Q_f, \delta)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $Q_f \subseteq Q$ 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 $q \in Q_f$. Note that this latter configuration does not allow additional transitions. 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 partial 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 $\vdash$ and $\dashv$ of the input.

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: $\delta \subseteq Q \times \overline{A} \times B^* \times Q \times \{\leftarrow, \rightarrow\}$. Without loss of generality, we suppose in this paper that two distinct transitions in a transducer are either on different input letter, or between distinct pair of states. 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 $\vdash u \dashv$.

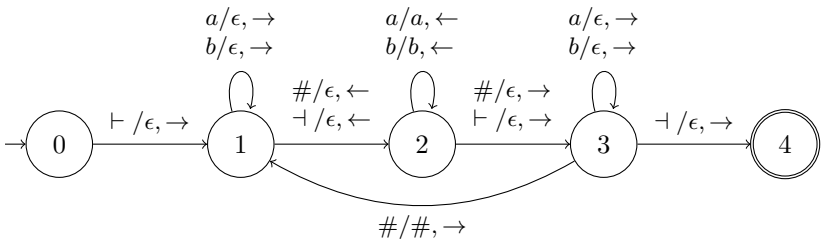

Fig. 1. A 2NFT realizing the function mirror* of Example 2.

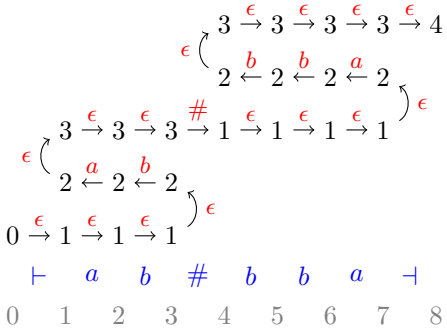

Fig. 2. A run of the 2NFT of Example 2.

We say that a 2NFT $\mathcal{T}$ is *functional* if the relation $R(\mathcal{T})$ is a partial 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 $\mathcal{T}$ is functional.

Theorem 1. [13] *Functional, deterministic and unambiguous* 2NFT *are expressively equivalent, and strictly more expressive than functional sweeping transducers, which in turn are strictly more expressive than 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* 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 $v_i =$* 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.*

2.2. *Crossing sequence automaton construction*

Crossing sequences. A standard tool to analyse two-way machines is the notion of *crossing sequence* [16], and the associated notions of traversals of the run. 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 $(1, 2, 3)$.

More precisely, given a run $(q_0, i_0, d_0) \dots (q_p, i_p, d_p)$, the crossing sequence at position i is defined as follows. Let $0 \leq j_0 < j_1 < \ldots < j_{k-1} \leqslant p$ be the sequence of all indices such that $i_{j_\ell} = i$ for $\ell \in [k]$. Then the crossing sequence at position i is defined as the tuple $(q_{j_0}, \dots, q_{j_{k-1}})$.

Formally, a *crossing sequence* is a non-empty sequence of states $(q_0, q_1, \dots, q_{k-1})$ with $k \geq 1$ and $q_i \in Q$ for every $i \in [k]$ such that:

(1) k is odd
(2) $\forall 0 \leqslant i \neq j < k$, if $q_i = q_j$ then i is odd and j is even, or conversely.

It is well known that crossing sequences encountered along accepting runs of unambiguous 2NFA satisfy Property (2). Indeed, a state cannot appear twice in a crossing sequence at two indices of same parity, otherwise a loop is entered. As a consequence, crossing sequences have size bounded by $2|Q|$.

Flows. 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. Formally, a flow F of size (n, m), with n, m two positive odd integers, is a directed graph 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. A vertex (x, D) is said *even* if x is even, it is said *odd* if x is odd. An edge is a *crossing edge* if it is between two vertices on different sides, it is a *return edge* otherwise. The edges should satisfy the following additional constraints that intuitively imply that the flow represents a stretch of run between two crossing sequences (see [4]):

- a return edge on the left is always between two vertices (x, L) and $(x+1, L)$ with x even,
- a return edge on the right is always between two vertices (x, R) and $(x+1, R)$ with x odd,
- every crossing edge from (x, L) to (y, R) is such that x and y are even,
- every crossing edge from (x, R) to (y, L) is such that x and y are odd,
- every vertex has exactly one adjacent edge (either incoming or outgoing),
- no crossing edges cross over.

Examples of flows are depicted in Figure 3.

Any flow F can be partitioned according to the type of edges :

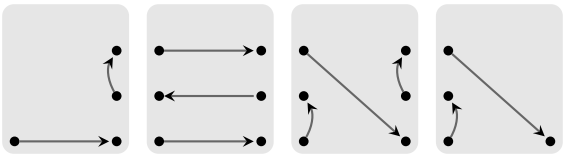

Fig. 3. Examples of flows. Considering Example 2, the first flow is associated with symbol $\vdash$, the second with symbols a and b, the third with $\#$, and the fourth with $\dashv$.

- F_{LR} consists of all crossing edges of F with even vertices,
- F_{RL} consists of all crossing edges of F with odd vertices,
- F_{LL} consists of all return edges of F with even source,
- F_{RR} consists of all return edges of F with odd source,

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 set of flows. Clearly, $\mathbb{F}$ is finite. It is well-known that the set of flows can be equipped with an operation of composition that is associative, yielding a monoid. Intuitively, this operation can be understood as the identification of paths in the concatenation of the two graphs. We denote by $F \circ F'$ this composition.

We state here a first property of the composition 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.*

Proof. Let $G = F \circ F'$. Consider an edge in G_{LR}. It necessarily involves an edge in F_{LR} and an edge in F'_{LR}. Hence, $|G_{LR}| \leqslant |F_{LR}|$ and $|G_{LR}| \leqslant |F'_{LR}|$. Similarly, any edge in G_{RL} involves an edge in F_{RL} and an edge in F'_{RL}, and thus $|G_{RL}| \leqslant |F_{RL}|$ and $|G_{RL}| \leqslant |F'_{RL}|$. If we consider the number of crossing edges of G, *i.e.* the size of $G_{LR} \cup G_{RL}$, we obtain that this number is bounded by $|F_{LR}| + |F_{RL}|$ and by $|F'_{LR}| + |F'_{RL}|$, *i.e.* bounded by $\min(k, l)$, as expected. □

Crossing sequence automaton. Consider an unambiguous two-way automaton $\mathcal{A} = (Q, q_0, Q_f, \delta)$. The *crossing sequence automaton* of $\mathcal{A}$ is an equivalent one-way automaton $\mathcal{B} = (P, s_0, P_f, \delta_B)$ defined as follows:

- states are crossing sequences of $\mathcal{A}$,
- the initial state is the initial crossing sequence, *i.e.* $s_0 = (q_0)$
- final states are crossing sequences of the form $\overrightarrow{p} = (q)$, with $q \in Q_f$
- the transition relation δ_B is defined as follows: there is a transition from $\overrightarrow{p} = (p_0, \ldots, p_{m-1})$ to $\overrightarrow{q} = (q_0, \ldots, q_{n-1})$ on letter $a \in A$ if, and only if, there exists a flow $F \in \mathbb{F}$ of size $(|\overrightarrow{p}|, |\overrightarrow{q}|)$ such that:
  - for every return edge $((x, L), (x{+}1, L))$ in F_{LL}, we have $(p_x, a, p_{x+1}, \leftarrow) \in \delta$,

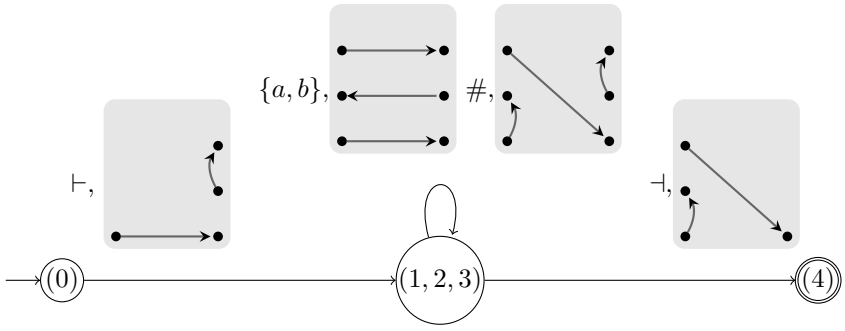

Fig. 4. The crossing sequence automaton of the underlying 2NFA of Example 2. Flows associated with transitions are indicated for clarity.

- for every return edge $((x, R), (x + 1, R))$ in F_{RR}, we have $(q_x, a, q_{x+1}, \rightarrow) \in \delta$,
- for every crossing edge $((x, L), (y, R))$ in F_{LR}, we have $(p_x, a, q_y, \rightarrow) \in \delta$,
- for every return edge $((x, R), (y, L))$ in F_{RL}, we have $(q_x, a, p_y, \leftarrow) \in \delta$.

This automaton may be of exponential size. W.l.o.g., we assume that the crossing sequence automaton is trimmed, meaning that every state appears in some accepting run. Note that as $\mathcal{A}$ is unambiguous, for every transition in the trimmed part of the crossing sequence automaton of $\mathcal{A}$, there is a single flow that can be associated with it. Otherwise, this would contradict the unambiguity of $\mathcal{A}$. As a consequence, the crossing sequence automaton of $\mathcal{A}$ is unambiguous too. An example is depicted in Figure 4.

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 [10, 17]). 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 a function f such that $\mathrm{dom}(f)$ is unambiguously iterable, the *Kleene star* f^* and the *left Kleene star* $f^{\overset{\leftarrow}{*}}$ are such that $\mathrm{dom}(f^*) = \mathrm{dom}(f^{\overset{\leftarrow}{*}}) = L^*$, $f^*(\varepsilon) = f^{\overset{\leftarrow}{*}}(\varepsilon) = \varepsilon$, and:

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

From these operators, we also define the *Kleene plus* (resp. *left Kleene plus*) as $f^+ = f \bullet f^*$ (resp. as $f^{\overset{\leftarrow}{+}} = f \overset{\leftarrow}{\bullet} f^{\overset{\leftarrow}{*}}$).

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:

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

Remark 4. *The Kleene star can be expressed using the Chained star, and the Hadamard and Cauchy products. Indeed, if f is a Reg-expression of domain L, then*

$$f \bullet f \bullet f^* \equiv \langle f \bullet L/\varepsilon, L\rangle^{\circledast} \otimes (L^*/\varepsilon \bullet f).$$

From this expression, f^* can be obtained using the sum operator. Observe the use of the construction $f \bullet L/\varepsilon$ which allows to prolong the domain of a partial function f by juxtaposing words from L. We will use this construct later in this work.

We consider the following grammars:

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

where L is a regular language over A and $v \in B^*$. A function expression obtained from some grammar G is called a G-expression. *Reg*-expressions are called *regular function expressions.*

Example 5. *We give examples to illustrate these operators:*

- *the identity function mapping on A can be defined as* $\mathsf{id}_A = \bigoplus_{a\in A}\{a\}/a$,
- *the identity function mapping on A^* can be defined as* $\mathsf{id}_{A^*} = (\mathsf{id}_A)^*$,
- *the mirror image on A^* can be defined as* $\mathsf{mirror}_{A^*} = (\mathsf{id}_A)^{\overleftarrow{*}}$,
- *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 6. *The following equivalences hold:*

- *Rational functions are equivalent to Rat-expressions* [6].
- *Regular functions are equivalent to Reg-expressions* [3].

3. Function Expression Flow Automata

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

Definition 7. *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 $\mathbb{W}$.*

Let W be a word flow. Following previous notations, we denote by W_{LR} the set of crossing edges of W from left to right. Such an edge is denoted by a triple (x, v, y) where x is a left vertex, $v \in B^*$ and y is a right vertex. Similarly, we define W_{LL}, W_{RL} and W_{RR}.

Composition of flows can be lifted to word flows. This operation can be understood as the identification of labelled paths in the concatenation of the two labelled graphs. Formally, let W, W' be two word flows in $\mathbb{W}$ of size (n, m) and (m, k) respectively. Then the composition $W \circ W'$ is a word flow $G \in \mathbb{W}$ of size (n, k) defined as follows:

- if there are $l \geq 0$ and a sequence of edges $(x_i, v_i, x_{i+1})_{0\leq i<2l+2}$ of $W_{LR} \times (W'_{LL} \times W_{RR})^l \times W'_{LR}$ (resp. $W'_{RL} \times (W_{RR} \times W'_{LL})^l \times W_{RL}$) then $(x_0, v_0 \cdots v_{2l+1}, x_{2l+2}) \in G_{LR}$ (resp. in G_{RL});
- if there is an edge (x, v, x') of W_{LL} (resp. W'_{RR}) then $(x, v, x') \in G_{LL}$ (resp. in G_{RR});
- if there are $l \geq 0$ and a sequence of edges $(x_i, v_i, x_{i+1})_{0\leq i<2l+3}$ of $W_{LR} \times W'_{LL} \times (W_{RR} \times W'_{LL})^l \times W_{RL}$ (resp. $W'_{RL} \times W_{RR} \times (W'_{LL} \times W_{RR})^l \times W'_{LR}$) then $(x_0, v_0 \cdots v_{2l+2}, x_{2l+3}) \in G_{LL}$ (resp. in G_{RR});

Figure 5 illustrates such a composition.

Definition 8. *A* function expression flow (FEF) E *over domain L is a flow of $\mathbb{F}$ whose edges are labelled by Reg-expressions from A^* to B^* of domain L.*

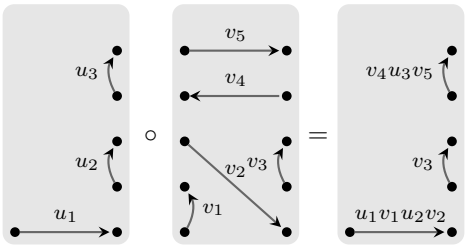

Fig. 5. Composition of word flows.

We denote by $\mathbb{E}$ the set of all FEFs, by $\mathrm{dom}(E)$ the domain of a FEF E and by $\mathrm{flow}(E)$ its underlying flow ($\mathrm{flow}(E) \in \mathbb{F}$). We extend the notations of edges of flows to FEFs. We denote by E_{LR} the set of all *labelled* crossing edges of a FEF E with even vertices, by E_{RL} the set of all *labelled* crossing edges of E with odd vertices, and so on.

A FEF $E \in \mathbb{E}$ 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 9. *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: $\mathrm{dom}(\mathcal{L}) = \biguplus_{E \in \mathcal{L}} \mathrm{dom}(E)$. Since the FEFs of a label are pairwise disjoint, 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 10. *Let n, m be two positive odd integers. An* (n, m)-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 $\mathsf{size}(q_0) = n$, $\mathsf{size}(q_f) = m$ 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 an (n, m)-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 in δ 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 an (n, m)-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 in the definition of FEFA

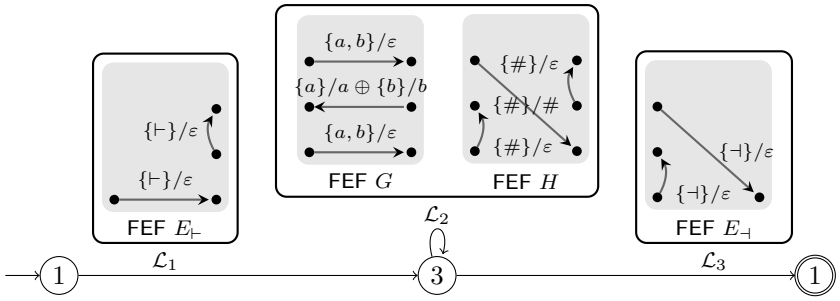

Fig. 6. A $(1,1)$-FEFA for the transducer of Figure 1. States are labelled with their size, $\mathrm{dom}(\mathcal{L}_1) = \{\vdash\}$, $\mathrm{dom}(\mathcal{L}_2) = \mathrm{dom}(G) \uplus \mathrm{dom}(H) = \{a, b\} \uplus \{\#\}$ and $\mathrm{dom}(\mathcal{L}_3) = \{\dashv\}$.

ensure that this composition is well defined and, moreover, that $\mathrm{out}(u)$ is of size (n, m). From now on, we assume all our FEFA are unambiguous.

Noteworthy, the semantics of a $(1,1)$-FEFA $\mathcal{A}$ can be identified with a functional transduction from A^* to B^*. Indeed, given a word u with some accepting execution which produces the word flow $\mathrm{out}(u)$, $\mathrm{out}(u)$ 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 the crossing sequence automaton construction:

Lemma 11. *Given an unambiguous* 2NFT*, one can build an equivalent unambiguous* $(1,1)$*-FEFA.*

Proof. First observe that in the crossing sequence automaton, one could merge the different final states, as there is no outgoing transition from final states. Starting then from this modified construction of the crossing sequence automaton, it is easy to obtain a translation of an unambiguous 2NFT into an unambiguous FEFA. The set of states is the same, as well as the initial and final states. Given a transition $(\overrightarrow{p}, a, \overrightarrow{q})$ of the crossing sequence automaton, we build a transition in the FEFA as follows. We consider the flow used to build the transition in the crossing sequence automaton (as observed earlier, this flow is unique as the 2NFT is unambiguous), and we label the edges of this flow by the function expression $\{a\}/v$ whenever the corresponding transition of the 2NFT has v as output word.

It remains to observe that the resulting FEFA has the expected properties. By construction, the initial state has no incoming edge, and the final state has no outgoing edge. The third property of δ is ensured by the crossing sequences labelling the states of the FEFA. Last, the unambiguity of the FEFA follows from that of the 2NFT. Similarly, the initial and final flows do satisfy the expected size properties.□

Figure 6 depicts the unambiguous $(1,1)$-FEFA obtained from the transducer of Figure 1 by using the crossing sequence automaton construction.

4. Elementary operations on FEFs and labels

The standard Brzozowski and McCluskey (BMC for short) algorithm takes as input a one-way finite-state automaton $\mathcal{A}$. W.l.o.g., we assume it is *normalised*, *i.e.* it has a single initial (resp. final) state with no incoming transitions, resp. no outgoing transitions. All states are supposed to be accessible and co-accessible. The algorithm outputs a regular expression equivalent to $\mathcal{A}$. It 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 $\mathcal{A}$, the transitions $q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_2 \xrightarrow{e_3} q_3$ and $q_1 \xrightarrow{e_4} q_3$ are replaced 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 state and the final state. The regular expression 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.

4.1. *Sum, concatenation and chained star of FEFs*

Definition 12 (Sum of FEFs) *Let E and E' be two FEFs. The* sum of E and E'*, denoted $E \oplus E'$, is defined if E and E' have the same underlying flow and if they have disjoint domains. In this case, $E \oplus E'$ is the mapping from $\mathrm{dom}(E) \uplus \mathrm{dom}(E')$ to $\mathbb{W}$ mapping a word u to $E(u)$ if $u \in \mathrm{dom}(E)$, and to $E'(u)$ if $u \in \mathrm{dom}(E')$.*

Proposition 13. *Let E and E' be two FEFs that have the same underlying flow and disjoint domains. One can compute a FEF G that defines the same function as $E \oplus E'$.*

Proof. By assumption, E and E' have the same underlying flow F. G also has F as underlying flow, and its set of edges is $\{(x, f \oplus f', y) \mid (x, f, y) \in E \text{ and } (x, f', y) \in E'\}$. □

Definition 14 (Concatenation of FEFs) *Let E and E' be two FEFs. The* concatenation of E and E'*, denoted $E \bullet E'$, is defined if E and E' have unambiguously concatenable domains and if the composition $\mathrm{flow}(E) \circ \mathrm{flow}(E')$ is well defined. In this case, $E \bullet E'$ is the mapping from $\mathrm{dom}(E) \bullet \mathrm{dom}(E')$ to $\mathbb{W}$ such that:*

$$\forall u = vv' \text{ with } v \in \mathrm{dom}(E), v' \in \mathrm{dom}(E'), \quad E \bullet E'(u) = E(v) \circ E'(v').$$

Proposition 15. *Let E and E' be two FEFs that have unambiguously concatenable domains and such that the composition* $\mathrm{flow}(E) \circ \mathrm{flow}(E')$ *is well defined. One can compute a FEF G that defines the same function as $E \bullet E'$.*

Proof. We define the FEF G as follows. First, its underlying flow is $F = \mathrm{flow}(E) \circ \mathrm{flow}(E')$.

As the sizes of flows are bounded, it is not difficult to observe that the function expressions labelling the edges of G can be obtained from the ones of E and E' using the Hadamard and Cauchy product operators. Formally:

- if there are $l \geq 0$ and a sequence of edges $(x_i, f_i, x_{i+1})_{0 \leq i < 2l+2}$ of $E_{LR} \times (E'_{LL} \times E_{RR})^l \times E'_{LR}$ (resp. $E'_{RL} \times (E_{RR} \times E'_{LL})^l \times E_{RL}$) then (x_0, f, x_{2l+2}) is an edge in G_{LR} (resp. in G_{RL}) with
$$f = \bigotimes_{0 \leq i \leq l} (f_{2i} \bullet f_{2i+1}) \qquad (\text{resp. } f = \bigotimes_{0 \leq i \leq l} (f_{2i+1} \overset{\leftarrow}{\bullet} f_{2i})).$$
- if there is an edge (x_1, f_1, x_2) of E_{LL} (resp. in E'_{RR}) then (x_1, f, x_2) is an edge in G_{LL} (resp. in G_{RR}) with
$$f = f_1 \bullet (\mathrm{dom}(E')/\varepsilon) \qquad (\text{resp. } f = (\mathrm{dom}(E)/\varepsilon) \bullet f_1)).$$
- if there are $l \geq 0$ and a sequence of edges $(x_i, f_i, x_{i+1})_{0 \leq i < 2l+3}$ of $E_{LR} \times E'_{LL} \times (E_{RR} \times E'_{LL})^l \times E_{RL}$ (resp. $E'_{RL} \times E_{RR} \times (E'_{LL} \times E_{RR})^l \times E'_{LR}$) then (x_0, f, x_{2l+3}) is an edge in G_{LL} (resp. in G_{RR}) with
$$f = \left(\textstyle\bigotimes_{0 \leq i \leq l} (f_{2i} \bullet f_{2i+1})\right) \otimes (f_{2l+2} \bullet (\mathrm{dom}(E')/\varepsilon))$$
$$(\text{resp. } f = \left(\textstyle\bigotimes_{0 \leq i \leq l} (f_{2i+1} \overset{\leftarrow}{\bullet} f_{2i})\right) \otimes ((\mathrm{dom}(E)/\varepsilon) \bullet f_{2l+2})). \qquad \square$$

We now define the chained star for a restricted class of FEFs.

Definition 16. *A flow is* sweeping *if it consists only of crossing edges.*

A word flow or a FEF is sweeping if its underlying flow is.

Definition 17 (Chained star of simple FEFs) *Given a sweeping FEF E and an unambiguously iterable language L such that $L^2 \subseteq \mathrm{dom}(E)$, the* chained star of E w.r.t. L, *denoted by $\langle E, L \rangle^{\circledast}$, is the mapping from $L^{\geqslant 2}$ to $\mathbb{W}$ such that for all words $u = u_1 u_2 \cdots u_n \in L^{\geqslant 2}$ with $u_i \in L$ for $i \in \{1, \ldots, n\}$, we have*

$$\langle E, L \rangle^{\circledast}(u) = E(u_1 u_2) \circ E(u_2 u_3) \circ \ldots \circ E(u_{n-1} u_n).$$

Proposition 18. *Let E be a sweeping FEF and L be an unambiguously iterable language L such that $L^2 \subseteq \mathrm{dom}(E)$. One can compute a FEF G that defines the same function as $\langle E, L \rangle^{\circledast}$.*

Proof. The FEF G has the same underlying sweeping flow as E. It is obtained by applying the chained star operator (or its left version) on each (crossing) edge of E:

if there is an edge (x, f_1, y) in E_{LR} then $(x, \langle f_1, L\rangle^{\circledast}, y)$ is an edge in $\langle E, L\rangle^{\circledast}_{LR}$; if there is an edge (x, f_1, y) in E_{RL} then $(x, \langle f_1, L\rangle^{\overleftarrow{\circledast}}, y)$ is an edge in $\langle E, L\rangle^{\circledast}_{RL}$. □

Chained star of sweeping FEFs will be useful in Section 5 to compute the Kleene plus of a special kind of labels.

4.2. *Sum, concatenation and Kleene plus of labels*

Definition 19 (Sum of labels) *Let $\mathcal{L}_1, \mathcal{L}_2$ be two labels of $\mathbb{L}$. The* sum of $\mathcal{L}_1$ and $\mathcal{L}_2$*, denoted by $\mathcal{L}_1 \oplus \mathcal{L}_2$, is defined if $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same size and disjoint domains. In this case, $\mathcal{L}_1 \oplus \mathcal{L}_2$ is the mapping from $\mathrm{dom}(\mathcal{L}_1) \uplus \mathrm{dom}(\mathcal{L}_2)$ to $\mathbb{W}$ mapping a word u to $\mathcal{L}_1(u)$ if $u \in \mathrm{dom}(\mathcal{L}_1)$, and to $\mathcal{L}_2(u)$ if $u \in \mathrm{dom}(\mathcal{L}_2)$.*

Proposition 20. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two labels that have the same size and disjoint domains. One can compute a label $\mathcal{L}$ that defines the same function as $\mathcal{L}_1 \oplus \mathcal{L}_2$.*

Proof. This label is simply obtained by taking the union of FEFs of $\mathcal{L}_1$ and $\mathcal{L}_2$.□

We say that two labels $\mathcal{L}_1$ and $\mathcal{L}_2$, of size (n, m) and (k, l) respectively, are *compatible for composition* if we have $m = k$.

Definition 21 (Concatenation of labels) *Let $\mathcal{L}_1, \mathcal{L}_2$ be two labels of $\mathbb{L}$. The* concatenation of $\mathcal{L}_1$ and $\mathcal{L}_2$*, denoted by $\mathcal{L}_1 \bullet \mathcal{L}_2$, is defined if $\mathcal{L}_1$ and $\mathcal{L}_2$ are compatible for composition and if they have unambiguously concatenable domains. In this case, $\mathcal{L}_1 \bullet \mathcal{L}_2$ is the mapping from $\mathrm{dom}(\mathcal{L}_1) \bullet \mathrm{dom}(\mathcal{L}_2)$ to $\mathbb{W}$ such that:*

$$\forall u = u_1u_2 \text{ with } u_1 \in \mathrm{dom}(\mathcal{L}_1), u_2 \in \mathrm{dom}(\mathcal{L}_2), \quad \mathcal{L}_1 \bullet \mathcal{L}_2(u) = \mathcal{L}_1(u_1) \circ \mathcal{L}_2(u_2).$$

Proposition 22. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two labels that are compatible for composition and have disjoint domains. One can compute a label $\mathcal{L}$ that defines the same function as $\mathcal{L}_1 \bullet \mathcal{L}_2$.*

Proof. The concatenation of labels is obtained by cartesian product as:

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

This is well-defined because the domains of $\mathcal{L}_1$ (of size (n, m)) and $\mathcal{L}_2$ (of size (m, k)) are supposed to be unambiguously concatenable. □

Definition 23 (Kleene plus of labels) *Let $\mathcal{L}$ be a label. The* Kleene plus of $\mathcal{L}$*, denoted by $\mathcal{L}^+$, is defined if $\mathcal{L}$ has size (m, m) for some m, and if its domain is unambiguously iterable. In this case, $\mathcal{L}^+$ is the mapping from $\mathrm{dom}(\mathcal{L})^+$ to $\mathbb{W}$ such that:*

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

Computing a label realizing the function $\mathcal{L}^+$ is actually a technical challenge. In Section 5, we solve this problem for a particular class of labels. In Section 6, we show how the general case can be reduced to this subclass.

5. Computing Kleene plus of simple labels

In this section, we present a construction called **KleenePlus** that outputs the Kleene plus of a given label. This construction is defined for *simple* labels only.

Definition 24. *A label* $\mathcal{L} \in \mathbb{L}$ *is* simple *if for all* $E, E' \in \mathcal{L}$, $\mathrm{flow}(E \bullet E') = \mathrm{flow}(E)$.

Let $\mathcal{L}$ be a simple label of size (m, m) with an unambiguously iterable domain $L = \mathrm{dom}(\mathcal{L})$. Given some $E \in \mathcal{L}$, we claim we can build a FEF $\tilde{E}$ with domain $\mathrm{dom}(E)L^+$, size (m, m) and underlying flow $F = \mathrm{flow}(E)$, that "simulates" all sequences of FEFs of $\mathcal{L}$ beginning by E. More precisely, we will prove in Proposition 30 that $\tilde{E}$ has the same semantics as the function ϕ_E defined from $\mathrm{dom}(E)L^+$ to $\mathbb{W}$ by

$$\phi_E(u) = E(u_0) \circ E_1(u_1) \circ \cdots \circ E_n(u_n) \tag{1}$$

where $u_0 u_1 \cdots u_n$ is the unique decomposition of u and $(E_1, \ldots, E_n)$ is the unique sequence of FEFs of $\mathcal{L}$ such that $u_0 \in \mathrm{dom}(E)$ and $u_i \in \mathrm{dom}(E_i)$ for all i. Unicity holds because L is unambiguously iterable and, moreover, all FEFs of $\mathcal{L}$ have disjoint domains (by definition of labels).

The construction of $\tilde{E}$ is detailed in Section 5.2.

We denote by $\mathbf{KleenePlus}(\mathcal{L})$ the label $\mathcal{L} \oplus \{\tilde{E} \mid E \in \mathcal{L}\}$. Note that the sum is well-defined because $\mathrm{dom}(\tilde{E}) = \mathrm{dom}(E)L^+$ and L is unambiguously iterable. As a consequence of this definition and of Proposition 30, we obtain:

Proposition 25. *Let* $\mathcal{L}$ *be a simple label. The label* $\boldsymbol{KleenePlus}(\mathcal{L})$ *defines the same function as* $\mathcal{L}^+$.

5.1. *Some basic properties of simple labels*

The construction of $\tilde{E}$ relies on some important properties of simple labels.

First, Lemma 26 states that all FEFs in a simple label are alike. In this lemma, $\mathrm{flow}(\mathcal{L})$ refers to the set of flows that appear in a label $\mathcal{L}$: $\mathrm{flow}(\mathcal{L}) = \{\mathrm{flow}(E) \mid E \in \mathcal{L}\}$.

Lemma 26. *Let* $\mathcal{L}$ *be a simple label of size* (m, m) *with an unambiguously iterable domain.*

(1) For every $E_1, \ldots, E_n \in \mathcal{L}$, $\mathrm{flow}(E_1 \bullet \cdots \bullet E_n) \in \mathrm{flow}(\mathcal{L})$.
(2) The flows of $\mathrm{flow}(\mathcal{L})$ *have the same number of crossing edges.*
(3) The flows of $\mathrm{flow}(\mathcal{L})$ *have the same right return edges.*

Proof. The first item immediately follows the definition of simple labels.

Let E, E' be two FEFs of $\mathcal{L}$ of underlying flows F and F' respectively.

Item 2: Suppose that F and F' have k and l crossing edges respectively. Since $\mathcal{L}$ is simple, $F \circ F' = F$ and $F' \circ F = F'$. Then by Lemma 3, we obtain $k \leqslant l$ and $l \leqslant k$, hence $k = l$.

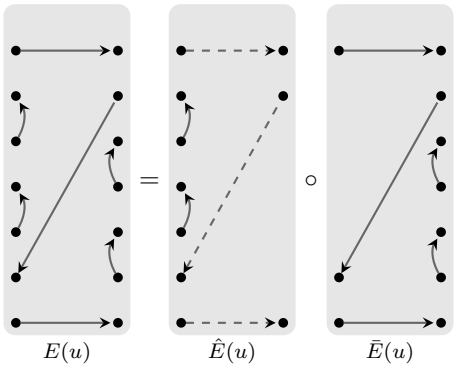

Fig. 7. Decomposition of a FEF. For readability, the *Reg*-expressions in the FEFs are omitted. Dashed edges are labelled by $\mathrm{dom}(E)/\varepsilon$. The other edges are labelled as in E.

Item 3: Since $F \circ F' = F$, the right return edges of F' are all present in F. Similarly, since $F' \circ F = F'$, the right return edges of F are all present in F'. □

Then we present decomposition properties of the FEFs of a simple label. They will be used as a trick when constructing $\tilde{E}$. Let E be a FEF. We can define from E two new FEFs $\bar{E}$ and $\hat{E}$ of domain $\mathrm{dom}(E)$ (see Figure 7). The FEF $\bar{E}$ is obtained from E by deleting all return edges on the left (and the corresponding vertices). The FEF $\hat{E}$ is obtained by deleting from E all return edges on the right (and the corresponding vertices) and, moreover, by replacing every Reg-expression labelling a crossing edge of E with $\mathrm{dom}(E)/\varepsilon$. We have the next key remarks.

Lemma 27. *Let E be a FEF and $u \in \mathrm{dom}(E)$. We have $E(u) = \hat{E}(u) \circ \bar{E}(u)$.*

Proof. By construction, the word flow $\hat{E}(u) \circ \bar{E}(u)$ has the same flow as E. Its left (resp. right) return edges are those of $\hat{E}(u)$ (resp. $\bar{E}(u)$), which are the same as those of $E(u)$. If the i-th LR (resp. RL) crossing edge of E is labelled by the regular function expression f then the i-th LR (resp. RL) crossing edge of $\hat{E}(u) \circ \bar{E}(u)$ is labelled by $\mathrm{dom}(E)/\varepsilon(u) \cdot f(u) = f(u)$ (resp. $f(u) \cdot \mathrm{dom}(E)/\varepsilon(u) = f(u)$). □

Lemma 28. *Let $\mathcal{L}$ be a simple label of size (m, m). There is a sweeping flow F such that $\mathrm{flow}(\bar{E}_1 \bullet \hat{E}_2) = F$ for all $E_1, E_2 \in \mathcal{L}$.*

Proof. By Lemma 26-2, all FEFs of $\mathcal{L}$ has the same number c of crossing edges. By construction, for all $E' \in L$, $\bar{E}'$ is a FEF of size (c, m) with no left return edge, and $\hat{E}'$ is a FEF of size (m, c) with no right return edge. Consequently, for all $E_1, E_2 \in \mathcal{L}$, $\bar{E}_1 \bullet \hat{E}_2$ is the FEF of size (c, c) with no return edge. □

5.2. *Construction of $\tilde{E}$*

Let $\mathcal{L}$ be a simple label and E be a FEF of $\mathcal{L}$. The construction of $\tilde{E}$ is based on the next idea : any word flow $\phi_E(u)$ can be decomposed using Lemmas 27 and 28 into

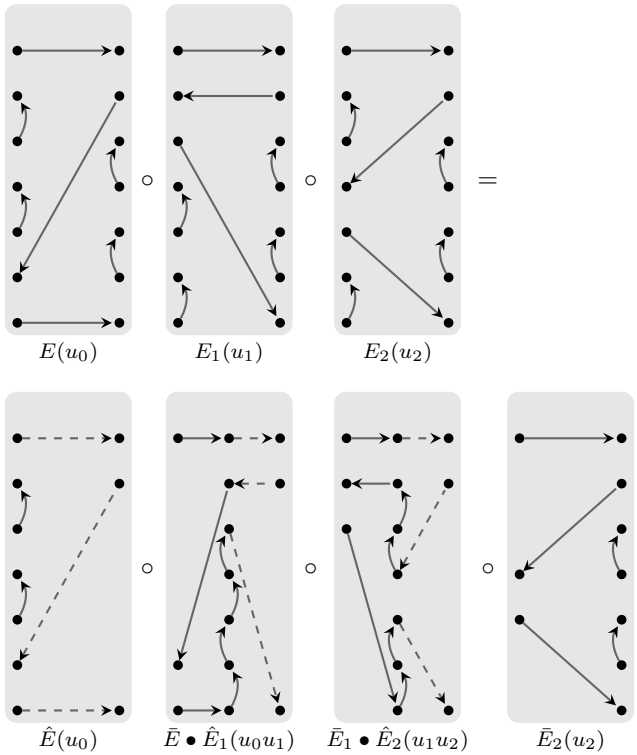

Fig. 8. Illustration of the decomposition used to build $\tilde{E}$ from a label $\{E, E_1, E_2\}$. For readability, the *Reg*-expressions in the FEFs are omitted.

a sequence of word flows which are images of sweeping FEFs of domain L^2 (except for the first and the last one). Figure 8 illustrates this idea.

We detail the construction of $\tilde{E}$ for a FEF $E \in \mathcal{L}$. As a consequence of Lemma 28, we can build the FEF $\diamondsuit = \bigoplus_{E_1,E_2\in\mathcal{L}}(\bar{E}_1 \bullet \hat{E}_2)$ of underlying flow F and domain L^2. Since F is sweeping, we can also build $\clubsuit = \langle \diamondsuit, L\rangle^{\circledast}$ of domain $L^{\geq 2}$. Observe that $\clubsuit$ does not depend on E, so it is computed once.

By Lemmas 26-2 and 26-3, for all FEFs $E_1, E_2 \in \mathcal{L}$, $\bar{E}_1$ and $\bar{E}_2$ have the same underlying flow. They also have disjoint domains because E_1 and E_2 have disjoint domains (by definition of labels). Thus we can build the FEF $\bigoplus_{E'\in\mathcal{L}} \bar{E}'$ of domain L. We extend it to a FEF $\spadesuit$ with domain $L^{\geq 2}$, by replacing each of its Reg-expressions f with $\mathrm{dom}(E)/\varepsilon \bullet (L^*/\varepsilon) \bullet f$. Again, note that $\spadesuit$ does not depend on E.

Finally, the FEF $\hat{E}$ is also extended to a FEF $\heartsuit_E$ with domain $\mathrm{dom}(E)L^+$ by replacing each of its Reg-expressions f with $f \bullet (L^+/\varepsilon)$.

Lemma 29. *For all $u \in \mathrm{dom}(E)L^+$, we have $\heartsuit_E(u) \circ \clubsuit(u) \circ \spadesuit(u) = \phi_E(u)$.*

Proof. Let $u \in \mathrm{dom}(E)L^+$. We recall that $\mathcal{L}$ is unambiguously iterable and that all FEFs of $\mathcal{L}$ have disjoint domains (by definition of labels). Then, there exist a

unique decomposition $u_0u_1\cdots u_n$ of u and a unique sequence $(E_1,\dots,E_n)$ of FEFs of $\mathcal{L}$ such that $u_0 \in \mathrm{dom}(E)$ and $u_i \in \mathrm{dom}(u_i)$ for all $i \in \{1,\dots,n\}$. It follows that $\heartsuit_E(u_0\cdots u_n) = \hat{E}(u_0)$, $\spadesuit(u_0\cdots u_n) = \bigoplus_{E'\in\mathcal{L}} \bar{E}'(u_n) = \bar{E}_n(u_n)$ and

$$\begin{aligned}\clubsuit(u_0\cdots u_n) &= \langle \diamondsuit, L\rangle^{\circledast}(u_0\cdots u_n)\\ &= \diamondsuit(u_0u_1)\circ\diamondsuit(u_1u_2)\circ\cdots\circ\diamondsuit(u_{n-1}u_n)\\ &= (\bar{E}\bullet\hat{E}_1)(u_0u_1)\circ(\bar{E}_1\bullet\hat{E}_2)(u_1u_2)\circ\cdots\circ(\bar{E}_{n-1}\bullet\hat{E}_n)(u_{n-1}u_n)\\ &= \bar{E}(u_0)\circ\hat{E}_1(u_1)\circ\bar{E}_1(u_1)\circ\hat{E}_2(u_2)\circ\cdots\circ\bar{E}_{n-1}(u_{n-1})\circ\hat{E}_n(u_n)\ .\end{aligned}$$

Then the result follows Lemma 27. □

We recall that $\heartsuit_E$, $\clubsuit$ and $\spadesuit$ have the same number of crossing edges as $\mathrm{flow}(E)$. In addition, $\mathrm{flow}(\heartsuit_E)$ (resp. $\mathrm{flow}(\spadesuit)$) has the same left return edges (resp. right return edges) as $\mathrm{flow}(E)$. We combine them to construct $\tilde{E}$ as follows:

- the flow of $\tilde{E}$ is the same as $\mathrm{flow}(E)$,
- the left return edges of $\tilde{E}$ are those of $\heartsuit_E$,
- the right return edges of $\tilde{E}$ are those of $\spadesuit$,
- the k-th LR-crossing edge of $\tilde{E}$ is labelled by the Reg-expression $f_\clubsuit \otimes f_\spadesuit$ where $f_\clubsuit$ and $f_\spadesuit$ are the labels of the k-th LR-crossing edges of $\clubsuit$ and $\spadesuit$, respectively,
- the k-th RL-crossing edge of $\tilde{E}$ is labelled by the Reg-expression $g_\spadesuit \otimes g_\clubsuit$ where $g_\clubsuit$ and $g_\spadesuit$ are the labels of the k-th RL-crossing edges of $\clubsuit$ and $\spadesuit$, respectively.

Proposition 30. *The domain of $\tilde{E}$ is $\mathrm{dom}(E)L^+$. Furthermore, for all $u \in \mathrm{dom}(\tilde{E})$, we have $\tilde{E}(u) = \phi_E(u)$.*

Proof. The domain of $\tilde{E}$ is $\mathrm{dom}(E)L^+$ since it is the domain of all its Reg-expressions (we recall that the domain of $f \otimes g$ is $\mathrm{dom}(f)\cap\mathrm{dom}(g)$).

Let $u \in \mathrm{dom}(\tilde{E})$. By Lemma 29, it suffices to show that the word flows $\tilde{E}(u)$ and $\heartsuit_E(u)\circ\clubsuit(u)\circ\spadesuit(u)$ are identical to prove that $\tilde{E}(u) = \phi_E(u)$. The left and right return edges of $\tilde{E}(u)$ are trivially the same as those of $\heartsuit_E(u)\circ\clubsuit(u)\circ\spadesuit(u)$. Suppose that $f_\clubsuit$ and $f_\spadesuit$ are the labels of the k-th LR-crossing edges of $\clubsuit$ and $\spadesuit$, respectively. Then the k-th LR-crossing edge of $\tilde{E}(u)$ is labelled with the word $(f_\clubsuit \otimes f_\spadesuit)(u) = f_\clubsuit(u)f_\spadesuit(u)$. Since $\heartsuit_E$ has no right return edge and $\spadesuit$ has no left return edge, the word labelling the k-th LR-crossing edge of $\heartsuit_E(u)\circ\clubsuit(u)\circ\spadesuit(u)$ is simply the concatenation of the words labelling the k-th LR-crossing edges of $\heartsuit_E(u)$, $\clubsuit(u)$ and $\spadesuit(u)$. This word is also equal to $f_\clubsuit(u)f_\spadesuit(u)$ because all crossing edges of $\heartsuit_E$ are labelled with $\mathrm{dom}(E)/\varepsilon \bullet L^+/\varepsilon$. Similar arguments show that the k-th RL-crossing edges of $\tilde{E}(u)$ and $\heartsuit_E(u)\circ\clubsuit(u)\circ\spadesuit(u)$ carry the same word. □

5.3. *Application to an example*

Example 31. *We consider the FEF G of Figure 6. Since $\mathrm{flow}(G\bullet G) = G$, the label $\{G\}$ is simple. The Kleene plus of $\{G\}$ is the label $\{G\}\oplus\{\tilde{G}\}$. Since G and*

$\tilde{G}$ have the same underlying flow, this label is equivalent to the label $\{G \oplus \tilde{G}\}$. The two crossing edges in $(G \oplus \tilde{G})_{LR}$ are labelled with the Reg-expression

$$\{a,b\}/\varepsilon \oplus \left(\langle \{a,b\}/\varepsilon \bullet \{a,b\}/\varepsilon, \{a,b\}\rangle^{\circledast} \otimes (\{a,b\}/\varepsilon \bullet \{a,b\}^*/\varepsilon \bullet \{a,b\}/\varepsilon)\right)$$

which is equivalent to $\{a,b\}^+/\varepsilon$. The crossing edge in $(G \oplus \tilde{G})_{RL}$ is labelled with

$$\begin{array}{c} (\{a\}/a \oplus \{b\}/b) \\ \oplus \\ \left(\begin{array}{c} (\{a,b\}/\varepsilon \bullet \{a,b\}^*/\varepsilon \bullet (\{a\}/a \oplus \{b\}/b)) \\ \otimes \\ \left\langle (\{a\}/a \oplus \{b\}/b) \overset{\leftarrow}{\bullet} \{a,b\}/\varepsilon, \{a,b\} \right\rangle^{\overset{\leftarrow}{\circledast}} \end{array} \right) \end{array}$$

which is equivalent to mirror_{A^+} with $A = \{a,b\}$.

6. A Brzozowski and McCluskey-like Algorithm for FEFA

In this section, we design a Brzozowski and McCluskey-like algorithm that takes as input an unambiguous FEFA $\mathcal{A}$ and returns a label $\mathcal{L}$ semantically equivalent to $\mathcal{A}$. This algorithm, called BMC-FEFA, can easily be used to turn any unambiguous 2NFT $\mathcal{T}$ into an equivalent regular function expression f in the following way:

(1) From $\mathcal{T}$, build an equivalent pruned unambiguous $(1,1)$-FEFA $\mathcal{A}$ (Lemma 11).
(2) Apply BMC-FEFA to $\mathcal{A}$. We get as output an equivalent label $\mathcal{L}$ (Theorem 34).
(3) Since $\mathcal{A}$ is a $(1,1)$-FEFA, all FEFs in $\mathcal{L}$ have size $(1,1)$ and have disjoint domains. So the FEF $F_{\text{out}} = \oplus_{F \in \mathcal{L}} F$ is well defined and has size $(1,1)$. It is trivially equivalent to $\mathcal{L}$ (using the definition of the sum $\oplus$).
(4) The (1,1)-FEF F_{out} consists of one crossing edge only. It carries the expected regular function expression f equivalent to $\mathcal{T}$.

6.1. *A first incomplete algorithm*

We first present an adaptation of Brzozowski and McCluskey-like algorithm to FEFA when self-loops always carry simple labels. It takes as input a *normalised* pruned unambiguous (n,m)-FEFA $\mathcal{A}$. In addition, it takes also as input a state-elimination strategy to guide the algorithm. The algorithm ouputs a label equivalent to $\mathcal{A}$. By definition, this algorithm I-BMC-FEFA (see Algorithm 1) only needs to compute the Kleene plus operator for simple labels by using the construction KleenePlus of Section 5. The soundness of this theorem immediately follows from Proposition 25. The overall algorithm is simply a loop over states of the FEFA, and hence necessarily terminates. We can thus state the following theorem:

Theorem 32. *Algorithm I-BMC-FEFA always terminates. If it does not end with* `abort`*, then the label it returns is equivalent to the FEFA taken as input.*

Algorithm 1 Incomplete BMC Algorithm for FEFA

Input: a normalised (n, m)-FEFA $\mathcal{A}$ with initial state ι and final state β, and a state-elimination strategy ν.

Output: a label that defines the same function as $\mathcal{A}$

function I-BMC-FEFA($\mathcal{A}, \nu$)
  for state q in $Q \setminus \{\iota, \beta\}$ picked w.r.t. ν **do**
    Let write transitions $q \xrightarrow{\mathcal{L}_2} q$
    if $\mathcal{L}_2$ is not simple **then**
      `abort`
    else
      $\mathcal{L}_2^+ = \text{KleenePlus}(\mathcal{L}_2)$
    end if
    for states q_1, q_2 in $Q \setminus \{q\}$ **do**
      Let write transitions $q_1 \xrightarrow{\mathcal{L}_1} q$, $q \xrightarrow{\mathcal{L}_3} q_2$, $q_1 \xrightarrow{\mathcal{L}_4} q_2$
      Add transition $q_1 \xrightarrow{\mathcal{L}} q_2$ with $\mathcal{L} = \mathcal{L}_1\mathcal{L}_2 + \mathcal{L}_1\mathcal{L}_2^+\mathcal{L}_3 + \mathcal{L}_4$.
    end for
    Remove state q and all incident transitions
  end for
  Let write transition $\iota \xrightarrow{\mathcal{L}} \beta$
  return $\mathcal{L}$
end function

6.2. *The complete BMC-FEFA algorithm*

To complete the I-BMC-FEFA algorithm, we need to compute the Kleene plus for non simple labels $\mathcal{L}$. The intuitive idea is to use the previously presented I-BMC-FEFA algorithm on a suitable FEFA $\mathcal{A}^{\mathcal{L}}_{\text{Unf}}$ that describes the same semantics as $\mathcal{L}^{\geq 2}$. By suitable, we mean that there exists a state-elimination strategy $\nu^{\mathcal{L}}$ of $\mathcal{A}^{\mathcal{L}}_{\text{Unf}}$ such that the I-BMC-FEFA algorithm ends without `abort`. The complete BMC algorithm for FEFA is described in Algorithm 2. It is similar to the I-BMC-FEFA algorithm except that the instruction `abort` is replaced with the instructions required to compute the Kleene plus of a non simple label.

We present now how to build the FEFA $\mathcal{A}^{\mathcal{L}}_{\text{Unf}}$ and how to choose the state-elimination strategy $\nu^{\mathcal{L}}$ when $\mathcal{L}$ is an arbitrary label of size (m, m) with unambiguously iterable domain $L = \text{dom}(\mathcal{L})$. We first consider an unambiguous $(m, m)-$FEFA $\mathcal{A}^{\mathcal{L}}$ 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 (m, m)-FEFA $\mathcal{A}^{\mathcal{L}}_{\text{Unf}} = (Q_{\text{Unf}}, \iota, \beta, \longrightarrow_{\text{Unf}})$ of $\mathcal{A}^{\mathcal{L}}$ equivalent to $\mathcal{A}^{\mathcal{L}}$, together with a state-elimination strategy ν such that the I-BMC-FEFA algorithm ends without `abort`.

Let $2N + 1$ be the greatest number of crossing edges among the FEFs of $\mathcal{L}$. By Lemma 3, every FEF finitely generated by concatenation of FEFs in $\mathcal{L}$ has at most $2N + 1$ crossing edges. For each $i \in [N + 1]$, we let S_i be the set of underlying flows

Algorithm 2 Complete BMC Algorithm for FEFA

Input: a normalised (n, m)-FEFA $\mathcal{A}$ with initial state ι and final state β.
Output: a label that defines the same function as $\mathcal{A}$

function BMC-FEFA($\mathcal{A}$)
  for state q in $Q \setminus \{\iota, \beta\}$ **do**
    Let write transitions $q \xrightarrow{\mathcal{L}_2} q$
    if $\mathcal{L}_2$ is not simple **then**
      build $\mathcal{A}_{\mathrm{Unf}}^{\mathcal{L}_2}$ and $\nu^{\mathcal{L}_2}$
      $\mathcal{L}_2^+ = \mathcal{L}_2 \oplus$ I-BMC-FEFA$(\mathcal{A}_{\mathrm{Unf}}^{\mathcal{L}_2}, \nu^{\mathcal{L}_2})$
    else
      $\mathcal{L}_2^+ = \mathrm{KleenePlus}(\mathcal{L}_2)$
    end if
    for states q_1, q_2 in $Q \setminus \{q\}$ **do**
      Let write transitions $q_1 \xrightarrow{\mathcal{L}_1} q$, $q \xrightarrow{\mathcal{L}_3} q_2$, $q_1 \xrightarrow{\mathcal{L}_4} q_2$
      Add transition $q_1 \xrightarrow{\mathcal{L}} q_2$ with $\mathcal{L} = \mathcal{L}_1\mathcal{L}_2 + \mathcal{L}_1\mathcal{L}_2^+\mathcal{L}_3 + \mathcal{L}_4$.
    end for
    Remove state q and all incident transitions
  end for
  Let write transition $\iota \xrightarrow{\mathcal{L}} \beta$
  return $\mathcal{L}$
end function

with $2i+1$ crossing edges finitely generated by $\mathcal{L}$ with concatenation of FEFs. We endow the set $\mathbb{F}$ of flows with an additional identity element $\top$ with respect to the flow composition, and define the set of states as $Q_{\mathrm{Unf}} = \{\beta\} \cup \prod_{i\in 0,\dots,N}(S_i \cup \{\top\})$. The initial state is set to $\iota = (\top)_{0\le i\le N}$. For $j \le N$, we define Q_j as the set of all states $(s_0, \dots, s_j, \top, \dots, \top) \in Q_{\mathrm{Unf}}$ with $s_j \neq \top$. Given a state $q \in \bigcup_j Q_j$, we denote by level(q) the unique j such that $q \in Q_j$, and we say that j is the *level* of state q. Intuitively, a state $q = (s_i)_{0\le i\le N}$ of level k keeps partial information about its accessibility: for all $j \in [k]$, all executions in $\mathcal{A}_{\mathrm{Unf}}^{\mathcal{L}}$ from a state of level j to state q using inner states of level strictly greater than j produce word flows of flow $F = s_{j+1} \circ \cdots \circ s_N$. The transition function updates this information. It uses an alphabet that consists of subsets of $\mathcal{L}$: For all $q = (s_0, \dots, s_N) \in Q_{\mathrm{Unf}} - \{\beta\}$, $j \in [N+1]$ and $F \in \bigcup_{i\in[N+1]} S_i$, we define $\mathcal{L}_{q,j,F}$ as the set of FEFs $E \in \mathcal{L}$ such that $F = s_j \circ s_{j+1} \circ \cdots \circ s_N \circ \mathrm{flow}(E) \in S_j$ and for all $i > j$, $s_i \circ s_{i+1} \circ \cdots \circ s_N \circ \mathrm{flow}(E) \notin S_i$. Formally the transition function $\longrightarrow_{\mathrm{Unf}}$ is defined as follows:

- $q \xrightarrow{\mathcal{L}}_{\mathrm{Unf}} \beta$ for all $q \in Q_{\mathrm{Unf}} - \{\iota, \beta\}$;
- let $q = (s_0, \dots, s_N) \in Q_{\mathrm{Unf}} - \{\beta\}$, $j \le N$ and $F \in \bigcup_{i\in[N+1]} S_i$. If $\mathcal{L}_{q,j,F} \neq \emptyset$, then $q \xrightarrow{\mathcal{L}_{q,j,F}}_{\mathrm{Unf}} (s_i')_{0\le i\le N}$, with $s_j' = F$, $s_i' = s_i$ if $i < j$, and $s_i' = \top$ if $i > j$.

Note that all labels of $\mathcal{A}_{\mathrm{Unf}}^{\mathcal{L}}$ have size (m, m) because $\mathcal{L}$ does. So, we consistently

set $\mathsf{size}(q) = m$ for all $q \in Q_{\mathrm{Unf}}$.

An example of such an unfolding is depicted in Figure 9.

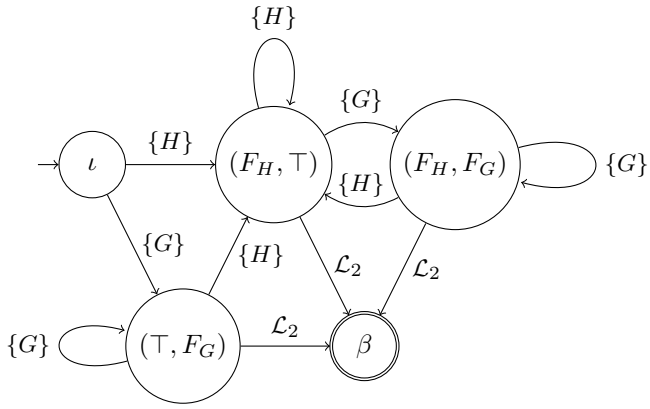

Fig. 9. The unfolding of the label $\mathcal{L}_2 = \{G, H\}$ of Figure 6. F_H and F_G stand for flow(H) and flow(G) respectively. All self-loops carry simple labels.

Lemma 33. *$\mathcal{A}^{\mathcal{L}}_{Unf}$ is unambiguous and defines the same function as $\mathcal{L}^{\geq 2}$.*

Proof. We start with two facts.

Fact 1: For every sequence $(E_0, \ldots, E_n)$ of FEFs of $\mathcal{L}$ ($n \geq 2$), we can associate exactly one accepting execution $(q_i \xrightarrow{\mathcal{L}_i} q_{i+1})_{0 \leq i \leq n}$ of $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ such that $E_i \in \mathcal{L}_i$ for all $i \in [n+1]$. This is because for each fixed state $q \in Q_{\mathrm{Unf}} - \{\beta\}$, the non-empty subsets $\mathcal{L}_{q,j,F}$'s of $\mathcal{L}$ form a partition of $\mathcal{L}$.

Fact 2: There exist for each word $u \in L^{\geq 2}$ a unique sequence $(E_0, \ldots, E_n)$ of FEFs of $\mathcal{L}$ and a unique decomposition of u into $u = u_0 \cdots u_n$ such that $u_i \in \mathrm{dom}(E_i)$ for all $i \in [n+1]$. This is because L is unambiguously iterable and all FEFs of $\mathcal{L}$ have disjoint domains.

We prove that $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ is unambiguous and define the same function as $\mathcal{L}^{\geq 2}$. Clearly, the domain of $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ is included in $\mathrm{dom}(\mathcal{L}^{\geq 2})$ because every label in $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ is a subset of $\mathcal{L}$. Let $u \in \mathrm{dom}(\mathcal{L}^{\geq 2})$. By fact 2, there exist a unique sequence $(E_0, \ldots, E_n)$ of FEFs of $\mathcal{L}$ and a unique decomposition of u into $u = u_0 \cdots u_n$ such that $u_i \in \mathrm{dom}(E_i)$ for all $i \in [n+1]$. It follows from Fact 1 that there is exactly one accepting execution $\sigma = (q_i \xrightarrow{\mathcal{L}_i} q_{i+1})_{0 \leq i \leq n}$ of $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ such that $E_i \in \mathcal{L}_i$ for all $i \in [n+1]$. Altogether, this means that σ is an accepting execution on u, and the only one : $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}(u)$ is defined and $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ is unambiguous. Besides, since all the FEFs in a label have disjoint domains, we have :

$$\mathcal{L}^{\geq 2}(u) = \mathcal{L}(u_0) \circ \cdots \circ \mathcal{L}(u_n) = E_0(u_0) \circ \cdots \circ E_n(u_n) \qquad \text{and}$$
$$\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}(u) = \mathcal{L}_1(u_0) \circ \cdots \circ \mathcal{L}_n(u_n) = E_0(u_0) \circ \cdots \circ E_n(u_n).$$

So $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ and $\mathcal{L}^{\geq 2}$ define the same function. □

Finally, we use a preprocessing for removing the inaccessible and co-inaccessible states of $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$, and we choose for the state-elimination strategy $\nu^{\mathcal{L}}$ anyone that successively eliminates states in Q_j, for j ranging from N to 0.

Theorem 34. *Algorithm BMC-FEFA always terminates and returns a label equivalent to the FEFA taken as input.*

Proof. The soundness of the algorithm follows from Lemma 33 and Theorem 32. Its completeness is proved in Subsection 6.3. □

6.3. *Completeness of the BMC-FEFA algorithm*

To prove the completeness of the BMC-FEFA algorithm, it suffices to prove that the I-BMC-FEFA algorithm never aborts while processing. This is actually the case because, as we will show in the sequel, removing states of $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ by *decreasing* level is a strategy that ensures that all self-loops considered when processing I-BMC-FEFA have a simple label.

We write $q_0 \overset{\mathcal{L}_\sigma}{\rightsquigarrow} q_n$ to denote that there exists an execution $(q_i \xrightarrow{\mathcal{L}_i} q_{i+1})_{0\leq i<n}$ of $\mathcal{A}^{L}_{\mathrm{Unf}}$ with $\mathcal{L}_\sigma = \mathcal{L}_0 \bullet \cdots \bullet \mathcal{L}_{n-1}$. Let $j \geqslant 0$. If moreover the inner states have level greater than j, namely if $\mathrm{level}(q_i) \geqslant j$ for every $i \in \{1, \ldots, n-1\}$, then we write $q_0 \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q_n$. Similarly, we write $q_0 \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q_n$ if $\mathrm{level}(q_i) > j$ for every $i \in \{1, \ldots, n-1\}$. At last, we define the length of an execution as its number of transitions. Lemma 35 specifies the type of information that a state carries. The first property corresponds to the intuition given when defining the unfolding automaton $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$.

Lemma 35. *Let $q = (s_i)_{0\leq i\leq N}$ in Q_j and $q' = (s'_i)_{0\leq i\leq N}$ in Q_k such that $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$. The following properties hold:*

(1) If $k > j$ then

(a) for all $i \leq j$, $s'_i = s_i$
(b) for all $E \in \mathcal{L}_\sigma$, $\mathrm{flow}(E) = s'_{j+1} \circ \cdots \circ s'_N$

(2) If $k = j$ then for all $E \in \mathcal{L}_\sigma$, $s_j \circ \mathrm{flow}(E) = s'_j$ and $\mathrm{flow}(E)$ has $2j+1$ crossing edges.

Proof. *Item 1:* We prove statement by induction on the length l of the execution from q. Consider the base case where $l = 1$. Then the execution $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$ is a transition $q \xrightarrow{\mathcal{L}_{q,k,F}}_{\mathrm{Unf}} q'$ for some flow F. The property (a) follows from the definition

of $\longrightarrow_{\mathrm{Unf}}$. Property (b) holds because for all $E \in \mathcal{L}_{q,k,F}$, we have that

$$\begin{aligned}
\mathrm{flow}(E) &= s_k \circ \cdots \circ s_N \circ \mathrm{flow}(E) && (s_i = \top \text{ for all } i > j \text{ because } q \in Q_j)\\
&= F && (\text{definition of } \mathcal{L}_{q,k,F})\\
&= s_{j+1} \circ \cdots \circ s_{k-1} \circ F && (s_i = \top \text{ for all } i > j \text{ because } q \in Q_j)\\
&= s'_{j+1} \circ \cdots \circ s'_k && (\text{definition of } \longrightarrow_{\mathrm{Unf}})\\
&= s'_{j+1} \circ \cdots \circ s'_N && (s'_i = \top \text{ for all } i > k \text{ because } q' \in Q_k)
\end{aligned}$$

Suppose that $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$ is an execution of length $l+1$. Then it can be decomposed into $q \overset{\mathcal{L}'}{\rightsquigarrow}_{>j} q'' \overset{\mathcal{L}_{q'',k,F}}{\longrightarrow}_{\mathrm{Unf}} q'$ where $q \overset{\mathcal{L}'}{\rightsquigarrow}_{>j} q''$ is an execution of length l with $q'' = (s''_0, \ldots, s''_N) \in Q_r$ for some $r > j$ (because of the definition of $\rightsquigarrow_{>j}$).

We start with property (a). By induction, for all $i \leq j$, $s''_i = s_i$. By definition of $\longrightarrow_{\mathrm{Unf}}$, we have $s''_i = s'_i$ for all $i < k$. The property follows from the hypothesis $k > j$.

We turn to property (b). Consider $E' \in \mathcal{L}'$ and $E \in \mathcal{L}_{q'',k,F}$. We have:

$$\begin{aligned}
\mathrm{flow}(E' \bullet E) &= \mathrm{flow}(E') \circ \mathrm{flow}(E)\\
&= s''_{j+1} \circ \cdots \circ s''_N \circ \mathrm{flow}(E) && (\text{induction property})\\
&= s''_{j+1} \circ \cdots \circ s''_{k-1} \circ s''_k \circ \cdots \circ s''_N \circ \mathrm{flow}(E)\\
&= s''_{j+1} \circ \cdots \circ s''_{k-1} \circ F && (\text{definition of } \mathcal{L}_{q'',k,F})\\
&= s'_{j+1} \circ \cdots \circ s'_{k-1} \circ s'_k && (\text{definition of } \longrightarrow_{\mathrm{Unf}})\\
&= s'_{j+1} \circ \cdots \circ s'_{k-1} \circ s'_k \circ \cdots \circ s'_N && (s'_i = \top \text{ for all } i > k)\\
&= s'_{j+1} \circ \cdots \circ s'_N
\end{aligned}$$

Item 2: There are two cases. If the execution is of length 1, *i.e.* if $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$ is equal to $q \overset{\mathcal{L}_\sigma}{\longrightarrow}_{\mathrm{Unf}} q'$, then the properties follow from the definition of $\longrightarrow_{\mathrm{Unf}}$.

Otherwise $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$ can be decomposed into $q \overset{\mathcal{L}'}{\rightsquigarrow}_{>j} q'' \overset{\mathcal{L}_{q'',k,F}}{\longrightarrow}_{\mathrm{Unf}} q'$ with $q'' = (s''_0, \ldots, s''_N) \in Q_r$ for some $r > j$.

Consider $E' \in \mathcal{L}'$ and $E \in \mathcal{L}_{q'',k,F}$. We have:

$$\begin{aligned}
s_j \circ \mathrm{flow}(E' \bullet E) &= s_j \circ \mathrm{flow}(E') \circ \mathrm{flow}(E)\\
&= s''_j \circ \mathrm{flow}(E') \circ \mathrm{flow}(E) && (\text{Lemma 35-(1).(a) on } q \overset{\mathcal{L}'}{\rightsquigarrow}_{>j} q'')\\
&= s''_j \circ s''_{j+1} \circ \cdots \circ s''_N \circ \mathrm{flow}(E) && (\text{Lemma 35-(1).(b) on } q \overset{\mathcal{L}'}{\rightsquigarrow}_{>j} q'')\\
&= F && (\text{definition of } \mathcal{L}_{q'',k,F})\\
&= s'_j && (\text{definition of } \longrightarrow_{\mathrm{Unf}}, q' \in Q_j)
\end{aligned}$$

It remains to prove that $\mathrm{flow}(E' \bullet E)$ has exactly $2j+1$ crossing edges. As shown above, we have $s'_j = s''_j \circ \mathrm{flow}(E' \bullet E)$. By definition, $s'_j \in S_j$ has $2j+1$ crossing edges. Lemma 3 entails that $\mathrm{flow}(E' \bullet E)$ has at least $2j+1$ crossing edges.

Conversely, we proceed by contradiction and assume that there exists $m > j$ such that $\mathrm{flow}(E' \bullet E)$ has $2m+1$ crossing edges. We have already established earlier in the proof that $\mathrm{flow}(E' \bullet E) = s''_{j+1} \circ \cdots \circ s''_N \circ \mathrm{flow}(E)$. The assumption, together with Lemma 3, entails that $s''_l = \top$ for all $j < l < m$, and then

$$\mathrm{flow}(E' \bullet E) = s''_m \circ \cdots \circ s''_N \circ \mathrm{flow}(E) \in S_m.$$

Thus, there exists a greatest integer $g > j$ such that $s''_g \circ \cdots \circ s''_N \circ \mathrm{flow}(E) \in S_g$. Using this fact, we can deduce from the definition of $\mathcal{L}_{q'',k,F}$ that $k = g$ and $F = s''_g \circ \cdots \circ s''_N \circ \mathrm{flow}(E) \in S_g$. Since we have supposed that $q'' \xrightarrow{\mathcal{L}_{q'',k,F}}_{\mathrm{Unf}} q'$, we get also that $s'_g = F$ and $s'_l = \top$ for all $l > g$. In other words, we get that $q' \in Q_g \neq Q_j$, which contradicts the hypothesis. This concludes the proof. □

Now we can prove a slightly more general version of Lemma 35-(2) where the inner states of an execution have level greater than or *equal to* j.

Lemma 36. *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 $E \in \mathcal{L}_\sigma$*,* $s_j \circ \mathrm{flow}(E) = s'_j$
(2) $\mathrm{flow}(E)$ *has* $2j + 1$ *crossing edges.*

Proof.

By induction on the length l of $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q'$. If $l = 1$, then $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q'$ consists of one transition and no inner state. So we have also $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{>j} q'$, and the result holds from Lemma 35-(2).

Let $l > 1$. We suppose that at least one intermediate state q'' belongs to Q_j, otherwise the result holds by Lemma 35-(2). Then $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q'$ can be split into $q \overset{\mathcal{L}'}{\rightsquigarrow}_{\geqslant j} q'' \overset{\mathcal{L}''}{\rightsquigarrow}_{>j} q'$ where $q'' \overset{\mathcal{L}''}{\rightsquigarrow}_{>j} q'$ is possibly one transition of $\longrightarrow_{\mathrm{Unf}}$. By induction hypothesis applied on $q \overset{\mathcal{L}'}{\rightsquigarrow}_{\geqslant j} q''$ and by Lemma 35-(2) applied on $q'' \overset{\mathcal{L}''}{\rightsquigarrow}_{>j} q'$, we have, for all $E' \in \mathcal{L}'$ and $E'' \in \mathcal{L}''$:

- $s_j \circ \mathrm{flow}(E') = s''_j$ and $\mathrm{flow}(E')$ has $2j + 1$ crossing edges;
- $s''_j \circ \mathrm{flow}(E'') = s'_j$ and $\mathrm{flow}(E'')$ has $2j + 1$ crossing edges.

Altogether, $s'_j = s_j \circ \mathrm{flow}(E') \circ \mathrm{flow}(E'') = s_j \circ \mathrm{flow}(E' \bullet E'')$, which has $2j+1$ crossing edges (because s'_j belongs to S_j). By Lemma 3, this means that $\mathrm{flow}(E' \bullet E'')$ has exactly $2j + 1$ crossing edges. □

Consider now two loops $q \overset{\mathcal{L}_\sigma}{\rightsquigarrow}_{\geqslant j} q$ and $q \overset{\mathcal{L}_\beta}{\rightsquigarrow}_{\geqslant j} q$ around state $q = (s_i)_{0\leq i\leq N} \in Q_j$. Since $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ is unambiguous, the domains of $\mathcal{L}_\sigma$ and $\mathcal{L}_\beta$ are disjoint, and their sum is possible. Lemma 37 shows that $\mathcal{L}_\sigma \oplus \mathcal{L}_\beta$ is a simple label.

Lemma 37. *Let* $j \geq 0$ *and* $q \in Q_j$*. If* $\{\mathcal{L}_1, \ldots, \mathcal{L}_k\}$ *is a set of labels such that* $q \overset{\mathcal{L}_i}{\rightsquigarrow}_{\geqslant j} q$ *for all* $i \in \{1, \ldots, k\}$*, then* $\oplus_{i\in\{1,\ldots,k\}} \mathcal{L}_i$ *is a simple label.*

Proof. We have to prove that for all $E, E' \in \bigcup_{i\in\{1,\ldots,k\}} \mathcal{L}_i$, we have $\mathrm{flow}(E \bullet E') = \mathrm{flow}(E)$. Lemma 36 implies the following properties:

(1) $s_j \circ \mathrm{flow}(E) = s_j$ and $s_j \circ \mathrm{flow}(E') = s_j$;
(2) $s_j \circ \mathrm{flow}(E \bullet E') = s_j$ (direct consequence of item 1);
(3) $\mathrm{flow}(E)$ and $\mathrm{flow}(E')$ have exactly $2j + 1$ crossing edges.

From the second and third items, together with Lemma 3, we deduce that flow($E \bullet E'$) has exactly $2j+1$ crossing edges (we recall that $q \in Q_j$, which means that s_j is distinct from $\top$ and has $2j+1$ crossing edges). In particular, this implies that none of the crossing edges of flow(E) and flow(E') are "used" into a return edge in flow($E \bullet E'$). As a consequence, flow(E) and flow($E \bullet E'$) have exactly the same left return edges. Last, we can deduce from the first item that flow(E) and flow(E') have the same right return edges (the ones of s_j). This entails the expected equality. ∎

Proposition 38. *Let $\mathcal{L}$ be a label. The I-BMC-FEFA algorithm applied to $\mathcal{A}^{\mathcal{L}}_{Unf}$ and $\nu^{\mathcal{L}}$ always ends without* `abort`*.*

Proof. The state-elimination strategy $\nu^{\mathcal{L}}$ successively eliminates states in Q_j, for j ranging from N to 0. As we observed earlier in Lemma 26, when $\mathcal{L}_x$ is a simple label, the flows associated with $\mathcal{L}_x^+$ are exactly those of $\mathcal{L}_x$. Consider the removal of a self-loop around some state $q \in Q_j$ at some point of the process. The flows appearing in its label are obtained as paths around q in $\mathcal{A}^{\mathcal{L}}_{\mathrm{Unf}}$ that only go through states in $\bigcup_{k \geqslant j} Q_k$. This observation, together with Lemma 37, ensures that we always remove self-loops with simple labels. Hence the I-BMC-FEFA algorithm ends without `abort`. □

6.4. *A complete example*

We consider the transducer of Figure 1 realizing the function mirror^* of Example 2. As already mentioned, the crossing sequence automaton construction produces the $(1,1)$-FEFA of Figure 6 that defines the same function. We start from this FEFA and apply a first instance of the BMC algorithm to it. According to this algorithm, we need to compute the Kleene plus of $\mathcal{L}_2 = \{G, H\}$.

However $\mathcal{L}_2$ is not simple. So we consider its unfolding $\mathcal{A}_{\mathrm{Unf}}$ of Figure 9. It defines the same function as $\mathcal{L}_2^{\geq 2}$. The states $(\top, F_G)$ and (F_H, F_G) have level 1, whereas $(F_H, \top)$ has level 0. We now apply instances of the simple BMC algorithm to $\mathcal{A}_{\mathrm{Unf}}$ to remove states $(\top, F_G)$, (F_H, F_G) and $(F_H, \top)$ in this order. In what follows, A denotes the set $\{a, b\}$. For readability, we write cancel_L instead of L/ε.

Elimination of $(\top, F_G)$. We recall that the label $\{G\}$ is simple and that its Kleene plus has already been computed in Example 31. Then the state $(\top, F_G)$ can be removed using the Kleene plus over simple labels. According to BMC algorithm, this yields two new transitions.

The first one, from ι to β, carries the label

$$(\{G\} \bullet \mathcal{L}_2) \oplus (\{G\} \bullet \{G\}^+ \bullet \mathcal{L}_2).$$

Making the sum of the FEFs with same underlying flow leads to the equivalent label $\{G_1, H_1\}$ where: G_1 has flow F_G, domain $A^{\geq 2}$, two LR-edges labelled by $\mathsf{cancel}_{A^{\geq 2}}$ and one RL-edge labelled by $\mathsf{mirror}_{A^{\geq 2}}$; H_1 has flow F_H, domain $A^+\{\#\}$, one LL-

edge labelled by $\mathsf{mirror}_{A^+} \bullet \mathsf{cancel}_{\{\#\}}$, one LR-edge labelled by $\mathsf{cancel}_{A^+} \bullet \mathsf{id}_{\{\#\}}$, and one RR-edge labelled by $\mathsf{cancel}_{A^+\{\#\}}$.

The second transition, from ι to $(F_H, \top)$, carries a label

$$\{H\} \oplus (\{G\} \bullet \{H\}) \oplus (\{G\} \bullet \{G\}^+ \bullet \{H\})$$

(note that the label $\{H\}$ at the beginning comes from the already present transition from ι to $(F_H, \top)$ in $\mathcal{A}_{\mathrm{Unf}}$). All FEFs of this expression have flow F_H. Making the sum of these FEFs, we get an equivalent label $\{H_2\}$ of flow F_H. H_2 has domain $A^*\{\#\}$, one LL-edge labelled by $\mathsf{mirror}_{A^*} \bullet \mathsf{cancel}_{\{\#\}}$, one LR-edge labelled by $\mathsf{cancel}_{A^*} \bullet \mathsf{id}_{\{\#\}}$, and one RR-edge labelled by $\mathsf{cancel}_{A^*\{\#\}}$.

Elimination of (F_H, F_G). Again, this yields two transitions. The first one is a self loop around $(F_H, \top)$ that also carries the label $\{H_2\}$. The second one, from $(F_H, \top)$ to β, carries a label $\mathcal{L}_2 \oplus \{G_1, H_1\}$, equivalent to $\{G_2, H_2\}$ where the FEF G_2 has flow F_G, domain A^+, two LR-edges labelled by cancel_{A^+} and one RL-edge labelled by mirror_{A^+}.

Elimination of $(F_H, \top)$. As expected, the self-loop around $(F_H, \top)$ carries a simple label (namely, $\{H_2\}$ of underlying flow F_H). The Kleene star $\{H_2\}^+$ consists of the FEFs H_2 and $\tilde{H}_2$. The FEF $\tilde{H}_2$ has flow F_H, domain $(A^*\{\#\})^{\geq 2}$, one LL-edge labelled by $\mathsf{mirror}_{A^*} \bullet \mathsf{cancel}_{\{\#\}(A^*\{\#\})^+}$, one RR-edge labelled by $\mathsf{cancel}_{(A^*\{\#\})^{\geq 2}}$, and one LR-edge labelled by

$$\tilde{f} = \left\langle \mathsf{cancel}_{A^*} \bullet \mathsf{id}_{\{\#\}} \bullet \mathsf{mirror}_{A^*} \bullet \mathsf{cancel}_{\{\#\}}, A^*\{\#\} \right\rangle^{\circledast}.$$

Then the elimination of $(F_H, \top)$ induces one transition from ι to β labelled by

$$\{G_1, H_1\} \oplus (\{H_2\} \bullet \{G_2, H_2\}) \oplus (\{H_2\} \bullet \{H_2\}^+ \bullet \{G_2, H_2\}).$$

This new label corresponds to $\mathcal{L}_2^{\geq 2}$. It contains 8 FEFs:

G_1 of domain $A^{\geq 2}$,
H_1 of domain $A^+\{\#\}$,
$H_2 \bullet G_2$ of domain $A^*\{\#\}A^+$,
$H_2 \bullet H_2$ of domain $(A^*\{\#\})^2$,
$H_2 \bullet H_2 \bullet G_2$ of domain $(A^*\{\#\})^2A^+$,
$H_2 \bullet H_2 \bullet H_2$ of domain $(A^*\{\#\})^3$,
$H_2 \bullet \tilde{H}_2 \bullet G_2$ of domain $(A^*\{\#\})^{\geq 3}A^+$,
$H_2 \bullet \tilde{H}_2 \bullet H_2$ of domain $(A^*\{\#\})^{\geq 4}$.

So $\mathcal{L}_2^{\geq 2}$ has domain $\{a, b, \#\}^{\geq 2}$ as expected.

Finally, come back to the $(1,1)$-FEFA of Figure 6 and the first instance of BMC algorithm. We get one transition, from the inital state to the final one, labelled by $\mathcal{L} = (\mathcal{L}_1 \bullet \mathcal{L}_3) \oplus (\mathcal{L}_1 \bullet \mathcal{L}_2 \bullet \mathcal{L}_3) \oplus (\mathcal{L}_1 \bullet \mathcal{L}_2^{\geq 2} \bullet \mathcal{L}_3)$. This label consists of 11 FEFs of size $(1,1)$. Each FEF is labelled by only one *REG*-expression. The sum of these expressions is the output result. It defines the function mirror^*. For instance, after some simplifications, the *REG*-expression labelling the FEF $E_{\vdash} \bullet H_2 \bullet \tilde{H}_2 \bullet H_2 \bullet E_{\dashv} \in$

$\mathcal{L}$ looks like

$$\begin{array}{c}\mathsf{cancel}_{\{\vdash\}} \bullet \mathsf{mirror}_{A^*} \bullet \\ \Big(\big(\mathsf{id}_{\{\#\}} \bullet \mathsf{mirror}_{A^*} \bullet \mathsf{cancel}_{\{\#\}(A^*\{\#\})^+}\big) \otimes \Big(\mathsf{cancel}_{\{\#\}} \bullet \tilde{f}\Big)\Big) \\ \bullet \mathsf{mirror}_{A^*} \bullet \mathsf{id}_{\{\#\}} \bullet \mathsf{cancel}_{\{\dashv\}}\end{array}$$

which is equivalent to mirror^* on domain $\{\vdash\}(A^*\{\#\})^{\geq 4}\{\dashv\}$.

7. The case of sweeping transducers

We now consider the case of sweeping transducers, namely transducers whose head can change direction only at the extremities $\vdash$ and $\dashv$ of the input. We introduce the two following grammars for function expressions:

$$\begin{array}{l}\overleftarrow{Rat} \ni f, g ::= L/v \mid f \oplus g \mid f \overleftarrow{\bullet} g \mid f^{\overleftarrow{*}} \\ Had(Rat, \overleftarrow{Rat}) \ni h ::= f \mid h \otimes f \quad \text{where } f \in Rat \cup \overleftarrow{Rat} \\ SW \ni s ::= h \mid s \oplus h \quad \text{where } h \in Had(Rat, \overleftarrow{Rat})\end{array}$$

where L is a regular language over A and $v \in B^*$.

Theorem 39. *Sweeping functions are equivalent to SW-expressions.*

Proof. We first show that any SW definable function is sweeping-definable (*i.e.* can be realised by a sweeping transducer). An expression built using the grammar SW is a finite sum of finite Hadamard products of rational and reverse-rational functions (*i.e.* generated using the grammar $\overleftarrow{Rat}$).

Rational and reverse-rational functions are clearly sweeping-definable, so are finite Hadamard products of them, using the sweeping property (we will do as many traversals of the input word as needed by the size of the finite Hadamard product). Last, the external finite sum is on disjoint domains, so remains sweeping-definable.

Conversely, let us show that any sweeping-definable function can be expressed by a regular function expression in the grammar SW. The proof is based on a Brzozowski & McCluskey algorithm. We observe that the (trimmed) crossing sequence automaton of a sweeping transducer has a very particular form. Indeed, there is a guess, on the first transition, of the number of passes k realized by the simulated accepting run. The initial flow has a single edge in F_{LR}, and $\lfloor k/2 \rfloor$ edges in F_{RR}. Then, all the flows encountered along the run correspond to sweeping FEFs: they have $\lceil k/2 \rceil$ edges in F_{LR} and $\lfloor k/2 \rfloor$ edges in F_{RL}. The flow of the last transition is as follows: a single edge in F_{LR}, and $\lfloor k/2 \rfloor$ edges in F_{LL}.

If we consider this crossing sequence automaton as a finite state automaton on a new alphabet (composed of FEFs with a single letter domain), we can apply to it the "classical" Brzozowski & McCluskey algorithm. The result is a regular expression on the new alphabet. We can actually evaluate the Kleene star in this setting, as all the FEFs it is applied to are sweeping FEFs, and in this case one can simply apply the Kleene star to each edge of the FEF. Following the above observation on the

shape of the flows of the automaton, one ends up with a regular function expression belonging to the grammar SW. □

8. 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 [14, 4], studying infinite words [2, 12] and identifying star-free expressions for first-order definable transformations [18, 15, 9] for instance.

Acknowledgments

We would like to thank the anonymous reviewers of this and the DLT'2018 submission for their valuable comments.

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

References

[1] R. Alur and P. Černý. Expressiveness of streaming string transducers. In *FSTTCS*, volume 8 of *LIPIcs.*, pages 1–12. Schloss Dagstuhl. Leibniz-Zent. Inform., 2010.
[2] R. Alur, E. Filiot, and A. Trivedi. Regular transformations of infinite strings. In *LICS*, pages 65–74, 2012.
[3] R. Alur, A. Freilich, and M. Raghothaman. Regular combinators for string transformations. In *CSL-LICS '14*, pages 9:1–9:10. ACM, 2014.
[4] F. Baschenis, O. Gauwin, A. Muscholl, and G. Puppis. Untwisting two-way transducers in elementary time. In *LICS'17*, pages 1–12. IEEE Computer Society, 2017.
[5] N. Baudru and P. Reynier. From two-way transducers to regular function expressions. In M. Hoshi and S. Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2018.
[6] J. Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979.
[7] M. Bojanczyk. Transducers with origin information. In *ICALP 2014*, volume 8573 of *LNCS*, pages 26–37. Springer, 2014.
[8] J. A. Brzozowski and E. J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. Electronic Computers*, 12(2):67–76, 1963.
[9] O. Carton and L. Dartois. Aperiodic two-way transducers and FO-transductions. In *CSL*, volume 41 of *LIPIcs*, pages 160–174. Schloss Dagstuhl. Leibniz-Zent. Inform., 2015.
[10] C. Choffrut and B. Guillon. An algebraic characterization of unary two-way transducers. In *MFCS*, volume 8634 of *LNCS*, pages 196–207. Springer, 2014.
[11] B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoret. Comput. Sci.*, 126(1):53–75, 1994.

[12] V. Dave, P. Gastin, and S. N. Krishna. Regular transducer expressions for regular transformations. In *LICS*, pages 315–324. IEEE Computer Society, 2018.
[13] J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.
[14] E. Filiot, O. Gauwin, P.-A. Reynier, and F. Servais. From two-way to one-way finite state transducers. In *LICS*, pages 468–477. IEEE Computer Society, 2013.
[15] E. Filiot, S. N. Krishna, and A. Trivedi. First-order definable string transformations. In *FSTTCS*, volume 29 of *LIPIcs*, pages 147–159. Schloss Dagstuhl. Leibniz-Zent. Inf., 2014.
[16] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
[17] S. Lombardy. Two-way representations and weighted automata. *RAIRO - Theor. Inf. and Applic.*, 50(4):331–350, 2016.
[18] R. McNaughton and S. Papert. *Counter-free automata*. The M.I.T. Press, Cambridge, Mass.-London, 1971.
[19] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. Electron. Comput.*, 9(1):39–47, 1960.
[20] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
[21] I. Simon. Factorization forests of finite height. *Theor. Comput. Sci.*, 72(1):65–94, 1990.