

Pebble transducers with unary output

Gaëtan Douéneau-Tabot ✉

IRIF, Université Paris Cité & Direction générale de l'armement - Ingénierie de projets

Abstract

Bojańczyk recently initiated an intensive study of deterministic pebble transducers, which are two-way automata that can drop marks (named "pebbles") on their input word, and produce an output word. They describe functions from words to words. Two natural restrictions of this definition have been investigated: marble transducers by Douéneau-Tabot et al., and comparison-free pebble transducers (that we rename here "blind transducers") by Nguyễn et al.

Here, we study the decidability of membership problems between the classes of functions computed by pebble, marble and blind transducers that produce a unary output. First, we show that pebble and marble transducers have the same expressive power when the outputs are unary (which is false over non-unary outputs). Then, we characterize 1-pebble transducers with unary output that describe a function computable by a blind transducer, and show that the membership problem is decidable. These results can be interpreted in terms of automated simplification of programs.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases polyregular functions, pebble transducers, marble transducers, streaming string transducers, factorization forests

Digital Object Identifier 10.4230/LIPIcs...XX

Acknowledgements The author is grateful to Olivier Carton for discussing about this work. He also thanks the reviewers of MFCS 2021 for their helpful comments and remarks.

1 Introduction

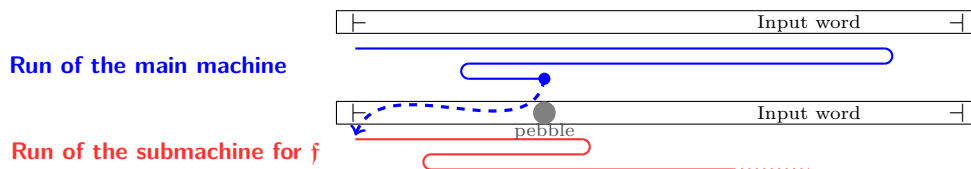
Regular languages can be described by several models such as deterministic, non-deterministic, or two-way (the reading head can move in two directions) finite automata [12]. A natural extension consists in adding an output mechanism to finite automata. Such machines, called *transducers*, describe functions from words to words (or relations when non-deterministic) and provide a natural way to model simple programs that produce outputs. The particular model of a *two-way transducer* consists in a two-way automaton enhanced with an output function. It describes the class of *regular functions* which has been intensively studied for its fundamental properties: closure under composition [5], logical characterization by monadic second-order transductions [7], decidable equivalence problem [9], etc.

Pebble transducers and their variants. The model of k -pebble transducer can be defined as an inductive extension of two-way transducers. A 0-pebble transducer is just a two-way transducer. For $k \geq 1$, a k -pebble transducer \mathcal{T} is a two-way transducer that, when in a given configuration, can "call" an *external function* f , computed by some $(k-1)$ -pebble transducer. \mathcal{T} gives as argument to f its input word together with a mark, named "pebble", on the position from which the call was performed, and uses the output of f within its own output.

The behavior of a 1-pebble transducer is depicted in Figure 1. Intuitively, a k -pebble transducer is some recursive program whose recursion depth is at most $k+1$. Equivalently, it can be seen as an iterative algorithm with "two-way for-loops", such that the maximal depth of nested loops is $k+1$. A k -pebble transducer can only produce an output whose length is polynomial in its input's length, more precisely $\mathcal{O}(n^{k+1})$ when n is the input's length (this is intuitive from the "nested loops" point of view). The functions computed by a k -pebble transducer for some $k \geq 0$ are thus called *polyregular functions* [3]. Several



properties of polyregular functions have been investigated: closure under composition [3], logical characterization by monadic second-order interpretations [4], etc. The equivalence problem (given two machines, do they compute the same function?) is however still open.



■ **Figure 1** Behavior of a 1-pebble transducer

Recently, two natural restrictions of pebble transducers have been introduced. First, the *k-marble transducers* of [6] only give as argument to their external function the prefix of the input word which ends in the calling position (see Figure 4). Second, the *k-blind transducers*¹ of [11] give the whole input word, but no pebble on the calling position (see Figure 3). The classes of functions they compute are strict subclasses of polyregular functions [6, 11].

Class membership problems. These various models of transducers raise several membership problems: given a function computed by a machine of model X , can it be computed by some machine of model Y ? When Y is a restriction of X , this problem reformulates as a program optimization question: given a "complex" algorithm in a class X , can we build an equivalent "simpler" one in class Y ? Thus it is of a foremost interest in practice.

Given a function f computed by an ℓ -pebble transducer, one can ask whether it is computable by a k -pebble transducer for a given $k < \ell$. The problem is open, but it is solved in the case of marble [6] and blind [11] transducers and it turns out that a necessary and sufficient condition for this membership is that $|f(w)| = \mathcal{O}(|w|^{k+1})$. Using the "nested loops" interpretation of pebble transducers, it means that an output of size $\mathcal{O}(|w|^{k+1})$ can always be produced with at most $k+1$ nested loops.

Contributions. In this paper, we study a different membership problem: can a function given by a k -pebble transducer be computed by a k -marble or k -blind transducer? It turns out to be a more difficult question, since there is no intuitive and machine-independent candidate for a membership condition (such as the size of the output). In general, membership problems for transducers are difficult, since contrary to regular languages, there is no "canonical" object known to represent a regular function. Hence, there can be several seemingly unrelated manners to produce the same function, and moving from one to another can be technical.

We focus on *transducers whose output alphabet is unary*, and our proof techniques are new. The first main result is that (when the outputs are unary) k -pebble transducers and k -marble transducers compute the same functions (one direction is obvious since k -marble is a restriction of k -pebble). The transformation is effective, but the way of producing the output must sometimes be completely modified (the transformation modifies the *origin semantics*, in the sense of [2]), which creates an additional difficulty. The correspondence fails as soon as the output is not over a unary alphabet, as detailed in Example 1.1.

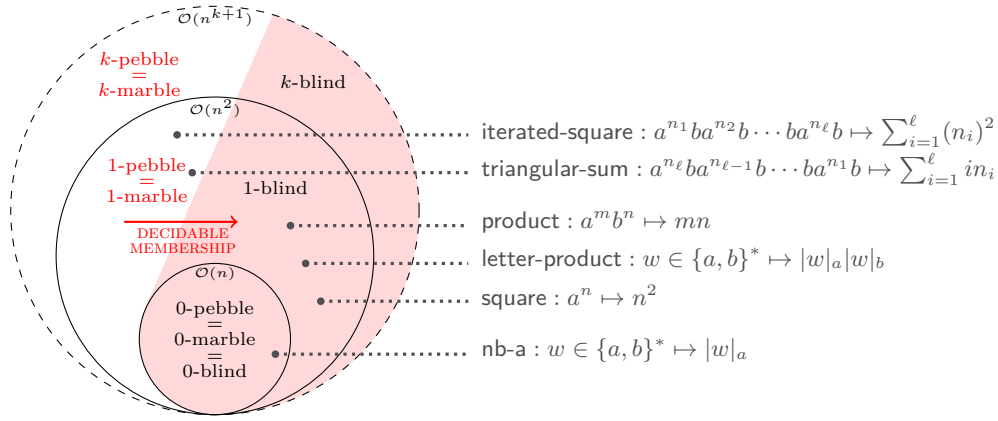
¹ The original terminology of [11] is *comparison-free pebble transducers*, but we strongly believe that the term "blind" is more adapted, since there are no pebbles in this model.

► **Example 1.1** ([6, 11]). The partial function $\{a, b\}^* \rightarrow \{a, b\}^*, a^m b^n \mapsto (b^n a)^m$ can be computed by a 1-pebble transducer, but not by a k -marble for any $k \geq 0$.

Since the equivalence problem is decidable for marble transducers, it follows from our result that it is also decidable for pebble transducers with unary output.

As a second main result, we show how to decide (when the outputs are unary) whether a function given by 1-pebble (\equiv 1-marble) transducer can be computed by a 1-blind transducer, or more generally by a k -blind transducer for some $k \geq 0$. The technical proof also gives a syntactical characterization of 1-marble transducers whose function verify this property: it describes a kind of "symmetry" in the production of the machine on its input. Furthermore, the conversion is effective when possible, but once more the manner of producing the output can be strongly modified. Our techniques heavily rely on the theory of *factorization forests*; this is, to our knowledge, the first time this notion is used for membership problems of transducers, and we believe this approach to be fruitful.

Our results are summarized in red in Figure 2. We also give some examples of functions (their outputs are non-negative integers, since we identify $\{a\}^*$ with \mathbb{N}).



■ **Figure 2** Classes of functions with unary output and results of this paper

Outline. We first recall in Section 2 the definitions of k -pebble, k -marble and k -blind transducers, simplified for the case of unary outputs. In Section 3, we define the notions of k -pebble, k -marble and k -blind bimachines, and show their equivalence with the transducer models. Bimachines are easier to handle in the proofs, due to the fact that they avoid two-way moves. In Section 4, we show that k -marble and k -pebble transducers are equivalent. Finally, we solve in Section 5 the class membership problem from 1-pebble to 1-blind. Due to space constraints, several proofs are sketched in the main paper, and we focus on the most significant lemmas and characterizations.

2 Preliminaries

\mathbb{N} is the set of nonnegative integers. If $0 \leq i \leq j$, the set $[i:j]$ denotes $\{i, i+1, \dots, j\} \subseteq \mathbb{N}$ (empty if $j < i$). Capital letters A, B denote finite sets of letters (alphabets). The empty word is denoted ε . If $w \in A^*$, let $|w| \in \mathbb{N}$ be its length, and for $1 \leq i \leq |w|$ let $w[i]$ be its i -th letter. If $I = \{i_1 < \dots < i_\ell\} \subseteq \{1, \dots, |w|\}$, let $w[I] := w[i_1] \dots w[i_\ell]$. If $a \in A$, we denote by $|w|_a$ the number of letters a occurring in w . Given $A = \{a, \dots\}$, let $\underline{A} := \{\underline{a}, \dots\}$ be a

XX:4 Pebble transducers with unary output

disjoint copy of A . For $1 \leq i \leq |w|$, we define $w \uparrow i := w[1:i-1] \underline{w[i]} w[i+1:|w|]$ as " w in which position i is underlined". We assume that the reader is familiar with the basics of automata theory, in particular the notion of two-way deterministic automaton.

Two-way transducers. A deterministic two-way transducer is a deterministic two-way automaton enhanced with the ability to produce outputs along its run. The class of functions described by these machines is known as *regular functions* [5, 7].

► **Definition 2.1.** A (deterministic) two-way transducer $(A, B, Q, q_0, F, \delta, \lambda)$ is:

- an input alphabet A and an output alphabet B ;
- a finite set of states Q with an initial state $q_0 \in Q$ and a set $F \subseteq Q$ of final states;
- a (partial) transition function $\delta : Q \times (A \uplus \{\vdash, \dashv\}) \rightarrow Q \times \{\triangleleft, \triangleright\}$;
- a (partial) output function $\lambda : Q \times (A \uplus \{\vdash, \dashv\}) \rightarrow B^*$ with same domain as δ .

When given as input a word $w \in A^*$, the two-way transducer disposes of a read-only input tape containing $\vdash w \dashv$. The marks \vdash and \dashv are used to detect the borders of the tape, by convention we denote them as positions 0 and $|w|+1$ of w . Formally, a *configuration* over $\vdash w \dashv$ is a tuple (q, i) where $q \in Q$ is the current state and $0 \leq i \leq |w|+1$ is the position of the reading head. The *transition relation* \rightarrow is defined as follows. Given a configuration (q, i) , let $(q', \star) := \delta(q, w[i])$. Then $(q, i) \rightarrow (q', i')$ whenever either $\star = \triangleleft$ and $i' = i - 1$ (move left), or $\star = \triangleright$ and $i' = i + 1$ (move right), with $0 \leq i' \leq |w|+1$. A *run* is a sequence of configurations $(q_1, i_1) \rightarrow \dots \rightarrow (q_n, i_n)$. Accepting runs are those that begin in $(q_0, 0)$ and end in a configuration of the form $(q, |w|+1)$ with $q \in F$ (and it never visits such a configuration before). The function $f : A^* \rightarrow B^*$ computed by the machine is defined as follows. Let $w \in A^*$, if there exists an accepting run on $\vdash w \dashv$, then $f(w)$ is the concatenation of the $\lambda(q, w[i])$ along this unique run on $\vdash w \dashv$. To make f a total function, we let $f(w) := \varepsilon$ if there is no accepting run (the language of words having an accepting run in a two-way transducer is regular [12], hence the domain does not matter).

► **Example 2.2.** $\text{reverse} : A^* \rightarrow A^*, abac \mapsto caba$ can be computed by a two-way transducer.

From now on, the output alphabet of the machines will always be a singleton. Up to identifying $\{a\}^*$ and \mathbb{N} , we assume that $\lambda : Q \times (A \uplus \{\vdash, \dashv\}) \rightarrow \mathbb{N}$ and $f : A^* \rightarrow \mathbb{N}$.

External functions. We now extend the notion of output function λ : it will not give directly an integer, but performs a call to an *external function* which returns an integer. For pebbles, the output of the external functions depends on the input word and the current position.

► **Definition 2.3.** A two-way transducer with external pebble functions $(A, Q, q_0, F, \delta, \mathfrak{F}, \lambda)$ is:

- an input alphabet A ;
- a finite set of states Q with an initial state $q_0 \in Q$ and a set $F \subseteq Q$ of final states;
- a (partial) transition function $\delta : Q \times (A \uplus \{\vdash, \dashv\}) \rightarrow Q \times \{\triangleleft, \triangleright\}$;
- a finite set \mathfrak{F} of external functions $\mathfrak{f} : (A \uplus \underline{A})^* \rightarrow \mathbb{N}$;
- a (partial) output function $\lambda : Q \times (A \uplus \{\vdash, \dashv\}) \rightarrow \mathfrak{F}$ with same domain as δ .

Configurations (q, i) and runs of two-way transducers with external functions are defined as for classical two-way transducers. The function $f : A^* \rightarrow \mathbb{N}$ computed by the machine is defined as follows. Let $w \in A^*$ such that there exists an accepting run on $\vdash w \dashv$. If $\lambda(q, w[i]) = \mathfrak{f} \in \mathfrak{F}$, we let $\nu(q, i) := \mathfrak{f}(w \uparrow i)$, that is the result of \mathfrak{f} applied to w marked in i . Finally, $f(w)$ is defined as the sum of the $\nu(q, i)$ along this unique accepting run on $\vdash w \dashv$. We similarly set $f(w) = 0$ if there is no accepting run.

► **Remark 2.4.** If the external functions are constant, we exactly have a two-way transducer.

► **Example 2.5.** Let $a, b \in A$, $f_b : w \in (A \uplus \underline{A})^* \mapsto |w|_b$ and $f_0 : w \in (A \uplus \underline{A})^* \mapsto 0$. The two-way transducer with external pebble functions, which makes a single pass on its input and calls f_b if reading a and f_0 otherwise, computes **letter-product** : $w \in A \mapsto |w|_a |w|_b$.

We define two other models. Their definition is nearly the same, except that the external functions of \mathfrak{F} have type $A^* \rightarrow \mathbb{N}$ and $\nu(q, i)$ is defined in a slightly different way:

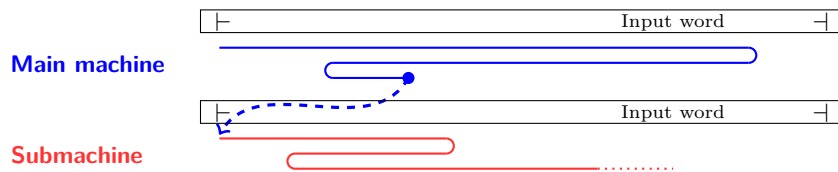
- in a *two-way transducer with external blind functions*, we define $\nu(q, i) := f(w)$. The external function is applied to w without marking the current position;
- in a *two-way transducer with external marble functions*, we define $\nu(q, i) := f(w[1:i])$. The external function is applied to the prefix of w stopping at the current position.

Pebble, blind and marble transducers. We now describe the transducer models using the formalism of external functions. These are not the original definitions from [3, 6, 11], but the correspondence is straightforward, as soon as we know that pebble automata can only recognize regular languages.

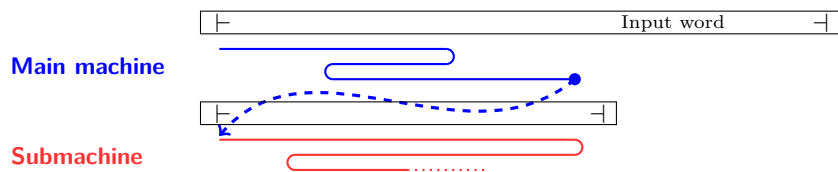
► **Definition 2.6.** For $k \geq 0$, a k -pebble (resp. k -blind, k -marble) transducer is:

- if $k = 0$, a two-way transducer;
- if $k \geq 1$, a two-way transducer with external pebble (resp. blind, marble) functions that are computed by $(k-1)$ -pebble (resp. $(k-1)$ -blind, $(k-1)$ -marble) transducers.

The intuitive behavior of a 1-pebble transducer is depicted in Figure 1 in Introduction. We draw in Figure 3 the behavior of a 1-blind transducer, which is the same except that the calling position is not marked for the machine computing the external function.



■ **Figure 3** Behavior of a 1-blind transducer



■ **Figure 4** Behavior of a 1-marble transducer

► **Example 2.7.** By restricting the functions f_b and f_0 of Example 2.5 to A^* , we see that **letter-product** : $w \in A^* \mapsto |w|_a |w|_b$ can be computed by a 1-blind transducer.

The intuitive behavior of a 1-marble transducer is depicted in Figure 4.

► **Example 2.8.** The function **letter-product** : $w \in A^* \mapsto |w|_a |w|_b$ can be computed by a 1-marble transducer as follows. Assume that $a \neq b$, let $f_a : w \mapsto |w|_a$ and $f_b : w \mapsto |w|_b$. The machine calls f_b when reading a and f_a when reading b . This way, each a is "counted" $|w|_b$

XX:6 Pebble transducers with unary output

times (from the call of f_b starting in this a which computes all the b before it, plus each time it is seen in a call of f_a starting from some b after this a).

The strategy for computing letter-product is really different between Examples 2.7 and 2.8. This illustrates the difficulty to obtain a "canonical" form for a transduction.

3 From two-way transducers to bimachines

Since we consider a commutative output monoid, the order in which the production is performed does not matter. It is thus tempting to simplify a two-way transducer in a one-way machine which visits each position only once. This is exactly what we do with bimachines, with the subtlety that they are able to check regular properties of the prefix (resp. suffix) starting (resp. ending) in the current position. From now on, we consider only total functions of type $A^+ \rightarrow \mathbb{N}$ (the output on ε can be treated separately and does not matter).

► **Definition 3.1.** A bimachine with external pebble functions $(A, M, \mu, \mathfrak{F}, \lambda)$ consists of:

- an input alphabet A ;
- a morphism into a finite monoid $\mu : A^* \rightarrow M$;
- a finite set \mathfrak{F} of external functions $f : (A \uplus A)^+ \rightarrow \mathbb{N}$;
- a total output function $\lambda : M \times A \times M \rightarrow \mathfrak{F}$.

Given $1 \leq i \leq |w|$ a position of $w \in A^*$, let $f_i := \lambda(\mu(w[1:i-1]), w[i], \mu(w[i+1:|w|])) \in \mathfrak{F}$. The bimachine defines a function $f : A^+ \rightarrow \mathbb{N}$ as follows:

$$f(w) := \sum_{1 \leq i \leq |w|} f_i(w \uparrow i).$$

As before, we can define *bimachines with external blind* (resp. *marble*) functions (in this case we have $f_i : A^+ \rightarrow \mathbb{N}$). We then let:

$$f(w) := \sum_{1 \leq i \leq |w|} f_i(w) \left(\text{resp. } f(w) := \sum_{1 \leq i \leq |w|} f_i(w[1:i]) \right).$$

As for two-way transducers, we define bimachines (without external functions) by setting $\lambda : M \times A \times M \rightarrow \mathbb{N}$. Equivalently, it corresponds to bimachines with external constant functions $f : w \mapsto n$. Going further, we define k -pebble bimachines by induction.

► **Definition 3.2.** For $k \geq 0$, a k -pebble (resp. k -blind, k -marble) bimachine is:

- if $k = 0$, a bimachine (without external functions);
- if $k \geq 1$, a bimachine with external pebble (resp. blind, marble) functions which are computed by $(k-1)$ -pebble (resp $(k-1)$ -blind, $(k-1)$ -marble) bimachines.

► **Example 3.3.** The function **triangular-sum** : $a^{n_\ell} b a^{n_{\ell-1}} b \dots b a^{n_1} b \mapsto \sum_{i=1}^{\ell} i n_i$ can be computed by a 1-marble bimachine. It uses the singleton monoid $M = \{1_M\}$ and the morphism $\mu : a, b \mapsto 1_M$ in all its bimachines. The output function of the main bimachine is defined by $\lambda(1_M, a, 1_M) := f_a$ and $\lambda(1_M, b, 1_M) := f_b$. For computing $f_a : w \mapsto 0$ we use output $\lambda_{f_a}(1_M, a, 1_M) := 0$ and $\lambda_{f_a}(1_M, b, 1_M) := 0$, and for computing $f_b : w \mapsto |w|_a$ we use output $\lambda_{f_b}(1_M, a, 1_M) := 1$ and $\lambda_{f_b}(1_M, b, 1_M) := 0$.

Standard proof techniques allow to relate bimachines and transducers.

► **Proposition 3.4.** k -pebble (resp. k -blind, k -marble) bimachines and k -pebble (resp. k -blind, k -marble) transducers compute the same functions, and both conversions are effective.

Proof sketch. Both directions are treated by induction. From bimachines to transducers, we show that a bimachine with external pebble functions can be transformed in an equivalent two-way transducer with the same external pebble functions (we use a *lookaround* [7] to simulate μ). From transducers to bimachines, the induction step shows that a two-way transducer with external pebble functions can be transformed in an equivalent bimachine with external pebble functions, by adapting the classical reduction from two-way to one-way automata [12]. However the new external functions can be linear combinations of the former ones, since we produce "all at once" the results of several visits in a position. We only need to use a finite number of combinations, since in its accepting runs, a two-way transducer can only visit each position a bounded number of times. ◀

4 Equivalence between k -pebble and k -marble transducers

The main goal of this section is to show equivalence between k -pebble and k -marble transducers, over unary outputs. We shall use another model which is equivalent to marble transducers [6]: a *streaming string transducer (with unary output)*, which consists in a deterministic automaton with a finite set \mathfrak{X} of registers that store integers. At each letter read, the values of the registers are updated by doing a linear combination of their former values, whose coefficients depend on the current state of the automaton. In our definition we focus on the registers and forget about the states, which corresponds to a weighted automaton over the semiring $(\mathbb{N}, +, \times)$ (it is shown in [6] that both models are equivalent). The update is represented by a matrix from $\mathbb{N}^{\mathfrak{X} \times \mathfrak{X}}$, which is chosen depending on the letter read.

► **Definition 4.1.** A streaming string transducer (SST) $\mathcal{T} = (A, \mathfrak{X}, I, T, F)$ is:

- an input alphabet A and a finite set \mathfrak{X} of registers;
- an initial row vector $I \in \mathbb{N}^{\mathfrak{X}}$;
- a register update function $T : A \rightarrow \mathbb{N}^{\mathfrak{X} \times \mathfrak{X}}$;
- an output column vector $F \in \mathbb{N}^{\mathfrak{X}}$.

T can be extended as a monoid morphism from A^* to $(\mathbb{N}^{\mathfrak{X} \times \mathfrak{X}}, \times)$. Given $w \in A^*$, the vector $IT(w)$ intuitively describes the values of the registers after reading w . To define the function $f : A^* \rightarrow \mathbb{N}$ computed by \mathcal{T} , we combine these values by the output vector:

$$f(w) := IT(w)F.$$

► **Example 4.2.** The function *triangular-sum* : $a^{n_\ell} b a^{n_{\ell-1}} b \cdots b a^{n_1} b \mapsto \sum_{i=1}^{\ell} in_i$ can be computed by an SST. We use two registers x, y and allow constants in the updates for more readability: x is initialized to 0 and updated $x \leftarrow x + 1$ on a and $x \leftarrow x$ on b , and y is initialized to 0 and updated $y \leftarrow y$ on a and $y \leftarrow y + x$ on b . Finally we output y .

We are now ready to state the main results of this section.

► **Theorem 4.3.** Given a k -pebble bimachine, one can build an equivalent SST.

The proof is done by induction on $k \geq 0$. Consider a bimachine whose external functions are computed by $(k-1)$ -pebble bimachines. By hypothesis, we can compute these functions by SSTs. The induction step is shown by Lemma 4.4, which uses new proof techniques.

► **Lemma 4.4.** Given a bimachine with external pebble functions computed by SSTs, one can build an equivalent SST (with no external functions).

Proof idea. Let \mathcal{T} be the SST computing an external function f . On input $w \in A^+$, the bimachine calls f on several positions $1 \leq i_1 < \dots < i_\ell \leq |w|$, which induces executions of \mathcal{T} on $w \uparrow i_1, \dots, w \uparrow i_\ell$. These executions are very similar: they only differ when reading the marked letter. Thus we build an SST which computes "simultaneously" all these executions, by keeping track of the sum of the values of the registers of \mathcal{T} along them. \blacktriangleleft

As a consequence of Theorem 4.3, we obtain equivalence between pebbles and marbles over unary outputs. The result is false over non-unary output alphabets [6, 11]. We also relate these functions with those computed by SST, assuming that the output is bounded by a polynomial in the input's length.

► **Corollary 4.5.** *For all $k \geq 0$ and $f : A^* \rightarrow \mathbb{N}$, the following conditions are equivalent:*

1. *f is computable by a k -pebble transducer;*
2. *f is computable by a k -marble transducer;*
3. *f is computable by an SST and $f(w) = \mathcal{O}(|w|^{k+1})$.*

Furthermore the transformations are effective.

Proof. Clearly a k -pebble transducer can simulate a k -marble transducer, hence $2 \Rightarrow 1$. Let f be computed by a k -pebble transducer, we have $f(w) = \mathcal{O}(|w|^{k+1})$ and by Theorem 4.3 one can build an SST for f . Thus $1 \Rightarrow 3$. Finally $3 \Rightarrow 2$ is shown in [6]. \blacktriangleleft

Another important consequence is that we can decide equivalence of pebble transducers with unary output, since we can do so for marble transducers [6].

► **Corollary 4.6.** *One can decide if two pebble transducers compute the same function.*

This has been an open question since [3], and it is still open for generic output alphabets.

5 Deciding if 1-pebble is 1-blind

Since the equivalence between marbles and pebbles is established, we now compare 1-pebble (which are 1-marble) transducers with 1-blind transducers. It turns out that 1-pebble are strictly more expressive; the main goal of this section is to show Theorem 5.1.

► **Theorem 5.1 (Membership).** *One can decide if a function given by a 1-marble (or 1-pebble) transducer can be computed by a k -blind transducer for some $k \geq 0$. If this condition holds, one can build a 1-blind transducer which computes it.*

Let us fix a function $f : A^+ \rightarrow \mathbb{N}$ described by a 1-marble bimachine $\mathcal{T} = (A, M, \mu, \mathfrak{F}, \lambda)$. For $f \in \mathfrak{F}$, let $\mathcal{T}_f := (A, M, \mu, \lambda_f)$ be the bimachine which computes it. We enforce the morphism μ to be surjective (up to considering the co-restriction to its image) and the same in all machines (up to taking the product of all morphisms used). Our goal is to give a decidable condition on \mathcal{T} for f to be computable by a 1-blind transducer. For this purpose, we define the notion of *bitype*. Intuitively, it describes two disjoint factors in an input word, together with a finite abstraction of their "context".

Let $\Lambda := 3|M|$ (it will be justified by Theorem 5.8).

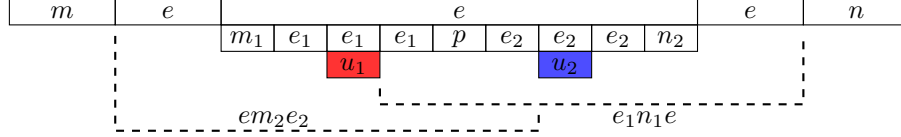
► **Definition 5.2.** A bitype $\Phi := m \langle \mathbf{u}_1 \rangle m' \langle \mathbf{u}_2 \rangle m''$ consists in $m, m', m'' \in M$, $\mathbf{u}_1, \mathbf{u}_2 \in A^+$.

We can define "the production performed in \mathbf{u}_1 by the calls from \mathbf{u}_2 ", in Φ . For $1 \leq i \leq |\mathbf{u}_1|$ and $1 \leq j \leq |\mathbf{u}_2|$, let $\Phi(i, j) := \lambda_{f_j}(m\mu(\mathbf{u}_1[1:i-1]), \mathbf{u}_1[i], \mu(\mathbf{u}_1[i+1:|\mathbf{u}_1|])m'\mu(\mathbf{u}_2[1:j])) \in \mathbb{N}$ where $f_j := \lambda(m\mu(\mathbf{u}_1)m'\mu(\mathbf{u}_2[1:j-1]), \mathbf{u}_2[j], \mu(\mathbf{u}_2[j+1:|\mathbf{u}_2|])m'')$. Then we set:

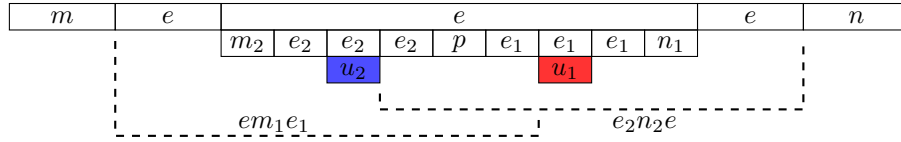
$$\text{prod}(m \langle \mathbf{u}_1 \rangle m' \langle \mathbf{u}_2 \rangle m'') := \sum_{\substack{1 \leq i \leq |\mathbf{u}_1| \\ 1 \leq j \leq |\mathbf{u}_2|}} \Phi(i, j) \in \mathbb{N}.$$

► **Definition 5.3.** The 1-marble bimachine \mathcal{T} is symmetrical whenever $\forall m, n, m_1, n_1, m_2, n_2 \in M$ and $u_1, u_2 \in A^+$ such that $|u_1|, |u_2| \leq 2^\Lambda$, $e_1 := \mu(u_1)$, $e_2 := \mu(u_2)$ and $e := m_1 e_1 n_1 = m_2 e_2 n_2$ are idempotents, there exists $K \geq 0$ such that $\forall p \in M$:

- if $m_1 e_1 p e_2 n_2 = e$, $e m_1 e_1 p e_2 = e m_2 e_2$ and $e_1 p e_2 n_2 e = e_1 n_1 e$,
then $\text{prod}(m e m_1 e_1 \langle u_1 \rangle e_1 p e_2 \langle u_2 \rangle e_2 n_2 e n) = K$;
- if $m_2 e_2 p e_1 n_1 = e$, $e m_2 e_2 p e_1 = e m_1 e_1$ and $e_2 p e_1 n_1 e = e_2 n_2 e$,
then $\text{prod}(m e m_2 e_2 \langle u_2 \rangle e_2 p e_1 \langle u_1 \rangle e_1 n_1 e n) = K$.



(a) Bitype $m e m_1 e_1 \langle u_1 \rangle e_1 p e_2 \langle u_2 \rangle e_2 n_2 e n$.



(b) Bitype $m e m_2 e_2 \langle u_2 \rangle e_2 p e_1 \langle u_1 \rangle e_1 n_1 e n$

■ **Figure 5** The bitypes used to define a symmetrical 1-marble bimachine

Symmetry means that, under some idempotent conditions, $\text{prod}(m \langle u_1 \rangle m' \langle u_2 \rangle m'')$ only depends on $m, m'', m' \mu(u_2) m''$ and $m \mu(u_1) m'$, that are the "contexts" of u_1 and u_2 , but not on the element m' which separates them. The same holds if we swap u_1 and u_2 . The bitypes considered in Definition 5.3 are depicted in Figure 5, together with the equations they satisfy.

Symmetry is the decidable condition we are looking for, as shown in Theorem 5.4. Recall that f is the function computed by the 1-marble bimachine \mathcal{T} .

► **Theorem 5.4 (Characterization).** The following conditions are equivalent:

1. f is computable by a k -blind transducer for some $k \geq 0$;
2. f is computable by a 1-blind transducer;
3. \mathcal{T} is symmetrical.

Theorem 5.1 follows from Theorem 5.4, since it suffices to check whether the machine is symmetrical, which can be decided by ranging over all monoid elements (including idempotents) and words of length at most 2^Λ .

► **Example 5.5.** Let us show that the bimachine of Example 3.3 computing triangular-sum is not symmetrical. Let $u_1 := a$, $u_2 := b$, $m, n, m_1, n_1, m_2, n_2, p, e = 1_M$, $e_1 := \mu(u_1) = 1_M$ and $e_2 := \mu(u_2) = 1_M$. Then $\text{prod}(m e m_1 e_1 \langle u_1 \rangle e_1 p e_2 \langle u_2 \rangle e_2 n_2 e n) = \text{prod}(1_M \langle a \rangle 1_M \langle b \rangle 1_M) = 1$ and $\text{prod}(m e m_2 e_2 \langle u_2 \rangle e_2 p e_1 \langle u_1 \rangle e_1 n_1 e n) = \text{prod}(1_M \langle b \rangle 1_M \langle a \rangle 1_M) = 0$. Furthermore the equations of Definition 5.3 hold, thus triangular-sum is not computable by a k -blind bimachine.

Lemma 5.6 shows $1 \Rightarrow 3$ in Theorem 5.4. It allows to show that some function cannot be computed by a k -marble transducer. Its proof is technical; a coarse intuition is that a 1-blind bimachine which makes a production on u_1 when called from u_2 cannot see the monoid element m' between u_1 and u_2 (since u_2 is not marked, its position is "forgotten").

► **Lemma 5.6.** If f is computable by a k -blind bimachine, then \mathcal{T} is symmetrical.

Since $2 \Rightarrow 1$ in Theorem 5.4 is obvious, it remains to show that if \mathcal{T} is symmetrical, then f is effectively computable by a 1-blind bimachine. This is the goal of the two following subsections. The main tool for the proof is the notion of factorization forest: using Lemma 5.17, it allows us to compute the function f without directly referring to a machine.

5.1 Factorization forests

Recall that $\mu : A^+ \rightarrow M$ is a fixed monoid morphism. A *factorization forest* [1] of $w \in A^+$ is an unranked tree structure which decomposes w following the image of its factors by μ .

- **Definition 5.7** ([13, 1]). A factorization (forest) of $w \in A^+$ is a tree defined as follows:
- if $w = a \in A$, it is a leaf a ;
 - if $|w| \geq 2$, then $(\mathcal{F}_1) \cdots (\mathcal{F}_n)$ is a factorization of w if each \mathcal{F}_i is a factorization of some $w_i \in A^+$ such that $w = w_1 \cdots w_n$, and either:
 - $n = 2$: the root is a binary node;
 - or $n \geq 3$ and $\mu(w_1) = \cdots = \mu(w_n)$ is an idempotent: the root is an idempotent node.

The set of factorizations over w is denoted $\text{Fact}(w)$. Recall that $\Lambda = 3|M|$.

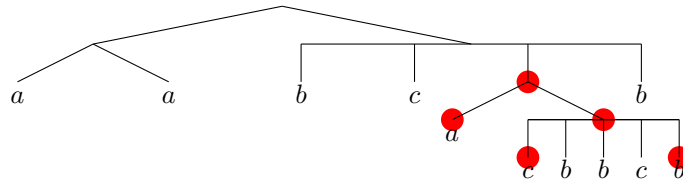
- **Theorem 5.8** ([13, 1]). For all $w \in A^+$, there is $\mathcal{F} \in \text{Fact}(w)$ of height at most Λ .

Let $\widehat{A} := A \uplus \{ (,) \}$. We have defined $\text{Fact}(w)$ as a set of tree structures, but we can assume that $\text{Fact}(w) \subseteq \widehat{A}^+$. Indeed, in Definition 5.7, a factorization of w can also be seen as "the word w with parentheses". There exists a rational function which computes factorizations, under this formalism. We reformulate this statement in Proposition 5.9 using a two-way transducer (which, exceptionally in this paper, has a non-unary output alphabet \widehat{A}).

- **Proposition 5.9** (Folklore). One can build a two-way transducer which computes a function $A^+ \rightarrow \widehat{A}^+, w \mapsto \mathcal{F} \in \text{Fact}(w)$ for some \mathcal{F} of height at most Λ .

We denote by $\text{Nodes}(\mathcal{F})$ the set of (idempotent or binary) nodes of \mathcal{F} . In order to simplify the statements, we identify a node with the subtree rooted in this node. Thus $\text{Nodes}(\mathcal{F})$ can also be seen as the set of subtrees of \mathcal{F} , and $\mathcal{F} \in \text{Nodes}(\mathcal{F})$. We shall use the standard tree vocabulary of "height" (a leaf is a tree of height 1), "parent node", "descendant" and "ancestor" (defined in a non-strict way: a node is itself one of its ancestors), "branch", etc.

- **Example 5.10.** Let $A = \{a, b, c\}$, $M = \{1_M, 2_M, 3_M\}$ with $2_M^2 = 1_M$, 3_M absorbing, $\mu(a) := 2_M$ and $\mu(b) := \mu(c) := 3_M$. Then $\mathcal{F} := (aa)(bc(a(cbbcb))b) \in \text{Fact}(aabcacbbcb)$ (we dropped the parens around single letters for more readability) is depicted in Figure 6. Idempotent nodes are drawn using a horizontal line.



■ **Figure 6** The factorization $(aa)(bc(a(cbbcb))b)$ of $aabcacbbcb$

We define $\text{Iterable-nodes}(\mathcal{F}) \subseteq \text{Nodes}(\mathcal{F})$ as the set of nodes which are the middle child of an idempotent node. Intuitively, such nodes can be copied without modifying their "context".

► **Definition 5.11.** Let $\mathcal{F} \in \text{Fact}(w)$, we define the set of iterable nodes of \mathcal{F} by induction:

- if $\mathcal{F} = a \in A$ is a leaf, $\text{Iterable-nodes}(\mathcal{F}) := \emptyset$;
- if $\mathcal{F} = (\mathcal{F}_1) \cdots (\mathcal{F}_n)$ is a binary or idempotent node, then:

$$\text{Iterable-nodes}(\mathcal{F}) := \{\mathcal{F}_i : 2 \leq i \leq n-1\} \uplus_{1 \leq i \leq n} \text{Iterable-nodes}(\mathcal{F}_i).$$

On the contrary, we now define sets of nodes which cannot be duplicated individually.

► **Definition 5.12.** Let $\mathcal{F} \in \text{Fact}(w)$, we define the dependency of \mathcal{F} as follows:

- if $\mathcal{F} = a \in A$ is a leaf, then $\text{Dep}(\mathcal{F}) := \{a\}$;
- if $\mathcal{F} = (\mathcal{F}_1) \cdots (\mathcal{F}_n)$ is binary or idempotent, then $\text{Dep}(\mathcal{F}) := \{\mathcal{F}\} \cup \text{Dep}(\mathcal{F}_1) \cup \text{Dep}(\mathcal{F}_n)$.

Intuitively, $\text{Dep}(\mathcal{F}) \subseteq \text{Nodes}(\mathcal{F})$ contains all the nodes of \mathcal{F} except those which are descendant of a middle child. If $\mathcal{I} \in \text{Nodes}(\mathcal{F})$, we consider $\text{Dep}(\mathcal{I}) \subseteq \text{Nodes}(\mathcal{I})$ as a subset of $\text{Nodes}(\mathcal{F})$. We then define the frontier of \mathcal{I} , denoted $\text{Fr}_{\mathcal{F}}(\mathcal{I}) \subseteq \{1, \dots, |w|\}$ as the set of positions of w which belong to $\text{Dep}(\mathcal{I})$ (when seen as leaves of \mathcal{F}).

► **Example 5.13.** In Figure 6, the top-most red node \mathcal{I} is iterable. Furthermore $\text{Dep}(\mathcal{I})$ is the set of red nodes, $\text{Fr}_{\mathcal{F}}(\mathcal{I}) = \{5, 6, 10\}$ and $w[\text{Fr}_{\mathcal{F}}(\mathcal{I})] = acb$.

The relationship between iterable nodes and dependencies is detailed below. We denote by $\text{Part}(\mathcal{F}) := \text{Iterable-nodes}(\mathcal{F}) \uplus \{\mathcal{F}\}$, the set of iterable nodes plus the root.

► **Lemma 5.14.** Let $\mathcal{F} \in \text{Fact}(w)$, then $\{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Part}(\mathcal{F})\}$ is a partition of $\text{Nodes}(\mathcal{F})$; and $\{\text{Fr}_{\mathcal{F}}(\mathcal{I}) : \mathcal{I} \in \text{Part}(\mathcal{F})\}$ is a partition of $\{1, \dots, |w|\}$.

We define $\text{prod}(i, j)$ in w as "the production performed in i when called from j ".

► **Definition 5.15.** Let $w \in A^+$ and $1 \leq i \leq j \leq |w|$ two positions of w . We define $\text{prod}(i, j) \in \mathbb{N}$ as $\lambda_{f_j}(\mu(w[1:i-1]), w[i], \mu(w[i+1:j]))$, where $f_j := \lambda(\mu(w[1:j-1]), w[j], \mu(w[j+1:|w|]))$.

We extend this definition to pairs of nodes: given $\mathcal{I}, \mathcal{J} \in \text{Nodes}(\mathcal{F})$, we define $\text{prod}(\mathcal{I}, \mathcal{J})$ "the sum of all productions performed in the frontier of \mathcal{I} , when called from the frontier of \mathcal{J} " as follows (we have to ensure that the calling positions are "on the right").

► **Definition 5.16.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ and $\mathcal{I}, \mathcal{J} \in \text{Nodes}(\mathcal{F})$. We define:

$$\text{prod}(\mathcal{I}, \mathcal{J}) := \sum_{\substack{i \in \text{Fr}_{\mathcal{F}}(\mathcal{I}) \\ j \in \text{Fr}_{\mathcal{F}}(\mathcal{J}) \\ i \leq j}} \text{prod}(i, j) \in \mathbb{N}.$$

If \mathcal{I} is an ancestor of \mathcal{J} (or the converse) then $\text{Fr}_{\mathcal{F}}(\mathcal{I})$ and $\text{Fr}_{\mathcal{F}}(\mathcal{J})$ are interleaved, hence we can have both $\text{prod}(\mathcal{I}, \mathcal{J}) \neq 0$ and $\text{prod}(\mathcal{J}, \mathcal{I}) \neq 0$. However, if \mathcal{I} and \mathcal{J} are not on the same branch, we have either $\text{prod}(\mathcal{I}, \mathcal{J}) = 0$ or $\text{prod}(\mathcal{J}, \mathcal{I}) = 0$.

Applying Lemma 5.14, it is not hard to compute $f(w)$ using the $\text{prod}(\mathcal{I}, \mathcal{J})$.

► **Lemma 5.17.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$. Then:

$$f(w) = \sum_{\mathcal{I}, \mathcal{J} \in \text{Part}(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}).$$

5.2 Typology of pairs of nodes

We intend to compute (if possible) f using a 1-blind transducer. Following Lemma 5.17, it is enough to consider the productions performed on the pairs of nodes of a factorization. For this study, we split the pairs depending on their relative position in the tree.

XX:12 Pebble transducers with unary output

Pairs separated by the frontier of the root. The frontier of the root $\text{Fr}_{\mathcal{F}}(\mathcal{F})$ plays a very specific role with respect to blind transducers. Indeed, over factorizations of height at most Λ , the size of the frontier is bounded, hence it splits the word in a bounded number of distinguishable "blocks". Formally, we define the notion of *basis*.

► **Definition 5.18.** *An idempotent node is a basis if it belongs to the dependency of the root.*

The following result is shown by induction.

► **Lemma 5.19.** *Let $w \in A^+$ and $\mathcal{F} \in \text{Fact}(w)$. Given $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$, there exists a unique basis, denoted $\text{basis}_{\mathcal{F}}(\mathcal{I})$, such that \mathcal{I} is the descendant of a middle child of $\text{basis}_{\mathcal{F}}(\mathcal{I})$.*

► **Definition 5.20.** *Given $w \in A^+$ and $\mathcal{F} \in \text{Fact}(w)$, we define $D(\mathcal{F}) \subseteq \text{Part}(\mathcal{F}) \times \text{Part}(\mathcal{F})$ by:*

$$D(\mathcal{F}) := \{(\mathcal{I}, \mathcal{J}) : \mathcal{I}, \mathcal{J} \in \text{Iterable-nodes}(\mathcal{F}) \text{ and } \text{basis}_{\mathcal{F}}(\mathcal{I}) \neq \text{basis}_{\mathcal{F}}(\mathcal{J})\}.$$

Intuitively $\text{basis}_{\mathcal{F}}(\mathcal{I}) \neq \text{basis}_{\mathcal{F}}(\mathcal{J})$ means that $\text{Fr}_{\mathcal{F}}(\mathcal{I})$ and $\text{Fr}_{\mathcal{F}}(\mathcal{J})$ belong to two different "blocks" of the input. Lemma 5.21 is shown by building a 1-blind bimachine which visits successively each basis \mathcal{B} , and for each iterable \mathcal{J} such that $\text{basis}_{\mathcal{F}}(\mathcal{J}) = \mathcal{B}$, calls a submachine which visits the \mathcal{I} such that $\text{basis}_{\mathcal{F}}(\mathcal{I}) \neq \mathcal{B}$ and produces $\text{prod}(\mathcal{I}, \mathcal{J})$. The key element for doing this operation without pebbles is that the number of bases is bounded.

► **Lemma 5.21.** *One can build a 1-blind bimachine computing:*

$$f_D : (\hat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in D(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Linked pairs. Let $U(\mathcal{F}) := \text{Part}(\mathcal{F}) \times \text{Part}(\mathcal{F}) \setminus D(\mathcal{F})$, it corresponds to the pairs of $\text{Iterable-nodes}(\mathcal{F})$ which have the same basis, plus all the pairs $(\mathcal{F}, \mathcal{I})$ and $(\mathcal{I}, \mathcal{F})$ for $\mathcal{I} \in \text{Part}(\mathcal{F})$. We now study the pairs of $U(\mathcal{F})$ which are "linked", in the sense that one node is (nearly) the ancestor of the other.

► **Definition 5.22.** *Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$. Let $L(\mathcal{F})$ be the set of all $(\mathcal{I}, \mathcal{J}) \in U(\mathcal{F})$ such that \mathcal{I} (or \mathcal{J}) is either the ancestor of, or the right/left sibling of an ancestor of \mathcal{J} (or \mathcal{I}).*

In particular, we have $(\mathcal{F}, \mathcal{F}), (\mathcal{I}, \mathcal{F}), (\mathcal{F}, \mathcal{I}), (\mathcal{I}, \mathcal{I}) \in L(\mathcal{F})$ for all $\mathcal{I} \in \text{Part}(\mathcal{F})$. If \mathcal{F} has height at most Λ , there are at most 3Λ nodes which are either an ancestor or the right/left sibling of an ancestor of \mathcal{I} . Lemma 5.23 follows from this boundedness.

► **Lemma 5.23.** *One can build a 0-blind bimachine computing:*

$$f_L : (\hat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Independent nodes. The remaining sum is the most interesting, since it is the only case where we use the assumption that \mathcal{T} to be symmetrical (and this assumption is crucial). Let $\mathcal{F} \in \text{Fact}(w)$, we define the set $I(\mathcal{F}) := U(\mathcal{F}) \setminus L(\mathcal{F})$. It contains the pairs $(\mathcal{I}, \mathcal{J})$ of iterable nodes such that $\text{basis}_{\mathcal{F}}(\mathcal{I}) = \text{basis}_{\mathcal{F}}(\mathcal{J})$ (i.e. they descend from a common "big" idempotent), and \mathcal{I} (or \mathcal{J}) is not an ancestor of \mathcal{J} (or \mathcal{I}), nor the left or right sibling of its ancestor.

► **Lemma 5.24.** *If \mathcal{T} is symmetrical, one can build a 1-blind bimachine computing:*

$$f_I : (\hat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Proof idea. We define $\text{type}_{\mathcal{F}}(\mathcal{I})$ for $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$ as a bounded abstraction of \mathcal{I} which describes the frontier and the location of \mathcal{I} in \mathcal{F} and in $\text{basis}_{\mathcal{F}}(\mathcal{I})$. Using symmetry, we show that for $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$, $\text{prod}(\mathcal{I}, \mathcal{J})$ only depends on $\text{type}_{\mathcal{F}}(\mathcal{I})$ and $\text{type}_{\mathcal{F}}(\mathcal{J})$, but not on their relative positions. Hence we build a 1-blind bimachine, whose main bimachine ranges over all possible \mathcal{J} and computes $\text{type}_{\mathcal{F}}(\mathcal{J})$, and whose submachines range over all possible \mathcal{I} (a special treatment has to be done to avoid \mathcal{I} such that $(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})$), compute $\text{type}_{\mathcal{F}}(\mathcal{I})$ and output $\text{prod}(\mathcal{I}, \mathcal{J})$. The submachines do not need to "see" \mathcal{J} . ◀

We finally show $3 \Rightarrow 2$ in Theorem 5.4. Given $w \in A^+$ we first compute \mathcal{F} of height at most Λ by Proposition 5.9. Then we use the machines from Lemmas 5.21, 5.23 and 5.24 and build a 1-blind transducer computing the sum of their outputs. The original function can be recovered since 1-blind transducers are closed under composition with two-way [11].

6 Conclusion and outlook

As a conclusion, we discuss future work. This paper introduces new proof techniques, in particular the use of factorization forests to study the productions of transducers. We believe that these techniques give a step towards other membership problems concerning pebble transducers. Among them, let us mention the membership problem from k -pebble to k -blind, at first over unary alphabets. Similarly, the membership from k -pebble to k -marble over non-unary alphabets is worth being studied (the answer seems to rely on combinatorial properties of the output, since unary outputs can always be produced using marbles).

References

- 1 Mikolaj Bojańczyk. Factorization forests. In *International Conference on Developments in Language Theory*, pages 1–17. Springer, 2009.
- 2 Mikolaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages, and Programming*, pages 26–37. Springer, 2014.
- 3 Mikolaj Bojańczyk. Polyregular functions. *arXiv preprint arXiv:1810.08760*, 2018.
- 4 Mikolaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, pages 106:1–106:14, 2019.
- 5 Michal P Chytil and Vojtěch Jákł. Serial composition of 2-way finite-state transducers and simple programs on strings. In *4th International Colloquium on Automata, Languages, and Programming, ICALP 1977*, pages 135–147. Springer, 1977.
- 6 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register transducers are marble transducers. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24–28, 2020, Prague, Czech Republic*, 2020.
- 7 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.
- 8 Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. In *International Workshop on Reachability Problems*, pages 75–86. Springer, 2017.
- 9 Eitan M Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM Journal on Computing*, 11(3):448–452, 1982.
- 10 Nathan Lhote. Pebble minimization of polyregular functions. In *35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*. IEEE, 2020.
- 11 Lê Thành Dung Nguyễn, Camille Noûs, and Pierre Pradic. Comparison-free polyregular functions. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, 2021.
- 12 John C Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.
- 13 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.

A Proof of Lemma 4.4

We show that given a bmachine with external pebble functions, which are computed by SSTs, one can build an equivalent SST.

A.1 SST with lookahead

We first define a variant of SST with the same expressive power. Intuitively, this model is similar to bmachines, in the sense that the register update not only depends on the current letter, but also on a finite abstraction of the prefix and suffix.

► **Definition A.1.** An SST with lookahead $\mathcal{T} = (A, \mathfrak{X}, M, \mu, I, \lambda, F)$ is:

- an input alphabet A and a finite set \mathfrak{X} of registers;
- a morphism into a finite monoid $\mu : A^* \rightarrow M$;
- an initial row vector $I \in \mathbb{N}^{\mathfrak{X}}$;
- a register update function $\lambda : M \times A \times M \rightarrow \mathbb{N}^{\mathfrak{X} \times \mathfrak{X}}$;
- an output column vector $F \in \mathbb{N}^{\mathfrak{X}}$.

Let us define its semantics. Intuitively, in position i of $w \in A^+$, we perform the register update $\lambda(\mu(w[1:i-1]), w[i], \mu(w[i+1:|w|]))$. Formally, for $0 \leq i \leq |w|$, we define $\mathcal{T}^{w,i} \in \mathbb{N}^{\mathfrak{X}}$ ("the values of the registers after reading $w[1:i]$ "²) as follows:

- $\mathcal{T}^{w,0} := I$;
- for $i \geq 1$, $\mathcal{T}^{w,i} := \mathcal{T}^{w,i-1} \times \lambda(\mu(w[1:i-1]), w[i], \mu(w[i+1:|w|]))$.

To define the function $f : A^+ \rightarrow \mathbb{N}$ computed by \mathcal{T} , we combine the final values by the output vector:

$$f(w) := \mathcal{T}^{w,|w|} F.$$

It is known that SST with lookahead are equivalent to SST (the proof is roughly a "determinisation" procedure for eliminating the rightmost argument of λ , and an encoding of the monoid in the registers for eliminating the leftmost argument).

► **Lemma A.2** ([8]). Given an SST with lookahead, we can build an equivalent SST.

Hence, it is sufficient to build an SST with lookahead.

A.2 Main proof of Lemma 4.4

Let $\mathcal{T} = (A, M, \mu, \lambda, \mathfrak{F})$ be the bmachine with external pebble functions. Each $\mathfrak{f} \in \mathfrak{F}$ is computed by an SST $\mathcal{T}_{\mathfrak{f}} := (A \uplus \underline{A}, \mathfrak{X}_{\mathfrak{f}}, I_{\mathfrak{f}}, T_{\mathfrak{f}}, F_{\mathfrak{f}})$.

► **Example A.3** (Running example). Let $\mathcal{T} = (A, M, \mu, \lambda, \{\mathfrak{f}\})$ with $M = \{1_M\}$ singleton and $\lambda(1_M, a, 1_M) = \mathfrak{f}$ for $a \in A$. Let $\mathcal{T}_{\mathfrak{f}}$ have two registers x, y with x initialized to 1 and y initialized to 0. When reading $a \in A \uplus \underline{A}$ it performs $x \leftarrow x, y \leftarrow y + x$. Finally it outputs y .

Then $\mathfrak{f}(w) = |w|$ and \mathcal{T} computes $f : w \mapsto |w|^2$.

► **Definition A.4.** Let $w \in A^*$ and $\mathfrak{f} \in \mathfrak{F}$. We define $\text{Calls}(w, \mathfrak{f})$ as the set of positions of w in which \mathfrak{f} is called, that is $\{1 \leq j \leq |w| : \lambda(\mu(w[1:j-1]), w[j], \mu(w[j+1:|w|])) = \mathfrak{f}\}$.

² Due to the fact that λ looks "on the right", $\mathcal{T}^{w,i}$ depends on the whole w and not only on $w[1:i]$.

XX:16 Pebble transducers with unary output

For $1 \leq j \leq i \leq |w|$, $I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j)$ corresponds to the value of the registers of $\mathcal{T}_{\mathfrak{f}}$ in position i when the call to \mathfrak{f} is performed from position j .

▷ **Claim A.5.** For all $w \in A^*$, the following holds:

$$f(w) = \sum_{\mathfrak{f} \in \mathfrak{F}} \sum_{\substack{1 \leq j \leq |w| \\ j \in \text{Calls}(w, \mathfrak{f})}} I_{\mathfrak{f}}T_{\mathfrak{f}}(w\uparrow j)F_{\mathfrak{f}}$$

Proof. By definition of external pebble functions we have:

$$f(w) := \sum_{1 \leq j \leq |w|} \mathfrak{f}_j(w\uparrow j)$$

where $\mathfrak{f}_j := \lambda(\mu(w[1:j-1]), w[j], \mu(w[j+1:|w|]))$ is "the external function called in j ". Hence by partitioning the sum depending on the external functions it follows:

$$f(w) := \sum_{\mathfrak{f} \in \mathfrak{F}} \sum_{\substack{1 \leq j \leq |w| \\ j \in \text{Calls}(w, \mathfrak{f})}} \mathfrak{f}(w\uparrow j)$$

And finally we note that $\mathfrak{f}(w\uparrow j) = I_{\mathfrak{f}}T_{\mathfrak{f}}(w\uparrow j)F_{\mathfrak{f}}$. ◀

Idea of the construction. Let us fix an external function \mathfrak{f} . Following Claim A.5, we want to build an SST with lookahead \mathcal{U} which computes the values of the vector:

$$\sum_{\substack{1 \leq j \leq |w| \\ j \in \text{Calls}(w, \mathfrak{f})}} I_{\mathfrak{f}}T_{\mathfrak{f}}(w\uparrow j) \in \mathbb{N}^{\mathfrak{X}}.$$

For this, it will keep track when in position i of the values of:

$$\sum_{\substack{1 \leq j \leq i \\ j \in \text{Calls}(w, \mathfrak{f})}} I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j) \in \mathbb{N}^{\mathfrak{X}}$$

and these values will be updated when going from i to $i + 1$.

► **Example A.6 (Running example).** We have $I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j)(x) = 1$ and $I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j)(y) = i$.

$$\text{Hence } \sum_{\substack{1 \leq j \leq i \\ j \in \text{Calls}(w, \mathfrak{f})}} I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j)(x) = i \text{ and } \sum_{\substack{1 \leq j \leq i \\ j \in \text{Calls}(w, \mathfrak{f})}} I_{\mathfrak{f}}T_{\mathfrak{f}}(w[1:i]\uparrow j)(y) = i^2.$$

Formal construction. Let $\mathcal{U} = (A, \mathfrak{X}, M, \mu, I, \kappa, F)$ be an SST with lookahead with:

- the set $\mathfrak{X} := \bigsqcup_{\mathfrak{f} \in \mathfrak{F}} \{\text{Sum}_x : x \in \mathfrak{X}_{\mathfrak{f}}\} \sqcup \{\text{Old}_x : x \in \mathfrak{X}_{\mathfrak{f}}\}$ of registers;
 - the morphism $\mu : A^* \rightarrow M$ used in \mathcal{T} ;
 - an initial column vector $I \in \mathbb{N}^{\mathfrak{X}}$ such that for all $\mathfrak{f} \in \mathfrak{F}$ and $x \in \mathfrak{X}_{\mathfrak{f}}$:
 - $I(\text{Sum}_x) = 0$;
 - $I(\text{Old}_x) = I_{\mathfrak{f}}(x)$;
 - the update $\kappa : M \times A \times M \rightarrow \mathbb{N}^{\mathfrak{X} \times \mathfrak{X}}$ as follows. Let $(m, a, n) \in M \times A \times M$ and $\mathfrak{f} := \lambda(m, a, n)$. Then $\kappa(m, a, n)$ performs the following updates:
 - for all $\mathfrak{g} \neq \mathfrak{f}$ and $x \in \mathfrak{X}_{\mathfrak{g}}$:
 - * $\text{Sum}_x \leftarrow \sum_{y \in \mathfrak{X}_{\mathfrak{g}}} \alpha_y \text{Sum}_y$;
 - * $\text{Old}_x \leftarrow \sum_{y \in \mathfrak{X}_{\mathfrak{g}}} \alpha_y \text{Old}_y$;
- where $x \leftarrow \sum_{y \in \mathfrak{X}_{\mathfrak{g}}} \alpha_y y$ is the update performed by $\mathcal{T}_{\mathfrak{g}}$ when reading a ;

- for all $x \in \mathfrak{X}_f$:
 - * $\text{Sum}_x \leftarrow \sum_{y \in \mathfrak{X}_f} \alpha_y \text{Sum}_y + \sum_{y \in \mathfrak{X}_f} \beta_y \text{Old}_y$;
 - * $\text{Old}_x \leftarrow \sum_{y \in \mathfrak{X}_f} \alpha_y \text{Old}_y$;
 where $x \leftarrow \sum_{y \in \mathfrak{X}_f} \alpha_y y$ is the update performed by \mathcal{T}_f when reading a ;
 and $x \leftarrow \sum_{y \in \mathfrak{X}_f} \beta_y y$ is the update performed by \mathcal{T}_f when reading \underline{a} .
 Intuitively, the sum with the β_y corresponds to what is "added" by the new call to f .
- the output line vector $F \in \mathbb{N}^{\mathfrak{X}}$ such that for all $f \in \mathfrak{F}$ and $x \in \mathfrak{X}_f$:
 - $F(\text{Sum}_x) = F_f(x)$;
 - $F(\text{Old}_x) = 0$.

► **Example A.7** (Running example). \mathcal{U} performs the following updates:

- $\text{Old}_x \leftarrow \text{Old}_x$, $\text{Old}_y \leftarrow \text{Old}_y + \text{Old}_x$;
- $\text{Sum}_x \leftarrow \text{Sum}_x + \text{Old}_x$, $\text{Sum}_y \leftarrow \text{Sum}_y + \text{Sum}_x + \text{Old}_y + \text{Old}_x$.

We can check that $\mathcal{U}^{w,i}(\text{Old}_x) = 1$, $\mathcal{U}^{w,i}(\text{Old}_y) = i$ and $\mathcal{U}^{w,i}(\text{Sum}_x) = i$, $\mathcal{U}^{w,i}(\text{Sum}_y) = i^2$.

Correctness of the construction. As the registers Old_x for $x \in \mathfrak{X}_f$ are updated following the updates of \mathcal{T}_f , it follows immediately that:

► **Claim A.8.** Given $x \in \mathfrak{X}_f$, for all $w \in A^+$ and $1 \leq i \leq |w|$ we have:

$$\mathcal{U}^{w,i}(\text{Old}_x) = I_f T_f(w[1:i])(x).$$

We can finally show that the registers Sum_x store the information we wanted.

► **Claim A.9.** Given $x \in \mathfrak{X}_f$, for all $w \in A^+$ and $0 \leq i \leq |w|$ we have:

$$\mathcal{U}^{w,i}(\text{Sum}_x) = \sum_{\substack{1 \leq j \leq i \\ j \in \text{Calls}(w,f)}} I_f T_f(w[1:i] \uparrow j)(x).$$

Proof. We proceed by induction on $0 \leq i \leq |w|$. For $i = 0$ both terms equal 0. For the induction step with $i \geq 1$ let $f \in \mathfrak{F}$ and $x \in \mathfrak{X}_f$.

Suppose that $\lambda(\mu(w[1:i-1]), w[i], \mu(w[i+1:|w|])) = f$ (the case when they differ is similar and even easier), then:

$$\sum_{\substack{1 \leq j \leq i \\ j \in \text{Calls}(w,f)}} I_f T_f(w[1:i] \uparrow j)(x) = I_f T_f(w[1:i] \uparrow i)(x) + \sum_{\substack{1 \leq j \leq i-1 \\ j \in \text{Calls}(w,f)}} I_f T_f(w[1:i] \uparrow j)(x). \quad (1)$$

- Let $x \leftarrow \sum_{y \in \mathfrak{X}_f} \alpha_y y$ be the update performed by \mathcal{T}_f when reading a .
 Then for $j \leq i-1$, $I_f T_f(w[1:i] \uparrow j)(x) = \sum_{y \in \mathfrak{X}_f} \alpha_y \times I_f T_f(w[1:i-1] \uparrow j)(y)$.
- Let $x \leftarrow \sum_{y \in \mathfrak{X}_f} \beta_y y$ be the update performed by \mathcal{T}_f when reading \underline{a} .
 Then $I_f T_f(w[1:i] \uparrow i)(x) = \sum_{y \in \mathfrak{X}_f} \beta_y \times I_f T_f(w[1:i-1])(y)$.

Hence we can rewrite Equation 1 using values in position $i-1$. By Claim A.8 and the induction hypothesis, this sum coincides with the update in \mathcal{U} . ◀

The fact that \mathcal{U} computes f follows from the definition of F and Claim A.5.

B

 Non-computability of iterated-square by a k -blind bimachine

In order to simplify the description, we do not deal with iterated-square but a slight variant:

$$f : a^{n_1}b \cdots ba^{n_\ell}b \mapsto \sum_{i=1}^{\ell} n_i(n_i + 1) = \text{iterated-square} + \text{nb-a}.$$

Indeed, by Claim J.10, if iterated-square is computable by a k -blind transducer, then so is f (since nb-a is computable by a 0-blind).

Description with a 1-marble bimachine. Let us first give a 1-marble bimachine for f . The monoid is $M = \{1_M, 0_M\}$ where 1_M is neutral and 0_M absorbing (both are idempotents). The morphism is $\mu : a \mapsto 1_M, b \mapsto 0_M$. In the main bimachine, we define the output function $\lambda(m, a, m) := f_a$ and $\lambda(m, b, m) := f_b$ for all $m \in M$.

Function f_b is constant to 0, its output is $\lambda_{f_b}(m, a, m) := 0$ and $\lambda_{f_b}(m, b, m) := 0$ for all $m \in M$. Function f_a outputs twice the number of a after the last b . Its output is $\lambda_{f_a}(m, a, 1_M) := 2$, $\lambda_{f_a}(m, a, 0_M) := 0$, and $\lambda_{f_a}(m, b, m) := 0$ for all $m \in M$.

Non-computability by 1-blind. We set $u_1 := a, u_2 := a$ and $m, n, m_1, n_1, m_2, n_2 = 0_M$. Let $e_1 := \mu(u_1) = 1_M, e_2 := \mu(u_2) = 1_M$, note that $e := m_1 e_1 n_1 = m_2 e_2 n_2 = 0_M$. Then:

- if $p = 0_M$, then $\text{prod}(mem_1e_1\langle u_1 \rangle e_1pe_2\langle u_2 \rangle e_2n_2en) = \text{prod}(0_M\langle a \rangle 0_M\langle a \rangle 0_M) = 0$
and $m_1e_1pe_2n_2 = e = 0_M, em_1e_1pe_2 = em_2e_2 = 0_M$ and $e_1pe_2n_2e = e_1n_1e = 0_M$;
- if $p = 1_M$, then $\text{prod}(mem_1e_1\langle u_1 \rangle e_1pe_2\langle u_2 \rangle e_2n_2en) = \text{prod}(0_M\langle a \rangle 1_M\langle a \rangle 0_M) = 2$
and $m_2e_2pe_1n_1 = e = 0_M, em_2e_2pe_1 = em_1e_1 = 0_M$ and $e_2pe_1n_1e = e_2n_2e = 0_M$.

It follows from Definition 5.3 that the bimachine is not symmetrical, hence by Lemma 5.6 the function f is not computable by a k -blind bimachine for any $k \geq 0$.

C

 Proof of Lemma 5.6 - blind \Rightarrow symmetrical

Lemma 5.6 is the core of the negative membership result, since it describes a necessary condition for $f : A^+ \rightarrow \mathbb{N}$ given by the 1-marble bimachine \mathcal{T} , to be computable by a k -blind bimachine. Assume that it is the case, we want to show that \mathcal{T} is symmetrical.

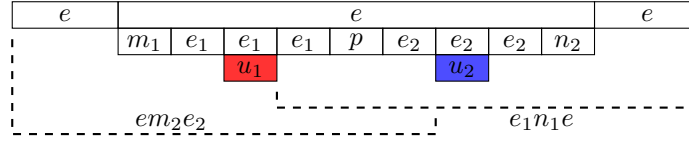
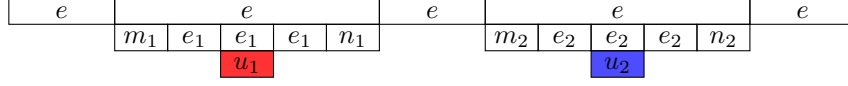
The elements $m, n \in M$ of Definition 5.3 define a "context", which should be reported everywhere in the proof, but it is never used in the results. Hence to simplify the notations, we suppose that $m = n$ is the neutral element of M (it will no longer be mentioned).

C.1 The production does not depend on p

This subsection is devoted to showing Lemma C.1. Intuitively, it says that the production on u_1 called from u_2 only depends their respective "contexts" (which are e_1, m_1, n_1 and e_2, m_2, n_2) but not on the p which separates them.

► **Lemma C.1.** *Let $m_1, n_1, m_2, n_2 \in M$ and $u_1, u_2 \in A^+$ such that $e_1 := \mu(u_1), e_2 := \mu(u_2)$ and $e := m_1 e_1 n_1 = m_2 e_2 n_2$ are idempotents. Let $p \in M$ be such that $m_1 e_1 p e_2 n_2 = e, em_1 e_1 p e_2 = em_2 e_2$ and $e_1 p e_2 n_2 e = e_1 n_1 e$. Then:*

$$\text{prod}(em_1e_1\langle u_1 \rangle e_1pe_2\langle u_2 \rangle e_2n_2e) = \text{prod}(em_1e_1\langle u_1 \rangle e_1n_1em_2e_2\langle u_2 \rangle e_2n_2e).$$

(a) The bitype $\Phi = em_1e_1\langle \mathbf{u}_1 \rangle e_1pe_2\langle \mathbf{u}_2 \rangle e_2n_2e$ (b) The bitype $\Phi' = em_1e_1\langle \mathbf{u}_1 \rangle e_1n_1em_2e_2\langle \mathbf{u}_2 \rangle e_2n_2e$ ■ **Figure 7** The bitypes considered in this subsection

We denote $\Phi := em_1e_1\langle \mathbf{u}_1 \rangle e_1pe_2\langle \mathbf{u}_2 \rangle e_2n_2e$ and $\Phi' := em_1e_1\langle \mathbf{u}_1 \rangle e_1n_1em_2e_2\langle \mathbf{u}_2 \rangle e_2n_2e$ the bitypes considered in Lemma C.1. They are depicted in Figure 7.

Let us prove Lemma C.1. Since μ is surjective, there exists $\alpha, \beta, \gamma \in A^*$ such that $\mu(\alpha) = em_1e_1$, $\mu(\beta) = e_1pe_2$, $\mu(\gamma) = e_2n_2e$. They describe an instance of the bitype Φ .

We are going to apply Lemma C.2 below with $\alpha, \beta, \gamma, \mathbf{u}_1$ and \mathbf{u}_2 . Intuitively, this lemma creates a family of words in which both the bitypes Φ and Φ' occurs, and describes what is the output of a k -blind bimachine on this family.

► **Lemma C.2.** *Let $\alpha, \beta, \gamma \in A^*$, $\mathbf{u}_1, \mathbf{u}_2 \in A^+$, and f computed by a k -blind bimachine. There exists $\omega \geq 1$ such that the following holds. Let $w := \alpha \mathbf{u}_1^\omega \beta \mathbf{u}_2^\omega \gamma$ and $G : \mathbb{N}^4 \rightarrow \mathbb{N}$ defined by:*

$$G(X, Y, X', Y') := f(w^{2\omega-1}(\alpha \mathbf{u}_1^{\omega X} \beta \mathbf{u}_2^{\omega Y} \gamma)w^{\omega-1}(\alpha \mathbf{u}_1^{\omega X'} \beta \mathbf{u}_2^{\omega Y'} \gamma)w^\omega)$$

Then there exists a polynomial $P : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for all $X, Y, X', Y' \geq 2$:

$$G(X, Y, X', Y') = P(X + X', Y + Y').$$

► **Remark C.3.** P has degree at most k .

► **Remark C.4.** Here we shall fix $X' = 2$, but the lemma will be re-used later with variable X' .

Let ω be fixed by Lemma C.2. It allows to describe the output of a k -blind bimachine on words of the form:

$$w^{2\omega-1}(\alpha \mathbf{u}_1^{\omega X} \beta \mathbf{u}_2^{\omega Y} \gamma)w^{\omega-1}(\alpha \mathbf{u}_1^{2\omega} \beta \mathbf{u}_2^{\omega Y'} \gamma)w^\omega$$

We now want to understand what is the output of our 1-marble bimachine \mathcal{T} , and relate this output with $\text{prod}(\Phi)$ and $\text{prod}(\Phi')$. This is the purpose of Lemma C.5 below, we set $w_0 := w^{2\omega-1}\alpha$, $v_1 := \mathbf{u}_1$, $w_1 = \beta$, $v_2 := \mathbf{u}_2$, $w_2 := \gamma w^{\omega-1}\alpha \mathbf{u}_1^{2\omega}\beta$, $v_3 := \mathbf{u}_2$ and $w_3 := \gamma w^\omega$.

► **Lemma C.5.** *Let $w_0, w_1, w_2, w_3 \in A^*$ and $v_1, v_2, v_3 \in A^+$. Let $f_1 := \mu(v_1)$, $f_2 := \mu(v_2)$ and $f_3 := \mu(v_3)$. We suppose that f_1, f_2, f_3 are idempotents. Let $H : \mathbb{N}^3 \rightarrow \mathbb{N}$ defined by:*

$$H(Z_1, Z_2, Z_3) = f(w_0 v_1^{Z_1} w_1 v_2^{Z_2} w_2 v_3^{Z_3} w_3).$$

Define the following bitypes:

- $\Psi_1^2 := \mu(w_0) f_1 \langle \mathbf{v}_1 \rangle f_1 \mu(w_1) f_2 \langle \mathbf{v}_2 \rangle f_2 \mu(w_2) f_3 \mu(w_3);$
- $\Psi_1^3 := \mu(w_0) f_1 \langle \mathbf{v}_1 \rangle f_1 \mu(w_1) f_2 \mu(w_2) f_3 \langle \mathbf{v}_3 \rangle f_3 \mu(w_3);$

XX:20 Pebble transducers with unary output

$$\blacksquare \Psi_2^3 := \mu(w_0)f_1\mu(w_1)f_2\langle \mathbf{v}_2 \rangle f_2\mu(w_2)f_3\langle \mathbf{v}_3 \rangle f_3\mu(w_3).$$

There exists a polynomial $Q(Z_1, Z_2, Z_3)$ without any term in Z_1Z_2 , Z_1Z_3 nor Z_2Z_3 such that for all $Z_1, Z_2, Z_3 \geq 4$:

$$H(Z_1, Z_2, Z_3) = Q(Z_1, Z_2, Z_3) + \text{prod}(\Psi_1^2)Z_1Z_2 + \text{prod}(\Psi_1^3)Z_1Z_3 + \text{prod}(\Psi_2^3)Z_2Z_3$$

► Remark C.6. Here we shall not use Ψ_2^3 , but the lemma will be re-used later.

Let $T_1, T_2, T'_2 \geq 4$, we have:

$$w_0v_1^{\omega T_1}w_1v_2^{\omega T_2}w_2v_3^{\omega T'_2}w_3 = w^{2\omega-1}(\alpha u_1^{\omega T_1}\beta u_2^{\omega T_2}\gamma)w^{\omega-1}(\alpha u_1^{2\omega}\beta u_2^{\omega T'_2}\gamma)w^{\omega}.$$

Thus by applying f to each member of the equality:

$$H(\omega T_1, \omega T_2, \omega T'_2) = G(T_1, T_2, 2, T'_2).$$

Hence by using the two previous lemmas:

$$P(T_1+2, T_2+T'_2) = Q(\omega T_1, \omega T_2, \omega T'_2) + \omega^2 \text{prod}(\Psi_1^2)T_1T_2 + \omega^2 \text{prod}(\Psi_1^3)T_1T'_2 + \omega^2 \text{prod}(\Psi_2^3)T_2T'_2.$$

Since $Q(\omega T_1, \omega T_2, \omega T'_2)$ has no term in T_1T_2 nor $T_1T'_2$, and P depends on $T_2 + T'_2$, it follows by identification that $\text{prod}(\Psi_1^2) = \text{prod}(\Psi_1^3)$. Finally $\text{prod}(\Phi) = \text{prod}(\Phi')$ since:

▷ Claim C.7. $\Psi_1^2 = \Phi$ and $\Psi_1^3 = \Phi'$.

Proof. With the elements chosen to apply Lemma C.5:

- $f_1 = \mu(u_1) = e_1$, $f_2 = e_2$, $f_3 = e_2$;
- $\mu(w) = \mu(\alpha u_1^{\omega} \beta u_2^{\omega} \gamma) = e$;
- $\mu(w_0) = e^{2\omega-1} e m_1 e_1 = e m_1 e_1$;
- $\mu(w_1) = e_1 p e_2$;
- $\mu(w_2) = e_2 n_2 (e m_1 e_1 p e_2) = e_2 n_2 e m_2 e_2$;
- $\mu(w_3) = e_2 n_2 e$;
- $\mu(w_0) f_1 = e m_1 e_1$;
- $f_1 \mu(w_1) f_2 = e_1 p e_2$;
- $f_2 \mu(w_2) f_3 \mu(w_3) = e_2 n_2 (e m_2 e_2 n_2 e) = e_2 n_2 e$;
- $f_1 \mu(w_1) f_2 \mu(w_2) f_3 = (e_1 p e_2 n_2 e) m_2 e_2 = e_1 n_1 e m_2 e_2$;
- $f_3 \mu(w_3) = e_2 n_2 e$.

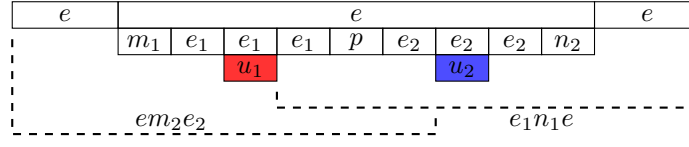
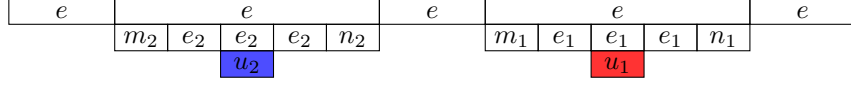
And the 5 last equalities give the equalities between the bitypes. ◀

C.2 The production does not change when swapping u_1 and u_2

This subsection is devoted to Lemma C.8. Intuitively, it says that the production from u_2 to u_1 is the same as that from u_1 to u_2 , if we keep the same "contexts" (which are independent from p).

► **Lemma C.8.** Let $m_1, n_1, m_2, n_2 \in M$ and $u_1, u_2 \in A^+$ such that $e_1 := \mu(u_1)$, $e_2 := \mu(u_2)$ and $e := m_1 e_1 n_1 = m_2 e_2 n_2$ are idempotents. Let $p \in M$ such that $m_1 e_1 p e_2 n_2 = e$, $e m_1 e_1 p e_2 = e m_2 e_2$ and $e_1 p e_2 n_2 = e_1 n_1 e$. Then:

$$\text{prod}(e m_1 e_1 \langle u_1 \rangle e_1 p e_2 \langle u_2 \rangle e_2 n_2 e) = \text{prod}(e m_2 e_2 \langle u_2 \rangle e_2 n_2 e m_1 e_1 \langle u_1 \rangle e_1 n_1 e).$$

(a) The bitype $\Phi = em_1e_1\langle \mathbf{u}_1 \rangle e_1pe_2\langle \mathbf{u}_2 \rangle e_2n_2e$ (b) The bitype $\Phi' = em_2e_2\langle \mathbf{u}_2 \rangle e_2n_2em_1e_1\langle \mathbf{u}_1 \rangle e_1n_1e$ ■ **Figure 8** The bitypes considered in this subsection

We denote $\Phi := em_1e_1\langle \mathbf{u}_1 \rangle e_1pe_2\langle \mathbf{u}_2 \rangle e_2n_2e$ and $\Phi' := em_2e_2\langle \mathbf{u}_2 \rangle e_2n_2em_1e_1\langle \mathbf{u}_1 \rangle e_1n_1e$ the bitypes considered in Lemma C.8. They are depicted in Figure 8.

The proof of Lemma C.8 is essentially similar to that of Lemma C.1, with the subtlety that we have to swap \mathbf{u}_1 and \mathbf{u}_2 at some point. Since μ is surjective, there exists $\alpha, \beta, \gamma \in A^*$ such that $\mu(\alpha) = em_1e_1$, $\mu(\beta) = e_1pe_2$, $\mu(\gamma) = e_2n_2e$. They describe an instance of the bitype Φ .

We first apply Lemma C.2 with $\alpha, \beta, \gamma, \mathbf{u}_1$ and \mathbf{u}_2 . We shall fix $Y' = 2$, but X' is variable. Let ω be fixed by Lemma C.2 and similarly $w := \alpha \mathbf{u}_1^\omega \beta \mathbf{u}_2^\omega \gamma$. It allows to describe the output of a k -blind bimachine on words of the form:

$$w^{2\omega-1}(\alpha \mathbf{u}_1^{\omega X} \beta \mathbf{u}_2^{\omega Y} \gamma) w^{\omega-1}(\alpha \mathbf{u}_1^{\omega X'} \beta \mathbf{u}_2^{2\omega} \gamma) w^\omega$$

We now want to understand what is the output of our 1-marble bimachine \mathcal{T} on such words and relate this output with $\text{prod}(\Phi)$ and $\text{prod}(\Phi')$.

For this purpose, we set $w_0 := w^{2\omega-1}\alpha$, $v_1 := \mathbf{u}_1$, $w_1 = \beta$, $v_2 := \mathbf{u}_2$, $w_2 := \gamma w^{\omega-1}\alpha$, $v_3 := \mathbf{u}_1$ and $w_3 := \beta \mathbf{u}_2^{2\omega} \gamma w^\omega$ and apply Lemma C.5. We shall use Ψ_2^3 , but not Ψ_1^3 .

Let $T_1, T_2, T'_1 \geq 4$, we have:

$$w_0 v_1^{\omega T_1} w_1 v_2^{\omega T_2} w_2 v_3^{\omega T'_1} w_3 = w^{2\omega-1}(\alpha \mathbf{u}_1^{\omega T_1} \beta \mathbf{u}_2^{\omega T_2} \gamma) w^{\omega-1}(\alpha \mathbf{u}_1^{\omega T'_1} \beta \mathbf{u}_2^{2\omega} \gamma) w^\omega.$$

Thus by applying f to each member of the equality:

$$H(\omega T_1, \omega T_2, \omega T'_1) = G(T_1, T_2, T'_1, 2).$$

Hence by using Lemma C.2 and Lemma C.5:

$$P(T_1+T'_1, T_2+2) = Q(\omega T_1, \omega T_2, \omega T'_1) + \omega^2 \text{prod}(\Psi_1^2) T_1 T_2 + \omega^2 \text{prod}(\Psi_1^3) T_1 T'_1 + \omega^2 \text{prod}(\Psi_2^3) T_2 T'_1.$$

Since $Q(\omega T_1, \omega T_2, \omega T'_1)$ has no term in $T_1 T_2$ nor $T_2 T'_1$ and P depends on $T_1 + T'_1$, it follows by identification that $\text{prod}(\Psi_1^2) = \text{prod}(\Psi_2^3)$. Finally $\text{prod}(\Phi) = \text{prod}(\Phi')$ since:

▷ Claim C.9. $\Psi_1^2 = \Phi$ and $\Psi_2^3 = \Phi'$.

C.3 Conclusion and proof of Lemma 5.6

Let $m_1, n_1, m_2, n_2 \in M$ and $\mathbf{u}_1, \mathbf{u}_2 \in A^+$ such that $e_1 := \mu(\mathbf{u}_1)$, $e_2 := \mu(\mathbf{u}_2)$ and $e := m_1 e_1 n_1 = m_2 e_2 n_2$ are idempotents.

XX:22 Pebble transducers with unary output

- **by applying Lemma C.1:** if $p \in M$ is such that
 $m_1 e_1 p e_2 n_2 = e$, $em_1 e_1 p e_2 = em_2 e_2$ and $e_1 p e_2 n_2 e = e_1 n_1 e$,
then $\text{prod}(em_1 e_1 \langle \mathbf{u}_1 \rangle e_1 p e_2 \langle \mathbf{u}_2 \rangle e_2 n_2 e) = \text{prod}(em_1 e_1 \langle \mathbf{u}_1 \rangle e_1 n_1 em_2 e_2 \langle \mathbf{u}_2 \rangle e_2 n_2 e)$;
- **by applying Lemma C.8 (after swapping 1 and 2):** if $p \in M$ is such that
 $m_2 e_2 p e_1 n_1 = e$, $em_2 e_2 p e_1 = em_1 e_1$ and $e_2 p e_1 n_1 e = e_2 n_2 e$,
then $\text{prod}(em_2 e_2 \langle \mathbf{u}_2 \rangle e_2 p e_1 \langle \mathbf{u}_1 \rangle e_1 n_1 e) = \text{prod}(em_1 e_1 \langle \mathbf{u}_1 \rangle e_1 n_1 em_2 e_2 \langle \mathbf{u}_2 \rangle e_2 n_2 e)$.

Finally, we just set $K := \text{prod}(em_1 e_1 \langle \mathbf{u}_1 \rangle e_1 n_1 em_2 e_2 \langle \mathbf{u}_2 \rangle e_2 n_2 e)$. The production is independent from p and of whether \mathbf{u}_1 is before or after \mathbf{u}_2 .

C.4 Technical tools: proofs of Lemma C.2 and Lemma C.5

C.4.1 Proof of Lemma C.2

We assume that $f : A^+ \rightarrow \mathbb{N}$ can be computed by a k -blind bimachine for some $k \geq 1$. Let $\nu : A^* \rightarrow N$ be the morphism of this bimachine (and all its sub-bimachines, without loss of generality). In this proof we forget about the fact that f was given by a 1-marble bimachine with morphism μ : here, the term "idempotent" will always mean "idempotent of N ".

Let ω be the idempotent index of N , that is the smallest integer such that m^ω is idempotent for all $m \in N$. In particular, $f_1 := \nu(\mathbf{u}_1^\omega)$, $f_2 := \nu(\mathbf{u}_2^\omega)$ are idempotents. Let $w := \alpha \mathbf{u}_1^\omega \beta \mathbf{u}_2^\omega \gamma$, then $f := \nu(w^\omega)$ is also an idempotent. Note that for all $X, Y \geq 1$, $\nu(\alpha \mathbf{u}_1^{\omega X} \beta \mathbf{u}_2^{\omega Y} \gamma) = \nu(w)$.

Now for $X, Y, X', Y' \geq 2$ let:

$$W(X, Y, X', Y') := w^{2\omega-1}(\alpha \mathbf{u}_1^{\omega X} \beta \mathbf{u}_2^{\omega Y} \gamma) w^{\omega-1}(\alpha \mathbf{u}_1^{\omega X'} \beta \mathbf{u}_2^{\omega Y'} \gamma) w^\omega \in A^+$$

We show Lemma C.2 by induction on $k \geq 0$, starting with the case $k = 0$.

► **Sublemma C.10.** *If $\mathcal{U} = (A, N, \nu, \lambda)$ is a 0-blind bimachine computing a function $g : A^+ \rightarrow \mathbb{N}$, there exists a polynomial $P : \mathbb{N}^2 \rightarrow \mathbb{N}$ of degree at most 1 such that:*

$$\forall X, Y, X', Y' \geq 2, f(W(X, Y, X', Y')) = P(X + X', Y + Y').$$

Proof. Intuitively, we only show that since \mathbf{u}_1^X and $\mathbf{u}_1^{X'}$ (resp. \mathbf{u}_2^Y and $\mathbf{u}_2^{Y'}$) are in the same "context", the productions are the same in both blocks. Formally, we extend the definition of $\lambda : N \times A \times N \rightarrow \mathbb{N}$ to $\lambda : N \times A^* \times N \rightarrow \mathbb{N}$ with:

$$\lambda(m, w, n) := \sum_{i=1}^{|w|} \lambda(m \nu(w[1:i-1]), w[i], \nu(w[i+1:|w|])n).$$

It follows by partitioning the sum defining $g(w)$ that:

▷ **Claim C.11.** If $w = w_1 \cdots w_\ell$ with $w_j \in A^*$, then:

$$g(w) = \sum_{j=1}^{\ell} \lambda(\nu(w_1 \cdots w_{j-1}), w_j, \nu(w_{j+1} \cdots w_\ell)).$$

By slicing the input $W(X, Y, X', Y')$ in several factors, we thus show that $g(W(X, Y, X', Y'))$ is the sum of several $\lambda(m, w_i, n)$:

$$\begin{aligned} & \blacksquare \lambda(\nu(\varepsilon), w^{2\omega-1} \alpha \mathbf{u}_1^\omega, \nu(\mathbf{u}_1^{\omega(X-1)} \beta \mathbf{u}_2^{\omega Y} \gamma w^{\omega-1} \alpha \mathbf{u}_1^{\omega X'} \beta \mathbf{u}_2^{\omega Y'} \gamma w^\omega)) \\ & = \lambda(\nu(\varepsilon), w^{2\omega-1} \alpha \mathbf{u}_1^\omega, f_1 \nu(\beta) f_2 \nu(\gamma) f) = C_1 \text{ constant with respect to } X, Y, X', Y'; \end{aligned}$$

$$\begin{aligned}
& \blacksquare \sum_{i=1}^{X-2} \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega i}), u_1^\omega, \nu(u_1^{\omega(X-1-i)} \beta u_2^{\omega Y} \gamma w^{\omega-1} \alpha u_1^{\omega X'} \beta u_2^{\omega Y'} \gamma w^\omega)) \\
& = \sum_{i=1}^{X-2} \lambda(f\nu(w^{\omega-1} \alpha) f_1, u_1^\omega, f_1 \nu(\beta) f_2 \nu(\gamma) f) \\
& = (X-2) \lambda(f\nu(w^{\omega-1} \alpha) f_1, u_1^\omega, f_1 \nu(\beta) f_2 \nu(\gamma) f) =: (X-2)C_X; \\
& \blacksquare \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega(X-1)}), u_1^\omega \beta u_2^\omega, \nu(u_2^{\omega(Y-1)} \gamma w^{\omega-1} \alpha u_1^{\omega X'} \beta u_2^{\omega Y'} \gamma w^\omega)) \\
& = \lambda(f\nu(w^{\omega-1} \alpha) f_1, u_1^\omega \beta u_2^\omega, f_2 \nu(\gamma) f) = C_2; \\
& \blacksquare \sum_{i=1}^{Y-2} \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega i}), u_2^\omega, \nu(u_2^{\omega(Y-1-i)} \gamma w^{\omega-1} \alpha u_1^{\omega X'} \beta u_2^{\omega Y'} \gamma w^\omega)) \\
& = \sum_{i=1}^{Y-2} \lambda(f\nu(w^{\omega-1} \alpha) f_1 \nu(\beta) f_2, u_2^\omega, f_2 \nu(\gamma) f) \\
& = (Y-2) \lambda(f\nu(w^{\omega-1} \alpha) f_1 \nu(\beta) f_2, u_2^\omega, f_2 \nu(\gamma) f) =: (Y-2)C_Y; \\
& \blacksquare \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega(Y-1)}), u_2^\omega \gamma w^{\omega-1} \alpha u_1^\omega, \nu(u_1^{\omega(X'-1)} \beta u_2^{\omega Y'} \gamma w^\omega)) = C_3; \\
& \blacksquare \sum_{i=1}^{X'-2} \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega Y} \gamma w^{\omega-1} \alpha u_1^{\omega i}), u_1^\omega, \nu(u_1^{\omega(X'-1-i)} \beta u_2^{\omega Y'} \gamma w^\omega)) \\
& = \sum_{i=1}^{X'-2} \lambda(f\nu(w^{\omega-1} \alpha) f_1, u_1^\omega, f_1 \nu(\beta) f_2 \nu(\gamma) f) = (X'-2)C_X; \\
& \blacksquare \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega Y} \gamma w^{\omega-1} \alpha u_1^{\omega(X'-1)}), u_1^\omega \beta u_2^\omega, \nu(u_2^{\omega(Y'-1)} \gamma w^\omega)) = C_4; \\
& \blacksquare \sum_{i=1}^{Y'-2} \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega Y} \gamma w^{\omega-1} \alpha u_1^{\omega X'} \beta u_2^{\omega i}), u_2^\omega, \nu(u_2^{\omega(Y'-1-i)} \gamma w^\omega)) \\
& = \sum_{i=1}^{Y'-2} \lambda(f\nu(w^{\omega-1} \alpha) f_1 \nu(\beta) f_2, u_2^\omega, f_2 \nu(\gamma) f) = (Y'-2)C_Y; \\
& \blacksquare \lambda(\nu(w^{2\omega-1} \alpha u_1^{\omega X} \beta u_2^{\omega Y} \gamma w^{\omega-1} \alpha u_1^{\omega X'} \beta u_2^{\omega(Y'-1)}), u_2^\omega \gamma w^\omega, \nu(\varepsilon)) = C_5.
\end{aligned}$$

Putting the terms together we get a polynomial of degree at most 1 in $X + X', Y + Y'$. ◀

For the induction step, let us consider a bimachine computing f with external blind functions f_1, \dots, f_n computed by $(k-1)$ -blind bimachines (with morphism ν). Then by induction we get polynomials $Q_1, \dots, Q_n : \mathbb{N}^2 \rightarrow \mathbb{N}$ of degree at most $k-1$ such that:

$$\forall X, Y, X', Y' \geq 2, f_i(W(X, Y, X', Y')) = Q_i(X + X', Y + Y').$$

By definition of external blind functions, there exists g_1, \dots, g_n computable by a 0-blind bimachine with monoid morphism ν (we define g_i as the function which counts "the number of times f_i is called by the main machine") such that for all $w \in A^+$:

$$f(w) = \sum_{i=1}^n g_i(w) f_i(w).$$

By Sublemma C.10, we get polynomials $Q'_1, \dots, Q'_n : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that:

$$\forall X, Y, X', Y' \geq 2, g_i(W(X, Y, X', Y')) = Q'_i(X + X', Y + Y').$$

The polynomial of degree at most k for f is:

$$Q := \sum_{i=1}^n Q'_i Q_i.$$

C.4.2 Proof of Lemma C.5

Let $W(Z_1, Z_2, Z_3) := w_0 v_1^{Z_1} w_1 v_2^{Z_2} w_2 v_3^{Z_3} w_3$. We want to describe $f(W(Z_1, Z_2, Z_3))$ as a polynomial. Recall that f is computed by the 1-marble bimachine $\mathcal{T} = (A, M, \mu, \mathfrak{F}, \lambda)$. For $\mathfrak{f} \in \mathfrak{F}$, let $\mathcal{T}_{\mathfrak{f}} := (A, M, \mu, \lambda_{\mathfrak{f}})$ be the auxiliary bimachine which computes it.

Decomposing bitypes. In order to study the production performed by \mathcal{T} on $W(Z_1, Z_2, Z_3)$, we first show how the productions on bitypes behave when iterating an idempotent.

▷ **Claim C.12.** Let $m, m', m'' \in M$ and $v, w \in A^+$ such that $e := \mu(v)$ is an idempotent. Then $\forall Z \geq 2$:

$$\begin{aligned} \text{prod}(m \langle \mathbf{v}^Z \rangle m' \langle \mathbf{w} \rangle m'') &= \text{prod}(m \langle \mathbf{v} \rangle e m' \langle \mathbf{w} \rangle m'') + \text{prod}(m e \langle \mathbf{v} \rangle m' \langle \mathbf{w} \rangle m'') \\ &\quad + (Z - 2) \text{prod}(m e \langle \mathbf{v} \rangle e m' \langle \mathbf{w} \rangle m''). \end{aligned}$$

Proof sketch. We consider the word v^Z as a concatenation $v \cdots v$ and look at the production performed on each v from w . We regroup the production depending on the bitypes. ◀

The following claim is obtained symmetrically.

▷ **Claim C.13.** Let $m, m', m'' \in M$ and $v, w \in A^+$ such that $\mu(v)$ is an idempotent. Then $\forall Z \geq 2$:

$$\begin{aligned} \text{prod}(m \langle \mathbf{w} \rangle m' \langle \mathbf{v}^Z \rangle m'') &= \text{prod}(m \langle \mathbf{w} \rangle m' e \langle \mathbf{v} \rangle m'') + \text{prod}(m \langle \mathbf{w} \rangle m' \langle \mathbf{v} \rangle e m'') \\ &\quad + (Z - 2) \text{prod}(m \langle \mathbf{w} \rangle m' e \langle \mathbf{v} \rangle e m''). \end{aligned}$$

Finally we can study the case of two idempotents iterated independently.

▷ **Claim C.14.** Let $m, m', m'' \in M$ and $v_1, v_2 \in A^+$ such that $e_1 := \mu(v_1)$ and $e_2 := \mu(v_2)$ are idempotents. There exists $C_1, C_2, C \in \mathbb{N}$ such that $\forall Z_1, Z_2 \geq 2$:

$$\begin{aligned} \text{prod}(m \langle \mathbf{v}_1^{Z_1} \rangle m' \langle \mathbf{v}_2^{Z_2} \rangle m'') &= \text{prod}(m e_1 \langle \mathbf{v}_1 \rangle e_1 m' e_2 \langle \mathbf{v}_2 \rangle e_2 m'') \times Z_1 Z_2 \\ &\quad + C_1 Z_1 + C_2 Z_2 + C. \end{aligned}$$

Proof. Apply successively claims C.12 and C.13. ◀

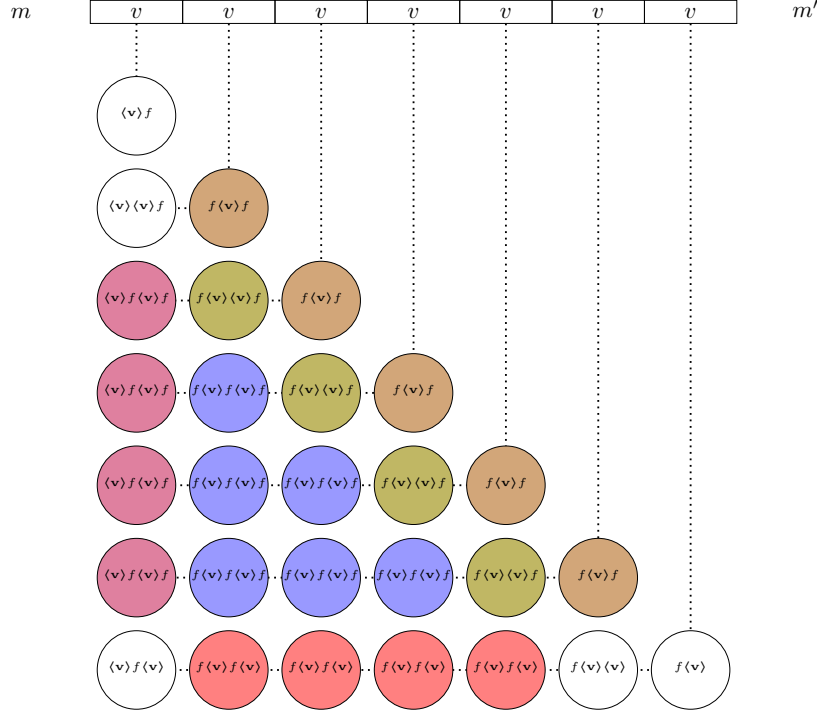
Monotypes. In order to obtain a complete description of the production performed on $W(Z_1, Z_2, Z_3)$, we also need to describe "the production on v from itself". This notion is covered by monotypes, which are defined below similarly to bitypes.

► **Definition C.15.** A monotype $\Pi := m \langle \mathbf{v} \rangle m'$ consists in $m, m' \in M$ and $v \in A^+$.

As for bitypes, we define "the production performed in v by the calls from v ", in Π . For $1 \leq i \leq j \leq |v|$, we define $\Pi(i, j) := \lambda_{\mathfrak{f}_j}(m \mu(v[1:i-1]), v[i], \mu(v[i+1:j]) m') \in \mathbb{N}$ where $\mathfrak{f}_j := \lambda(m \mu(v[1:j-1]), v[j], \mu(v[j+1:|v|]) m')$. Then we set:

$$\text{prod}(\Pi) := \sum_{1 \leq i \leq j \leq |v|} \Pi(i, j) \in \mathbb{N}.$$

We now describe the production on a monotype given by an iterated idempotent.



■ **Figure 9** A decomposition of production on the monotype $m\langle v^7 \rangle m'$

► **Lemma C.16.** *Let $m, m' \in M$ and $v \in A^+$ such that $\mu(v)$ is an idempotent. There exists a polynomial $P : \mathbb{N} \rightarrow \mathbb{N}$ of degree at most 2 such that $\forall Z \geq 4$:*

$$\text{prod}(m\langle v^Z \rangle m') = P(Z).$$

Proof. We claim that for all $Z \geq 4$:

$$\begin{aligned} \text{prod}(m\langle v^Z \rangle m') &= \text{prod}(m\langle v \rangle f m') + \text{prod}(m\langle v \rangle \langle v \rangle f m') + \text{prod}(m\langle v \rangle f \langle v \rangle f m') \\ &\quad + \text{prod}(m f \langle v \rangle \langle v \rangle m') + \text{prod}(m f \langle v \rangle) \\ &\quad + (Z-3) \text{prod}(m f \langle v \rangle f \langle v \rangle m') + (Z-3) \text{prod}(m\langle v \rangle f \langle v \rangle f m') \\ &\quad + (Z-3) \text{prod}(m f \langle v \rangle \langle v \rangle f m') + (Z-2) \text{prod}(m f \langle v \rangle f m') \\ &\quad + \frac{(Z-4)(Z-3)}{2} \text{prod}(m f \langle v \rangle f \langle v \rangle f m'). \end{aligned} \quad (2)$$

In order to show Equation 2, we first decompose the production of v^Z in several blocks, which are for $1 \leq i \leq j \leq Z$ "the production on the i -th v when called from the j -th v ":

$$\begin{aligned} \text{prod}(m\langle v^Z \rangle m') &= \sum_{1 \leq i < j \leq Z} \text{prod}(m\mu(v^{i-1})\langle v \rangle \mu(v^{j-i-1})\langle v \rangle \mu(v^{Z-j})m') \\ &\quad + \sum_{1 \leq j \leq Z} \text{prod}(m\mu(v^{j-1})\langle v \rangle \mu(v^{Z-j})m') \end{aligned} \quad (3)$$

Finally, we can obtain Equation 2 by regrouping the terms that occur in the right member of Equation 3, depending on their bitype. In order to convince the reader without burdening the proof, we decomposed $\text{prod}(m\langle v^7 \rangle m')$ in Figure 9, using a matrix which describes the

XX:26 Pebble transducers with unary output

terms of Equation 3: in cell (i, j) for $i \leq j$, we have written "the bitype of the i -th v when called from the j -th v ". The bitypes which are equal are identified with the same colors. It is not hard to see that we get the coefficients of Equation 2. \blacktriangleleft

Proof of Lemma C.5. We have $\mu(v_1) = f_1$, $\mu(v_2) = f_2$, $\mu(v_3) = f_3$. Define $m_0 := \mu(w_0)$, $m_1 := \mu(w_1)$, $m_2 := \mu(w_2)$, $m_3 := \mu(w_3)$. By batching the productions $\text{prod}(i, j)$ performed when computing $f(W(Z_1, Z_2, Z_3))$, we obtain Equation 4 for all $Z_1, Z_2, Z_3 \geq 4$. Applying the claims C.12, C.13, C.14 and Lemma C.16 to the various terms of this equation yields Lemma C.5. The framed terms give the monomials in $Z_1 Z_2$, $Z_1 Z_3$ and $Z_2 Z_3$.

$$\begin{aligned}
f(W(Z_1, Z_2, Z_3)) = & \text{prod}(\langle \mathbf{w}_0 \rangle f_1 m_1 f_2 m_2 f_3 m_3) + \text{prod}(\langle \mathbf{w}_0 \rangle \langle \mathbf{v}_1^{Z_1} \rangle m_1 f_2 m_2 f_3 m_3) \\
& + \text{prod}(\langle \mathbf{w}_0 \rangle f_1 \langle \mathbf{w}_1 \rangle f_2 m_2 f_3 m_3) + \text{prod}(\langle \mathbf{w}_0 \rangle f_1 m_1 \langle \mathbf{v}_2^{Z_2} \rangle m_2 f_3 m_3) \\
& + \text{prod}(\langle \mathbf{w}_0 \rangle f_1 m_1 f_2 \langle \mathbf{w}_2 \rangle f_3 m_3) + \text{prod}(\langle \mathbf{w}_0 \rangle f_1 m_1 f_2 m_2 \langle \mathbf{v}_3^{Z_3} \rangle m_3) \\
& + \text{prod}(\langle \mathbf{w}_0 \rangle f_1 m_1 f_2 m_2 f_3 \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle m_1 f_2 m_2 f_3 m_3) + \text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle \langle \mathbf{w}_1 \rangle f_2 m_2 f_3 m_3) \\
& + \boxed{\text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle m_1 \langle \mathbf{v}_2^{Z_2} \rangle m_2 f_3 m_3)} + \text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle m_1 f_2 \langle \mathbf{w}_2 \rangle f_3 m_3) \\
& + \boxed{\text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle m_1 f_2 m_2 \langle \mathbf{v}_3^{Z_3} \rangle m_3)} + \text{prod}(m_0 \langle \mathbf{v}_1^{Z_1} \rangle m_1 f_2 m_2 f_3 \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 f_1 \langle \mathbf{w}_1 \rangle f_2 m_2 f_3 m_3) + \text{prod}(m_0 f_1 \langle \mathbf{w}_1 \rangle \langle \mathbf{v}_2^{Z_2} \rangle m_2 f_3 m_3) \\
& + \text{prod}(m_0 f_1 \langle \mathbf{w}_1 \rangle f_2 \langle \mathbf{w}_2 \rangle f_3 m_3) + \text{prod}(m_0 f_1 \langle \mathbf{w}_1 \rangle f_2 m_2 \langle \mathbf{v}_3^{Z_3} \rangle m_3) \\
& + \text{prod}(m_0 f_1 \langle \mathbf{w}_1 \rangle f_2 m_2 f_3 \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 f_1 m_1 \langle \mathbf{v}_2^{Z_2} \rangle m_2 f_3 m_3) + \text{prod}(m_0 f_1 m_1 \langle \mathbf{v}_2^{Z_2} \rangle \langle \mathbf{w}_2 \rangle f_3 m_3) \\
& + \boxed{\text{prod}(m_0 f_1 m_1 \langle \mathbf{v}_2^{Z_2} \rangle m_2 \langle \mathbf{v}_3^{Z_3} \rangle m_3)} + \text{prod}(m_0 f_1 m_1 \langle \mathbf{v}_2^{Z_2} \rangle m_2 f_3 \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 f_1 m_1 f_2 \langle \mathbf{w}_2 \rangle f_3 m_3) + \text{prod}(m_0 f_1 m_1 f_2 \langle \mathbf{w}_2 \rangle \langle \mathbf{v}_3^{Z_3} \rangle m_3) \\
& + \text{prod}(m_0 f_1 m_1 f_2 \langle \mathbf{w}_2 \rangle f_3 \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 f_1 m_1 f_2 m_2 \langle \mathbf{v}_3^{Z_3} \rangle m_3) + \text{prod}(m_0 f_1 m_1 f_2 m_2 \langle \mathbf{v}_3^{Z_3} \rangle \langle \mathbf{w}_3 \rangle) \\
& + \text{prod}(m_0 f_1 m_1 f_2 m_2 f_3 \langle \mathbf{w}_3 \rangle)
\end{aligned} \tag{4}$$

D Proof of Lemma 5.14

We show that $\text{Dep}(\mathcal{F}) \uplus \{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})\}$ partitions $\text{Nodes}(\mathcal{F})$. Applying this result specifically to the leaves will directly give a partition of the positions of w by the frontiers of these nodes. The proof is done by induction on the structure of \mathcal{F} :

- if $\mathcal{F} = (a)$ is a leaf the result is true;
- if $\mathcal{F} = (\mathcal{F}_1) \dots (\mathcal{F}_n)$ then $\text{Dep}(\mathcal{F}) = \{\mathcal{F}\} \uplus \text{Dep}(\mathcal{F}_1) \uplus \text{Dep}(\mathcal{F}_n)$.

$$\text{Iterable-nodes}(\mathcal{F}) = \{\mathcal{F}_i : 2 \leq i \leq n-1\} \biguplus_{1 \leq i \leq n} \text{Iterable-nodes}(\mathcal{F}_i)$$

Furthermore, by induction hypothesis $\text{Dep}(\mathcal{F}_i) \uplus \{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}_i)\}$ partitions $\text{Nodes}(\mathcal{F}_i)$ for all $1 \leq i \leq n$, hence:

$$\begin{aligned}
 \text{Nodes}(\mathcal{F}) &= \{\mathcal{F}\} \biguplus_{1 \leq i \leq n} \text{Nodes}(\mathcal{F}_i). \\
 &= \{\mathcal{F}\} \biguplus_{1 \leq i \leq n} (\text{Dep}(\mathcal{F}_i) \uplus \{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}_i)\}) \\
 &= \{\mathcal{F}\} \uplus \text{Dep}(\mathcal{F}_1) \uplus \text{Dep}(\mathcal{F}_n) \\
 &\quad \biguplus_{2 \leq i \leq n-1} \text{Dep}(\mathcal{F}_i) \biguplus_{1 \leq i \leq n} \{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}_i)\} \\
 &= \text{Dep}(\mathcal{F}) \uplus \{\text{Dep}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})\}
 \end{aligned}$$

E Proof of Lemma 5.17

Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$, we show that:

$$f(w) = \sum_{\mathcal{I}, \mathcal{J} \in \text{Part}(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}).$$

We first note that by definition:

$$\sum_{\mathcal{I}, \mathcal{J} \in \text{Part}(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) = \sum_{\substack{\mathcal{I} \in \text{Part}(\mathcal{F}) \\ i \in \text{Fr}_{\mathcal{F}}(\mathcal{I})}} \sum_{\substack{\mathcal{J} \in \text{Part}(\mathcal{F}) \\ j \in \text{Fr}_{\mathcal{F}}(\mathcal{J}) \\ i \leq j}} \text{prod}(i, j)$$

But since by Lemma 5.14 $\{\text{Fr}_{\mathcal{F}}(\mathcal{I}) : \mathcal{I} \in \text{Part}(\mathcal{F})\}$ partitions the positions of w :

$$\sum_{\mathcal{I}, \mathcal{J} \in \text{Part}(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) = \sum_{1 \leq i \leq |w|} \sum_{i \leq j \leq |w|} \text{prod}(i, j)$$

Finally by definition of a 1-blind bimachine it follows:

$$\sum_{1 \leq i \leq j \leq |w|} \text{prod}(i, j) = f(w).$$

F Proof of Lemma 5.19

We show that each iterable node is the middle child of a unique basis. The result is obtained by induction on the structure of the factorization:

- if $\mathcal{F} = (a)$ is a leaf the result is true by emptiness;
- if $\mathcal{F} = (\mathcal{F}_1) \dots (\mathcal{F}_n)$ then $\text{Dep}(\mathcal{F}) = \{\mathcal{F}\} \uplus \text{Dep}(\mathcal{F}_1) \uplus \text{Dep}(\mathcal{F}_n)$.
 - by induction hypothesis, the iterable nodes of \mathcal{F}_1 (resp. \mathcal{F}_n) are the descendant of a middle child of a unique basis of \mathcal{F}_1 (resp. \mathcal{F}_n). These nodes are still bases of \mathcal{F} . Hence the iterable nodes of \mathcal{F}_1 and \mathcal{F}_n are the descendant of a unique basis of \mathcal{F} .
 - the iterable nodes from \mathcal{F}_i for $2 \leq i \leq n-1$ are the descendant of a middle child of a unique basis of \mathcal{F} , which is \mathcal{F} itself (if $n \geq 2$ it is an idempotent node).

G Further properties of factorizations

In this section, we develop several tools which are useful in our factorization techniques.

G.1 Representation of factorizations

The set $(\widehat{A})^+$ not only contains factorizations of height at most Λ , it also contains factorizations of bigger height, and other words which are not factorizations. However, such inputs will often be considered as "invalid" in our constructions, thus we need to isolate them.

► **Definition G.1.** Let $\text{Fact}_{\leq \Lambda}$ be the set of factorizations of height at most Λ .

$\text{Fact}_{\leq \Lambda}$ can also be seen as a language of $(\widehat{A})^*$. Using this point of view, the following lemma follows from an easy inductive construction.

► **Lemma G.2** (Folklore). $\text{Fact}_{\leq \Lambda}$ is a regular language of $(\widehat{A})^*$.

As a consequence, a function defined over $\text{Fact}_{\leq \Lambda}$ can be extended to $(\widehat{A})^+$, while preserving its computability by some bimachine. Thus in the proofs of lemmas 5.21, 5.23 and 5.24 we shall always consider that the inputs belong to $\text{Fact}_{\leq \Lambda}$.

► **Lemma G.3.** Let $k \geq 0$, $g : \text{Fact}_{\leq \Lambda} \rightarrow \mathbb{N}$ computed by a k -pebble (resp. k -blind, k -marble) bimachine, and:

$$h : (\widehat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} g(\mathcal{F}) & \text{if } \mathcal{F} \in \text{Fact}_{\leq \Lambda}; \\ 0 & \text{otherwise.} \end{cases}$$

Then one can build a k -pebble (resp. k -blind, k -marble) bimachine which computes h .

Proof idea. Using its monoid morphism, the main bimachine can check a regular property of their input and have a different behavior depending on it. ◀

G.2 Frontiers and dependencies

Size of the sets. As evoked in the main body of this paper, over factorizations of height at most Λ , dependencies and frontiers have a bounded size.

► **Lemma G.4.** Let $\mathcal{F} \in \text{Fact}_{\leq \Lambda}$ and $\mathcal{I} \in \text{Nodes}(\mathcal{F})$. Then $|\text{Dep}(\mathcal{I})|, |\text{Fr}_{\mathcal{F}}(\mathcal{I})| \leq 2^\Lambda$.

Proof. We show by induction that if \mathcal{I} has height at most h , then $|\text{Dep}(\mathcal{I})| \leq 2^h$. Thus here $|\text{Dep}(\mathcal{I})| \leq 2^\Lambda$. Considering the leaves gives the result for the frontier. ◀

Minimum and maximum. Intuitively, taking the dependency of a node \mathcal{I} consists in pruning the subtree \mathcal{I} , in a way that shall "preserve" some information on it. First, it preserves the "borders" of the words.

► **Definition G.5.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ and $\mathcal{I} \in \text{Nodes}(\mathcal{F})$. By construction, \mathcal{I} is a factorization of some portion of w between two positions denoted $\min_{\mathcal{F}}(\mathcal{I})$ and $\max_{\mathcal{F}}(\mathcal{I})$.

In other words, $\mathcal{I} \in \text{Fact}(w[\min_{\mathcal{F}}(\mathcal{I}) : \max_{\mathcal{F}}(\mathcal{I})])$.

► **Lemma G.6.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ and $\mathcal{I} \in \text{Nodes}(\mathcal{F})$. Let $\{i_1 < \dots < i_\ell\} := \text{Fr}_{\mathcal{F}}(\mathcal{I})$. Then $\min_{\mathcal{F}}(\mathcal{I}) = i_1$ and $\max_{\mathcal{F}}(\mathcal{I}) = i_\ell$.

Proof idea. By induction since $\text{Dep}(\mathcal{I})$ contains its rightmost and leftmost children. ◀

► **Remark G.7.** Thus $\min_{\mathcal{F}}(\mathcal{F}) = 1$ and $\max_{\mathcal{F}}(\mathcal{F}) = |w|$ since \mathcal{F} is a factorization of the whole w .

Values. More interestingly, the frontier preserves the image by μ . This explains why considering the frontier of \mathcal{I} is a sufficient abstraction of \mathcal{I} when looking at monoids elements.

► **Lemma G.8.** *Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ and $\mathcal{I} \in \text{Nodes}(\mathcal{F})$, then:*

$$\mu(w[\min_{\mathcal{F}}(\mathcal{I}) : \max_{\mathcal{F}}(\mathcal{I})]) = \mu(w[\text{Fr}_{\mathcal{F}}(\mathcal{I})]).$$

We denote this element $\text{value}(\mathcal{I})$.

Proof idea. By induction since $\text{Dep}(\mathcal{I})$ only removes "middle children" of \mathcal{I} . ◀

H

 Proof of Lemma 5.21 - pairs separated by the frontier of the root

We build a 1-blind bimachine computing:

$$f_D : (\hat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in D(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Using Lemma G.3, we restrict our construction to factorizations of height at most Λ . Let $\mathcal{F} \in \text{Fact}(w)$ of height at most Λ . Let $\{p_1 < \dots < p_{\ell(\mathcal{F})}\} := \text{Fr}_{\mathcal{F}}(\mathcal{F})$ be the frontier of its root node \mathcal{F} . By lemmas G.4 and G.6, $\ell(\mathcal{F}) \leq 2^\Lambda$ and $p_1 = 1$ and $p_{\ell(\mathcal{F})} = |w|$.

Intuitively, the p_k split the word w in at most $\ell(\mathcal{F})-1$ blocks (some of them can be empty), and if $(\mathcal{I}, \mathcal{J}) \in D(\mathcal{F})$, then $\text{Fr}_{\mathcal{F}}(\mathcal{I})$ and $\text{Fr}_{\mathcal{F}}(\mathcal{J})$ are in two distinct blocks.

▷ **Claim H.1.** The following holds:

$$f_D(\mathcal{F}) = \sum_{\substack{1 \leq k' < k < \ell(\mathcal{F}) \\ p_{k'} < i < p_{k'+1} \\ p_k < j < p_{k+1}}} \text{prod}(i, j).$$

Proof. We first show that $\forall \mathcal{I}, \mathcal{J} \in \text{Iterable-nodes}(\mathcal{F})$, $\text{basis}_{\mathcal{F}}(\mathcal{I}) = \text{basis}_{\mathcal{F}}(\mathcal{J})$ if and only if there exists k such that $p_k < \text{Fr}_{\mathcal{F}}(\mathcal{I}) < p_{k+1}$ and $p_k < \text{Fr}_{\mathcal{F}}(\mathcal{J}) < p_{k+1}$. Second, using Lemma 5.14 get for all $1 \leq i \leq |w|$ that $i \notin \text{Fr}_{\mathcal{F}}(\mathcal{F})$ if and only if $i \in \text{Fr}_{\mathcal{F}}(\mathcal{I})$ for some $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$. The result follows by rewriting the sum which defines f_D . ◀

We now use Claim H.1 to compute f_D using a 1-blind bimachine in Algorithm 1. Intuitively, the algorithm ranges over all possible j and for each of them, it calls an external function which ranges over all i in blocks "on the left", and produces the $\text{prod}(i, j)$.

We use the notation of *bitypes*, see Definition 5.2 on page 8. Note that by definition $\text{prod}(i, j) = \text{prod}(\mu(w[1:i-1])\langle \mathbf{w}[i] \rangle \mu(w[i+1:j-1])\langle \mathbf{w}[j] \rangle \mu(w[j+1:|w|]))$.

Correctness of the algorithm. We first claim that if \mathcal{F} is a factorization of w and $k < \ell(\mathcal{F})$, then $f_k^{m\langle \mathbf{a} \rangle n}(\mathcal{F})$ computes the sum of productions over all i which are in a block $k' < k$, when called from a j in block k and in "context" m, n . Formally:

$$f_k^{m\langle \mathbf{a} \rangle n}(\mathcal{F}) = \sum_{\substack{1 \leq k' < k \\ p_{k'} < i < p_{k'+1}}} \text{prod}(\mu(w[1:i-1])\langle \mathbf{w}[i] \rangle \mu(w[i+1:p_k])m\langle \mathbf{a} \rangle n\mu(w[p_{k+1}:|w|])).$$

Finally with Claim H.1, it is easy to show that $\text{Main}(\mathcal{F}) = f_D(\mathcal{F})$.

■ **Algorithm 1** Computing f_D with a 1-blind bimachine

```

Function Main( $\mathcal{F}$ )
   $w \leftarrow$  word factored by  $\mathcal{F}$ ;
  for  $j$  in  $\{1, \dots, |w|\}$  do
    if  $p_k < j < p_{k+1}$  for some  $1 \leq k < \ell(\mathcal{F})$  then
       $a \leftarrow w[j]$ ; /* Letter in position  $j$ . */
       $m \leftarrow \mu(w[p_k+1:j-1])$ ; /* Left "context" of  $j$  in block  $k$ . */
       $n \leftarrow \mu(w[j+1:p_{k+1}-1])$ ; /* Right "context" of  $j$  in block  $k$ . */
      Output  $f_k^{m\langle a \rangle n}(\mathcal{F})$ 
    end
  end

Function  $f_k^{m\langle a \rangle n}(\mathcal{F})$ 
   $w \leftarrow$  word factored by  $\mathcal{F}$ ;
  for  $i$  in  $\{1, \dots, |w|\}$  do
    if  $p_{k'} < i < p_{k'+1}$  for some  $0 \leq k' < k$  then
       $a' \leftarrow w[i]$ ; /* Letter in position  $i$ . */
       $m' \leftarrow \mu(w[p_{k'}+1:i-1])$ ; /* Left "context" of  $i$  in block  $k'$ . */
       $n' \leftarrow \mu(w[i+1:p_{k'+1}-1])$ ; /* Right "context" of  $i$  in block  $k'$ . */
       $m_1 \leftarrow \mu(w[1:p_{k'}])$ ; /* Context before block  $k'$ . */
       $m_2 \leftarrow \mu(w[p_{k'}+1:p_k])$ ; /* Context between blocks  $k'$  and  $k$ . */
       $m_3 \leftarrow \mu(w[p_{k+1}:|w|])$ ; /* Context between blocks  $k'$  and  $k$ . */
      Output  $\text{prod}(m_1(m'\langle a' \rangle n')m_2(m\langle a \rangle n)m_3)$ 
    end
  end

```

Implementation by a 1-blind bimachine. We justify how Algorithm 1 can be implemented with a 1-blind bimachine. It uses finitely many external functions $f_k^{m\langle a \rangle n}$ for $1 \leq k \leq 2^\Lambda - 1$, $m, n \in M$ and $a \in A$ and uses its morphism to "detect" the frontier of the root (recall that the frontier has a bounded size). This detection is detailed below.

► **Definition H.2.** Given $w \in A^+$ and $\mathcal{F} \in \text{Fact}(w)$, let $\pi : \{1, \dots, |w|\} \rightarrow \{1, \dots, |\mathcal{F}|\}$ be the function mapping a position of w to the corresponding leaf of \mathcal{F} , seen as a position.

► **Remark H.3.** The image of π is exactly the set of positions of \mathcal{F} labelled by A .

We explain how to "detect" if the current position belongs to the frontier of the root.

▷ **Claim H.4.** One can build a regular language $P \subseteq (\hat{A} \uplus \hat{A})^*$ such that for all $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ of height at most Λ and $1 \leq j \leq |w|$:

$$\mathcal{F} \uparrow \pi(j) \in P \text{ if and only if } j \in \text{Fr}_{\mathcal{F}}(\mathcal{F}).$$

Proof. We build by induction a language P^h which works for factorizations of height at most h . For $h = 1$, $P^1 := \{\underline{a} : a \in A\}$. Assume that $h \geq 2$ and P^{h-1} is built. Then we define $P^h := P^{h-1} \cup \{(\mathcal{F}_1) \dots (\mathcal{F}_n) : n \geq 0 \text{ and } (\mathcal{F}_1 \in P^h \text{ or } \mathcal{F}_n \in P^h)\}$. ◀

In a similar way, it is possible to detect the number of the current block, and describe the "context" of the current position in this block.

▷ **Claim H.5.** One can build a regular language $P_k^{m,n} \subseteq (\widehat{A} \uplus \underline{A})^*$ such that for all $w \in A^+$, $\mathcal{F} \in \text{Fact}_{\leq \Lambda}$ over w and $1 \leq j \leq |w|$:

$$\begin{aligned} \mathcal{F} \uparrow \pi(j) &\in P_k^{m,n} \text{ if and only if} \\ p_k < j < p_{k+1} \text{ and } \mu(w[p_k+1:j-1]) &= m \text{ and } \mu(w[j+1:p_{k+1}-1]) = n. \end{aligned}$$

Using Claim H.5, we can implement the function Main using a bimachine with external blind functions. Its morphism is built from the regular languages $P_k^{m,n}$ for $1 \leq k \leq 2^\Lambda$ and $m, n \in M$, and it calls the corresponding $\mathfrak{f}_k^{m(\mathbf{a})n}$ as external function.

The construction of a bimachine the $\mathfrak{f}_k^{m(\mathbf{a})n}$ is similar.

I Proof of Lemma 5.23 - linked pairs

We want to build a 0-blind bimachine computing:

$$f_L : (\widehat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

We shall build a 1-pebble bimachine computing f_L . Why not a 0-blind? Because from a 1-pebble bimachine we can build an equivalent 1-marble bimachine (Corollary 4.5). But as claimed below, f_L has a "linear growth", hence the results of [6] (which "minimize" the number of marbles used) will automatically build a 0-marble (= 0-blind) bimachine³.

▷ **Claim I.1.** $f_L(\mathcal{F}) = \mathcal{O}(|\mathcal{F}|)$.

Proof. Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ of height at most Λ (by definition of f_L , it is enough to only consider such inputs). Given $\mathcal{I} \in \text{Nodes}(\mathcal{F})$, we denote by $\text{Up}(\mathcal{I})$ the set of nodes $\mathcal{J} \in \text{Part}(\mathcal{F})$ which are either an ancestor of \mathcal{I} , or the right/left sibling of an ancestor of \mathcal{I} . Since the height is at most Λ , it follows that $|\text{Up}(\mathcal{I})| \leq 3\Lambda$. Furthermore:

$$f_L(\mathcal{F}) = \sum_{\mathcal{I} \in \text{Part}(\mathcal{F})} \left(\text{prod}(\mathcal{I}, \mathcal{I}) + \sum_{\mathcal{J} \in \text{Up}(\mathcal{I}) \setminus \{\mathcal{I}\}} \text{prod}(\mathcal{I}, \mathcal{J}) + \text{prod}(\mathcal{J}, \mathcal{I}) \right).$$

Since the frontiers have size at most 2^Λ (Lemma G.4), there exists $B \geq 0$ independent from \mathcal{F} and w such that $\text{prod}(\mathcal{I}, \mathcal{J}) \leq B$ for all $\mathcal{I}, \mathcal{J} \in \text{Nodes}(\mathcal{F})$.

Finally $f_L(\mathcal{F}) \leq |\mathcal{F}| \times (1 + 2 \times 3\Lambda) \times B$. ◀

It remains to describe how f_L can be computed by a 1-pebble bimachine. We restrict our construction to factorizations of height at most Λ (using Lemma G.3). The 1-pebble bimachine is described by Algorithm 2. Intuitively, it ranges over all possible i and j positions of w , and for each pair, it checks whether $i \in \text{Fr}_{\mathcal{F}}(\mathcal{I})$ and $j \in \text{Fr}_{\mathcal{F}}(\mathcal{J})$ for some $(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})$. If it is the case, it outputs $\text{prod}(i, j)$. Note that we use an external *pebble* function and not a blind one: it can "see" the calling position i on the input \mathcal{F} .

Correctness of Algorithm 2 follows from the definitions. To see how it can be implemented by a 1-pebble bimachine, we only need to show that the condition $(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})$ can be checked using a regular language, as claimed below. We re-use the notation π from the proof of Lemma 5.21. The claim follows from similar constructions of regular languages.

³ A direct construction of a 0-blind bimachine is possible, but somehow more complex to explain.

XX:32 Pebble transducers with unary output

■ **Algorithm 2** Computing f_L with a 1-pebble bimachine

```

Function Main( $\mathcal{F}$ )
   $w \leftarrow$  word factored by  $\mathcal{F}$ ;
  for  $i$  in  $\{1, \dots, |w|\}$  do
    | Output  $\mathfrak{f}(\mathcal{F}, i)$ 
  end

Function  $\mathfrak{f}(\mathcal{F}, i)$ 
   $w \leftarrow$  word factored by  $\mathcal{F}$ ;
  for  $j$  in  $\{1, \dots, |w|\}$  do
    |  $\mathcal{I} \leftarrow$  unique node in  $\text{Part}(\mathcal{F})$  such that  $i \in \text{Fr}_{\mathcal{F}}(\mathcal{I})$ ;
    |  $\mathcal{J} \leftarrow$  unique node in  $\text{Part}(\mathcal{F})$  such that  $j \in \text{Fr}_{\mathcal{F}}(\mathcal{J})$ ;
    | if  $(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})$  then
      | | Output  $\text{prod}(i, j)$ 
    | end
  end

```

▷ **Claim 1.2.** One can build a regular language $P \subseteq (\widehat{A} \uplus \underline{A})^*$ such that for all $w \in A^+$, $1 \leq i \leq j \leq |w|$ and factorization \mathcal{F} of height at most Λ over w :

$\mathcal{F} \uparrow \pi(i) \uparrow \pi(j) \in P$ if and only if

when $\mathcal{I}, \mathcal{J} \in \text{Part}(\mathcal{F})$ are such that $i \in \text{Fr}_{\mathcal{F}}(\mathcal{I}), j \in \text{Fr}_{\mathcal{F}}(\mathcal{J})$, we have $(\mathcal{I}, \mathcal{J}) \in L(\mathcal{F})$.

J Proof of Lemma 5.24 - Independent pairs

Assume that \mathcal{T} is symmetrical. We build a 1-blind bimachine computing:

$$f_I : (\widehat{A})^+ \rightarrow \mathbb{N}, \mathcal{F} \mapsto \begin{cases} \sum_{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})} \text{prod}(\mathcal{I}, \mathcal{J}) & \text{if } \mathcal{F} \text{ factorization of height at most } \Lambda; \\ 0 & \text{otherwise.} \end{cases}$$

Once more, using Lemma G.3, we only consider "valid" inputs.

As explained in the body of this paper, we shall see that since \mathcal{T} is symmetrical, $\text{prod}(\mathcal{I}, \mathcal{J})$ does not depend on the relative position of \mathcal{I} and \mathcal{J} , when $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$. It remains to formalize this intuition by defining the *type*. We use the notations of Subsection G.2.

Left and right. In this paragraph, we define the "left" and "right" of a node, as the monoid elements which are "before" and "after" it.

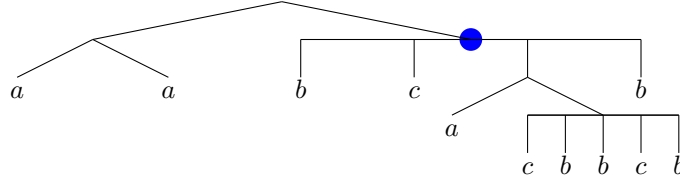
► **Definition J.1.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$, and $\mathcal{I} \in \text{Nodes}(\mathcal{F})$. We define the left and right of \mathcal{I} in \mathcal{F} as follows:

$$\begin{cases} \text{left}_{\mathcal{F}}(\mathcal{I}) := \mu(w[1: \min_{\mathcal{F}}(\mathcal{I}) - 1]) \\ \text{right}_{\mathcal{F}}(\mathcal{I}) := \mu(w[\max_{\mathcal{F}}(\mathcal{I}) + 1: |w|]) \end{cases}$$

► **Remark J.2.** $\text{left}_{\mathcal{F}}(\mathcal{F}) = \text{right}_{\mathcal{F}}(\mathcal{F}) = \mu(\varepsilon)$.

► **Example J.3.** In the factorization \mathcal{F} of Figure 10, we consider the basis $\mathcal{I} \in \text{Nodes}(\mathcal{F})$ colored in blue. Then $\text{left}_{\mathcal{F}}(\mathcal{I}) = \mu(w[1:2]) = \mu(aa)$ and $\text{right}_{\mathcal{F}}(\mathcal{I}) = \mu(w[12:11]) = \mu(\varepsilon)$.

To get our result, we shall define a more precise abstraction of the position of an iterable node, that we call its *type*.



■ **Figure 10** The factorization $(aa)(bc(a(cbbcb))b)$ of $aabcacbbcb$

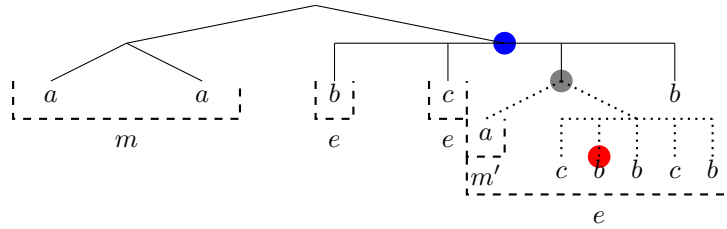
Types. Let $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$, it is the descendant of a middle child of $\text{basis}_{\mathcal{F}}(\mathcal{I})$. We denote this middle child $\text{middle}_{\mathcal{F}}(\mathcal{I})$ (note that it is an iterable node). Intuitively, the *type* of \mathcal{I} is an abstraction of the left and right of $\text{basis}_{\mathcal{F}}(\mathcal{I})$, plus the context of the idempotent parent of \mathcal{I} within $\text{middle}_{\mathcal{F}}(\mathcal{I})$ (seen as a subtree).

► **Definition J.4.** Let $w \in A^+$, $\mathcal{F} \in \text{Fact}(w)$ and $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$. We define $\text{type}_{\mathcal{F}}(\mathcal{I}) = (d, m, n, e, m', n', u) \in \mathbb{N} \times M^5 \times A^+$ as follows:

- $1 \leq d$ is the depth of the node \mathcal{I} in \mathcal{F} ;
- $m := \text{left}_{\mathcal{F}}(\text{basis}_{\mathcal{F}}(\mathcal{I}))$ and $n := \text{right}_{\mathcal{F}}(\text{basis}_{\mathcal{F}}(\mathcal{I}))$;
- $e := \text{value}(\text{middle}_{\mathcal{F}}(\mathcal{I})) = \text{value}(\text{basis}_{\mathcal{F}}(\mathcal{I}))$ (it is an idempotent);
- if $\text{middle}_{\mathcal{F}}(\mathcal{I}) = \mathcal{I}$ (i.e. \mathcal{I} is a middle children of its basis), $m' := n' := \mu(\varepsilon)$;
- if $\text{middle}_{\mathcal{F}}(\mathcal{I}) \neq \mathcal{I}$ (i.e. $\mathcal{I} \in \text{Iterable-nodes}(\text{middle}_{\mathcal{F}}(\mathcal{I}))$), let \mathcal{P} be its parent, then $m' := \text{left}_{\text{middle}_{\mathcal{F}}(\mathcal{I})}(\mathcal{P})$ and $n' := \text{right}_{\text{middle}_{\mathcal{F}}(\mathcal{I})}(\mathcal{P})$;
- $u = w[\text{Fr}_{\mathcal{F}}(\mathcal{I})]$.

We denote by $T := [1, \Lambda] \times M^5 \times \{w \in A^+ : |w| \leq 2^\Lambda\}$. This set is finite, furthermore, if \mathcal{F} has height at most Λ then $\text{type}_{\mathcal{F}}(\mathcal{I}) \in T$.

► **Example J.5.** In the factorization \mathcal{F} of Figure 11, we consider $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$ colored in red. Then $\text{basis}_{\mathcal{F}}(\mathcal{I})$ is colored in blue and $\text{middle}_{\mathcal{F}}(\mathcal{I})$ is colored in gray (the associated subtree is dotted). Then $\text{type}_{\mathcal{F}}(\mathcal{I}) = (d, m, n, e, m', n', u)$ where $d = 5$; $m = \mu(aa)$; $n = \mu(\varepsilon)$; $e = \mu(bcacbbcb) = \mu(b) = \mu(c) = \mu(acbbcb)$; $m' = \mu(a)$; $n' = \mu(\varepsilon)$; $u = b$.



■ **Figure 11** The factorization $(aa)(bc(a(cbbcb))b)$ of $aabcacbbcb$

Relating types and bitypes. We now show that when $\mathcal{I}, \mathcal{J} \in I(\mathcal{F})$, then $\text{prod}(\mathcal{I}, \mathcal{J})$ only depends on the types of \mathcal{I} and \mathcal{J} . This is the purpose of Lemma J.8 below, whose proof (given in the next subsection) crucially relies on the symmetry of \mathcal{T} .

► **Definition J.6.** If $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$, we write $\mathcal{I} < \mathcal{J}$ if $\max_{\mathcal{F}}(\mathcal{I}) < \min_{\mathcal{F}}(\mathcal{J})$.

► **Remark J.7.** Since $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$, \mathcal{I} and \mathcal{J} are not on the same branch of \mathcal{F} , thus either $\mathcal{I} < \mathcal{J}$ or $\mathcal{J} < \mathcal{I}$ (their frontiers cannot be interleaved).

XX:34 Pebble transducers with unary output

In the following, we write $\{\tau_1, \tau_2\} \subseteq T$ to describe a set of 2 or 1 elements of T . Caution: we can have $\tau_1 = \tau_2$; this abuse of notation makes the statements more readable.

► **Lemma J.8.** *Let $\{\tau_1, \tau_2\} \subseteq T$. There exists $K \geq 0$ such that for all factorization \mathcal{F} of height at most Λ , if $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$, $\mathcal{I} < \mathcal{J}$ and $\{\tau_1, \tau_2\} = \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\}$ then $\text{prod}(\mathcal{I}, \mathcal{J}) = K$. We thus define $\text{prod}(\{\tau_1, \tau_2\}) := K$.*

Proof. See Subsection J.1. ◀

► **Remark J.9.** Since we use sets, $\text{prod}(\{\tau_1, \tau_2\}) = \text{prod}(\{\tau_2, \tau_1\})$. The symmetry of the definition means that the calls from τ_1 to τ_2 give the same production as those from τ_2 to τ_1 .

Lemma J.8 still holds if we let the depths of the nodes be variable. However, the depths will be used for the computation by a 1-blind bimachine.

Computing f_I with types. We can now decompose f_I as a linear combination. Recall that in our notations, we can have $\tau_1 = \tau_2$. Equation 5 follows by partitioning the sum defining f_I depending on $\{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\}$ and applying Lemma J.8.

$$\begin{aligned} f_I(\mathcal{F}) &= \sum_{\{\tau_1, \tau_2\} \subseteq T} \sum_{\substack{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) \\ \mathcal{I} < \mathcal{J} \\ \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\} = \{\tau_1, \tau_2\}}} \text{prod}(\mathcal{I}, \mathcal{J}) \\ &= \sum_{\{\tau_1, \tau_2\} \subseteq T} f_{\{\tau_1, \tau_2\}} \times \text{prod}(\{\tau_1, \tau_2\}) \end{aligned} \quad (5)$$

where $f_{\{\tau_1, \tau_2\}} := |\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \mathcal{I} < \mathcal{J}, \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\} = \{\tau_1, \tau_2\}\}|$.

Since the number of sets $\{\tau_1, \tau_2\} \subseteq T$ is bounded, we only need to describe how to compute each $f_{\{\tau_1, \tau_2\}}$. Indeed, we can recombine them using Claim J.10.

▷ **Claim J.10.** Let $\alpha, \beta \in \mathbb{N}$ and $g, h : B^* \rightarrow \mathbb{N}$ computable by a k -blind bimachine, so is:

$$\alpha g + \beta h : w \mapsto \alpha \times g(w) + \beta \times h(w).$$

Proof idea. We build a k -blind transducer which simulates sequentially g , and then h . ◀

From now we fix $\tau_1 = (d_1, m_1, n_1, e_1, m'_1, n'_1, u_1)$ and $\tau_2 = (d_2, m_2, n_2, e_2, m'_2, n'_2, u_2)$. We assume that $\tau_1 \neq \tau_2$ (the case of equality is similar and even easier). Up to switching 1 and 2, we assume that $d_1 \geq d_2$, that is τ_1 is "deeper" than τ_2 . Then:

$$\begin{aligned} f_{\{\tau_1, \tau_2\}}(\mathcal{F}) &= |\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \mathcal{I} < \mathcal{J}, \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\} = \{\tau_1, \tau_2\}\}| \\ &= \frac{1}{2} |\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\} = \{\tau_1, \tau_2\}\}| \\ &= |\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1, \text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2\}| \end{aligned}$$

The second line is justified since the function $\sigma = (\mathcal{I}, \mathcal{J}) \mapsto (\mathcal{J}, \mathcal{I})$ is an involution without fixpoint (since $(\mathcal{I}, \mathcal{I}) \notin I(\mathcal{F})$) of the set $\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\} = \{\tau_1, \tau_2\}\}$ which reverses the ordering of the pair (that is, if $\mathcal{I} < \mathcal{J}$ then $\mathcal{J} > \mathcal{I}$). For the third line, we similarly use σ since it reverses the types $\text{type}_{\mathcal{F}}(\mathcal{I})$ and $\text{type}_{\mathcal{F}}(\mathcal{J})$ (we use $\tau_1 \neq \tau_2$ here).

By partitioning the last set, we then get:

$$\begin{aligned}
 f_{\{\tau_1, \tau_2\}}(\mathcal{F}) &= \sum_{\substack{\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}) \\ \text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1}} |\{(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}) : \text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2\}| \\
 &= \sum_{\substack{\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}) \\ \text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1}} |\{\mathcal{J} : (\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}), \text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2\}|
 \end{aligned} \tag{6}$$

For all \mathcal{I} , we thus count all the $\mathcal{J} \in \text{Iterable-nodes}(\mathcal{F})$ such that $\text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2$ and $\text{basis}_{\mathcal{F}}(\mathcal{J}) = \text{basis}_{\mathcal{F}}(\mathcal{I})$, except those which are in "nearly" on the same branch as \mathcal{I} . However this set of \mathcal{J} seems to depend on \mathcal{I} , which we want to avoid.

We want to rewrite Equation 6 in order to remove this dependency.

▷ **Claim J.11.** Let \mathcal{F} of height at most Λ , for all $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$ with $\text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1$; then $\{\mathcal{J} : (\mathcal{I}, \mathcal{J}) \in I(\mathcal{F}), \text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2\} = J_{\mathcal{F}}(\text{basis}_{\mathcal{F}}(\mathcal{I})) \setminus A_{\mathcal{F}}(\mathcal{I})$ where:

- $J_{\mathcal{F}}(\mathcal{B}) := \{\mathcal{J} \in \text{Iterable-nodes}(\mathcal{F}) : \text{basis}_{\mathcal{F}}(\mathcal{J}) = \mathcal{B}, \text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2\};$
- $A_{\mathcal{F}}(\mathcal{I})$ is the set of nodes \mathcal{J} such that $\text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2$, $\text{basis}_{\mathcal{F}}(\mathcal{J}) = \text{basis}_{\mathcal{F}}(\mathcal{I})$ and \mathcal{J} is either an ancestor of \mathcal{I} , or the right/left sibling of an ancestor of \mathcal{I} .

Proof. Let \mathcal{I} be fixed as in the claim. Let $\mathcal{J} \in J_{\mathcal{F}}(\mathcal{I})$. Since $d_1 \geq d_2$, then \mathcal{I} is "deeper" than \mathcal{J} and either $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$; or \mathcal{J} is either an ancestor of \mathcal{I} , or the right/left sibling of an ancestor of \mathcal{I} . This case disjunction gives a disjoint union. ◀

Using Claim J.11 and Equation 6, we finally get:

$$f_{\{\tau_1, \tau_2\}}(\mathcal{F}) = \sum_{\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})} |J_{\mathcal{F}}(\text{basis}_{\mathcal{F}}(\mathcal{I}))| - |A_{\mathcal{F}}(\mathcal{I})| \tag{7}$$

Let us explain why Equation 7 is satisfactory. First, the set $J_{\mathcal{F}}(\text{basis}_{\mathcal{F}}(\mathcal{I}))$ does not depend on \mathcal{I} , but only on $\text{basis}_{\mathcal{F}}(\mathcal{I})$, which is a bounded information that can be given to an external blind function. Second, the set $A_{\mathcal{F}}(\mathcal{I})$ still depends on \mathcal{I} , but:

▷ **Claim J.12.** $|A_{\mathcal{F}}(\mathcal{I})| \leq 3$

Proof. The nodes of $A_{\mathcal{F}}(\mathcal{I})$ have type τ_2 . Thus $A_{\mathcal{F}}(\mathcal{I})$ is included in the set containing the ancestor of \mathcal{I} at height d_2 , and its potential right/left siblings. ◀

Hence, it is also a bounded information which can be given to a external function. We can now describe in Algorithm 3 how to compute f_{τ_1, τ_2} with a 1-blind bimachine.

Correctness of the algorithm. It follows by definition that $\mathfrak{f}_{\mathcal{B}}^{-c}(\mathcal{F}) = \max(0, |J_{\mathcal{B}}(\mathcal{F})| - c)$. Using Equation 7, we get $\text{Main}(\mathcal{F}) = f_{\{\tau_1, \tau_2\}}(\mathcal{F})$.

Implementation by a 1-blind bimachine. We justify how Algorithm 3 can be implemented with a 1-blind bimachine. First, it uses finitely many external functions $\mathfrak{f}_{\mathcal{B}}^{-c}$ for $0 \leq c \leq 3$, and \mathcal{B} a base of the forest (there is a bounded number of bases).

It seems however that \mathcal{B} depends on the input \mathcal{F} , but we can naturally order the bases depending on "the first position below an iterable node having this base":

▷ **Claim J.13.** The bases of \mathcal{F} can be ordered $\mathcal{B}_1, \dots, \mathcal{B}_{\ell(\mathcal{F})}$, in the strictly increasing order of the values $\min \{\min_{\mathcal{I}}(\mathcal{I}) : \mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}) \text{ and } \text{basis}_{\mathcal{F}}(\mathcal{I}) = \mathcal{B}_i\}$.

■ **Algorithm 3** Computing $f_{\{\tau_1, \tau_2\}}$ with a 1-blind bimachine

```

Function Main( $\mathcal{F}$ )
  for  $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$  do
    if  $\text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1$  then
       $c \leftarrow |A_{\mathcal{F}}(\mathcal{I})|$ ; /* Number of nodes of type  $\tau_2$  "above"  $\mathcal{I}$ . */
       $\mathcal{B} \leftarrow \text{basis}_{\mathcal{F}}(\mathcal{I})$ ; /* Base of  $\mathcal{I}$ . */
      Output  $\mathfrak{f}_{\mathcal{B}}^{-c}(\mathcal{F})$ 
    end
  end

Function  $\mathfrak{f}_{\mathcal{B}}^{-c}(\mathcal{F})$ 
  counter  $\leftarrow 0$ ;
  for  $\mathcal{J} \in \text{Iterable-nodes}(\mathcal{F})$  do
    if  $\text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2$  and  $\text{basis}_{\mathcal{F}}(\mathcal{J}) = \mathcal{B}$  then
      if counter  $< c$  then
        counter  $\leftarrow$  counter + 1; /* No output on the  $c$  first nodes. */
      else
        Output 1
      end
    end
  end

```

Proof. The set $S_{\mathcal{B}} := \{\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F}) \text{ and } \text{basis}_{\mathcal{F}}(\mathcal{I}) = \mathcal{B}\}$ is not empty since idempotent nodes have at least one middle children. We only need to show that $S_{\mathcal{B}}$ are $S_{\mathcal{B}'}$ are disjoint when $\mathcal{B} \neq \mathcal{B}'$, what is clear since every iterable node has a unique base. ◀

Hence we do not really index the external function by the bases, but by their number in this ordering (the range of numbers is bounded independently from \mathcal{F}).

The second issue is that Algorithm 3 iterates on $\text{Iterable-nodes}(\mathcal{F})$, but not on the positions $\{1, \dots, |\mathcal{F}|\}$ (that's what a bimachine does). Recall the function π from Definition H.2. The idea is to code the iterable node \mathcal{I} by the position $\xi(\mathcal{I}) := \pi(\min_{\mathcal{F}}(\mathcal{I})) \in \{1, \dots, |\mathcal{F}|\}$. Due to the partition, each position of $\{1, \dots, |\mathcal{F}|\}$ codes at most one iterable node.

We then need to check if a position codes for an iterable node \mathcal{I} of type τ_1 , such that $|A_{\mathcal{F}}| = c$ and $\mathcal{B}_i = \text{basis}_{\mathcal{F}}(\mathcal{I})$. This is the purpose of Claim J.14, which is used to build the monoid morphism of the main bimachine (similarly to what we did for Lemma 5.21). Its proof is an easy construction of regular languages.

▷ **Claim J.14.** One can build a regular language $P_i^{-c} \subseteq (\widehat{A} \uplus \widehat{A})^*$ such that for all $w \in A^+$, $\mathcal{F} \in \text{Fact}_{\leq \Lambda}$ over w and $1 \leq j \leq |\mathcal{F}|$:

$\mathcal{F} \uparrow j \in P_i^{-c}$ if and only if $\xi(\mathcal{I}) = j$ for some $\mathcal{I} \in \text{Iterable-nodes}(\mathcal{F})$ such that $\text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1$, $|A_{\mathcal{F}}(\mathcal{I})| = c$ and $\text{basis}_{\mathcal{F}}(\mathcal{I}) = \mathcal{B}_i$.

The construction of the bimachines for the external functions $\mathfrak{f}_{\mathcal{B}}^{-c}$ is similar. However, we need to discuss the implementation of counter. The idea is to use the ordering given by $\xi(\mathcal{J})$ on the nodes $\mathcal{J} \in \text{Iterable-nodes}(\mathcal{F})$, and to produce 1 each time $\text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2$ and $\text{basis}_{\mathcal{F}}(\mathcal{J}) = \mathcal{B}$, except for the c first ones (which is a regular property since $c \leq 3$).

J.1 Proof of Lemma J.8

Let $\tau_1, \tau_2 \in T$ be two types.

▷ **Claim J.15.** If there exists a factorization \mathcal{F} of height at most Λ and $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$ such that $\{\tau_1, \tau_2\} = \{\text{type}_{\mathcal{F}}(\mathcal{I}), \text{type}_{\mathcal{F}}(\mathcal{J})\}$, then $\exists 1 \leq d_1, d_2 \leq \Lambda, m, n, e, m_1, n_1, m_2, n_2 \in M, u_1, u_2 \in A^+$ with $|u_1|, |u_2| \leq 2^\Lambda$ such that:

$$\begin{cases} \tau_1 = (d_1, m, n, e, m_1, n_1, u_1); \\ \tau_2 = (d_2, m, n, e, m_2, n_2, u_2). \end{cases}$$

Proof. If $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$ then $\text{basis}_{\mathcal{F}}(\mathcal{I}) = \text{basis}_{\mathcal{F}}(\mathcal{J})$. Hence they these nodes have the same left m , right n and value e . ◀

We thus assume that τ_1 and τ_2 are as in Claim J.15, since otherwise Lemma J.8 is true by emptiness. The general idea is to reduce the proof to the definition of a symmetrical 1-marble bimachine. For this, we relate pairs of nodes with bitypes.

Let $w \in A^+$ and $\mathcal{F} \in \text{Fact}(w)$ be a factorization of height at most Λ . If $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$ with $\mathcal{I} < \mathcal{J}$, we let:

$$\begin{aligned} \text{bitype}(\mathcal{I}, \mathcal{J}) := & \mu(w[1:\min_{\mathcal{F}}(\mathcal{I})-1]) \langle \mathbf{w}[\text{Fr}_{\mathcal{F}}(\mathcal{I})] \rangle \mu(w[\max_{\mathcal{F}}(\mathcal{I})+1:\min_{\mathcal{F}}(\mathcal{J})-1]) \\ & \langle \mathbf{w}[\text{Fr}_{\mathcal{F}}(\mathcal{J})] \rangle \mu(w[\max_{\mathcal{F}}(\mathcal{J})+1:|w|]). \end{aligned}$$

The following lemma, shown in Subsubsection J.1.1, justifies this definition.

► **Lemma J.16.** If $(\mathcal{I}, \mathcal{J}) \in I(\mathcal{F})$ and $\mathcal{I} < \mathcal{J}$, then $\text{prod}(\mathcal{I}, \mathcal{J}) = \text{prod}(\text{bitype}(\mathcal{I}, \mathcal{J}))$.

We finally want to relate $\text{bitype}(\mathcal{I}, \mathcal{J})$ with the types τ_1 and τ_2 .

From now assume without loss of generality that $\text{type}_{\mathcal{F}}(\mathcal{I}) = \tau_1$ and $\text{type}_{\mathcal{F}}(\mathcal{J}) = \tau_2$. Let $e_1 := \mu(u_1)$ and $e_2 := \mu(u_2)$, the following claim is immediate by definition of a type.

▷ **Claim J.17.** $u_1 = w[\text{Fr}_{\mathcal{F}}(\mathcal{I})]$, $u_2 = w[\text{Fr}_{\mathcal{F}}(\mathcal{J})]$ and $m_1 e_1 n_1 = m_2 e_2 n_2 = e$.

Similarly, we can show the following (the proof uses ideas from that of Lemma J.19, it is in fact easier, hence we do not detail it).

▷ **Claim J.18.** $\mu(w[1:\min_{\mathcal{F}}(\mathcal{I})-1]) = m e m_1 e_1$ and $\mu(w[\max_{\mathcal{F}}(\mathcal{J})+1:|w|]) = e_2 n_2 e n$.

We finally create some $p \in M$, which fits the conditions of Definition 5.3. This result relies on the definition of $I(\mathcal{F})$, that is that the nodes are not "nearly on the same branch".

► **Lemma J.19.** There exists $p \in M$ such that:

$$\begin{cases} m_1 e_1 p e_2 n_2 = e; \\ e m_1 e_1 p e_2 = e m_2 e_2; \\ e_1 p e_2 n_2 e = e_1 n_1 e; \\ \mu(w[\max_{\mathcal{F}}(\mathcal{I})+1:\min_{\mathcal{F}}(\mathcal{J})-1]) = e_1 p e_2; \end{cases}$$

Proof. Since $\mathcal{I}, \mathcal{J} \in \text{Iterable-nodes}(\mathcal{F})$, then \mathcal{I} (resp. \mathcal{J}) has a right sibling \mathcal{I}' (resp. a left sibling \mathcal{J}') and $\text{value}(\mathcal{I}') = \text{value}(\mathcal{I}) = \mu(u_1) = e_1$ (resp. $\text{value}(\mathcal{J}') = \text{value}(\mathcal{J}) = \mu(u_2) = e_2$). Two cases occur:

- either $\mathcal{I}' = \mathcal{J}'$, that is $\mathcal{I}, \mathcal{I}', \mathcal{J}$ are successive siblings. In that case $m_1 = m_2$, $n_1 = n_2$ and $e_1 = e_2$. We set $p := \mu(\varepsilon)$, the conditions are easy to check;
- or $\mathcal{I}' \neq \mathcal{J}'$. In that case:

XX:38 Pebble transducers with unary output

▷ Claim J.20. $\max_{\mathcal{F}}(\mathcal{I}') < \min_{\mathcal{F}}(\mathcal{J}')$

Proof. Consider the least common ancestor \mathcal{A} of \mathcal{I} and \mathcal{J} . It cannot be \mathcal{I} nor \mathcal{J} itself by definition of $I(\mathcal{F})$, thus one of the following occurs:

- either \mathcal{A} is neither the direct parent of \mathcal{I} nor that of \mathcal{J} , the result is immediate;
- or \mathcal{A} it is the parent of \mathcal{I} . But by definition of $I(\mathcal{F})$, \mathcal{J} is not a descendant of \mathcal{I}' , hence \mathcal{J} is the descendant of a sibling on the right of \mathcal{I}' and the result follows;
- or \mathcal{A} it is the parent of \mathcal{J} , treated symmetrically. ◀

■ We define:

$$p := \mu(w[\max_{\mathcal{F}}(\mathcal{I}') + 1 : \min_{\mathcal{F}}(\mathcal{J}') - 1]).$$

It is always clear that $\mu(w[\max_{\mathcal{F}}(\mathcal{I}) + 1 : \min_{\mathcal{F}}(\mathcal{J}) - 1]) = e_1 p e_2$. The other equations can be checked by a (rather long) case disjunction. Let us detail one case.

Assume that \mathcal{I} is a middle child of $\text{basis}_{\mathcal{F}}(\mathcal{I}) = \text{basis}_{\mathcal{F}}(\mathcal{J})$, i.e. $\mathcal{I} = \text{middle}_{\mathcal{F}}(\mathcal{I})$. Then $m_1 = n_1 = \mu(\varepsilon)$ and $e_1 = e$. Let $\mathcal{I}, \mathcal{I}', \mathcal{I}'', \mathcal{I}'''$ be the successive right siblings below $\text{basis}_{\mathcal{F}}(\mathcal{I})$ and assume that \mathcal{J} is a descendant of \mathcal{I}''' . Then $p = em_2 e_2$ or $p = em_2$ (depending on whether \mathcal{J}' is a middle child or not). Then $m_1 e_1 p e_2 n_2 = em_2 e_2 n_2 = e$; $em_1 e_1 p e_2 = em_2 e_2$; and $e_1 p e_2 n_2 e = em_2 e_2 n_2 e = e = e_1 n_1 e$. ◀

Finally we get $\text{bitype}(\mathcal{I}, \mathcal{J}) = em_1 e_1 \langle \mathbf{u}_1 \rangle e_1 p e_2 \langle \mathbf{u}_2 \rangle e_2 n_2 e$. To conclude the proof, we use the K of Definition 5.3 whose conditions are met. Switching τ_1 and τ_2 gives the same K , using the second case of Definition 5.3.

J.1.1 Proof of Lemma J.16

Let $\mathcal{I} < \mathcal{J}$ and $(\mathcal{I}, \mathcal{J}) \in I(F)$. Let $\{i_1, \dots, i_\ell\} := \text{Fr}_{\mathcal{F}}(\mathcal{I})$ and $\{j_1, \dots, j_{\ell'}\} := \text{Fr}_{\mathcal{F}}(\mathcal{J})$. Hence $i_\ell = \max_{\mathcal{F}}(\mathcal{I}) < \min_{\mathcal{F}}(\mathcal{J}) = j_1$. Furthermore:

$$\text{prod}(\mathcal{I}, \mathcal{J}) = \sum_{\substack{1 \leq k \leq \ell \\ 1 \leq k' \leq \ell'}} \text{prod}(i_k, j_{k'})$$

Let $q := \mu([1:i_1-1])$, $q' := \mu(w[i_\ell+1:j_1-1])$ and $q'' := \mu(w[j_{\ell'}+1:|w|])$. Let $u := w[\text{Fr}_{\mathcal{F}}(\mathcal{I})]$ and $u' := w[\text{Fr}_{\mathcal{F}}(\mathcal{J})]$ be the words described by their frontiers.

By definition, $\text{bitype}(\mathcal{I}, \mathcal{J}) = q \langle \mathbf{u} \rangle q' \langle \mathbf{u}' \rangle q''$. Hence:

$$\text{prod}(\text{bitype}(\mathcal{I}, \mathcal{J})) = \sum_{\substack{1 \leq k \leq \ell \\ 1 \leq k' \leq \ell'}} P(k, k') \text{ where} \tag{8}$$

$$P(k, k') := q \mu(u[1:k-1]) \langle \mathbf{u}[k] \rangle \mu(u[k+1:\ell]) q' \mu(u'[1:k'-1]) \langle \mathbf{u}'[k'] \rangle \mu(u'[k'+1:\ell']) q''.$$

Hence to show Lemma J.16, it is enough to show $\forall k, k' \text{ prod}(i_k, i_{k'}) = P(k, k')$. And:

$$\begin{aligned} \text{prod}(i_k, i_{k'}) &= \text{prod}(\mu(w[1:i_k-1]) \langle \mathbf{w}[i_k] \rangle \mu(w[i_k+1:j_{k'}-1]) \langle \mathbf{w}[j_{k'}] \rangle \mu(w[j_{k'}+1:|w|])) \\ &= \text{prod}(\mu(w[1:i_k-1]) \langle \mathbf{u}[k] \rangle \mu(w[i_k+1:j_{k'}-1]) \langle \mathbf{u}'[k'] \rangle \mu(w[j_{k'}+1:|w|])) \end{aligned} \tag{9}$$

Comparing Equations 8 and 9, we see that we only need to show that $\forall k, k'$:

$$\begin{cases} \mu(u[1:k-1]) = \mu(w[i_1:i_k-1]) \\ \mu(u[k+1:\ell]) = \mu(w[i_k+1:i_\ell]) \\ \mu(u'[1:k'-1]) = \mu(w[j_1:j_{k'}-1]) \\ \mu(u'[k'+1:\ell']) = \mu(w[j_{k'}+1:j_{\ell'}]) \end{cases}$$

We only treat the first case, since the others are similar. It follows by applying the claim below to the node $\mathcal{I} \in \mathbf{Fact}(w[i_1:i_\ell])$ seen as a factorization.

▷ **Claim J.21.** Let $\mathcal{F} \in \mathbf{Fact}(x)$, $y := x[\mathbf{Fr}_{\mathcal{F}}(\mathcal{F})]$ and $\{1 = i_1 < \dots < i_\ell = |x|\} := \mathbf{Fr}_{\mathcal{F}}(\mathcal{F})$. Then $\forall 1 \leq k \leq \ell$, $\mu(x[1:i_k-1]) = \mu(y[1:k-1])$.

Proof. The result is shown by induction on the factorization. The only interesting case is when $\mathcal{F} = (\mathcal{F}_1) \dots (\mathcal{F}_n)$ starts with an idempotent node. Let $x = x_1 \dots x_n$ be such that \mathcal{F}_i is a factorization of x_i . Then $e := \mu(x_i)$ is an idempotent.

By definition of the dependency and Lemma G.6, there exists $1 \leq p \leq \ell$ such that $i_p = |x_1|$ and $i_{p+1} = |x| - |x_n| + 1$. Hence:

$$y = x_1[i_1] \dots x_1[i_p] x_n[i_{p+1}-|x|+|x_n|] \dots x_n[i_\ell-|x|+|x_n|]$$

Let $1 \leq k \leq \ell$, we assume that $p+1 \leq k$ (the case $k \leq p$ is easier). Then:

$$y[1:k-1] = x_1[i_1] \dots x_n[i_p] x_n[i_{p+1}-|x|+|x_n|] \dots x_n[i_{k-1}-|x|+|x_n|].$$

By Lemma G.6:

$$\mu(x_1[i_1] \dots x_1[i_{p-1}] x_1[i_p]) = \mu(x_1) = e$$

And by induction hypothesis:

$$\mu(x_n[i_{p+1}-|x|+|x_n|] \dots x_n[i_{k-1}-|x|+|x_n|]) = \mu(x_n[i_{p+1}-|x|+|x_n|:i_{k-1}-|x|+|x_n|])$$

Finally since $e = \mu(x_2) = \dots = \mu(x_{n-1})$ is idempotent, it follows that:

$$\begin{aligned} \mu(x[1:i_k-1]) &= e \mu(x_n[i_{p+1}-|x|+|x_n|:i_{k-1}-|x|+|x_n|]) \\ &= \mu(x_1[i_1] \dots x_1[i_{p-1}] x_1[i_p]) \mu(x_n[i_{p+1}-|x|+|x_n|] \dots x_n[i_{k-1}-|x|+|x_n|]) \\ &= \mu(y[1:k-1]). \end{aligned}$$

◀

K Proof of Theorem 5.4 - characterization

We only show that $3 \Rightarrow 2$. Assume that the 1-blind bimachine \mathcal{T} is symmetrical.

By Proposition 5.9, one can build a two-way transducer (with non-unary output) that given $w \in A^+$, computes some factorization $\mathcal{F} \in \mathbf{Fact}(w)$ of height at most Λ , described as a word, that is $\mathcal{F} \in \widehat{A}^+$. We denote this function $f_{\mathbf{Fact} \leq \Lambda}$.

Using lemmas 5.21, 5.23, 5.24, the functions f_D, f_L and f_I are computable by a 1-blind transducer. We want to compute their sum, in the sense of Claim J.10. It follows from this claim that $f_D + f_L + f_I$ can be computed by a 1-blind bimachine, or equivalently a 1-blind transducer. By Lemma 5.17 and the definitions of f_D, f_L and f_I , it follows that:

$$(f_D + f_L + f_I) \circ f_{\mathbf{Fact} \leq \Lambda} = f.$$

It follows from [11] that (even for non-unary alphabets) 1-blind transducers are effectively closed under composition with two-way transducers, hence $(f_D + f_L + f_I) \circ f_{\mathbf{Fact} \leq \Lambda}$ can be computed by a 1-blind transducer.