Adding Pebbles to Weighted Automata: Easy Specification & Efficient Evaluation

Paul Gastin, Benjamin Monmege

Adding Pebbles to Weighted Automata Easy Specification & Efficient Evaluation*

Paul Gastin and Benjamin Monmege

LSV, ENS Cachan, CNRS, Inria, France
firstname.lastname@lsv.ens-cachan.fr

Abstract. We extend weighted automata and weighted rational expressions with 2-way moves and (reusable) pebbles. We show with examples from natural language modeling and quantitative model-checking that weighted expressions and automata with pebbles are more expressive and allow much more natural and intuitive specifications than classical ones. We extend Kleene-Schützenberger theorem showing that weighted expressions and automata with pebbles have the same expressive power. We focus on an efficient translation from expressions to automata. We also prove that the evaluation problem for weighted automata can be done very efficiently if the number of (reusable) pebbles is low.

1 Introduction

Regular expressions have always been used to specify patterns. Popular because they propose a concise and intuitive way of denoting such patterns, they have also a long history in the formal language community. A seminal result, known as Kleene's theorem, establishes that the (denotational) regular expressions have the same expressive power as the (operational) finite state automata. Efficient translation algorithms of regular expressions into finite automata are crucial since expressions are convenient to denote patterns and automata are amenable to efficient algorithms. Regular expressions and finite automata have been extended in several directions, e.g., tree (walking) automata, (regular) XPath, etc.

Nowadays, quantitative models and analysis are intensively studied, resulting in a revision of the foundation of computer science where classical yes/no answers are replaced by quantities such as probability, energy consumption, reliability, cost, etc. In the 60s, Schützenberger provided a generic way of turning qualitative into quantitative systems, starting the theory of weighted automata [32] (see [18,16,3] for recent books on this theory). Indeed, probabilistic automata and word transducers appear as instances of that framework, which found its way into numerous application areas such as natural language processing, speech recognition or digital image compression. Schützenberger proved the equivalence between weighted automata and weighted regular expressions, extending Kleene's theorem. Various translation algorithms can be extended from the Boolean framework to the weighted case, see [28,30] for surveys about

* Supported by LIA INFORMEL.

these methods, and [23] which obtains Schützenberger's theorem as a corollary of Kleene's theorem.

In Sections 4 and 5, we extend weighted expressions and automata with 2-way moves and pebbles. There are several motivations for these extensions. First, as shown in Section 2 for applications in natural language processing and quantitative model-checking, 2-way moves and pebbles allow more natural and more concise descriptions of the quantitative expressions we need to evaluate. Second, in the weighted case, 2-way and pebbles do increase the expressive power as already observed in [8] in relation with weighted logics or in [27] in the probabilistic setting. This is indeed in contrast with the Boolean case where 2-way and pebbles do not add expressive power over words (see, e.g., [20]) even though they allow more succinct descriptions (see, e.g., [4]). Our work is also inspired by pebble tree-walking automata and in particular their links with powerful logics, XPath formalisms and caterpillar expressions on trees [17,10,6,31,5].

In Sections 6 and 7, we generalize Kleene and Schützenberger theorems to weighted expressions and automata with 2-way moves and pebbles. We establish their expressive power equivalence by providing effective translations in both directions. Showing how to transform an *operational* automaton into an equivalent *denotational* expression is indeed very interesting from a theoretical point of view, but is less useful in practice. On the other hand, we need highly efficient translations from the convenient denotational formalism of expressions to operational automata which, as stated above, are amenable to efficient algorithms. Efficiency is measured both with respect to the size of the resulting automaton, and the space and time complexities of the translation. We show that, Glushkov's [21] or Berry-Sethi [2] translations, which are among the best ones in the Boolean case, can be extended to weighted expressions with 2-may moves and pebbles. The constructions for the rational operations (sum, product, star) can be adapted easily to cope with 2-way moves, even though the correctness proofs are more involved and require new theoretical grounds such as series over a partial monoid as explained in Section 4.1. The main novelty in Sections 6 and 7 is indeed the treatment of pebbles in the translations between expressions and automata.

To complete the picture, we study in Section 8 the evaluation problem of a weighted automaton with 2-way moves and *reusable* pebbles over a given word. The algorithm is polynomial in the size of the word, where the degree is 1 plus the number of reusable pebbles. We can even decrease the degree by 1 for *strongly layered* automata. This applies when we only have one reusable pebble, and we obtain an algorithm which is linear in the size of the input word. This is in particular the case for automata derived from weighted LTL.

The paper includes intuitive explanations and examples for a better understanding of weighted expressions with 2-way moves and pebbles, and of the translations between automata and expressions. An extended abstract of this work appeared in [19].

2 Motivations

We give in this section two motivating examples for studying weighted expressions and automata with 2-way moves and pebbles.

2.1 Language modeling

Since decades, weighted automata have been extensively used in Natural Language Processing (see [22]), in particular for automatic translation, speech recognition or transliteration. All these tasks have in common to split the problem into independent parts, certain directly related to the specific task and others related to the knowledge of the current language. For example, in the translation task from French sentences to English sentences, one splits the problem into first knowing translation of single words and then modeling English sentences (knowledge which is independent from the translation task). The second part, namely to know whether a sequence of words is a *good* English sentence, is known as *language modeling*. Often this knowledge is learned from a large corpus of English texts, and stored into a formal model, e.g., a weighted finite state automaton representing the probability distribution $\mathbb{P}$ of well-formed English sentences. The translation task is then resolved by first generating several English sentences from the original French one (due to ambiguity of the word-by-word translation task), and then choosing among this set of sentences the ones with highest probability.

One broadly used language model is the n-gram model, where the probability of a word in a sentence depends only on the previous $n-1$ words: for example in a 1-gram model, only the individual word frequencies are relevant to generate well-formed English sentences, whereas in a 2-gram model, the probability of a word depends on the very same frequency distribution and also the previous word. To formally describe these models, and further study them, let us define them using regular expressions. Let D denote the dictionary of words in the language. Suppose we are given the conditional probability distributions $\mathbb{P}(u_n \mid u_1, \ldots, u_{n-1})$ in the n-gram model (with $u_i \in D$ for all i). The probability of a sentence $(u_i)_{1 \le i \le m} \in D^m$ can be given by the following weighted regular expression in a 1-gram model and a 3-gram model:

$$E_1 = \Big(\sum_{u \in D} u\mathbb{P}(u) \Big)^* \qquad E_3 = {\rightarrow}{\rightarrow} \Big(\sum_{u,v,w \in D} {\leftarrow}{\leftarrow} uvw\mathbb{P}(w \mid u, v) \Big)^*$$

where symbols $\rightarrow$ and $\leftarrow$ denote a right or left move, respectively, no matter what word it is reading. Expression E_1 is a classical weighted regular expression where the Kleene star iterates the computation of the inner expression, which here computes the probability of the current word u. Expression E_3 has the opportunity to move forward and backward: this allows to easily recover the context whereas a 1-way automaton would have to store the context in its states. Notice that expression E_3 is quite readable and intuitive. One could write an equivalent 1-way expression, but imagine how intricate it would be since positions

would have to encode the context, i.e., the last two words. This is an important motivation for studying 2-way expressions and automata.

Actually, expression E_3 is not small since the sum hides the very big set D^3: for a dictionary of size 1 million, this seems already unpracticable. But in practice, a much smaller expression could be sufficient. First, for many words, the frequency distribution of the word w is a sufficiently good approximation of the conditional probability $\mathbb{P}(w \mid u, v)$. Let us denote D_0 this set of words. For instance, the probability of observing the word `the` may not really depend on the previous words. Then, let D_1 be the set of words (disjoint from D_0) such that only the previous word is necessary to describe the probability. Finally, let D_2 be the rest of the dictionary. Now, we may replace expression E_3 by the following expression, whose size is much smaller if D_0 and D_1 contain enough words:

$$\Big(\sum_{w \in D_0} w\mathbb{P}(w) + \sum_{w \in D_1, v \in D} {\leftarrow} vw\mathbb{P}(w \mid v) + \sum_{w \in D_2, u, v \in D} {\leftarrow}{\leftarrow} uvw\mathbb{P}(w \mid u, v) \Big)^*$$

To motivate the introduction of pebbles, let us add internationalization, which means that the user has the ability to write/speak alternately in two or more languages, e.g., English and French. All tasks such as automatic translation or speech recognition are now more complex since there is no a priori knowledge of the current language of the speaker. Again, splitting the problem into independent parts, we have to know the probability distributions $\mathbb{P}_L$ for every involved language L, and, assuming a current language L, we should be able to solve the language processing task with a procedure Task_L. Then, before processing the next word, we start a computation which re-reads the current prefix of the text in order to compute using $\mathbb{P}_L$ the probability that the current language L is still valid. The next word is then processed with the current or the alternate language (see Figure 1). In order to compute the probability that the current language is still valid, we mark the current position with a pebble ($\downarrow_x$) and read the current prefix of the text with the automaton modeling the current language. Then we return to the marked position and lift the pebble ($\uparrow$) in order to resume the top level computation. Details of these constructs will be developed in Section 5.

2.2 Weighted Linear Temporal Logic

Whereas weighted automata and weighted expressions have been extensively studied, logical formalisms adapted to the weighted case still need deeper understanding. This is especially true for weighted *linear* temporal logics [24], whereas weighted *branching* temporal logics have received more attention [13,12,26,7].

We would like to illustrate that using pebbles in weighted expressions or automata is a natural and powerful way to deal with nesting in LTL formulas. Temporal logics implicitely use a free variable to denote the position where the formula has to be evaluated. We will mark this position with a pebble, say x, in expressions $E_\varphi(x)$ or automata $\mathcal{A}_\varphi(x)$ associated with an LTL formula φ.

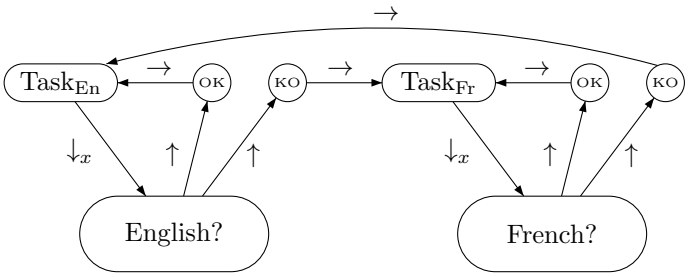
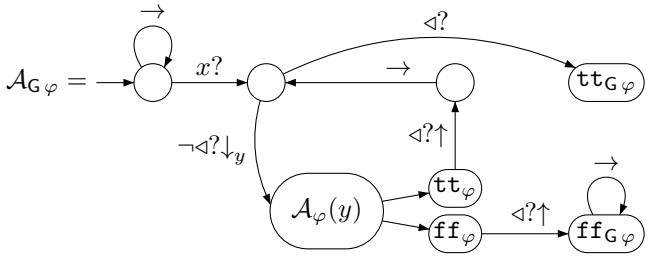


Fig. 1. Pebble automaton for the multi-language modeling task.

For this motivating example, we only consider finite words and the probabilistic setting. Given an LTL formula φ, a word $u \in A^+$ and a position i in u, the aim is to compute the probability $\mathbb{P}(\varphi, u, i)$ that φ holds on position i. Hence, the expression $E_\varphi(x)$ and the automaton $\mathcal{A}_\varphi(x)$ associated with φ should compute this probability if pebble x marks position i of u.

For instance, the formula $\mathsf{G}\,\varphi$ (*globally* φ) means that φ should always hold in the future of the current position. Hence, $\mathbb{P}(\mathsf{G}(\frac{1}{3}a \vee \frac{3}{4}b), aba, 0) = \frac{1}{3} \cdot \frac{3}{4} \cdot \frac{1}{3}$ and $\mathbb{P}(\mathsf{G}(\frac{1}{3}a \vee \frac{3}{4}b), aabb, 2) = \frac{9}{16}$. More generally, $\mathbb{P}(\mathsf{G}\,\varphi, u, i) = \prod_{j \geq i} \mathbb{P}(\varphi, u, j)$. Assume that we have already constructed an automaton $\mathcal{A}_\varphi(y)$ with 2 designated terminal states $\{\mathtt{tt}_\varphi, \mathtt{ff}_\varphi\}$, such that runs ending in $\mathtt{tt}_\varphi$ computes the probability that φ holds and those ending in $\mathtt{ff}_\varphi$ computes the probability that $\neg\varphi$ holds. Then, the automaton for $\mathsf{G}\,\varphi$ with the same property on terminal states $\mathtt{tt}_{\mathsf{G}\,\varphi}$ and $\mathtt{ff}_{\mathsf{G}\,\varphi}$ is depicted below:



We can also give expressions for $\mathsf{G}\,\varphi$ and $\neg\,\mathsf{G}\,\varphi$:

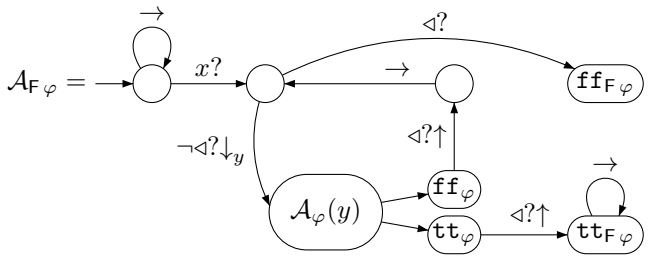
$$E_{\mathsf{G}\,\varphi}(x) = \triangleright?{\rightarrow}^* x?\big((y!E_\varphi(y)){\rightarrow}\big)^* \triangleleft?$$
$$E_{\neg\,\mathsf{G}\,\varphi}(x) = \triangleright?{\rightarrow}^* x?\big((y!E_\varphi(y)){\rightarrow}\big)^* (y!E_{\neg\varphi}(y)){\rightarrow}^* \triangleleft?\,.$$

The expression $E_{\mathsf{G}\,\varphi}(x)$ starts at the beginning of the word ($\triangleright?$), moves right (${\rightarrow}^*$) until it sees the marked position ($x?$). Then, it iterates the computation of φ with the current position marked with y ($y!E_\varphi(y)$), moving right (${\rightarrow}$) between two computations. The iteration stops at the end of the word ($\triangleleft?$). The expression for $\neg\,\mathsf{G}\,\varphi$ starts similarly, but it may exit the iteration at any point, computing the probability that φ does not hold ($y!E_{\neg\varphi}(y)$).

The dual formula $\mathsf{F}\,\varphi$ (*Finally* φ) states that φ should eventually hold in the future of the current position. The semantics is obtained inductively with $\mathbb{P}(\mathsf{F}\,\varphi, u, i) = \mathbb{P}(\varphi, u, i) + \mathbb{P}(\neg\varphi, u, i) \times \mathbb{P}(\mathsf{F}\,\varphi, u, i+1)$: either φ holds immediately ($\mathbb{P}(\varphi, u, i)$) or ($+$) it does not ($\mathbb{P}(\neg\varphi, u, i)$) and ($\times$) it should be satisfied later ($\mathbb{P}(\mathsf{F}\,\varphi, u, i+1)$). For instance, $\mathbb{P}(\mathsf{F}(\frac{1}{3}a), abba, 0) = \frac{1}{3} + \frac{2}{3}(0 + \frac{2}{3}(0 + \frac{2}{3} \cdot \frac{1}{3}))$. The following dualities hold: $\mathsf{F}\,\varphi = \neg\,\mathsf{G}\,\neg\varphi$ and $\neg\,\mathsf{F}\,\varphi = \mathsf{G}\,\neg\varphi$. Hence, we have

$$E_{\mathsf{F}\,\varphi}(x) = \triangleright?{\to}^* x?\big((y!E_{\neg\varphi}(y)){\to}\big)^*(y!E_{\varphi}(y)){\to}^*\triangleleft?$$
$$E_{\neg\,\mathsf{F}\,\varphi}(x) = \triangleright?{\to}^* x?\big((y!E_{\neg\varphi}(y)){\to}\big)^*\triangleleft?\,.$$

The automaton $\mathcal{A}_{\mathsf{F}\,\varphi}(x)$ is obtained from $\mathcal{A}_{\mathsf{G}\,\varphi}(x)$ by switching the terminal states:



Using 2-way moves, it is not difficult to extend these constructions to cope with LTL including both future and past modalities.

3 Preliminaries

Words. The set of nonempty words over a finite alphabet A is denoted A^+. We write $u = u_0 \cdots u_{n-1} \in A^+$ a nonempty word of length $|u| = n \geq 1$ with $u_i \in A$ for $0 \leq i < |u|$. The set of *positions* of u is $\mathrm{pos}(u) = \{0, 1, \ldots, |u|\}$. In particular, we include $|u|$ in $\mathrm{pos}(u)$ even though the last letter is on position $|u| - 1$.

Semirings. A semiring is a set $\mathbb{S}$ equipped with two binary internal operations denoted $+$ and $\times$, and two neutral elements 0 and 1 such that $(\mathbb{S}, +, 0)$ is a commutative monoid, $(\mathbb{S}, \times, 1)$ is a monoid, $\times$ distributes over $+$ and $0 \times s = s \times 0 = 0$ for every $s \in \mathbb{S}$. If the monoid $(\mathbb{S}, \times, 1)$ is commutative, the semiring itself is called commutative. See [15,28] for more discussions about semirings, especially complete and continuous ones, as we describe now.

A semiring $\mathbb{S}$ is *complete* if every family $(s_i)_{i \in I}$ of elements of $\mathbb{S}$ over an arbitrary indexed set I is summable to some element in $\mathbb{S}$ denoted $\sum_{i \in I} s_i$ and called *sum* of the family, such that the following conditions are satisfied:

- $\sum_{i \in \emptyset} s_i = 0$, $\sum_{i \in \{1\}} s_i = s_1$ and $\sum_{i \in \{1,2\}} s_i = s_1 + s_2$;
- if $I = \bigcup_{j \in J} I_j$ is a partition, $\sum_{j \in J} \big(\sum_{i \in I_j} s_i\big) = \sum_{i \in I} s_i$;
- $\big(\sum_{i \in I} s_i\big) \times \big(\sum_{j \in J} t_j\big) = \sum_{(i,j) \in I \times J} (s_i \times t_j)$.

Intuitively, this means that it is possible to define infinite sums that extends the binary addition and satisfies infinite versions of associativity and distributivity.

In a complete semiring, for every $s \in \mathbb{S}$, the element $s^* = \sum_{i \in \mathbb{N}} s^i$ exists (where s^i is defined recursively by $s^0 = 1$ and $s^{i+1} = s^i \times s$). Here are some examples of complete semirings.

- The Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$ with $\sum$ defined as an infinite disjunction.
- $(\mathbb{R}_{\geq 0} \cup \{\infty\}, +, \times, 0, 1)$ with $\sum$ defined as usual for positive (not necessarily convergent) series: in particular, $s^* = \infty$ if $s \geq 1$ and $s^* = 1/(1-s)$ if $0 \leq s < 1$.
- $(\mathbb{N} \cup \{\infty\}, +, \times, 0, 1)$ as a complete subsemiring of the previous one.
- $(\mathbb{R} \cup \{-\infty\}, \min, +, -\infty, 0)$ with $\sum = \inf$ and $(\mathbb{R} \cup \{\infty\}, \max, +, \infty, 0)$ with $\sum = \sup$.
- Complete lattices such as $([0,1], \min, \max, 0, 1)$.
- The semiring of languages over an alphabet A: $(2^{A^*}, \cup, +, \emptyset, \{\varepsilon\})$ with $\sum$ defined as (infinite) union.

In this paper, we consider *continuous* semirings which are complete semirings in which infinite sums can be approximated by finite partial sums. Formally, a complete semiring $\mathbb{S}$ is *continuous* if the relation $\leq$ defined over $\mathbb{S}$ by $a \leq b$ if $b = a + c$ for some $c \in \mathbb{S}$ is an order relation; and for every family $(s_i)_{i \in I}$ in $\mathbb{S}$, the sum $\sum_{i \in I} s_i$ is the least upper bound of the finite sums $\sum_{i \in J} s_i$ for $J \subseteq I$ finite. All the above complete semirings are also continuous.

Series and polynomials Let Z be a set. A series f over Z is a map $f \colon Z \to \mathbb{S}$. We denote by $\mathbb{S}\langle\langle Z \rangle\rangle$ the set of series over Z with coefficients in $\mathbb{S}$. The *support* of a series $f \in \mathbb{S}\langle\langle Z \rangle\rangle$ is the set $\{z \in Z \mid f(z) \neq 0\}$. A series with a finite support is called a *polynomial*. We denote by $\mathbb{S}\langle Z \rangle$ the set of polynomials over Z with coefficients in $\mathbb{S}$.

We can lift addition from $\mathbb{S}$ to $\mathbb{S}\langle\langle Z \rangle\rangle$ pointwise by $(f+g)(z) = f(z) + g(z)$ for all $z \in Z$. Then, $(\mathbb{S}\langle\langle Z \rangle\rangle, +, 0)$ is a commutative monoid where 0 is the series mapping every element $z \in Z$ to 0. If Z is equipped with a structure of monoid and the semiring $\mathbb{S}$ is complete, we can also define the (Cauchy) product of two series by $(f \times g)(z) = \sum_{z=xy} f(x)g(y)$ for all $z \in Z$. This sum may be infinite, but is well-defined since the semiring is complete. The Cauchy product is associative and admits as unit the characteristic function (denoted 1) of the neutral element of Z. Hence, $(\mathbb{S}\langle\langle Z \rangle\rangle, +, \times, 0, 1)$ is a semiring. When $\mathbb{S}$ is continuous, we can also lift infinite sums pointwise to $\mathbb{S}\langle\langle Z \rangle\rangle$ which becomes a continuous semiring.

4 Weighted Expressions with pebbles

The syntax of our weighted expressions is carefully chosen so that an efficient translation to weighted automata can be obtained, essentially based on Glushkov's construction as we will see in Section 7. Formally, for a (continuous)

semiring $\mathbb{S}$, an alphabet A and a set Peb of pebbles, the syntax is given by the grammar:

$$E ::= s \mid \varphi \mid \rightarrow \mid \leftarrow \mid x!E \mid E + E \mid E \cdot E \mid E^+$$
$$\varphi ::= a? \mid \triangleright? \mid \triangleleft? \mid x? \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

with $s \in \mathbb{S}$, $a \in A$, $x \in \text{Peb}$. We denote by Test the set of *test* formulas φ defined by the second line of the grammar above. For instance, one can check with $\triangleright?$ and $\triangleleft?$ whether we are at the beginning or at the end of the word. This is indeed useful since we have 2-way expressions. We denote by pebWE the set of weighted expressions with pebbles. Below, we give the intuitive meaning of our weighted expressions. We start without pebbles (i.e., without $x!E$). Then, we introduce pebbles. The formal semantics is given in Table 1.

Notice that from the *irreflexive* iteration $E^+ = \sum_{n>0} E^n$, we get also the classical Kleene star: $E^* \stackrel{\text{def}}{=} 1 + E^+$. Indeed, we also have $E^+ = E \cdot E^*$ but if we apply Glushkov's construction (blindly) to $E \cdot E^*$ we get an automaton with twice the number of states needed for E^+. This is basically why we prefer to have E^+ as a primary construct.

We have chosen to distinguish between *checking* the current position with some *test* φ and *moving* to the right or left position with $\rightarrow$ and $\leftarrow$. This is in the spirit of XPath in trees. This allows to write concise expressions, e.g., $E = \rightarrow^+ a? \leftarrow^+ b? \rightarrow^+ c? \leftarrow^+ d? \rightarrow^+$ to describe patterns consisting of an a having in its past a b, having in its future a c, having in its past a d. In the Boolean semantics, this expression defines words having this pattern. In the semiring $\mathbb{N}$ of natural numbers, the expression counts the number of occurrences of the pattern, e.g., $[\![E]\!](cabcdbadcbab) = 8$. Indeed we may write an equivalent 1-way expression for this pattern but it would be less concise and harder to decipher (see e.g., [4] for succinctness results in the Boolean case).

Let $u = u_0 u_1 \cdots u_{|u|-1} \in A^+$. A test φ will be evaluated at a position $i \in \text{pos}(u)$: $\triangleright?$ holds if $i = 0$, $\triangleleft?$ holds if $i = |u|$ and $a?$ holds if $i < |u|$ and $u_i = a$.

With the 2-way mechanism, a sub-expression such as $a? \leftarrow^+ b? \rightarrow^+ c? \leftarrow^+ d?$ may start from position i, end in position j and still visit the whole word. In order to inductively define the semantics of expressions, we assign to triples (u, i, j) a value $[\![E]\!](u, i, j) \in \mathbb{S}$.

It is also convenient to *check-and-move* so we introduce the macros $a \stackrel{\text{def}}{=} a? \rightarrow$ and $\overline{a} \stackrel{\text{def}}{=} a? \leftarrow$. Then, we can write $\rightarrow^* \texttt{blue} \leftarrow^+ \triangleright? \rightarrow^* \texttt{black} \rightarrow^*$ to define words having both `blue` and `black` as subwords. This allows to write classical (1-way) regular expressions such as $(ab)^+aa$. In order to get the classical semantics for usual 1-way expressions, the evaluation of an expression on a whole word is defined as $[\![E]\!](u) = [\![E]\!](u, 0, |u|)$. For instance, $[\![a]\!](a) = [\![a? \rightarrow]\!](a, 0, 1) = 1$, $[\![\rightarrow^* a \rightarrow^*]\!](baaba) = [\![\rightarrow^* a? \rightarrow \rightarrow^*]\!](baaba, 0, 5) = 3$, and $[\![(2\rightarrow)^+]\!](u) = 2^{|u|}$.

Our 2-way expressions are uncomparable with expressions over the free group. Indeed, the expression $a\overline{a}b$ always evaluates to 0 in our setting, whereas over the free group it would evaluate to 1 on $b = a\overline{a}b$.

Notice that with the 2-way mechanism we may write $E = E_1 \triangleleft? \leftarrow^* \triangleright? E_2$ to compute the product (intersection in the boolean semantics) of the values computed by E_1 and E_2: $[\![E]\!](u) = [\![E_1]\!](u) \times [\![E_2]\!](u)$.

The 2-way mechanism together with iteration gives rise to infinite sums. This may be useful for probabilistic systems. For instance, in the *continuous* semiring $(\mathbb{R}^\infty_{\geq 0}, +, \times, 0, 1)$, consider the expression $E = (\neg\triangleleft?(s \rightarrow + (1-s)\neg\triangleright?\leftarrow))^* \triangleleft?$ with $0 < s < 1$ some probability. Expression E describes a random walk[1] and it will be used again in Section 5. Let $F = \neg\triangleleft?(s \rightarrow + (1-s)\neg\triangleright?\leftarrow)$ so that $E = F^* \triangleleft?$. Let u be a word of length $m \geq 2$. We can easily see that for all $i, j \in \mathrm{pos}(u)$ and all $n > |j - i|$, the expression F^n computes a *positive* value on (u, i, j). Therefore, the expression F^* computes an infinite sum on (u, i, j). In the present case $(0 < s < 1)$, the series $\sum_{n\geq 0} [\![F^n]\!](u, i, j)$ converges and $[\![F^*]\!](u, i, j) \in \mathbb{R}_{\geq 0}$. On the other hand, for the expression $G = \neg\triangleleft?\rightarrow + \neg\triangleright?\leftarrow$, we can check that the series $\sum_{n\geq 0} [\![G^n]\!](u, i, j)$ diverges and we get $[\![G^*]\!](u, i, j) = \infty$. Since we are considering *complete* semirings, infinite sums exist and the semantics of an iteration E^* or E^+ is always well-defined.

We explain now the pebble mechanism used in our expressions. The construct $x!E$ marks with x the current position *in* u and evaluates E on the marked word, from beginning to end. Indeed, we can use $x?$ in E to test whether the current position is marked. For instance, consider

$$E = \rightarrow^+ a?\, x!\Big((\neg x?\rightarrow)^*\, b?\, (\neg x?\rightarrow)^+\, c?\leftarrow^+ d?\rightarrow^+\Big)\rightarrow^*$$

which is a variant of our first example. Here the pattern consists of an a for which the corresponding prefix contains a b, having in its future a c, having in its past a d. In particular, the c must be on the left of the current a which is marked with x. Hence, we get $[\![E]\!](\mathit{cabcdbadcbab}) = 4$.

As another example, on a word u, the expression $(x!((2\rightarrow)^+)\rightarrow)^+$ computes $2^{|u|^2}$ over the natural semiring[2]. Actually, the pebble is not tested in this expression: it is only used to restart the computation $|u|$ times.

We give now the formal semantics of tests and of pebWE. For each word $u \in A^+$, valuation $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(u)$ and position $i \in \mathrm{pos}(u)$, the semantics $u, \sigma, i \models \varphi$ of tests is defined inductively. The Boolean connectives are as usual. For the atoms, $\triangleright?$ holds if $i = 0$, $\triangleleft?$ holds if $i = |u|$, $a?$ holds if $i < |u|$ and $u_i = a$ and $x?$ holds if $\sigma(x) = i < |u|$ (the last position $|u|$ cannot be marked).

A *marked* word is a tuple (u, σ, i, j) where $u \in A^+$ is a word, $\sigma : \mathrm{Peb} \to \mathrm{pos}(u)$ is a valuation and $i, j \in \mathrm{pos}(u)$ are positions. We denote by $\mathrm{Mk}(A^+)$ the set of marked words (we will see below that it forms a partial monoid).

[1] With $\alpha = \frac{1-s}{s}$, one can show that $[\![E]\!](u, 0, |u|) = \frac{1}{1+\alpha+\ldots+\alpha^{|u|}}$.

[2] This function cannot be computed without pebble by a classical 1-way weighted expression. We can see this using Schützenberger's theorem since weighted automata only compute values in $2^{\mathcal{O}(|u|)}$.

$$[\![s]\!](u,\sigma,i,j) = \begin{cases} s & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \qquad [\![\varphi]\!](u,\sigma,i,j) = \begin{cases} 1 & \text{if } j = i \wedge u,\sigma,i \models \varphi \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\rightarrow]\!](u,\sigma,i,j) = \begin{cases} 1 & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases} \qquad [\![\leftarrow]\!](u,\sigma,i,j) = \begin{cases} 1 & \text{if } j = i-1 \\ 0 & \text{otherwise} \end{cases}$$

$$[\![x!E]\!](u,\sigma,i,j) = \begin{cases} [\![E]\!](u,\sigma[x \mapsto i],0,|u|) & \text{if } j = i < |u| \\ 0 & \text{otherwise} \end{cases}$$

$$[\![E+F]\!](u,\sigma,i,j) = [\![E]\!](u,\sigma,i,j) + [\![F]\!](u,\sigma,i,j)$$

$$[\![E \cdot F]\!](u,\sigma,i,j) = \sum_{k \in \mathrm{pos}(u)} [\![E]\!](u,\sigma,i,k) \times [\![F]\!](u,\sigma,k,j)$$

$$[\![E^+]\!](u,\sigma,i,j) = \sum_{n>0} [\![E^n]\!](u,\sigma,i,j)$$

Table 1. Semantics of weighted expressions

The semantics[3] of a pebWE E is a map $[\![E]\!] \colon \mathrm{Mk}(A^+) \to \mathbb{S}$, i.e., a series over marked words: $[\![E]\!] \in \mathbb{S}\langle\langle \mathrm{Mk}(A^+) \rangle\rangle$. It is is defined in Table 1. Note that, since we are considering *complete* semirings, the infinite sum in the semantics of E^+ is always well-defined. If the expression has no free pebbles then we omit the valuation and simply write $[\![E]\!](u,i,j)$. For whole words we let $[\![E]\!](u,\sigma) = [\![E]\!](u,\sigma,0,|u|)$ and $[\![E]\!](u) = [\![E]\!](u,0,|u|)$ as explained above.

Notice that for tests φ_1 and φ_2, the expressions $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \cdot \varphi_2$ are equivalent, but $\varphi_1 \vee \varphi_2$ and $\varphi_1 + \varphi_2$ are not equivalent in general. One can check that $\varphi_1 + \neg\varphi_1 \cdot \varphi_2$ is equivalent to the disjunction $\varphi_1 \vee \varphi_2$. Hence, conjunctions and disjunctions in tests are not necessary for the expressive power of pebWE and they could have been defined as macros.

Similar to the star-height of an expression, we define the *pebble*-depth:

$$\mathrm{pebd}(s) = \mathrm{pebd}(\varphi) = \mathrm{pebd}(\leftarrow) = \mathrm{pebd}(\rightarrow) = 0$$
$$\mathrm{pebd}(E+F) = \mathrm{pebd}(E \cdot F) = \max(\mathrm{pebd}(E), \mathrm{pebd}(F))$$
$$\mathrm{pebd}(E^+) = \mathrm{pebd}(E) \qquad \mathrm{pebd}(x!E) = 1 + \mathrm{pebd}(E)\,.$$

4.1 Series over a partial monoid

We show in this subsection that the set of marked words can be endowed with a *partial monoid* structure which allows to define a Cauchy product on series in $\mathbb{S}\langle\langle \mathrm{Mk}(A^+) \rangle\rangle$. Since the sums can be lifted pointwise from $\mathbb{S}$ to series over $\mathbb{S}$, we show that $\mathbb{S}\langle\langle \mathrm{Mk}(A^+) \rangle\rangle$ is actually a *continuous* semiring. Indeed, the semantics defined for sum, product and iteration of pebWE in Table 1 corresponds to sum,

[3] We may also define the semantics $[\![E]\!]_{\mathcal{V}}$ of an expression E using valuations over a subset $\mathcal{V} \subseteq \mathrm{Peb}$, provided it contains the free pebbles of E.

Cauchy product and star in the continuous semiring $\mathbb{S}\langle\langle \mathrm{Mk}(A^+)\rangle\rangle$. This more formal view of the semantics of pebWE is especially useful for proofs, but since proofs are omitted in this paper this section may be skipped in a first reading.

Pebble weighted expressions and pebble weighted automata introduce two new difficulties. The first one comes from the 2-way navigation mechanism which prevents the computation of the behavior of an expression (or an automaton) using the concatenation of words in the underlying monoid, here the free monoid A^+. The second one comes indeed from pebbles which allow to restart the computation. To address both problems, we had to fix the word when defining the semantics and we no more use the monoid structure of A^+. Here, we define a *partial* monoid structure on the *marked words* and show how this allows us to reuse existing results from the classical theory of rational series.

A *partial* monoid is a triple $(Z, \cdot, Y)$ where Z is the set of elements, $\cdot\colon Z^2 \to Z$ is a *partially defined* associative concatenation[4] and $Y \subseteq Z$ is a set of *partial* units satisfying:

$$\begin{array}{llll} \forall x, z \in Z & \forall y \in Y & x \cdot y = z & \implies x = z \\ \forall x, z \in Z & \forall y \in Y & y \cdot x = z & \implies x = z \\ \forall z \in Z & \exists! y \in Y & y \cdot z = z & \\ \forall z \in Z & \exists! y \in Y & z \cdot y = z\,. & \end{array}$$

Indeed, a classical monoid is a partial monoid with the concatenation being totally defined and with the set of partial units being the singleton set consisting of the (real) unit.

We are especially interested in the partial monoid $(\mathrm{Mk}(A^+), \cdot, \mathrm{Unit}(A^+))$ of *marked words* over A^+ where

$$\begin{aligned} \mathrm{Mk}(A^+) &= \{(u, \sigma, i, j) \mid u \in A^+,\ \sigma\colon \mathrm{Peb} \to \mathrm{pos}(u),\ i, j \in \mathrm{pos}(u)\} \\ \mathrm{Unit}(A^+) &= \{(u, \sigma, i, i) \mid u \in A^+,\ \sigma\colon \mathrm{Peb} \to \mathrm{pos}(u),\ i \in \mathrm{pos}(u)\} \end{aligned}$$

and the partial concatenation is defined for all $u \in A^+$, $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(u)$ and $i, j, k \in \mathrm{pos}(u)$ by $(u, \sigma, i, k) \cdot (u, \sigma, k, j) = (u, \sigma, i, j)$ and it is undefined in all other cases. We can see that this partial concatenation is associative and that the above requirements for partial units are satisfied.

Note that a partial monoid needs not be graded and in particular, the partial monoid of marked words defined above is not graded. Hence, we cannot apply directly the theory of rational series over graded monoids as developped e.g. in [29]. Instead, we will use the theory of rational series over a *continuous* semiring $\mathbb{S}$ (see e.g., [28, III.5]). We first show that, even if the monoid Z, and more specifically $\mathrm{Mk}(A^+)$, is only partial, we can define (infinite) sums and (Cauchy) product on series over Z so that $\mathbb{S}\langle\langle Z\rangle\rangle$ forms a continuous semiring.

Let $\mathbb{S}$ be a continuous semiring and $(Z, \cdot, Y)$ be a partial monoid. As described in Section 3, infinite sums may be lifted from $\mathbb{S}$ to series in $\mathbb{S}\langle\langle Z\rangle\rangle$. We may also

[4] for all $x, y, z \in Z$, $(x \cdot y) \cdot z$ is defined iff $x \cdot (y \cdot z)$ is defined, and in this case $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

define the Cauchy product as usual. Note that, even though the concatenation in Z may be partially defined, the Cauchy product in $\mathbb{S}\langle\langle Z\rangle\rangle$ is always defined for $f, g \in \mathbb{S}\langle\langle Z\rangle\rangle$ and $z \in Z$ by $(f \times g)(z) = \sum_{x,y \in Z, z=x\cdot y} f(x) \times g(y)$. The sum ranges over all pairs (x, y) for which the concatenation is defined and such that $x \cdot y = z$. The sum may be finite or infinite but it is nonempty considering the left and right partial units of z. Finally, we let 1_Y be the characteristic function of the set Y of partial units of Z and we can easily check that it is a unit for the Cauchy product:

$$(f \times 1_Y)(z) = \sum_{\substack{x,y \in Z \\ z=x\cdot y}} f(x)1_Y(y) = \sum_{\substack{x \in Z, y \in Y \\ z=x\cdot y}} f(x) = \sum_{\substack{y \in Y \\ z=z\cdot y}} f(z) = f(z)\,.$$

Mimicking the proof for classical monoids, we can show the following.

Proposition 1. *If $\mathbb{S}$ is a continuous semiring and $(Z, \cdot, Y)$ is a partial monoid then the series $\mathbb{S}\langle\langle Z\rangle\rangle$ forms a continuous semiring $(\mathbb{S}\langle\langle Z\rangle\rangle, +, \times, 0, 1_Y)$.*

This allows to apply the theory of rational series over continuous semirings (see e.g., [28, III.5]). In particular, a star operation may be defined.

We can check that the semantics of pebWE in the continuous semiring $\mathbb{S}\langle\langle \mathrm{Mk}(A^+)\rangle\rangle$ as defined in Table 1 satisfies

$$\begin{aligned} [\![E + F]\!] &= [\![E]\!] + [\![F]\!] \qquad & [\![E^*]\!] &= [\![E]\!]^* \\ [\![E \cdot F]\!] &= [\![E]\!] \times [\![F]\!] \qquad & [\![E^+]\!] &= [\![E]\!]^+ \,. \end{aligned} \tag{1}$$

5 Weighted Automata with Pebbles

We fix a finite set Peb of pebbles and a (continuous) semiring $\mathbb{S}$. We denote by $\mathrm{Move} = \{\leftarrow, \rightarrow, \uparrow\} \cup \{\downarrow_x \mid x \in \mathrm{Peb}\}$ the set of possible moves of an automaton.

A *pebble weighted automaton* (pebWA) is a tuple $\mathcal{A} = (Q, A, I, M, T)$ with Q a finite set of states, A a finite alphabet, $I \in \mathbb{S}^Q$ a row vector assigning an initial weight to each state, $T \in \mathbb{S}\langle\mathrm{Test}\rangle^Q$ a column vector assigning to each state a polynomial over tests, and $M \in (\mathbb{S}\langle\mathrm{Test}\rangle\langle\mathrm{Move}\rangle)^{Q\times Q}$ the transition matrix.

We explain first the semantics of a pebWA on the automaton $\mathcal{A}_1$ represented in Figure 2 with its matrix representation on the right.

Intuitively, we enter state 1 with weight 5. We can loop on state 1 if the current letter is either an a or a b, in which case we move right in the word. The weight of this loop is 2 or 3 depending on the current letter. If $\mathcal{A}_1$ reads letter c while being in state 1, then it drops pebble x and restarts at the beginning of the word in state 2. There, it moves right in the word, either staying in state 2 with weight 1 (provided the current position does not carry the pebble), or going to state 3 with weight 7. Once we reach state 3, we must lift the pebble and go to state 4. Then, we move right coming back to state 1.

An accepting run of $\mathcal{A}_1$ must start in state 1 and end in state 1. The weight of a run is the product of the weights of its transitions. Over the natural semiring $(\mathbb{N}^\infty, +, \times, 0, 1)$, each accepting run of $\mathcal{A}_1$ has weight $5 \times 2^{|u|_a} \times 3^{|u|_b} \times 7^{|u|_c}$.

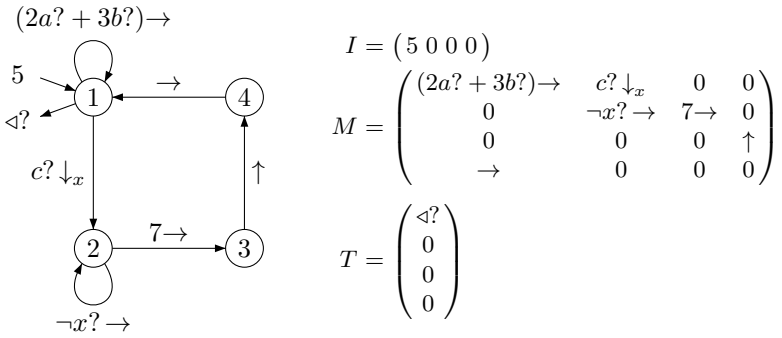


Fig. 2. A pebWA and its matrix representation.

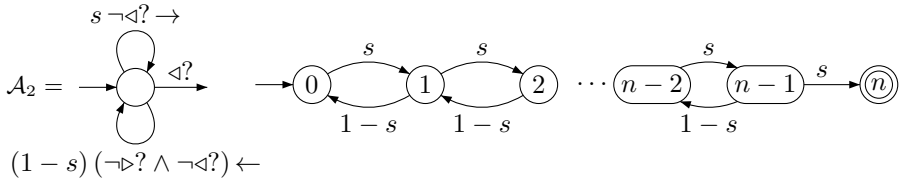


Fig. 3. Markov Chain obtained by synchronizing $\mathcal{A}_2$ with a word of length n

The non-deterministic choice in state 2 induces several runs. The semantics of the automaton is as usual the sum of the weights of all accepting runs. In our example,

$$[\![\mathcal{A}_1]\!](u) = 5 \times 2^{|u|_a} \times 3^{|u|_b} \times 7^{|u|_c} \times \prod_{\substack{i \in \mathrm{pos}(u) \\ u_i = c}} (i+1)\,.$$

Consider also the 2-way automaton $\mathcal{A}_2$ over the semiring $(\mathbb{R}^{\infty}_{\geq 0}, +, \times, 0, 1)$, with $0 < s < 1$. The matrix M of $\mathcal{A}_2$ admits as unique coefficient the polynomial $s\ \neg\triangleleft? \rightarrow + (1-s)\ (\neg\triangleright? \wedge \neg\triangleleft?) \leftarrow$, which, for clarity, we preferred to draw with two loops in Figure 3. This is a compact and elegant way of representing a Markov chain describing a random walk, see Figure 3. The same example was described with a pebWE in Section 4.

Remark 2. Notice that 2-way weighted expressions/automata are strictly more expressive than their 1-way versions (where left moves are disabled). The automaton $\mathcal{A}_2$ above for the random walk yields a counter-example. In fact, for $s = \frac{1}{2}$, the expression/automaton associates to a word u of length n the value $\frac{1}{n+1}$. This is not achievable by a 1-way weighted automaton over the semiring $(\mathbb{R}^{\infty}_{\geq 0}, +, \times, 0, 1)$ with weights in $\mathbb{Q}$. Intuitively, this is a consequence of the fact that the semantics of such an automaton is of the form $\frac{a}{b^c}$ with $a, b \in \mathbb{Z}$, $b \neq 0$ and $c \in \mathbb{N}$: here b can be defined as the least common denominator of the weights appearing over transitions of the automaton. Hence, taking a word u of length $p + 1$ with p a prime number greater than b is sufficient to prove that such a

1-way weighted automaton cannot map word u to weight $\frac{1}{p}$. Notice that this result strongly relies on the fact that we are dealing with infinite sums (because of the Kleene star, or the infinite number of runs). Indeed, it was shown in [8] that 2-way WA are not more expressive than 1-way WA when the semantics is restricted to non-looping runs.

As for expressions, we allow macros in M and T: for $a \in A$, we use $a \stackrel{\text{def}}{=} a? \cdot \rightarrow$ and $\overline{a} \stackrel{\text{def}}{=} a? \cdot \leftarrow$, for $d \in \text{Move}$, we write d instead of $\mathtt{tt}? \cdot d$. For instance, the label of the loop on state 1 of $\mathcal{A}_1$ could be written $2a + 3b$.

For each $p, q \in Q$ and $d \in \text{Move}$, we denote by $M^d_{p,q} \in \mathbb{S}\langle\text{Test}\rangle$ the coefficient of move d in $M_{p,q}$. For instance, $M^{\downarrow_x}_{1,2} = c?$ in $\mathcal{A}_1$. We collect these coefficients in matrices $M^d = (M^d_{p,q}) \in (\mathbb{S}\langle\text{Test}\rangle)^{Q \times Q}$.

We turn now to the formal definition of the semantics of pebWA. A *configuration* of $\mathcal{A}$ is a tuple (u, σ, q, i, π) with $u \in A^+$ a word, $\sigma \colon \text{Peb} \to \text{pos}(u)$ a valuation, $q \in Q$ the current state, $i \in \text{pos}(u)$ the current position, and $\pi \in (\text{Peb} \times \text{pos}(u))^*$ the stack of pebbles currently dropped. Since pebbles are reusable, the stack of pebbles may contain several occurrences of the same pebble dropped on different positions. In this case, only the last occurrence of each pebble is still visible for the automaton, older occurrences being hidden. This mechanism mimics the ability in pebWE to reuse the same pebble x in nested expressions $x!E$. We extract the visible pebbles from the stack π of dropped pebbles and the underlying valuation σ, hence defining a valuation σ_π by induction over π by $\sigma_\varepsilon = \sigma$ and $\sigma_{\pi(x,i)} = \sigma_\pi[x \mapsto i]$.

We define the semantics of pebWA in terms of a weighted transition system $\text{TS}(\mathcal{A})$ whose locations are the configurations of the automaton. The weight of $(u, \sigma, p, i, \pi) \rightsquigarrow (u, \sigma, q, j, \pi')$ is defined by

$$[\![M^{\rightarrow}_{p,q}]\!](u, \sigma_\pi, i, i) \quad \text{if } j = i + 1 \text{ and } \pi' = \pi \tag{S1}$$

$$[\![M^{\leftarrow}_{p,q}]\!](u, \sigma_\pi, i, i) \quad \text{if } j = i - 1 \text{ and } \pi' = \pi \tag{S2}$$

$$[\![M^{\downarrow_x}_{p,q}]\!](u, \sigma_\pi, i, i) \quad \text{if } j = 0,\ i < |u| \text{ and } \pi' = \pi(x, i) \tag{S3}$$

$$[\![M^{\uparrow}_{p,q}]\!](u, \sigma_\pi, i, i) \quad \text{if } \pi = \pi'(y, j) \text{ for some } y \in \text{Peb} \tag{S4}$$

where $[\![M^d_{p,q}]\!]$ is the semantics of $M^d_{p,q} \in \mathbb{S}\langle\text{Test}\rangle$, seen as a pebWE. Note from (S3) that a pebble cannot be dropped on position $|u|$ in agreement with the convention adopted for weighted expressions.

The set of transitions of $\text{TS}(\mathcal{A})$ consists of those $(u, \sigma, p, i, \pi) \rightsquigarrow (u, \sigma, q, j, \pi')$ with a non-zero weight: hence $\text{TS}(\mathcal{A})$ is a disjoint union of transition systems depending on the pair (u, σ) considered. A run of $\mathcal{A}$ is a path ρ in $\text{TS}(\mathcal{A})$. Its weight is the product of the weights of its transitions from left to right.

Given a marked word $(u, \sigma, i, j) \in \text{Mk}(A^+)$ and two states $p, q \in Q$, we define $[\![\mathcal{A}_{p,q}]\!](u, \sigma, i, j) = \sum_\rho \mathsf{weight}(\rho)$ where the sum ranges over all runs ρ from configuration $(u, \sigma, p, i, \varepsilon)$ to configuration $(u, \sigma, q, j, \varepsilon)$. This sum could be infinite, but is well defined since the semiring is complete. The semantics of $\mathcal{A}$

also use the initial and terminal weights:

$$[\![\mathcal{A}]\!](u,\sigma,i,j) = \sum_{p,q\in Q} I_p \times [\![\mathcal{A}_{p,q}]\!](u,\sigma,i,j) \times [\![T_q]\!](u,\sigma,j,j)\,.$$

When reading the whole word, we simply write $[\![\mathcal{A}]\!](u,\sigma)$ for $[\![\mathcal{A}]\!](u,\sigma,0,|u|)$. Note that we can compute the set of free pebbles of an automaton, i.e., the set of pebbles x that may be tested with $x?$ before being dropped with $\downarrow_x$. If the automaton has no free pebble, then the underlying valuation σ is not necessary and we simply write $[\![\mathcal{A}]\!](u)$ for the semantics.

Layered automata. As observed in automaton $\mathcal{A}_1$, it is handy, if possible, to visualize a pebWA in terms of layers, where each layer contains subruns where no pebble is dropped or lifted. We will require in the following that there are a finite number of such layers: intuitively, this means that the depth of the current stack of pebbles is bounded by a fixed parameter K. Remark however that the stack may contain several occurrences of the same pebble. Also, due to the 2-way mechanism, runs may still be of unbounded size. More formally, we assume given a function $\ell\colon Q \to \{0,\dots,K\}$ mapping each state to its layer. The top layer is K so $\ell(q)$ is the number of pebbles that can still be dropped on top of the stack. We want to start and end the computation at the top layer so we suppose that for all $q \in Q$, if $I_q \neq 0$ or $T_q \neq 0$ then $\ell(q) = K$. To maintain syntactically the condition along every possible run, we also suppose for all $p,q \in Q$ that if $M^{\leftarrow}_{p,q} \neq 0$ or $M^{\rightarrow}_{p,q} \neq 0$ then $\ell(q) = \ell(p)$; if $M^{\uparrow}_{p,q} \neq 0$ then $\ell(q) = \ell(p)+1$; and for all $x \in \mathrm{Peb}$, if $M^{\downarrow_x}_{p,q} \neq 0$ then $\ell(q) = \ell(p)-1$. An automaton $\mathcal{A}$ verifying these conditions will be called *K-layered* in the following. If we order states by decreasing layers, a 2-layered automaton $\mathcal{A} = (Q, A, I, M, T)$ is thus of the form

$$I = \begin{pmatrix} I^{(2)} & 0 & 0 \end{pmatrix},\ M = \begin{pmatrix} N^{(2)} & D^{(2)} & 0 \\ L^{(1)} & N^{(1)} & D^{(1)} \\ 0 & L^{(0)} & N^{(0)} \end{pmatrix},\ T = \begin{pmatrix} T^{(2)} \\ 0 \\ 0 \end{pmatrix} \tag{2}$$

where entries in $N^{(i)}$ are in $\mathbb{S}\langle\mathrm{Test}\rangle\langle\{\leftarrow,\rightarrow\}\rangle$, entries in $L^{(i)}$ are in $\mathbb{S}\langle\mathrm{Test}\rangle\langle\{\uparrow\}\rangle$, and entries in $D^{(i)}$ are in $\mathbb{S}\langle\mathrm{Test}\rangle\langle\{\downarrow_x \mid x \in \mathrm{Peb}\}\rangle$. The entries of $I^{(2)}$ and $T^{(2)}$ are as usual in $\mathbb{S}$ and $\mathbb{S}\langle\mathrm{Test}\rangle$ respectively.

6 From Automata to Expressions

In this section, we prove that every K-layered pebble weighted automaton admits an equivalent pebble weighted expression. We first show that, for 2-way weighted automata (or 0-layered pebWA), we can use the classical constructions, e.g., the state elimination method of Brzozowski and McCluskey [11], the procedure of McNaughton and Yamada [25] or the recursive algorithm [14]. We refer to the

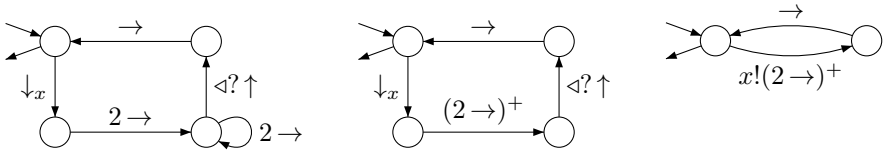


Fig. 4. A pebWA and two equivalent generalized pebWA.

survey of Sakarovitch [30, Section 6.2] where these methods are presented and compared for 1-way weighted automata.

In the state elimination method, states are progressively suppressed and transitions are labeled with (weighted) rational expressions. To deal with pebbles, we will also eliminate the lower layers and subsume their computations with expressions of the form $x!E$. Therefore, it is convenient to consider automata allowing pebWE in the labels of transitions.

We first introduce these *generalized* pebWA. Then, we show how to compute pebWE equivalent to the behaviors of 0-layered generalized pebWA. Finally, we explain how to deal with drop and lift moves of K-layered automata.

6.1 Generalized Pebble Automata

We start with an example presented in Figure 4. The loop of the left automaton gives rise to the iteration $(2\rightarrow)^+$ on the middle automaton. Moreover, the drop/lift process has even been replaced with the $x!-$ feature of pebWE in the right automaton. This gives already the intuition of the construction of a pebWE equivalent to a pebWA. Note that the right automaton has a single layer whereas the left and middle ones have 2 layers.

Formally, a *generalized* pebWA (GpebWA) is a tuple $\mathcal{A} = (Q, A, I, M, T)$ with $I \in \mathbb{S}^Q$, $M \in (\text{pebWE} + \mathbb{S}\langle\text{Test}\rangle\langle\{\uparrow\} \cup \{\downarrow_x \mid x \in \text{Peb}\}\rangle)^{Q\times Q}$ and $T \in \mathbb{S}\langle\text{Test}\rangle^Q$. Intuitively, the entries $M^{\leftarrow}_{p,q} \cdot \leftarrow + M^{\rightarrow}_{p,q} \cdot \rightarrow$ have been extended to arbitrary pebWE $M^{\text{pebWE}}_{p,q}$. The semantics of pebWA is easily extended to GpebWA. In fact, we only have to replace (S1-S2) by (G1-2) below:

$$[\![M^{\text{pebWE}}_{p,q}]\!](u, \sigma_\pi, i, j) \quad \text{if } \pi' = \pi\,. \tag{G1-2}$$

The definition of K-layered automata can easily be extended to GpebWA. Layered automata are still of the form given in (2), the only difference being that the entries of matrices $N^{(i)}$ are now pebWE instead of simple polynomials in $\mathbb{S}\langle\text{Test}\rangle\langle\{\leftarrow, \rightarrow\}\rangle$. It is clear from the definition that every (K-layered) pebWA can be seen as a (K-layered) GpebWA.

6.2 Automata to expressions: 0-layered generalized pebWA

We deal in this section with GpebWA $\mathcal{A} = (Q, A, I, M, T)$ with no drop or lift transitions, i.e., 0-layered GpebWA where the entries of the transition matrix are all pebWE: $M \in \text{pebWE}^{Q\times Q}$.

We first relate the semantics of the automaton with the star of the semantic matrix $[\![M]\!]$. For each $p, q \in Q$, the entry $M_{p,q}$ is a pebWE and its semantics $[\![M_{p,q}]\!] \in \mathbb{S}\langle\langle \mathrm{Mk}(A^+)\rangle\rangle$ is a series over marked words. Hence, the semantic matrix $[\![M]\!] = ([\![M_{p,q}]\!])_{(p,q)\in Q^2} \in \mathbb{S}\langle\langle \mathrm{Mk}(A^+)\rangle\rangle^{Q\times Q}$.

Recall that $\mathbb{K} = \mathbb{S}\langle\langle \mathrm{Mk}(A^+)\rangle\rangle$ is a continuous semiring by Proposition 1. For each finite set Q, the semiring of matrices $\mathbb{K}^{Q\times Q}$ is also continuous. Hence, given a matrix H in $\mathbb{K}^{Q\times Q}$, the *star* matrix $H^* = \sum_{n\geq 0} H^n \in \mathbb{K}^{Q\times Q}$ is well-defined. Applying this to $H = [\![M]\!]$ we can generalize the classical relation between the semantics of a 1-way WA and the star of a matrix of series.

Theorem 3. *Let $\mathcal{A} = (Q, A, I, M, T)$ be a 0-layered* GpebWA*. For all states $p, q \in Q$, we have $[\![\mathcal{A}_{p,q}]\!] = ([\![M]\!]^*)_{p,q}$, i.e., $[\![\mathcal{A}_{p,q}]\!](u, \sigma, i, j) = ([\![M]\!]^*)_{p,q}(u, \sigma, i, j)$ for all marked words $(u, \sigma, i, j) \in \mathrm{Mk}(A^+)$.*

Proof. We show by induction on $n \geq 0$ that for every $(u, \sigma, i, j) \in \mathrm{Mk}(A^+)$, $([\![M]\!]^n)_{p,q}(u, \sigma, i, j)$ computes the sum of the weights of runs of length n from configuration $(u, \sigma, p, i, \varepsilon)$ to configuration $(u, \sigma, q, j, \varepsilon)$.

This is true for $n = 0$, as $1_{\mathrm{Unit}(A^+)}(u, \sigma, i, j) = 1$ if and only if $i = j$. Then, assuming the property for $n - 1 \geq 0$, we prove it for n. A run of length $n > 0$ starts with a transition followed by a run of length $n - 1$. Hence, the sum of the weights of runs of length n from configuration $(u, \sigma, p, i, \varepsilon)$ to configuration $(u, \sigma, q, j, \varepsilon)$ is computed by

$$\sum_{r\in Q, k\in \mathrm{pos}(u)} [\![M]\!]_{p,r}(u, \sigma, i, k) \times ([\![M]\!]^{n-1})_{r,q}(u, \sigma, k, j)$$

which is equal to $([\![M]\!]^n)_{p,q}(u, \sigma, i, j)$ by definition of the matrix multiplication induced by the Cauchy product. □

Now, it is well known that the entries of the star of a matrix H are in the rational closure[5] of the entries of H [14]. Hence, we obtain:

Theorem 4. *Let $\mathcal{A} = (Q, A, I, M, T)$ be a 0-layered* GpebWA*. We can construct a matrix $\Phi(M) \in \mathrm{pebWE}^{Q\times Q}$ which is equivalent to the automaton, i.e., such that $[\![\Phi(M)]\!] = [\![M]\!]^*$. Moreover, the entries of $\Phi(M)$ are in the* rational closure *of the entries of M.*

The matrix $\Phi(M)$ can be constructed from M using one of the classical algorithm, e.g., the recursive algorithm used in the proof below. We can also apply McNaughton-Yamada algorithm, or the state elimination method, or the system resolution method starting from any initial state p and final state q.

Proof. We use the recursive method, due to Conway, to construct a matrix $\Phi(M)$ of pebWE such that $[\![\Phi(M)]\!] = [\![M]\!]^*$.

[5] The rational closure is the closure under sum (+) concatenation (·) and star (*).

Let $N \in \mathbb{K}^{Q\times Q}$ be a matrix over a continuous semiring $\mathbb{K}$. For any block decomposition $N = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ with A and D square matrices, we have [14]

$$N^* = \begin{pmatrix} A' & A'BD' \\ D'CA' & D' + D'CA'BD' \end{pmatrix}. \tag{3}$$

with $D' = D^*$ and $A' = (A + BD'C)^*$.

Following (3) we define $\Phi(M)$ inductively. If $|Q| = 1$, i.e., when $M \in \text{pebWE}$, then we simply let $\Phi(M) = M^*$. From the compositionality of the semantics of pebWE (1), we obtain $[\![\Phi(M)]\!] = [\![M]\!]^*$.

Next, if $|Q| > 1$ we consider the block decomposition $M = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$ with H of size 1. Then, we let

$$\Phi(M) = \begin{pmatrix} E' & E'FH' \\ H'GE' & H' + H'GE'FH' \end{pmatrix}. \tag{4}$$

with $H' = H^*$ and $E' = \Phi(E + FH'G)$. We apply (3) with $\mathbb{K} = \mathbb{S}\langle\langle \text{Mk}(A^+)\rangle\rangle$, $N = [\![M]\!]$ and the block decomposition with D of size 1. Hence, we have $[\![E]\!] = A$, $[\![F]\!] = B$, $[\![G]\!] = C$ and $[\![H]\!] = D$. From the compositionality of the semantics of pebWE (1), we obtain $[\![H']\!] = [\![H]\!]^* = D^* = D'$. By induction we get $[\![E']\!] = [\![\Phi(E + FH'G)]\!] = [\![E + FH'G]\!]^*$. Using again the compositionality (1), we get $[\![E']\!] = [\![E+FH'G]\!]^* = A'$, as well as $[\![E'FH']\!] = A'BD'$ and $[\![H'GE']\!] = D'CA'$. From (3-4) we deduce $[\![\Phi(M)]\!] = [\![M]\!]^*$. □

6.3 Dynamically Marked Words

In order to deal also with drop and lift moves of GpebWA, we first describe the semantics at a higher level, as we did in Section 4.1 for pebWE. If we only consider weighted automata with two-way navigation, marked words are suitable to define an equivalent semantics of pebWA. However, as soon as we introduce drop and lift moves, it is necessary to encode the contents of the stack of pebbles in the marked words, as it will change throughout the computation. Hence, we consider the partial monoid $(\text{DMk}(A^+), \cdot, \text{DUnit}(A^+))$ of *dynamically* marked words over A^+ where

$$\begin{aligned}\text{DMk}(A^+) = \{(u, \sigma, i, \pi, i', \pi') \mid u \in A^+,\ \sigma\colon \text{Peb} \to \text{pos}(u),\\ i, i' \in \text{pos}(u),\ \pi, \pi' \in (\text{Peb} \times \text{pos}(u))^*\}\\ \text{DUnit}(A^+) = \{(u, \sigma, i, \pi, i, \pi) \mid u \in A^+,\ \sigma\colon \text{Peb} \to \text{pos}(u),\\ i \in \text{pos}(u),\ \pi \in (\text{Peb} \times \text{pos}(u))^*\}\end{aligned}$$

and the partial concatenation is defined for all $u \in A^+$, $\sigma\colon \text{Peb} \to \text{pos}(u)$, $i, i', i'' \in \text{pos}(u)$, $\pi, \pi', \pi'' \in (\text{Peb} \times \text{pos}(u))^*$ by

$$(u, \sigma, i, \pi, i', \pi') \cdot (u, \sigma, i', \pi', i'', \pi'') = (u, \sigma, i, \pi, i'', \pi'')$$

and it is undefined in all other cases. We can check that this is indeed a partial monoid, so that, by Proposition 1, $(\mathbb{S}\langle\langle \text{DMk}(A^+)\rangle\rangle, +, \times, 0, 1_{\text{DUnit}(A^+)})$ is a continuous semiring.

6.4 Automata to expressions: K-layered generalized pebWA

It is now possible to give a natural semantics of drop and lift moves as series over $\mathrm{DMk}(A^+)$:

$$[\![\downarrow_x]\!](u,\sigma,i,\pi,i',\pi') = \begin{cases} 1 & \text{if } i < |u|,\ i' = 0,\ \pi' = \pi(x,i) \\ 0 & \text{otherwise,} \end{cases}$$

$$[\![\uparrow]\!](u,\sigma,i,\pi,i',\pi') = \begin{cases} 1 & \text{if } \pi = \pi'(y,i') \text{ for some } y \in \mathrm{Peb} \\ 0 & \text{otherwise.} \end{cases}$$

Actually, it is also possible to introduce *dynamic* pebble weighted expressions (DpebWE) simply by replacing the $x!-$ construct by new atoms for drop and lift moves:

$$F ::= s \mid \varphi \mid \rightarrow \mid \leftarrow \mid \downarrow_x \mid \uparrow \mid F + F \mid F \cdot F \mid F^* .$$

The *dynamic* semantics $[\![F]\!]$ of a DpebWE is a series over $\mathrm{DMk}(A^+)$. The semantics of $\downarrow_x$ and $\uparrow$ is already given above. For another atom F, it is inherited from the semantics of pebWE:

$$[\![E]\!](u,\sigma,i,\pi,i',\pi') = \begin{cases} [\![E]\!](u,\sigma_\pi,i,i') & \text{if } \pi' = \pi \\ 0 & \text{otherwise.} \end{cases}$$

Finally, since $\mathbb{S}\langle\langle \mathrm{DMk}(A^+) \rangle\rangle$ is a continuous semiring, the semantics of sum, concatenation and star is obtained compositionnally:

$$[\![E + F]\!] = [\![E]\!] + [\![F]\!] \qquad [\![E \cdot F]\!] = [\![E]\!] \times [\![F]\!] \qquad [\![E^*]\!] = [\![E]\!]^* .$$

We can now mimick Section 6.2 in order to translate automata into DpebWE. For instance, the DpebWE $(\downarrow_x (2\rightarrow)^+ \lhd ?\uparrow\rightarrow)^*$ is equivalent to the pebWA on the left of Figure 4.

Actually, we can deal with DpebWA which are automata $\mathcal{A} = (Q, A, I, M, T)$ where transitions are labelled with DpebWE, i.e., $M \in \mathrm{DpebWE}^{Q\times Q}$. The semantics is obtained as for pebWA via a weighted transition system $\mathrm{TS}(\mathcal{A})$: the weight of a transition $(u,\sigma,p,i,\pi) \rightsquigarrow (u,\sigma,q,i',\pi')$ is now $[\![M_{p,q}]\!](u,\sigma,i,\pi,i',\pi')$. Then, for $(u,\sigma,i,\pi,i',\pi') \in \mathrm{DMk}(A^+)$, we define $[\![\mathcal{A}_{p,q}]\!](u,\sigma,i,\pi,i',\pi')$ as the sum of weights of the runs from configuration (u,σ,p,i,π) to configuration (u,σ,q,i',π'). Copying, mutatis mutandis, the proofs of Theorems 3 and 4 we obtain:

Proposition 5. *Let $\mathcal{A} = (Q, A, I, M, T)$ be a* DpebWA*. For all $p, q \in Q$, we have $[\![\mathcal{A}_{p,q}]\!] = ([\![M]\!]^*)_{p,q}$. Moreover, the matrix $\Phi(M) \in \mathrm{DpebWE}^{Q\times Q}$ (whose entries are in the rational closure of the entries of M) satisfies $[\![\Phi(M)]\!] = [\![M]\!]^*$.*

In particular, the semantics of a DpebWA verifies: $[\![\mathcal{A}]\!] = [\![I]\!] \times [\![M]\!]^* \times [\![T]\!]$.

Remark 6. The partial monoid $\mathrm{Mk}(A^+)$ is embedded in $\mathrm{DMk}(A^+)$ by identifying the marked word (u,σ,i,i') with the dynamically marked word $(u,\sigma,i,\varepsilon,i',\varepsilon)$. Also, any pebWA $\mathcal{A} = (Q, A, I, M, T)$ is a DpebWA and the semantics over $\mathrm{Mk}(A^+)$ coincide: $[\![\mathcal{A}_{p,q}]\!](u,\sigma,i,i') = [\![\mathcal{A}_{p,q}]\!](u,\sigma,i,\varepsilon,i',\varepsilon)$.

Hence, Proposition 5 constructs a DpebWE equivalent over $\mathrm{Mk}(A^+)$ to the pebWA $\mathcal{A}$. But our aim is to construct a pebWE which is equivalent to $\mathcal{A}$. This is achieved below using as a tool the detour via DpebWE.

First, we show that pebWE can be seen as a fragment of DpebWE if we interpret $x!E$ as a macro for $\downarrow_x \cdot E \cdot \triangleleft? \cdot \uparrow$. Indeed with this interpretation, the semantics coincide:

Lemma 7. *Let E be a* pebWE *and let $(u, \sigma, i, \pi, i', \pi') \in \mathrm{DMk}(A^+)$. Then, $[\![E]\!](u, \sigma, i, \pi, i', \pi') \neq 0$ implies $\pi' = \pi$. Moreover, if $\pi' = \pi$ then*

$$[\![E]\!](u, \sigma, i, \pi, i', \pi') = [\![E]\!](u, \sigma_\pi, i, i')\,.$$

Proof. We proceed by structural induction on the pebWE. For atoms of pebWE, the result is clear from the definition of $[\![-]\!]$. For sum, concatenation and star, the result is trivial since both semantics are compositional. The interesting case is $x!E$ which is interpreted as $\downarrow_x \cdot E \cdot \triangleleft? \cdot \uparrow$ in DpebWE. By definition of $[\![-]\!]$ and the Cauchy product in $\mathbb{S}\langle\!\langle \mathrm{DMk}(A^+)\rangle\!\rangle$ we have

$$s = [\![\downarrow_x \cdot E \cdot \triangleleft? \cdot \uparrow]\!](u, \sigma, i, \pi, i', \pi') = \sum_{y \in \mathrm{Peb}} [\![E]\!](u, \sigma, 0, \pi(x, i), |u|, \pi'(y, i'))\,.$$

Since $E \in$ pebWE, we obtain by induction that each term of the sum above is null if $\pi(x, i) \neq \pi'(y, i')$. Hence, $s \neq 0$ implies $\pi' = \pi$. Moreover, if $\pi' = \pi$ then either $i' \neq i$ and $s = 0 = [\![x!E]\!](u, \sigma_\pi, i, i')$, or $i' = i$ and

$$s = [\![E]\!](u, \sigma, 0, \pi(x, i), |u|, \pi(x, i)) = [\![E]\!](u, \sigma_{\pi(x,i)}, 0, |u|) = [\![x!E]\!](u, \sigma_\pi, i, i)$$

where the second equality holds by induction and the third one follows from $\sigma_{\pi(x,i)} = \sigma_\pi[x \mapsto i]$. □

We will extend Theorem 4 to any K-layered GpebWA $\mathcal{A} = (Q, A, I, M, T)$. For $i \leq K$, we let $Q^{(i)} = \ell^{-1}(i)$ be the set of states in layer i. Note that a GpebWA is also a DpebWA hence we may apply the above results.

Proposition 8. *Let $\mathcal{A} = (Q, A, I, M, T)$ be a 1-layered* GpebWA*. We can construct a 0-layered* GpebWA *$\mathcal{A}^{(1)} = (Q^{(1)}, A, I^{(1)}, M^{(1)}, T^{(1)})$ which is equivalent to $\mathcal{A}$: $[\![\mathcal{A}_{p,q}]\!] = [\![\mathcal{A}^{(1)}_{p,q}]\!]$ for all $p, q \in Q^{(1)}$.*

We use the layered decomposition given in (2). To simplify the notation, we write $N = N^{(1)}$, $D = D^{(1)}$, $L = L^{(0)}$ and $P = N^{(0)}$ so that $M = \begin{pmatrix} N & D \\ L & P \end{pmatrix}$.

Let $p, q \in Q^{(1)}$ be in layer 1 and $p', q' \in Q^{(0)}$ be in layer 0. Then, D is a *drop*-matrix whose (p, p')-entry can be written $\sum_{x \in \mathrm{Peb}} d^x_{p,p'} \cdot \downarrow_x$ with $d^x_{p,p'} \in \mathbb{S}\langle\mathrm{Test}\rangle$. The (q', q)-entry of the *lift*-matrix L can be written $e_{q',q} \cdot \uparrow$ with $e_{q',q} \in \mathbb{S}\langle\mathrm{Test}\rangle$. Now, P is a $Q^{(0)} \times Q^{(0)}$ matrix of pebWE and we may apply Proposition 5 in order to get a matrix $\Phi(P)$ of pebWE which is equivalent to the iteration of P: $[\![\Phi(P)]\!] = [\![P]\!]^*$. From (D, P, L), we define the $Q^{(1)} \times Q^{(1)}$ pebWE-matrix G by

$$G_{p,q} = \sum_{p',q'} \sum_{x \in \mathrm{Peb}} d^x_{p,p'} \cdot x!\big(\Phi(P)_{p',q'} \cdot e_{q',q} \cdot \rightarrow^*\big)\,.$$

The matrix G is also denoted $C(D, P, L)$ below. Note that the maximal pebble-depth of the entries of G is at most 1 plus the maximal pebble-depth of the entries of P since the construction $\Phi(P)$ does not increase the pebble-depth.

Lemma 9. *We have* $[\![G]\!] = [\![D]\!] \times [\![P]\!]^* \times [\![L]\!]$.

Proof. Recall that when viewing pebWE as DpebWE we interpret the expression $x!E$ as $\downarrow_x \cdot E \cdot \triangleleft? \cdot \uparrow$. Now, it is easy to check that $[\![\rightarrow^* \cdot \triangleleft? \cdot \uparrow]\!] = [\![\uparrow]\!]$, hence the pebWE $x!\big(\Phi(P)_{p',q'} \cdot e_{q',q} \cdot \rightarrow^*\big)$ is equivalent to $\downarrow_x \cdot \Phi(P)_{p',q'} \cdot e_{q',q} \cdot \uparrow$. We deduce that

$$[\![G_{p,q}]\!] = \sum_{p',q'} [\![D_{p,p'}]\!] \times [\![\Phi(P)_{p',q'}]\!] \times [\![L_{q',q}]\!] \,.$$

Since $[\![\Phi(P)]\!] = [\![P]\!]^*$ by Proposition 5, the result follows. □

To conclude the proof of Proposition 8, we simply set $M^{(1)} = N + G$. Now, for all $p, q \in Q^{(1)}$, we have $[\![\mathcal{A}_{p,q}]\!] = ([\![M]\!]^*)_{p,q}$ by Proposition 5. Moreover, the upper-left of $[\![M]\!]^*$ is $([\![N]\!] + [\![D]\!] \times [\![P]\!]^* \times [\![L]\!])^*$ which is, by Lemma 9, equal to $([\![N]\!] + [\![G]\!])^* = [\![M^{(1)}]\!]^*$. Using again Proposition 5 we get for all $p, q \in Q^{(1)}$ that $[\![\mathcal{A}^{(1)}_{p,q}]\!] = ([\![M^{(1)}]\!]^*)_{p,q} = ([\![M]\!]^*)_{p,q} = [\![\mathcal{A}_{p,q}]\!]$. Using Remark 6, we deduce that $[\![\mathcal{A}_{p,q}]\!] = [\![\mathcal{A}^{(1)}_{p,q}]\!]$.

Proposition 8 can then be generalized to an arbitrary number of layers.

Proposition 10. *Let* $\mathcal{A} = (Q, A, I, M, T)$ *be a* K*-layered* GpebWA. *We can construct a* 0*-layered* GpebWA $\mathcal{A}^{(K)} = (Q^{(K)}, A, I^{(K)}, M^{(K)}, T^{(K)})$ *which is equivalent to* $\mathcal{A}$*:* $[\![\mathcal{A}_{p,q}]\!] = [\![\mathcal{A}^{(K)}_{p,q}]\!]$ *for all* $p, q \in Q^{(K)}$.

Proof. We use again the notation of the layered decomposition. The proof is by induction on K. When $K = 0$ we simply have $\mathcal{A}^{(0)} = \mathcal{A}$, i.e., $M^{(0)} = N^{(0)}$. For $K > 0$, we set $M^{(K)} = N^{(K)} + C(D^{(K)}, M^{(K-1)}, L^{(K-1)})$ where the matrix $M^{(K-1)}$ is obtained by induction. Correctness follows from Proposition 8. □

Finally, from Theorem 4 and Proposition 10 we deduce:

Theorem 11. *Let* $\mathcal{A} = (Q, A, I, M, T)$ *be a* K*-layered* GpebWA. *The matrix* $H = \Phi(M^{(K)})$ *of* pebWE *satisfies* $[\![H_{p,q}]\!] = [\![\mathcal{A}_{p,q}]\!]$ *for all* $p, q \in Q^{(K)}$.
Therefore, the pebWE $E(\mathcal{A}) = I \times H \times T$ *is equivalent to* $\mathcal{A}$*:* $[\![E(\mathcal{A})]\!] = [\![\mathcal{A}]\!]$.
Moreover, the pebble-depth of $E(\mathcal{A})$ *is at most* K *if* $\mathcal{A}$ *is a* K*-layered* pebWA.

7 From Expressions to Automata

We describe in this section how to transform a weighted expression with pebbles to an equivalent weighted automaton with pebbles. Expressions are very convenient to denote in a rather clear and intuitive way the quantitative functions that we want to compute. On the other hand, automata are much more amenable to efficient algorithms, e.g., for evaluation as shown in Section 8. Hence, we need efficient translations from expressions to automata. Such translations have been

well-studied both in the boolean and in the weighted (1-way) cases. Glushkov's translation (or Berry-Sethi) is acknowledged to be among the best ones. The good news is that this construction can be adapted to cope with 2-way moves and pebbles as we will show in this section. The construction is by structural induction on the expression.

Theorem 12. *For each* pebWE E *we can construct a* layered pebWA $\mathcal{A}(E)$ *such that* $[\![\mathcal{A}(E)]\!] = [\![E]\!]$*, i.e., for all* $(u, \sigma, i, j) \in \mathrm{Mk}(A^+)$ *we have*

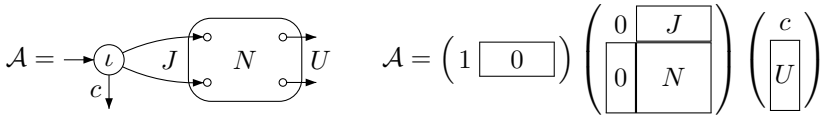
$$[\![\mathcal{A}(E)]\!](u, \sigma, i, j) = [\![E]\!](u, \sigma, i, j)\,.$$

Moreover, the number of layers in $\mathcal{A}(E)$ *is the pebble-depth of* E*.*

We define the *literal-length* $\ell\ell(E)$ of an expression as the number of occurrences of moves ($\leftarrow$ or $\rightarrow$) plus twice the number of occurrences of ! (in $x!-$). We will see that the number of states of $\mathcal{A}(E)$ will be $1 + \ell\ell(E)$. For a 2-way expression E of pebble-depth 0 (2-way-WE) the literal-length is simply the number of moves, which are the positions to be marked for Glushkov's construction.

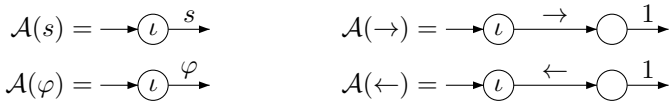
For the rational operations ($+$, $\cdot$, *, and $^+$), we can still use the classical constructions even though we are working with pebWA. We recall these constructions below for the sake of completeness. The main novelty is indeed the treatment of pebbles.

We adopt the presentation of standard automata by Sakarovitch [30]. A standard automaton $\mathcal{A} = (Q, A, I, M, T)$ has a single initial state ι with (initial) weight 1, all other states have initial weight 0. Moreover, the initial state ι has no ingoing transition. We use both the graphical representation and the matrix representation of an automaton:



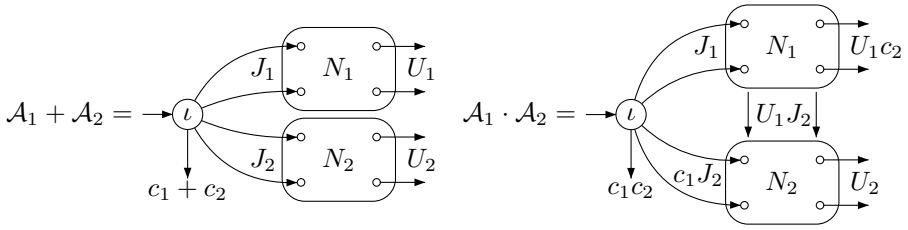
Since terminal weights allow polynomials over Test with the mapping $T\colon Q \to \mathbb{S}\langle\mathrm{Test}\rangle$, we will be able to cope with expressions of the form $E \cdot \varphi?$ and $E \cdot s$ without adding unnecessary states. For $s \in \mathbb{S}$ and $\varphi \in \mathrm{Test}$, we simply write s for $s\mathtt{tt}?$ and φ for 1φ, and also $\rightarrow$ for $1\mathtt{tt}?\rightarrow$ and $\leftarrow$ for $1\mathtt{tt}?\leftarrow$.

We start with atoms. Compared to the classical (1-way) translation, a slight difference is that we are using tests (φ) and moves ($\leftarrow$,$\rightarrow$) instead of letters ($a = a?\rightarrow$) for the atoms. The automata for the atoms are defined as



and we can easily see that they are equivalent to the corresponding atoms: if E is an atom then $[\![E]\!](u, \sigma, i, j) = [\![\mathcal{A}(E)]\!](u, \sigma, i, j)$ for all $(u, \sigma, i, j) \in \mathrm{Mk}(A^+)$.

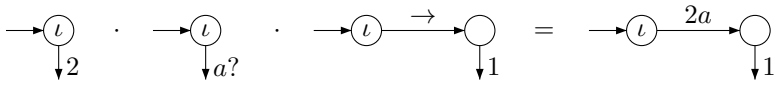
The constructions for sum and concatenation are as usual.



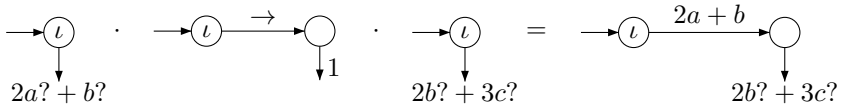
In the concatenation, we are overloading the product notation as follows. The product of two monomials $s_1\varphi_1$ and $s_2\varphi_2$ from $\mathbb{S}\langle\text{Test}\rangle$ should be understood as $(s_1 s_2)(\varphi_1 \wedge \varphi_2)$ to stay in $\mathbb{S}\langle\text{Test}\rangle$. Hence c_1c_2 and the entries of U_1c_2 are in $\mathbb{S}\langle\text{Test}\rangle$. Similarly, in U_1J_2, the product of a monomial $s_1\varphi_1 \in \mathbb{S}\langle\text{Test}\rangle$ and a monomial $s_2\varphi_2 d$ (with $d \in \text{Move}$) is defined as $(s_1 s_2)(\varphi_1 \wedge \varphi_2)d$. Hence, the entries of the matrices c_1J_2 and U_1J_2 are in $\mathbb{S}\langle\text{Test}\rangle\langle\text{Move}\rangle$. The matrix representation is therefore:

$$\mathcal{A}_1 \cdot \mathcal{A}_2 = \left(1 \;\middle|\; \begin{array}{c|c} 0 & 0 \end{array}\right) \left(\begin{array}{c|cc} 0 & J_1 & c_1J_2 \\ \hline 0 & N_1 & U_1J_2 \\ \hline 0 & 0 & N_2 \end{array}\right) \left(\begin{array}{c} c_1c_2 \\ \hline U_1c_2 \\ \hline U_2 \end{array}\right)$$

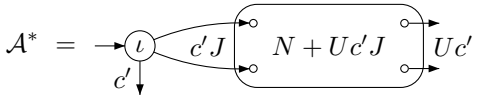
For instance, the automaton for $2a = 2 \cdot a? \cdot \to$ is computed as follows:



Similarly, for the expression $E = (2a? + b?) \to (2b? + 3c?)$ we compute the concatenation of 3 automata as follows:



Finally, the star is also computed as usual with the following construction.



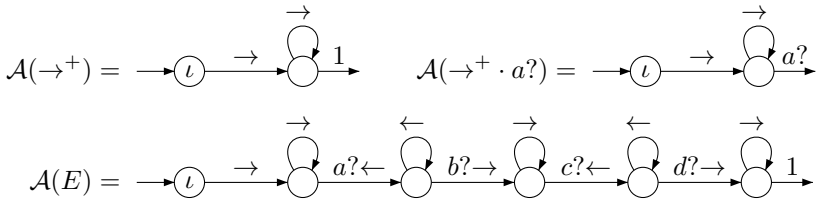
$$\mathcal{A}^* = \left(1 \;\middle|\; \begin{array}{c|c} 0 & 0 \end{array}\right) \left(\begin{array}{c|c} 0 & c'J \\ \hline 0 & N + Uc'J \end{array}\right) \left(\begin{array}{c} c' \\ \hline Uc' \end{array}\right)$$

where $c' \in \mathbb{S}\langle\text{Test}\rangle$ is defined to be equivalent to c^* as follows. Since $c \in \mathbb{S}\langle\text{Test}\rangle$ and test formulas are closed under boolean connectives, we find an equivalent

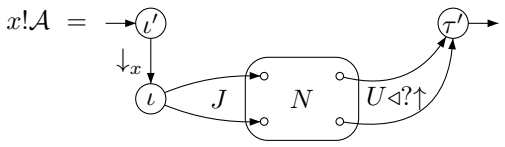
expression $c \equiv \sum_i s_i\varphi_i$ with $(\varphi_i)_i$ pairwise incompatible test formulas ($i \neq j$ implies $\varphi_i \wedge \varphi_j$ unsatisfiable). Then we can easily check that $[\![c]\!]^* = [\![c']\!]$ with $c' = \sum_i s_i^*\varphi_i$. Notice that $s_i^* \in \mathbb{S}$ is well-defined since the semiring is complete.

As for the concatenation, we can check that the entries of Uc' are in $\mathbb{S}\langle\text{Test}\rangle$ and the entries of $Uc'J$ are in $\mathbb{S}\langle\text{Test}\rangle\langle\text{Move}\rangle$. The strict iteration $\mathcal{A}^+$ is computed similarly by simply changing the final weight of ι to $c'' = \sum_i s_i^+\varphi_i$ (note that $0^+ = 0$), but keeping the other occurrences of c' in $c'J$, $Uc'J$ and Uc'.

For instance, for expression $E = \rightarrow^+ a? \leftarrow^+ b? \rightarrow^+ c? \leftarrow^+ d? \rightarrow^+$ introduced in Section 4, we can compute the automaton as follows:



Finally, we give the construction for $x!E$ which should drop the pebble on the current position, evaluate E from beginning to end ($\triangleleft?$) of the word and finally lift the pebble. From a standard automaton $\mathcal{A}$ equivalent to E, we construct the following standard automaton $x!\mathcal{A}$:

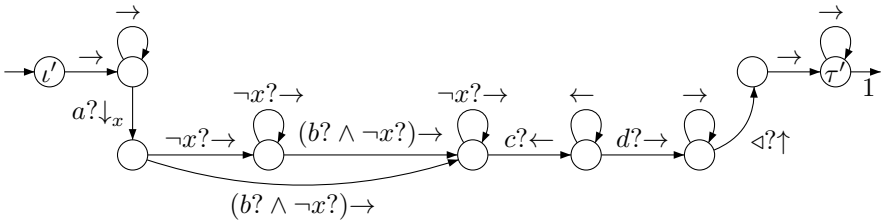


$$x!\mathcal{A} = \left(1\ 0 \middle| 0\ \boxed{0}\right) \left(\begin{array}{cc|c|c} 0 & 0 & \downarrow_x & \boxed{0} \\ 0 & 0 & 0 & \boxed{0} \\ \hline 0 & 0 & 0 & \boxed{J} \\ \hline \boxed{0} & \boxed{U\triangleleft?\uparrow} & \boxed{0} & \boxed{N} \end{array}\right) \left(\begin{array}{c} 0 \\ 1 \\ \hline 0 \\ \hline \boxed{0} \end{array}\right) \tag{5}$$

For instance, consider again expression E below used in Section 4:

$$E = \rightarrow^+ a?\, x!\Big((\neg x? \rightarrow)^*\, b?\, (\neg x? \rightarrow)^+\, c? \leftarrow^+ d? \rightarrow^+\Big) \rightarrow^* .$$

The construction applied to E gives the following pebWA.



Let us briefly discuss the complexity of our translation. Clearly, the number of states of the automaton $\mathcal{A}(E)$ is 1 plus the literal-length $\ell\ell(E)$ of expression E. The time complexity is cubic in the length of E. It should be possible to get a quadratic algorithm by generalizing the notion of star normal form introduced in [9] for word languages or the algorithm presented in [1] for classical weighted expressions and automata.

To conclude this section, we prove the correctness of the constructions. This correctness is trivial for the atoms. For sum, product and star, the proof is similar to the case of classical (1-way) weighted expressions and automata. For the sake of completeness, we give below the arguments for product and star, then we deal with the $x!-$ construction.

From Proposition 5, we know that $[\![\mathcal{A}]\!] = [\![I]\!] \times [\![M]\!]^* \times [\![T]\!]$. Using the special form of standard automata and (3), we have

$$[\![M]\!] = \begin{pmatrix} 0 & [\![J]\!] \\ 0 & [\![N]\!] \end{pmatrix} \qquad [\![M]\!]^* = \begin{pmatrix} 1 & [\![J]\!][\![N]\!]^* \\ 0 & [\![N]\!]^* \end{pmatrix}$$

and we get $[\![\mathcal{A}]\!] = [\![c]\!] + [\![J]\!][\![N]\!]^*[\![U]\!]$.

For the concatenation, the matrix N of $\mathcal{A} = \mathcal{A}_1 \cdot \mathcal{A}_2$ satisfies (applying (3)):

$$[\![N]\!] = \begin{pmatrix} [\![N_1]\!] & [\![U_1J_2]\!] \\ 0 & [\![N_2]\!] \end{pmatrix} \quad [\![N]\!]^* = \begin{pmatrix} [\![N_1]\!]^* & [\![N_1]\!]^*[\![U_1J_2]\!][\![N_2]\!]^* \\ 0 & [\![N_2]\!]^* \end{pmatrix}.$$

Hence we obtain

$$\begin{aligned}
[\![\mathcal{A}]\!] &= [\![c]\!] + [\![J]\!][\![N]\!]^*[\![U]\!] \\
&= [\![c_1c_2]\!] + [\![J_1]\!][\![N_1]\!]^*[\![U_1c_2]\!] \\
&\qquad + [\![J_1]\!][\![N_1]\!]^*[\![U_1J_2]\!][\![N_2]\!]^*[\![U_2]\!] + [\![c_1J_2]\!][\![N_2]\!]^*[\![U_2]\!] \\
&= ([\![c_1]\!] + [\![J_1]\!][\![N_1]\!]^*[\![U_1]\!])([\![c_2]\!] + [\![J_2]\!][\![N_2]\!]^*[\![U_2]\!]) = [\![\mathcal{A}_1]\!][\![\mathcal{A}_2]\!]
\end{aligned}$$

For the star construction, the correctness is obtained as follows using classical rational identities and $[\![c]\!]^* = [\![c']\!]$:

$$\begin{aligned}
[\![\mathcal{A}]\!]^* &= ([\![c]\!] + [\![J]\!][\![N]\!]^*[\![U]\!])^* = ([\![c]\!]^*[\![J]\!][\![N]\!]^*[\![U]\!])^*[\![c]\!]^* \\
&= ([\![c']\!][\![J]\!][\![N]\!]^*[\![U]\!])^*[\![c']\!] \\
&= [\![c']\!] + [\![c']\!][\![J]\!]([\![N]\!]^*[\![U]\!][\![c']\!][\![J]\!])^*[\![N]\!]^*[\![U]\!][\![c']\!] \\
&= [\![c']\!] + [\![c']\!][\![J]\!]([\![N]\!] + [\![U]\!][\![c']\!][\![J]\!])^*[\![U]\!][\![c']\!] \\
&= [\![c']\!] + [\![c'J]\!]([\![N + Uc'J]\!])^*[\![Uc']\!] = [\![\mathcal{A}^*]\!]
\end{aligned}$$

It remains to prove the correctness of the new construction for the $x!-$ operation. Assume for simplicity that $\mathcal{A}$ is a 0-layered automaton, then $\mathcal{A}' = x!\mathcal{A}$ is a 1-layered automaton and its layered decomposition is shown in (5). We

write the corresponding block decomposition of the transition matrix of $\mathcal{A}'$ as $M' = \begin{pmatrix} 0 & D \\ L & M \end{pmatrix}$ where M is the transition matrix of $\mathcal{A}$, D is the drop matrix with only non-zero entry being $D_{\iota',\iota} = \downarrow_x$, and L is the lift matrix with non-zero entries in the column $U\triangleleft?\uparrow$. By (3), the upper-left block of $[\![M']\!]^*$ is $[\![D]\!][\![M]\!]^*[\![L]\!]$, hence we have

$$[\![\mathcal{A}']\!] = ([\![M']\!]^*)_{\iota',\tau'} = ([\![D]\!][\![M]\!]^*[\![L]\!])_{\iota',\tau'} = [\![G]\!]_{\iota',\tau'}$$

where $G = C(D, M, L)$ (see Lemma 9). Let Q be the set of non-initial states of $\mathcal{A}$. We have,

$$G_{\iota',\tau'} = \textstyle\sum_{q\in Q} x!\big((\Phi(M)_{\iota,q}U_q)\triangleleft?\to^*\big) \equiv x!\Big(\sum_{q\in Q}\Phi(M)_{\iota,q}U_q\Big)\,.$$

We conclude using

$$[\![\mathcal{A}]\!] = [\![I]\!][\![M]\!]^*[\![T]\!] = [\![I]\!][\![\Phi(M)]\!][\![T]\!] = [\![c + \textstyle\sum_{q\in Q}\Phi(M)_{\iota,q}U_q]\!]$$

and $x!(c + F) \equiv x!F$ when $c \in \mathbb{S}\langle\mathrm{Test}\rangle$ and $F \in \mathrm{pebWE}$ since we only consider nonempty words and the $x!-$ construct requires to read the whole word.

8 Evaluation of pebble weighted automata

In this section, we study the evaluation problem of a K-layered pebWA $\mathcal{A}$ with reusable pebbles: given a word u and a valuation $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(u)$, compute $[\![\mathcal{A}]\!](u, \sigma)$. The challenge is important since, even if the word is fixed, the number of accepting runs may be infinite. We will prove in particular that the complexity of the evaluation problem is only linear in the size of the input word when the number p of reusable pebbles is at most 1.

The computations are done with matrix operations sum, product and star. But it is important to know the cost in scalar operations. This is well-known for sum and product of matrices. We show below that computing the star of a square matrix has the same complexity as computing the product of matrices. Indeed, we can improve the cubic complexity of product and star by using a better algorithm for computing the product of matrices.

Lemma 13. *Let $\mathbb{K}$ be a continuous semiring and $N \in \mathbb{K}^{n\times n}$ be a square matrix of dimension n. We can compute N^* with $\mathcal{O}(n)$ scalar star operations and $\mathcal{O}(n^3)$ scalar sum and product operations.*

Proof. We use (3) with a block decomposition of N such that D is a scalar, i.e., a block of size 1. Assuming that A' is already computed, (3) allows to compute N^* with one scalar star operation and a quadratic number of scalar sum and product operations. The result follows by induction. □

Before explaining, how to evaluate efficiently pebWA, let us recall how this can be done for classical finite-state automata. In case of a deterministic finite-state automaton, we can easily check whether a word u is accepted by following

the single computation in the automaton labeled by u, from left to right: at every position of the word, we only have to keep *one memory cell* containing the state reached after the current prefix of u. This memory cell can be updated with each letter in constant time using the transition function. On the overall, this leads to a complexity $\mathcal{O}(|u|)$ (in particular, independent of the automaton size). For a non-deterministic finite-state automaton, this complexity is not achievable anymore. A first solution consists of determinizing the automaton and then applying previous method: we obtain a complexity $\mathcal{O}(2^n + |u|)$ where n is the number of states of the NFA. This method cannot be extended to weighted automata, as they are not necessarily determinizable. Hence, we would rather use a dynamic programming method. We will pay the price of non-determinism by using more *memory cells* during the left-to-right computation of the runs of the automaton over the word. We use one memory cell s_q for every state q of the automaton. Cell s_q is a boolean which is true after reading a word u if and only if there exists a run labeled u leading from an initial state to state q. Cells are initially true for initial states and false otherwise. The update of the cells when reading a letter a is done by the multiplication of the row vector of cells (of dimension $1 \times |Q|$) with the adjacency matrix of the transitions of the automaton labeled with letter a. Hence, each update requires a number of operations $\mathcal{O}(n^2)$, leading to an overall complexity of $\mathcal{O}(n^2|u|)$. This method can naturally be extended to the weighted setting using memory cells containing values in $\mathbb{S}$. More precisely, cell s_q holds the sum of weights of runs starting in an initial state and ending in state q. The update now use the weighted transition matrix and can be done with the same complexity.

We now explain how to evaluate layered pebWA with similar methods. Let $\mathcal{A} = (Q, A, I, M, T)$ be a K-layered pebWA. As in Section 6.4, for $k \leq K$, we let $Q^{(k)} = \ell^{-1}(k)$ be the set of states in layer k. For each layer k we let $n_k = |Q^{(k)}|$. The total number of states is then $n = |Q| = \sum_{k=0}^{K} n_k$.

Theorem 14. *Given a K-layered* pebWA *with $p \geq 0$ reusable pebbles and a word $u \in A^+$, we can compute with $\mathcal{O}((p+1)|Q|^3|u|^{p+1})$ scalar operations (sum, product, star) the values $[\![\mathcal{A}_{q,q'}]\!](u, \sigma)$ for all layers $k \leq K$, states $q, q' \in Q^{(k)}$ and valuations $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$.*

Proof. We follow the same basic idea, recalled previously, used to evaluate classical weighted automata: memory cells are associated with each state computing weights of the runs from left to right. The 2-way navigation is then resolved by adding more memory cells (a quadratic number with respect to n), namely those that compute weights of the back and forth loops. Finally, we deal with layers inductively. In the whole proof, we fix a word $u \in A^+$.

Recall that, for all valuations $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$, layers $k \in \{0, \dots, K\}$ and states $q, q' \in Q^{(k)}$, the value $[\![\mathcal{A}_{q,q'}]\!](u, \sigma)$ is the sum of weights of the runs from configurations $(u, \sigma, q, 0, \varepsilon)$ to $(u, \sigma, q', |u|, \varepsilon)$: observe that the stack of pebbles is empty at the beginning of these runs, hence they stay in layers $k, k-1, \dots, 0$. In the following, these values (and others that will be defined later) will be

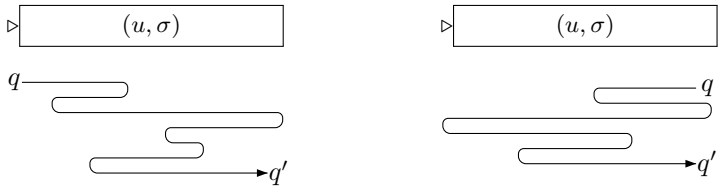


Fig. 5. Runs of a 2-way automaton

grouped into matrices indexed by subsets of states. In particular, we let $B_\sigma^{(k)} = (\llbracket \mathcal{A}_{q,q'} \rrbracket(u, \sigma))_{q,q' \in Q^{(k)}}$ be such a matrix indexed by $Q^{(k)} \times Q^{(k)}$.

Let $k \in \{0, \dots, K\}$ be a layer of the automaton. If $k > 0$, we assume by induction that for all valuations $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$ we have already computed the matrices $B_\sigma^{(k-1)}$. For each valuation $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$, we will compute the matrix $B_\sigma^{(k)}$ reading the word u from left to right. Formally, for $0 \leq i \leq |u|$, we define the matrix $B_\sigma^{\to i} = (B_{\sigma,q,q'}^{\to i})_{q,q' \in Q^{(k)}}$ where $B_{\sigma,q,q'}^{\to i}$ is the sum of weights of the runs from configuration $(u, \sigma, q, 0, \varepsilon)$ to $(u, \sigma, q', i, \varepsilon)$ with intermediary configurations of the form (u, σ, r, j, π) with $\pi \neq \varepsilon$ or $j \leq i$ (See left of Figure 5). These are the runs which move from the beginning of the word to position i, staying on the left of position i, unless some pebbles are currently dropped (if a pebble is dropped, then automaton can read the whole word). In order to compute inductively $B_\sigma^{\to i}$ using $B_\sigma^{\to i-1}$, we also define the matrices $B_\sigma^{\zeta i} = (B_{\sigma,q,q'}^{\zeta i})_{q,q' \in Q^{(k)}}$ where $B_{\sigma,q,q'}^{\zeta i}$ is the sum of weights of the runs from configuration $(u, \sigma, q, i, \varepsilon)$ to $(u, \sigma, q', i, \varepsilon)$ with intermediary configurations of the form (u, σ, r, j, π) with $\pi \neq \varepsilon$ or $j \leq i$ (See right of Figure 5). Again, these runs stay on the left of their starting position except when they drop pebbles.

By definition we have $B_\sigma^{\to 0} = B_\sigma^{\zeta 0}$. Moreover, for $0 < i \leq |u|$, a run from position 0 to position i can be decomposed as a run from position 0 to position $i-1$ followed by a right move, followed by a loop over position i:

$$B_\sigma^{\to i} = B_\sigma^{\to i-1} \times M_{\sigma,i-1}^{\to,(k)} \times B_\sigma^{\zeta i}\,.$$

Here and in the following, we will denote by $M_{\sigma,i}^{d,(k)}$ the $Q^{(k)} \times Q^{(k)}$-matrix with (q, q')-coefficient given by $\llbracket M_{q,q'}^{d} \rrbracket(u, \sigma, i, i)$ for $d \in \{\to, \leftarrow\}$: this coefficient denotes the weight of taking a transition with move d from state q to state q' on position i with current valuation σ. We will also need similar matrices for drop and lift moves. We denote by $M_{\sigma,i}^{\downarrow_x,(k)}$ the $Q^{(k)} \times Q^{(k-1)}$-matrix with (q, q')-coefficient given by $\llbracket M_{q,q'}^{\downarrow_x} \rrbracket(u, \sigma, i, i)$. Without loss of generality, we assume that lift moves only occur on position $|u|$ of the word u, hence we do not need to give position and valuation (as no pebble can be dropped on this position) to define similar matrix for lift: let $M^{\uparrow,(k)}$ be the $Q^{(k-1)} \times Q^{(k)}$-matrix with (q, q')-coefficient given by $\llbracket M_{q,q'}^{\uparrow} \rrbracket(u, \sigma, |u|, |u|)$ for any valuation σ.

We now explain how to compute the matrices $B_\sigma^{\zeta i}$ inductively from left to right. At the bottom level ($k = 0$) no pebble can be dropped and we get

$$\begin{aligned} B_\sigma^{\zeta 0} &= \mathrm{Id} \\ B_\sigma^{\zeta i} &= \left(M_{\sigma,i}^{\leftarrow,(k)} \times B_\sigma^{\zeta i-1} \times M_{\sigma,i-1}^{\rightarrow,(k)}\right)^* \qquad \text{if } 0 < i \leq |u|\,. \end{aligned}$$

If $k > 0$, the run may immediately drop some pebble $x \in \mathrm{Peb}$ on position $0 \leq i < |u|$ resulting in the *nested* computation of

$$N_{\sigma,i}^{(k)} = \sum_{x \in \mathrm{Peb}} M_{\sigma,i}^{\downarrow_x,(k)} \times B_{\sigma[x \mapsto i]}^{(k-1)} \times M^{\uparrow,(k)}\,.$$

If $i > 0$, the run may also start by moving left, hence we obtain

$$\begin{aligned} B_\sigma^{\zeta 0} &= \left(N_{\sigma,0}^{(k)}\right)^* \\ B_\sigma^{\zeta i} &= \left(N_{\sigma,i}^{(k)} + M_{\sigma,i}^{\leftarrow,(k)} \times B_\sigma^{\zeta i-1} \times M_{\sigma,i-1}^{\rightarrow,(k)}\right)^* \qquad \text{if } 0 < i < |u| \\ B_\sigma^{\zeta |u|} &= \left(M_{\sigma,|u|}^{\leftarrow,(k)} \times B_\sigma^{\zeta |u|-1} \times M_{\sigma,|u|-1}^{\rightarrow,(k)}\right)^* \end{aligned}$$

using the hypotheses ensuring that no pebble is dropped on the last position.

Finally, once all these matrices obtained, we can compute the behavior of layer k with the formula

$$B_\sigma^{(k)} = B_\sigma^{\rightarrow |u|}\,.$$

To conclude this proof, it remains to count the number of scalar operations in the whole computation. For this, we first count the number of matrix operations (sum, product and star) and then infer the number of scalar operations assuming standard algorithms on matrices (quadratic for sum, cubic for product and cubic for star using Lemma 13).

Fix a layer k. For all valuations σ (there are $|u|^p$ such valuations), matrices $B_\sigma^{\rightarrow i}$, $B_\sigma^{\zeta i}$ must be computed for every $0 \leq i \leq |u|$. When $k = 0$, the total number of matrix operations (sum, product, star) is $\mathcal{O}(|u|^p \times |u|)$, which corresponds to $\mathcal{O}(n_0^3 \times |u|^{p+1})$ scalar operations. For $k > 0$, the computation of $N_{\sigma,i}^{(k)}$ takes $\mathcal{O}(p(n_k n_{k-1}^2 + n_k^2 n_{k-1}))$ scalar sum and products for each σ and i. Hence, the total number of scalar operations for computing all matrices $B_\sigma^{\rightarrow i}$, $B_\sigma^{\zeta i}$ of layer k is now $\mathcal{O}(p(n_k n_{k-1}^2 + n_k^2 n_{k-1} + n_k^3)|u|^{p+1})$.

Summing over all $k \leq K$, we get a total number of $\mathcal{O}((p+1)n^3|u|^{p+1})$ scalar operations since $n_0^3 + \sum_{k=1}^K n_k n_{k-1}^2 + n_k^2 n_{k-1} + n_k^3 \leq n^3$. □

It is important to notice that the complexity with respect to the pebWA does not depend on the number K of layers but only on the total number of states. The number of pebbles occurs in the exponent but since we allow reusable pebbles, this number may be rather small. This is in the same vein as restricting the number of variable names, e.g., in first-order logic, without restricting the quantifier

depth. Restricting the number of variable names often results in much lower complexity. For instance, the complexity of the evaluation (model-checking) problem of first-order logic over relational structures drops from PSPACE to PTIME when the number of variable names is bounded [33,34].

We have seen in Section 2.2 that weighted LTL formulas can be described with pebWE using two pebbles x and y. Actually, the same constructions are valid if we reuse pebble x instead of y. For instance, *until* may be described with

$$E_{\varphi \mathsf{U} \psi}(x) = \triangleright ? \rightarrow^* x ? \big((x! (E_{\neg \psi}(x) \leftarrow^* E_\varphi(x))) \rightarrow \big)^* (x! E_\psi(x)) \rightarrow^* \triangleleft ? \,.$$

Therefore, any weighted LTL formula φ may be described with a pebWE E_φ using a single pebble x. The pebble-depth of E_φ being the nesting depth of modalities in φ. Using Theorem 12 we obtain a layered pebWA $\mathcal{A}_\varphi$ equivalent to E_φ. The number K of layers in $\mathcal{A}_\varphi$ is the pebble-depth of E_φ, i.e., the nesting depth of φ. Moreover, $\mathcal{A}_\varphi$ uses only one pebble. The number of states of $\mathcal{A}_\varphi$ is linear in the size of φ. Hence, Theorem 14 yields an evaluation algorithms using $\mathcal{O}(|\varphi|^3 |u|^2)$ scalar operations. We see below that there is an algorithm which is also linear in $|u|$.

We say that a K-layered pebWA $\mathcal{A} = (Q, A, I, M, T)$ is *strongly* K-layered if in each layer only a fixed pebble may be dropped: for all $i \leq K$, there is a pebble $x_i \in \mathrm{Peb}$ such that for all $q, q' \in Q$ and $x \in \mathrm{Peb}$, if $\ell(q) = i$ and $x \neq x_i$ then $M^{\downarrow_x}_{q,q'} = 0$.

Theorem 15. *Given a* strongly K*-layered* pebWA *with* $p > 0$ *reusable pebbles and a word* $u \in A^+$*, we can compute with* $\mathcal{O}(|Q|^3 |u|^p)$ *scalar operations (sum, product, star) the values* $[\![\mathcal{A}_{q,q'}]\!](u, \sigma)$ *for all layers* $k \leq K$*, states* $q, q' \in Q^{(k)}$ *and valuations* $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$*.*

Proof. The idea is to decrease by 1 the exponent in the complexity by reducing the total number of valuations σ for which we must compute the matrices used in the proof of Theorem 14. Indeed, knowing the unique pebble that may be dropped in a given layer permits to forget the precise position of this pebble when computing the matrices of this layer.

Let $0 \leq k \leq K$ be a layer of $\mathcal{A}$. In addition to matrices defined in the proof of Theorem 14, we introduce new matrices $B^{i\curvearrowright}_\sigma$ and $B^{i\rightarrow}_\sigma$ in order to compute the sum of weights of runs which stay on the right of their starting position (except when they drop pebbles). For every states $q, q' \in Q^{(k)}$ and $0 \leq i \leq |u|$, we denote by $B^{i\curvearrowright}_{\sigma,q,q'}$ the sum of weights of the runs from configuration $(u, \sigma, q, i, \varepsilon)$ to $(u, \sigma, q', i, \varepsilon)$ with intermediary configurations of the form (u, σ, r, j, π) with $\pi \neq \varepsilon$ or $j \geq i$. Finally, we denote by $B^{i\rightarrow}_{\sigma,q,q'}$ the sum of weights of the runs from configuration $(u, \sigma, q, i, \varepsilon)$ to $(u, \sigma, q', |u|, \varepsilon)$ with intermediary configurations of the form (u, σ, r, j, π) with $\pi \neq \varepsilon$ or $j \geq i$.

Apparently, it seems that we will compute twice as many matrices, however we will gain in complexity by replacing the usual valuation σ by its restriction σ' to the pebbles in $\mathrm{Peb} \backslash \{x_k\}$. Indeed, with $j = \sigma(x_k)$, we can compute the matrix $B^{(k)}_\sigma$ by splitting the word u into three parts: positions appearing before j,

position j and positions appearing after j. Hence, for $0 < j = \sigma(x_k) < |u|$, we can compute $B^{(k)}_\sigma$ with (notice that $\sigma'[x_k \mapsto j] = \sigma$)

$$B^{(k)}_\sigma = B^{\to j-1}_{\sigma'} \times M^{\to,(k)}_{\sigma',j-1} \times \Big(\quad M^{\downarrow_{x_k},(k)}_{\sigma,j} \times B^{(k-1)}_\sigma \times M^{\uparrow,(k)} \\ + M^{\leftarrow,(k)}_{\sigma,j} \times B^{\circlearrowleft j-1}_{\sigma'} \times M^{\to,(k)}_{\sigma',j-1} \\ + M^{\to,(k)}_{\sigma,j} \times B^{j+1\circlearrowright}_{\sigma'} \times M^{\leftarrow,(k)}_{\sigma',j+1} \Big)^* \times M^{\to,(k)}_{\sigma,j} \times B^{j+1\to}_{\sigma'} .$$

This formula can easily be adapted when $j = 0$. It remains to compute the matrices $B^{\to i}_{\sigma'}$ and $B^{\circlearrowleft i}_{\sigma'}$ for $0 \le i < |u|$, and the matrices $B^{i\circlearrowright}_{\sigma'}$ and $B^{i\to}_{\sigma'}$ for $0 < i \le |u|$. First, if $k > 0$ then pebble $x_k \in \mathrm{Peb}$ may be dropped on position $0 \le i < |u|$ (with $i \neq \sigma(x_k)$) resulting in the *nested* computation of

$$N^{(k)}_{\sigma',i} = M^{\downarrow_{x_k},(k)}_{\sigma',i} \times B^{(k-1)}_{\sigma'[x_k \mapsto i]} \times M^{\uparrow,(k)} .$$

We let $N^{(0)}_{\sigma',i} = 0$. Then, it is easy to verify that for $0 < i < |u|$:

$$B^{\circlearrowleft 0}_{\sigma'} = \Big(N^{(k)}_{\sigma',0}\Big)^* \qquad \text{and} \qquad B^{\to 0}_{\sigma'} = B^{\circlearrowleft 0}_{\sigma'}$$
$$B^{\circlearrowleft i}_{\sigma'} = \Big(N^{(k)}_{\sigma',i} + M^{\leftarrow,(k)}_{\sigma',i} \times B^{\circlearrowleft i-1}_{\sigma'} \times M^{\to,(k)}_{\sigma',i-1}\Big)^*$$
$$B^{\to i}_{\sigma'} = B^{\to i-1}_{\sigma'} \times M^{\to,(k)}_{\sigma',i-1} \times B^{\circlearrowleft i}_{\sigma'}$$
$$B^{|u|\circlearrowright}_{\sigma'} = \mathrm{Id} \qquad \text{and} \qquad B^{|u|\to}_{\sigma'} = \mathrm{Id}$$
$$B^{i\circlearrowright}_{\sigma'} = \Big(N^{(k)}_{\sigma',i} + M^{\to,(k)}_{\sigma',i} \times B^{i+1\circlearrowright}_{\sigma'} \times M^{\leftarrow,(k)}_{\sigma',i+1}\Big)^*$$
$$B^{i\to}_{\sigma'} = B^{i\circlearrowright}_{\sigma'} \times M^{\to,(k)}_{\sigma',i} \times B^{i+1\to}_{\sigma'} .$$

The computation of the four types of matrices, for all valid positions i and all partial valuations σ', requires globally $\mathcal{O}(|u|^{p-1} \times |u|)$ matrix operations. Hence, this improves the overall complexity as announced. □

Notice that if $p \le 1$ then any K-layered pebWA is *strongly* K-layered. In this case, we get an evaluation algorithm using $\mathcal{O}(|Q|^3|u|)$ scalar operations. This is in particular the case for pebWA arising from weighted LTL formulas.

9 Discussion

To conclude, let us briefly mention some interesting topics that could be studied in the future. As already stated in Section 7, one should try to obtain a quadratic algorithm for the translation of pebWE to pebWA. Next, as in Section 8 for the evaluation problem, one should develop efficient algorithms for quantitative model-checking, emptiness, containment, etc.

We have no restriction over the syntax of expressions or automata. In particular, 2-way moves may give rise to unbounded loops which is why we considered

continuous semirings. We believe that continuous semirings are suitable for most applications. But in case one needs to work without this hypothesis, it is possible to put restrictions on the syntax of expressions and automata in order to rule out unbounded loops and have a well-defined semantics in arbitrary semirings. For instance, one may restrict iterations to forward proper or backward proper expressions.

The correctness of our translations between pebWE and pebWA relies on the partial monoid structure of marked words, which does not use concatenation of words. We can also endow marked trees with such a partial monoid structure. Therefore, pebWE can be extended to trees with a semantics in the continuous semiring of series over marked trees. We obtain in this way a weighted extension of caterpillar expressions or Regular XPath. Similarly, one may define *tree-walking* pebWA. We believe that the translations presented in this paper also apply to pebWE over trees and tree-walking pebWA.

A more prospective problem is to replace the $x!-$ construction of pebWE with a *chop* product $E\,;F$ which evaluates E on the current prefix and F on the current suffix. We can easily simulate this relativization mechanism using a pebble to mark the current position. The converse is an interesting problem which needs to be investigated: is it possible to simulate pebbles with chop products?

Acknowledgements The authors would like to thank Benedikt Bollig and Jacques Sakarovitch for helpful discussions.

References

1. C. Allauzen and M. Mohri. A unified construction of the Glushkov, Follow, and Antimirov automata. In *Proceedings of MFCS'06*, volume 4162 of *LNCS*, pages 110–121. Springer, 2006.
2. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.
3. J. Berstel and Ch. Reutenauer. *Noncommutative rational series with applications*, volume 137 of *Encyclopedia of Mathematics & Its Applications*. Cambridge, 2011.
4. J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Theory of Computing Systems*, 26:237–269, 1993.
5. M. Bojańczyk. Tree-walking automata. In *Proceedings of LATA'08*, volume 5196 of *LNCS*, pages 1–2. Springer, 2008.
6. M. Bojańczyk, M. Samuelides, T. Schwentick, and L. Segoufin. Expressive power of pebble automata. In *Proceedings of ICALP'06*, volume 4051 of *LNCS*, pages 157–168. Springer, 2006.
7. B. Bollig and P. Gastin. Weighted versus probabilistic logics. In *Proceedings of DLT'09*, volume 5583 of *LNCS*, pages 18–38. Springer, 2009.
8. B. Bollig, P. Gastin, B. Monmege, and M. Zeitoun. Pebble weighted automata and transitive closure logics. In *Proceedings of ICALP'10*, volume 6199 of *LNCS*, pages 587–598. Springer, 2010.
9. A. Brüggeman-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:197–213, 1993.

10. A. Brüggeman-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2(1):81–106, 2000.
11. J. A. Brzozowski and E. J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. on Electronic Computers*, 12(9):67–76, 1963.
12. P. Buchholz and P. Kemper. Model checking for a class of weighted automata. *Discrete Event Dynamic Systems*, 20(1):103–137, Jan. 2009.
13. F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 333–355. Springer, 2004.
14. J. Conway. *Regular Algebra and Finite Machines*. Chapman & Hall, 1971.
15. M. Droste and W. Kuich. Semirings and formal power series. In *Handbook of Weighted Automata* [16], chapter 1, pages 3–27.
16. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009.
17. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In *Jewels are forever*, pages 72–83. Springer, 1999.
18. Z. Ésik and W. Kuich. *Modern Automata Theory*. 2007. Electronic book, http://dmg.tuwien.ac.at/kuich.
19. P. Gastin and B. Monmege. Adding pebbles to weighted automata. In *Proceedings of CIAA'12*, volume 7381 of *LNCS*, pages 28–51. Springer, 2012.
20. N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169:161–184, 1996.
21. V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.
22. K. Knight and J. May. Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata* [16], chapter 14, pages 555–579.
23. D. Kuske. Schützenberger's theorem on formal power series follows from kleene's theorem. *Theoretical Computer Science*, 401(1-3):243–248, 2008.
24. E. Mandrali. Weighted LTL with discounting. In *Proceedings of CIAA'12*, volume 7381 of *LNCS*, pages 353–360. Springer, 2012.
25. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. on Electronic Computers*, 9(1):39–47, 1960.
26. I. Meinecke. A weighted μ-calculus on words. In *Proceedings of DLT'09*, volume 5583 of *LNCS*, pages 384–395. Springer, 2009.
27. B. Ravikumar. On some variations of two-way probabilistic finite automata models. *Theoretical Computer Science*, 376(1-2):127–136, 2007.
28. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
29. J. Sakarovitch. Rational and recognisable power series. In *Handbook of Weighted Automata* [16], chapter 4, pages 103–172.
30. J. Sakarovitch. Automata and expressions. In *AutoMathA Handbook*. 2012. To appear.
31. M. Samuelides and L. Segoufin. Complexity of pebble tree-walking automata. In *Proceedings of FCT'07*, volume 4639 of *LNCS*, pages 458–469. Springer, 2007.
32. M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
33. M. Vardi. The complexity of relational query languages. In *Proceedings of STOC'82*, pages 137–146. ACM Press, 1982.
34. M. Vardi. On the complexity of bounded-variable queries. In *Proceedings of PODS'95*, pages 266–276. ACM Press, 1995.