Adding Pebbles to Weighted Automata*

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, establish 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 quantitative 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 [31] (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 [27,29] for surveys about these methods, and [22] 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 [26] 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., [19]) 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 [17,10,6,30,5].

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 wrt. the size of the resulting automaton and wrt. the space and time complexities of the translation. We show that, Glushkov's [20] 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 focuses on 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. Most proofs are omitted and will appear in a longer version.

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 [21]), 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 $(u_i)_{1 \leq i \leq m} \in D^m$ can be given by the following weighted regular expression in a 1-gram model and a 3-gram model:

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

where symbols $\to$ 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
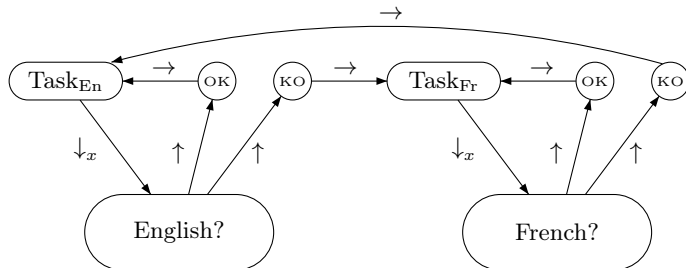
Fig. 1. Pebble automaton for the multi-language modeling task.

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:

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

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.

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 [23], whereas weighted *branching* temporal logics have received more attention [13,12,25,7].

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. For this motivating example, we only consider finite words and the probabilistic setting. Temporal logics implicitly 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.

Consider an LTL formula $\mathsf{F}\varphi$, for *Finally* φ. Given a word u and a position i in the word, we want to compute the probability $\mathbb{P}(\mathsf{F}\varphi, u, i)$ that φ holds on u at position i. For instance, with $\varphi = \frac{1}{3}a$, we should compute $\mathbb{P}(\mathsf{F}\varphi, abba, 0) = \frac{1}{3} + \frac{2}{3}(0 + \frac{2}{3}(0 + \frac{2}{3}(\frac{1}{3} + 0)))$: either φ is satisfied immediately with probability $\frac{1}{3}$ or it is not (probability $\frac{2}{3}$) and (product) it should be satisfied later. More generally, we have

$$\mathbb{P}(\mathsf{F}\varphi, u, i) = \mathbb{P}(\varphi, u, i) + \mathbb{P}(\neg\varphi, u, i)\Big(\mathbb{P}(\varphi, u, i+1)$$
$$+ \mathbb{P}(\neg\varphi, u, i+1)\big(\mathbb{P}(\varphi, u, i+2) + \cdots\big)\Big).$$

For every LTL formula φ, we want to give an equivalent expression $E_\varphi(x)$ which evaluates to $\mathbb{P}(\varphi, u, i)$ over word u when pebble x marks position i. For *Finally* φ, we set

$$E_{\mathsf{F}\varphi}(x) = \triangleright? \rightarrow^* x? \big((y! E_{\neg\varphi}(y)) \rightarrow\big)^* (y! E_\varphi(y)) \rightarrow^* \triangleleft?$$
$$= \triangleright? \rightarrow^* x? \sum_{n \geq 0} \big((y! E_{\neg\varphi}(y)) \rightarrow\big)^n (y! E_\varphi(y)) \rightarrow^* \triangleleft?.$$

The expression starts at the beginning of the word ($\triangleright?$), moves right ($\rightarrow^*$) until it sees the marked position ($x?$), for each possible $n \geq 0$ it iterates n times the computation of $\neg\varphi$ with the current position marked with y ($y! E_{\neg\varphi}(y)$) and moving right between two computations, and it finally computes φ with $y! E_\varphi(y)$ before moving to the end of the word ($\rightarrow^* \triangleleft?$).

Similarly, for *Globally* φ ($\mathsf{G}\varphi$), we have $\mathbb{P}(\mathsf{G}\varphi, u, i) = \prod_{j \geq i} \mathbb{P}(\varphi, u, j)$, leading to the simpler expression

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

Finally, based on the equivalence $\varphi \, \mathsf{U} \, \psi \equiv (\neg\psi \wedge \varphi) \, \mathsf{U} \, \psi$, the expression for the *Until* modality is

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

In terms of automata, let us assume that for every formula φ, there is an automaton $\mathcal{A}_\varphi$ with 2 designated terminal states $\{\mathrm{OK}, \mathrm{KO}\}$, such that runs ending in OK computes expression E_φ and those ending in KO computes expression $E_{\neg\varphi}$. We have depicted below automata for the modalities *Finally* and *Globally*.
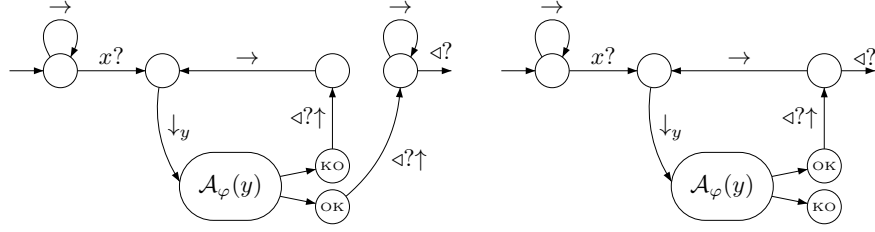
3 Preliminaries

Words. The set of non-empty words over a finite alphabet A is denoted A^+. We write $u = u_0 \cdots u_{n-1} \in A^+$ a non-empty 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 [15,27] 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, $\sum_{j \in J} \left(\sum_{i \in I_j} s_i \right) = \sum_{i \in I} s_i$;
- $\left(\sum_{i \in I} s_i \right) \times \left(\sum_{j \in J} t_j \right) = \sum_{(i,j) \in I \times J} (s_i \times t_j)$.

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 a monoid and the semiring 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 $s \in \mathbb{S}$, $a \in A$, $x \in$ 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: $E^* \overset{\text{def}}{=} 1 + E^+$. 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 $u = u_0u_1\cdots u_{|u|-1} \in A^+$. 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 $a \overset{\text{def}}{=} a?\rightarrow$ and $\overline{a} \overset{\text{def}}{=} a?\leftarrow$. Then, we can write $\rightarrow^*\texttt{blue}\leftarrow^+\triangleright?\rightarrow^*\texttt{black}\rightarrow^*$ to define words having both $\texttt{blue}$ and $\texttt{black}$ 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 $(\mathbb{R}_{\geq 0}^\infty, +, \times, 0, 1)$, 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 $i, j \in \text{pos}(u)$ 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.

---

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

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

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

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

$$\llbracket E + F \rrbracket(u, \sigma, i, j) = \llbracket E \rrbracket(u, \sigma, i, j) + \llbracket F \rrbracket(u, \sigma, i, j)$$

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

$$\llbracket E^+ \rrbracket(u, \sigma, i, j) = \sum_{n > 0} \llbracket E^n \rrbracket(u, \sigma, i, j)$$

Table 1. Semantics of weighted expressions

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 $\llbracket E \rrbracket(cabcdbadcbab) = 4$.

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, $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$ 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).

---

[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 $2^{\mathcal{O}(|u|)}$.

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:

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

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, 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*

---

[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.

[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 $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

units satisfying:

$$\forall x, z \in Z \quad \forall y \in Y \qquad x \cdot y = z \implies x = z$$
$$\forall x, z \in Z \quad \forall y \in Y \qquad y \cdot x = z \implies x = z$$
$$\forall z \in Z \quad \exists! y \in Y \qquad y \cdot z = z$$
$$\forall z \in Z \quad \exists! y \in Y \qquad z \cdot y = z\,.$$

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

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

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 developed e.g. in [28]. Instead, we will use the theory of rational series over a *continuous* semiring $\mathbb{S}$ (see e.g., [27, 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 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 by $(f \times g)(z) = \sum_{x,y \in Z, z = x \cdot y} f(x) \times g(y)$ for $f, g \in \mathbb{S}\langle\!\langle Z \rangle\!\rangle$ and $z \in Z$. 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 must be nonempty since we have the left and right partial units for 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. 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., [27, III.5]). In particular, a star operation may be defined.
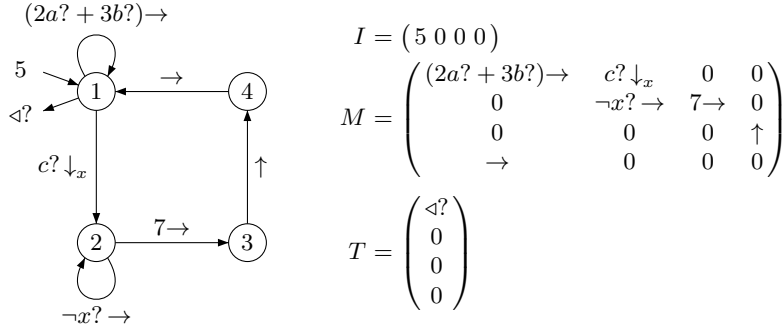
$$I = \begin{pmatrix} 5 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} (2a? + 3b?)\rightarrow & c?\downarrow_x & 0 & 0 \\ 0 & \neg x?\rightarrow & 7\rightarrow & 0 \\ 0 & 0 & 0 & \uparrow \\ \rightarrow & 0 & 0 & 0 \end{pmatrix}$$

$$T = \begin{pmatrix} \triangleleft? \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Fig. 2. A pebWA and its matrix representation.

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

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

5   Weighted Automata with Pebbles

We fix a finite set Peb of pebbles and a (continuous) semiring $\mathbb{S}$. We denote by 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 $5 \times 2^{|u|_a} \times 3^{|u|_b} \times 7^{|u|_c}$. 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
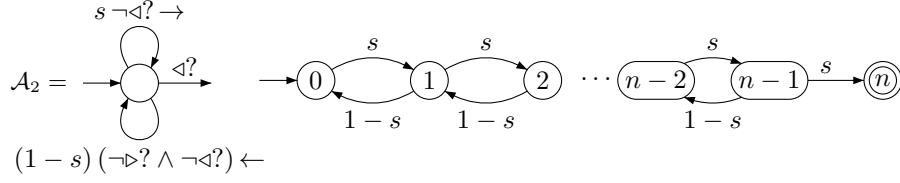
Fig. 3. Markov Chain obtained by synchronizing $\mathcal{A}_2$ with a word of length n

example,

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

Consider also the 2-way automaton $\mathcal{A}_2$ over the semiring $(\mathbb{R}_{\geq 0}^{\infty}, +, \times, 0, 1)$, with $0 < s < 1$. The matrix M of $\mathcal{A}_2$ admits as unique coefficient the polynomial $s \, \neg \vartriangleleft? \rightarrow + (1-s) \, (\neg \vartriangleright? \wedge \neg \vartriangleleft?) \leftarrow$, 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 decribed with a pebWE in Section 4.

As for expressions, we allow macros in M and T: for $a \in A$, we use $a \stackrel{\mathrm{def}}{=} a? \cdot \rightarrow$ and $\bar{a} \stackrel{\mathrm{def}}{=} a? \cdot \leftarrow$, for $d \in \mathrm{Move}$, 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 $d \in \mathrm{Move}$, we denote by $M_{p,q}^d \in \mathbb{S}\langle \mathrm{Test} \rangle$ the coefficient of move d in $M_{p,q}$. For instance, $M_{1,2}^{\downarrow_x} = c?$ in $\mathcal{A}_1$. We collect these coefficients in matrices $M^d = (M_{p,q}^d) \in (\mathbb{S}\langle \mathrm{Test} \rangle)^{Q \times Q}$.

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, $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(u)$ a valuation, $q \in Q$ the current state, $i \in \mathrm{pos}(u)$ the current position, and $\pi \in (\mathrm{Peb} \times \mathrm{pos}(u))^*$ 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 $\mathrm{TS}(\mathcal{A})$ 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

$$\llbracket M_{p,q}^{\rightarrow} \rrbracket(u, \sigma_\pi, i, i) \quad \text{if } j = i+1 \text{ and } \pi' = \pi \tag{S1}$$

$$\llbracket M_{p,q}^{\leftarrow} \rrbracket(u, \sigma_\pi, i, i) \quad \text{if } j = i-1 \text{ and } \pi' = \pi \tag{S2}$$

$$\llbracket M_{p,q}^{\downarrow_x} \rrbracket(u, \sigma_\pi, i, i) \quad \text{if } j = 0, \, i < |u| \text{ and } \pi' = \pi(x, i) \tag{S3}$$

$$\llbracket M_{p,q}^{\uparrow} \rrbracket(u, \sigma_\pi, i, i) \quad \text{if } \pi = \pi'(y, j) \text{ for some } y \in \mathrm{Peb} \tag{S4}$$

where $[\![M_{p,q}^d]\!]$ is the semantics of $M_{p,q}^d \in \mathbb{S}\langle\mathrm{Test}\rangle$, 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 $\mathrm{TS}(\mathcal{A})$ consists of those $(u,\sigma,p,i,\pi) \rightsquigarrow (u,\sigma,q,j,\pi')$ with a non-zero weight: hence $\mathrm{TS}(\mathcal{A})$ is a disjoint union of transition systems depending on the pair (u,σ) considered. A run of $\mathcal{A}$ is a path ρ in $\mathrm{TS}(\mathcal{A})$. Its weight is the product of the weights of its transitions from left to right.

Given a marked word $(u,\sigma,i,j) \in \mathrm{Mk}(A^+)$ 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 $\ell\colon Q \to \{0,\ldots,K\}$ 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 $M_{p,q}^{\leftarrow} \neq 0$ or $M_{p,q}^{\rightarrow} \neq 0$ then $\ell(q) = \ell(p)$; if $M_{p,q}^{\uparrow} \neq 0$ then $\ell(q) = \ell(p) + 1$; and for all $x \in \mathrm{Peb}$, if $M_{p,q}^{\downarrow_x} \neq 0$ 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

$$I = \left(\begin{array}{|c|c|c|} \hline I^{(2)} & 0 & 0 \\ \hline \end{array}\right),\ M = \left(\begin{array}{|c|c|c|} \hline N^{(2)} & D^{(2)} & 0 \\ \hline L^{(1)} & N^{(1)} & D^{(1)} \\ \hline 0 & L^{(0)} & N^{(0)} \\ \hline \end{array}\right),\ T = \left(\begin{array}{|c|} \hline T^{(2)} \\ \hline 0 \\ \hline 0 \\ \hline \end{array}\right) \quad (1)$$
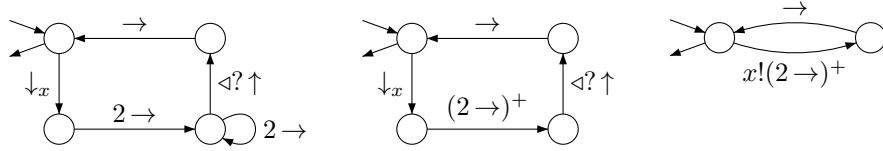
Fig. 4. A pebWA and two equivalent generalized pebWA.

where entries in $N^{(i)}$ are in $\mathbb{S}\langle\text{Test}\rangle\langle\{\leftarrow, \rightarrow\}\rangle$, entries in $L^{(i)}$ are in $\mathbb{S}\langle\text{Test}\rangle\langle\{\uparrow\}\rangle$, and entries in $D^{(i)}$ are in $\mathbb{S}\langle\text{Test}\rangle\langle\{\downarrow_x \mid x \in \text{Peb}\}\rangle$. The entries of $I^{(2)}$ and $T^{(2)}$ are as usual in $\mathbb{S}$ and $\mathbb{S}\langle\text{Test}\rangle$ 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 [24] and the recursive algorithm [14]. We refer to the survey of Sakarovitch [29, 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 $I \in \mathbb{S}^Q$, $T \in \mathbb{S}\langle\text{Test}\rangle^Q$ and $M \in (\text{pebWE} + \mathbb{S}\langle\text{Test}\rangle\langle\text{Move} \setminus \{\leftarrow, \rightarrow\}\rangle)^{Q\times Q}$. Intuitively, the entries $M_{p,q}^{\leftarrow} \cdot \leftarrow + M_{p,q}^{\rightarrow} \cdot \rightarrow$ have been extended to arbitrary pebWE $M_{p,q}^{\text{pebWE}}$. The semantics of pebWA is easily extended to GpebWA. In fact, we only have to replace (S1-S2) by (G1-2) below:

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

15

The definition of K-layered automata can easily be extended to GpebWA. Layered automata are still of the form given in (1), 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 M are all pebWE.

Theorem 2. *Let $\mathcal{A} = (Q, A, I, M, T)$ be a 0-layered GpebWA. We can construct a matrix $\Phi(M) \in \text{pebWE}^{Q \times Q}$ which is equivalent to the automaton: for all $p, q \in Q$, we have $[\![\Phi(M)_{p,q}]\!] = [\![\mathcal{A}_{p,q}]\!]$, i.e., for all $(u, \sigma, i, j) \in \text{Mk}(A^+)$*

$$[\![\Phi(M)_{p,q}]\!](u, \sigma, i, j) = [\![\mathcal{A}_{p,q}]\!](u, \sigma, i, j).$$

Moreover, the entries of $\Phi(M)$ are in the rational closure[5] *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 or McNaughton-Yamada algorithm. We can also apply the state elimination method or the system resolution method starting from any initial state p and final state q.

We quickly justify below the correctness of the construction based on the partial monoid structure of marked words introduced in Section 4.1. In a first reading, one may go directly to the next subsection.

Recall that $\mathbb{K} = \mathbb{S}\langle\!\langle\text{Mk}(A^+)\rangle\!\rangle$ is a continuous semiring by Proposition 1. For each finite set Q, the semiring of matrices $\mathbb{K}^{Q \times Q}$ is also continuous and, given a matrix H in $\mathbb{K}^{Q \times Q}$, the entries of the matrix $H^* = \sum_{n \geq 0} H^n \in \mathbb{K}^{Q \times Q}$ are in the rational closure of the entries of H. Moreover, H^* can be computed inductively: if $H = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ is a block decomposition, with A and D square matrices, then ([14])

$$H^* = \begin{pmatrix} (A + BD^*C)^* & A^*B(D + CA^*B)^* \\ D^*C(A + BD^*C)^* & (D + CA^*B)^* \end{pmatrix}. \tag{2}$$

We apply the above to $H = [\![M]\!] = ([\![M_{p,q}]\!])_{p,q \in Q} \in \mathbb{K}^{Q \times Q}$. As usual, the matrix M^n describes the paths of length n of the automaton $\mathcal{A}$ and the matrix $H^n = [\![M]\!]^n$ gives the semantics restricted to paths of length n. Summing over all paths, we obtain the full semantics:

$$[\![\mathcal{A}_{p,q}]\!] = \sum_{n \geq 0} ([\![M]\!]^n)_{p,q} = ([\![M]\!]^*)_{p,q}$$

Using recursively (2) on the matrix M, we obtain a matrix $\Phi(M)$ whose entries are in the rational closure of the entries of M and such that $[\![\Phi(M)]\!] = [\![M]\!]^*$.

---

[5] The rational closure is the closure under sum $(+)$ concatenation $(\cdot)$ and star $(^*)$.

6.3 Automata to expressions: Generalized pebWA

We will now extend Theorem 2 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.

Proposition 3. *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 Section 5 (1). To simplify the notation, we write $N = N^{(1)}$, $D = D^{(1)}$, $L = L^{(0)}$ and $P = N^{(0)}$ so that

$$
M = \left(\begin{array}{c|c} N & D \\ \hline L & P \end{array} \right) = \left(\begin{array}{c|c} N & 0 \\ \hline 0 & 0 \end{array} \right) + \left(\begin{array}{c|c} 0 & D \\ \hline L & P \end{array} \right).
$$

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 Theorem 2 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

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

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 4. *For all $p, q \in Q^{(1)}$ and all $(u, \sigma, i, j) \in \mathrm{Mk}(A^+)$, we have*

$$
[\![G_{p,q}]\!](u, \sigma, i, j) = \sum_\rho \mathsf{weight}(\rho)
$$

where the sum ranges over all runs ρ on (u, σ) from configuration (p, i, ε) to configuration (q, j, ε) and using only intermediate states in layer 0.

To conclude the proof of Proposition 3, we simply set

$$
M^{(1)} = N + G = N + C(D, P, L)
$$

and we can check using Lemma 4 that $[\![\mathcal{A}_{p,q}]\!] = [\![\mathcal{A}^{(1)}_{p,q}]\!]$ for all $p, q \in Q^{(1)}$.

Proposition 5. *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 $p, q \in Q^{(K)}$.*

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. The correctness follows from Proposition 3. From Theorem 2 and Proposition 5 we deduce:

Theorem 6. *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 tranlsation (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 7. *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 *litteral-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 litteral-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 [29]. 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:

$$\mathcal{A} = \rightarrow \iota \quad J \quad \boxed{N} \quad U \qquad \mathcal{A} = \begin{pmatrix} 1 & \boxed{\quad 0 \quad} \end{pmatrix} \begin{pmatrix} \begin{array}{c|c} 0 & J \\ \hline 0 & N \end{array} \end{pmatrix} \begin{pmatrix} c \\ \hline U \end{pmatrix}$$
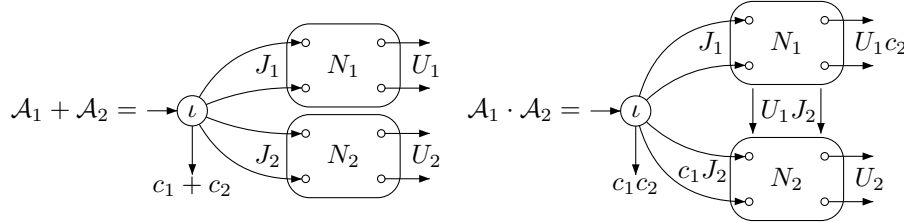
Since terminal weights allow polynomials over Test with the mapping $T\colon Q \to \mathbb{S}\langle \text{Test} \rangle$, we will be able to cope with expressions of the form $E \cdot \varphi?$ and $E \cdot s$ without adding unnecessary states. For $s \in \mathbb{S}$ and $\varphi \in \text{Test}$, we simply write s for $s\mathtt{tt}?$ and φ for 1φ, and also $\to$ for $1\mathtt{tt}?{\to}$ 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 ($\leftarrow, \to$) instead of letters ($a = a?{\to}$) for the atoms. The automata for the atoms are defined as

$$\mathcal{A}(s) = \rightarrow \iota \xrightarrow{\ s\ }$$
$$\mathcal{A}(\to) = \rightarrow \iota \xrightarrow{\ \to\ } \bigcirc \xrightarrow{\ 1\ }$$
$$\mathcal{A}(\varphi) = \rightarrow \iota \xrightarrow{\ \varphi\ }$$
$$\mathcal{A}(\leftarrow) = \rightarrow \iota \xrightarrow{\ \leftarrow\ } \bigcirc \xrightarrow{\ 1\ }$$

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.

$$\mathcal{A}_1 + \mathcal{A}_2 = \rightarrow \iota \quad \begin{array}{c} J_1 \ \boxed{N_1}\ U_1 \\ J_2 \ \boxed{N_2}\ U_2 \end{array} \qquad c_1 + c_2$$

$$\mathcal{A}_1 \cdot \mathcal{A}_2 = \rightarrow \iota \quad \begin{array}{c} J_1 \ \boxed{N_1}\ U_1 c_2 \\ U_1 J_2 \downarrow \\ c_1 J_2 \ \boxed{N_2}\ U_2 \end{array} \qquad c_1 c_2$$
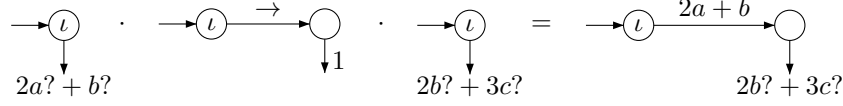
In the concatenation, we are overloading the product notation as follows. The product of two monomials $s_1\varphi_1$ and $s_2\varphi_2$ from $\mathbb{S}\langle \text{Test} \rangle$ should be understood as $(s_1 s_2)(\varphi_1 \wedge \varphi_2)$ to stay in $\mathbb{S}\langle \text{Test} \rangle$. Hence $c_1 c_2$ and the entries of $U_1 c_2$ are in $\mathbb{S}\langle \text{Test} \rangle$. Similarly, in $U_1 J_2$, 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 $c_1 J_2$ and $U_1 J_2$ are in $\mathbb{S}\langle \text{Test} \rangle\langle \text{Move} \rangle$. The matrix representation is therefore:

$$\mathcal{A}_1 \cdot \mathcal{A}_2 = \begin{pmatrix} 1 & \boxed{\ 0\ } & \boxed{\ 0\ } \end{pmatrix} \begin{pmatrix} \begin{array}{c|c|c} 0 & J_1 & c_1 J_2 \\ \hline 0 & N_1 & U_1 J_2 \\ \hline 0 & 0 & N_2 \end{array} \end{pmatrix} \begin{pmatrix} c_1 c_2 \\ \hline U_1 c_2 \\ \hline U_2 \end{pmatrix}$$

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

$$\rightarrow \iota \downarrow 2 \quad \cdot \quad \rightarrow \iota \downarrow a? \quad \cdot \quad \rightarrow \iota \xrightarrow{\ \to\ } \bigcirc \downarrow 1 \quad = \quad \rightarrow \iota \xrightarrow{\ 2a\ } \bigcirc \downarrow 1$$

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

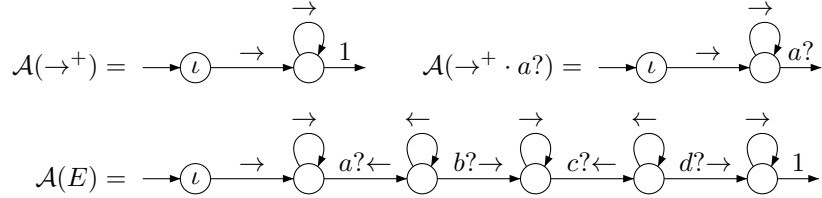$$\begin{array}{ccc} \xrightarrow{} \iota \\ \downarrow \\ 2a? + b? \end{array} \quad \cdot \quad \begin{array}{c} \to \iota \xrightarrow{\;\to\;} \bigcirc \\ \downarrow 1 \end{array} \quad \cdot \quad \begin{array}{c} \to \iota \\ \downarrow \\ 2b? + 3c? \end{array} \quad = \quad \begin{array}{c} \to \iota \xrightarrow{\;2a+b\;} \bigcirc \\ \downarrow \\ 2b? + 3c? \end{array}$$

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

$$\mathcal{A}^* \;=\; \to \iota \xrightarrow{\;c^*J\;} \boxed{N + Uc^*J} \xrightarrow{\;Uc^*\;} \\ \phantom{\mathcal{A}^*} \quad c^* \downarrow$$

$$\mathcal{A}^* \;=\; \begin{pmatrix} 1 & \boxed{0} \end{pmatrix} \begin{pmatrix} 0 & \boxed{c^*J} \\ 0 & \boxed{N + Uc^*J} \end{pmatrix} \begin{pmatrix} c^* \\ \boxed{Uc^*} \end{pmatrix}$$

Notice that $c^* \in \mathbb{S}$ is well-defined since the semiring is complete. As for the concatenation, we can check that the entries of Uc^* are in $\mathbb{S}\langle\mathrm{Test}\rangle$ and the entries of Uc^*J are in $\mathbb{S}\langle\mathrm{Test}\rangle\langle\mathrm{Move}\rangle$. The strict iteration $\mathcal{A}^+$ is computed similarly by simply changing the final weight of ι to c^+ (note that $0^+ = 0$), but keeping the other occurrences of c^* in c^*J, Uc^*J and Uc^*.

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

$$\mathcal{A}({\to}^+) = \to \iota \xrightarrow{\;\to\;} \overset{\to}{\bigcirc} \xrightarrow{\;1\;} \qquad \mathcal{A}({\to}^+ \cdot a?) = \to \iota \xrightarrow{\;\to\;} \overset{\to}{\bigcirc} \xrightarrow{\;a?\;}$$

$$\mathcal{A}(E) = \to \iota \xrightarrow{\;\to\;} \overset{\to}{\bigcirc} \xrightarrow{a? \leftarrow} \overset{\leftarrow}{\bigcirc} \xrightarrow{b? \to} \overset{\to}{\bigcirc} \xrightarrow{c? \leftarrow} \overset{\leftarrow}{\bigcirc} \xrightarrow{d? \to} \overset{\to}{\bigcirc} \xrightarrow{\;1\;}$$

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 $x!\mathcal{A}$:

$$x!\mathcal{A} \;=\; \to \iota \qquad\qquad \tau \to \\ \downarrow_x \qquad\qquad\qquad \nearrow \\ \iota' \xrightarrow{J} \boxed{N} \xrightarrow{U\triangleleft?\uparrow}$$

$$x!\mathcal{A} \;=\; \begin{pmatrix} 1 & 0 & 0 & \boxed{0} \end{pmatrix} \begin{pmatrix} 0 & 0 & \downarrow_x & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J \\ 0 & U\triangleleft?\uparrow & 0 & N \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

The correctness of this construction follows easily from Proposition 3. Assume for simplicity that $\mathcal{A}$ is a 0-layered automaton, then $x!\mathcal{A}$ is a 1-layered automaton and it's layer decomposition is shown both in the graphical and matricial representations above. Let $Q = \{\iota, \tau, \iota'\} \uplus Q'$ be the set of states of $x!\mathcal{A}$ where Q' are the non-initial states of $\mathcal{A}$. Using the notation of Section 6.3, the drop-matrix D of $x!\mathcal{A}$ contains a single non-zero entry which is $D_{\iota,\iota'} = \downarrow_x$, the non-zero entries of the lift-matrix L are in the $Q' \times \tau$ column $U \triangleleft ?\uparrow$, and F is the transition matrix of $\mathcal{A}$. Therefore, the matrix $G = C(D, F, L)$ has a single non-zero entry which is

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

As last example, consider again expression E below used in Section 4:

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

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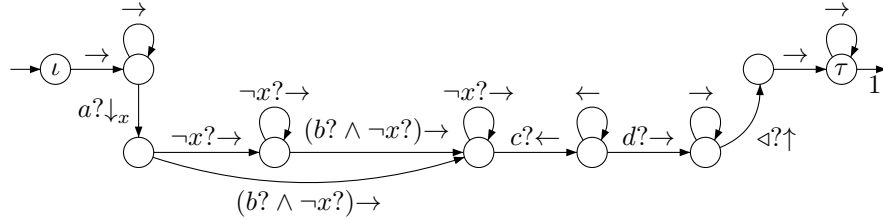
To close this section, we briefly discuss the complexity of our translation. Clearly, the number of states of the automaton $\mathcal{A}(E)$ is the litteral-length $\ell\ell(E)$ of expression E. The time complexity is cubic in the length of E. It should be possible to get a quadratic algorithm by generalizing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

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.

Let $\mathcal{A} = (Q, A, I, M, T)$ be a K-layered pebWA. As in Section 6.3, for $i \le K$, we let $Q^{(i)} = \ell^{-1}(i)$ be the set of states in layer i.

Theorem 8. *Given a K-layered pebWA with p pebbles and a word $w \in A^+$, we can compute with $\mathcal{O}((K+1)|w|^{p+1})$ matrix operations (sum, product, iteration) the values $[\![\mathcal{A}_{p,q}]\!](w, \sigma)$ for all $p, q \in Q^{(K)}$ and valuations $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(w)$.*

It is important to notice that the complexity only linearly depends on the number K of layers. The number of pebbles occurs in the exponent but since we allow reusable pebbles, this number may be much smaller than the number of layers. 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 [32,33].

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

$$E_{\varphi \cup \psi}(x) = \triangleright? \to^* x? \big((x!(E_{\neg \psi}(x) \leftarrow^* E_\varphi(x))) \to \big)^* (x!E_\psi(x)) \to^* \triangleleft? .$$

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 7 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. Theorem 8 yields an evaluation algorithms using $\mathcal{O}((K+1)|w|^2)$ matrix operations. We see below that there is an algorithm which is also linear in $|w|$.

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 $x_i \in \text{Peb}$ such that for all $q, q' \in Q$ and $x \in \text{Peb}$, if $\ell(q) = i$ and $x \neq x_i$ then $M_{q,q'}^{\downarrow_x} = 0$.

Theorem 9. *Given a* strongly *K-layered pebWA with p pebbles and a word $w \in A^+$, we can compute with $\mathcal{O}((K+1)|w|^{\max(1,p)})$ matrix operations (sum, product, iteration) the values $[\![\mathcal{A}_{p,q}]\!](w, \sigma)$ for all states $p, q \in Q^{(K)}$ and valuations $\sigma \colon \text{Peb} \to \text{pos}(w)$.*

Notice that if $p \leq 1$ then any K-layered pebWA is *strongly K-layered*. In this case, we get an evaluation algorithm using $\mathcal{O}((K+1)|w|)$ matrix 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?

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. N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169:161–184, 1996.
20. V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.
21. K. Knight and J. May. Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata* [16], chapter 14, pages 555–579.
22. D. Kuske. Schützenberger's theorem on formal power series follows from kleene's theorem. *Theoretical Computer Science*, 401(1-3):243–248, 2008.
23. E. Mandrali. Weighted LTL with discounting. In *Proceedings of CIAA'12*, LNCS. Springer, 2012. To Appear.
24. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. on Electronic Computers*, 9(1):39–47, 1960.
25. I. Meinecke. A weighted μ-calculus on words. In *Proceedings of DLT'09*, volume 5583 of *LNCS*, pages 384–395. Springer, 2009.
26. B. Ravikumar. On some variations of two-way probabilistic finite automata models. *Theoretical Computer Science*, 376(1-2):127–136, 2007.
27. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
28. J. Sakarovitch. Rational and recognisable power series. In *Handbook of Weighted Automata* [16], chapter 4, pages 103–172.
29. J. Sakarovitch. Automata and expressions. In *AutoMathA Handbook*. 2012. To appear.
30. 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.
31. M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
32. M. Vardi. The complexity of relational query languages. In *Proceedings of STOC'82*, pages 137–146. ACM Press, 1982.
33. M. Vardi. On the complexity of bounded-variable queries. In *Proceedings of PODS'95*, pages 266–276. ACM Press, 1995.