Weighted Specifications over Nested Words

Benedikt Bollig, Paul Gastin, and Benjamin Monmege

March 2013

Research report LSV-13-04

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Weighted Specifications over Nested Words *

Benedikt Bollig[1], Paul Gastin[1], and Benjamin Monmege[1]

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

Abstract. This paper studies several formalisms to specify quantitative properties of finite nested words (or equivalently finite unranked trees). These can be used for XML documents or recursive programs: for instance, counting how often a given entry occurs in an XML document, or computing the memory required for a recursive program execution. Our main interest is to translate these properties, as efficiently as possible, into an automaton, and to use this computational device to decide problems related to the properties (e.g., emptiness, model checking, simulation) or to compute the value of a quantitative specification over a given nested word. The specification formalisms are weighted regular expressions (with forward and backward moves following linear edges or call-return edges), weighted first-order logic, and weighted temporal logics. We introduce weighted automata walking in nested words, possibly dropping/lifting (reusable) pebbles during the traversal. We prove that the evaluation problem for such automata can be done very efficiently if the number of pebble names is small, and we also consider the emptiness problem.

1 Introduction

In this paper, we develop denotational formalisms to express quantitative properties of nested words. Nested words, introduced in [2], are strings equipped with a binary nesting relation. Just like trees, they have been used as a model of XML documents or recursive programs. Though nested words can indeed be encoded in trees (and vice versa), they are often more convenient to work with, e.g., in streaming applications, as they come with a linear order that is naturally given by an XML document [13] or the program execution. Moreover, nested words better reflect system runs of recursive programs where the nesting relation matches a procedure call with its corresponding return. There is indeed a wide range of works that address logics and automata over nested words to process XML documents or to model recursive programs e.g., [13, 1, 16].

Most previous approaches to nested words (or unranked trees) consider Boolean properties: logical formulae are evaluated to either true or false, or to a set of word positions if the formula at hand represents a unary query. Now, given an XML document in terms of a nested word, one can imagine *quantitative* properties that one would like to compute: What is the number of books of a

* This work was partially supported by LIA INFORMEL.

certain author? Are there more fiction than non-fiction books? What is the total number of entries? So, we would like to have flexible and versatile languages allowing us to evaluate arithmetic expressions, possibly guarded by logical conditions written in a standard language (e.g., first-order logic or XPath). To this aim, we introduce (1) weighted regular expressions, which can indeed be seen as a quantitative extension of XPath, (2) weighted first-order logic as already studied in [5] over words, and (3) weighted nested word temporal logic in the flavor of [1]. Their application is not restricted to XML documents, though. For instance, when nested words model recursive function calls, these specification languages can be used to quantify the call-depth of a given system run, i.e., the maximal number of open calls.

Thus, when one considers expressiveness and algorithmic issues, a natural question arises: Is there a robust automaton model to which it is possible to compile specifications written in these languages? Our answer will be positive: we actually obtain a Kleene-Schützenberger correspondence, stating the equivalence of weighted regular expressions with a model of automaton. Not only do we obtain this correspondence, but we also give complexity results concerning the size of the automaton derived from a regular expression, and the time and space used to construct it. We also prove that we can evaluate these automata efficiently and that emptiness problem is decidable (in case the underlying weight structure has no zero divisors). Towards a suitable operational device, we consider navigational automata with pebbles, for two reasons. First, weighted automata, the classical quantitative extensions of finite automata [19], are not expressive enough to encode powerful quantitative expressions, neither for words [7] nor for nested words or trees [17, 8]. Second, we are looking for a model that conforms with common query languages for nested words or trees, such as XPath or equivalent variants of first-order logic, aiming at a quantitative version of the latter and a suitable algorithmic framework. Indeed, tree-walking automata form an appropriate machine model for compiling XPath queries [3].

Contribution. In Section 3, we introduce *weighted expressions with pebbles* over nested words, mixing navigational constructs and rational arithmetic expressions. As an operational counterpart of weighted expressions, we then introduce *weighted automata with pebbles* in Section 4, which can traverse a nested word along nesting edges and direct successors in both directions, and occasionally place reusable pebbles. In a sense, these are extensions of the *tree-walking automata with invisible pebbles*, introduced in [11], to the weighted setting and to nested words. We extend results over words stated in [12], namely a Kleene-Schützenberger theorem showing correspondence between weighted expressions with pebbles and *layered* weighted automata with pebbles (i.e., those that can only use a bounded number of pebbles). We also show how to efficiently compute the value associated to a given nested word in a weighted automaton with pebbles, and prove decidability (not surprinsingly, with non-elementary complexity) of the emptiness problem in case the underlying weight structure has no zero divisor.

In order to allow more flexibility, we also discuss, in Section 5, logical quantitative formalisms like first-order logic and temporal logics, and show how to compile them efficiently into weighted automata with pebbles.

This report is an extended version of [4].

2 Preliminaries

2.1 Nested Words

We fix a finite alphabet A. For $n \in \mathbb{N}$, we let $[n] = \{0, 1, \ldots, n-1\}$. A *nested word* over A is a pair $W = (w, \curvearrowright)$ where $w = a_0 \cdots a_{n-1} \in A^+$ is a nonempty string and $\curvearrowright \subseteq ([n] \times [n]) \cap <$ is a *nesting relation*: for all $(i,j), (i',j') \in \curvearrowright$, we have (1) $i = i'$ iff. $j = j'$, and (2) $i < i'$ implies ($j < i'$ or $j > j'$). We will more often denote $(i,j) \in \curvearrowright$ as $i \curvearrowright j$. Moreover, the inverse of relation $\curvearrowright$ will be denoted as $\curvearrowleft$, so that $i \curvearrowright j$ iff. $j \curvearrowleft i$. For uniformity reasons, we denote as $i \to j$ the fact that j is the successor of i, i.e., $j = i+1$. In case we want to stress that i is the predecessor of j, we rather denote it $j \leftarrow i$. The *length* n of W is denoted $|W|$, and $\mathrm{pos}(W) = [n]$ is the set of *positions* of W. In order to ease some definitions of the paper, a virtual position n can be added to the positions: we will then denote $\overline{\mathrm{pos}}(W) = [n] \cup \{n\}$ the extended set of positions.

Let $\mathcal{T} = \{\mathsf{first}, \mathsf{last}, \mathsf{call}, \mathsf{ret}, \mathsf{int}\}$. Each position $i \in \overline{\mathrm{pos}}(W)$ in a nested word $W = (w, \curvearrowright)$ has a *type* $\tau(i) \subseteq \mathcal{T}$: $\mathsf{first} \in \tau(i)$ iff. $i = 0$; $\mathsf{last} \in \tau(i)$ iff. $i = |W|$; $\mathsf{call} \in \tau(i)$ iff. there exists j such that $i \curvearrowright j$; dually, $\mathsf{ret} \in \tau(i)$ iff. there exists j such that $j \curvearrowright i$; finally, $\mathsf{int} \in \tau(i)$ iff. $i < |W|$ and $\tau(i) \cap \{\mathsf{call}, \mathsf{ret}\} = \emptyset$.

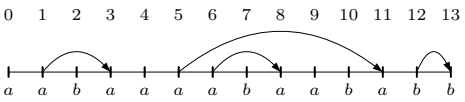


Fig. 1. A nested word

It is convenient to represent the pairs of the relation $\curvearrowright$ pictorially by curved lines which do not cross. Fig. 1 shows a nested word $W = (w, \curvearrowright)$ over $A = \{a, b\}$ of length $|W| = 14$. We have $w = aabaaaabaababb$ and the nesting relation defined by $1 \curvearrowright 3$, $5 \curvearrowright 11$, $6 \curvearrowright 8$ and $12 \curvearrowright 13$. Moreover, $\tau(0) = \{\mathsf{first}, \mathsf{int}\}$, $\tau(13) = \{\mathsf{ret}\}$ and $\tau(14) = \{\mathsf{last}\}$. In the examples of this paper, we will consider the *call-depth* c-d(j) of a position $j \in \overline{\mathrm{pos}}(W)$, i.e., the number of contexts in which position j lies. Formally, c-d$(j) = |\{(i,k) \mid i \curvearrowright k \wedge i < j < k\}|$. For example, position 7 has call-depth 2, whereas position 6 has call-depth 1. The call-depth of a nested word is the maximal call-depth among the positions: the call-depth of W is here 2.

2.2 Weights

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.

A semiring $\mathbb{S}$ is *complete* if every family $(s_i)_{i \in I}$ of elements of $\mathbb{S}$ over an arbitrary index 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 $I = \biguplus_{j \in J} I_j$ 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 extend 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 $s^{i+1} = s^i \times s$). We can easily prove that $s^* = 1 + s\, s^* = 1 + s^*\, s$ for all $s \in \mathbb{S}$.

Here are some examples of complete semirings.

- The Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$ with $\sum$ defined as a possibly 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: $(2^{A^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ with $\sum$ defined as possibly 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 $c \in \mathbb{S}$ for every $a, b \in \mathbb{S}$ 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. See [9] for more discussions about semirings, especially continuous ones. Thereafter, $\mathbb{S}$ will denote a continuous semiring.

3 Weighted Regular Expressions with Pebbles

In this section, we introduce weighted regular expressions with pebbles. Like classical regular expressions, their syntax employs operations $+$ and $\cdot$, as well as

a Kleene star. However, unlike in the Boolean setting, $+$ and $\cdot$ will be interpreted as sum and Cauchy product, respectively.

We introduce first these weighted regular expressions with an example. We consider a nested word over the alphabet $\{a, b\}$. The classical regular expression $(a+b)^*b(a+b)^*$ *checks* that the given nested word contains an occurrence of letter b. We will rather use the shortcut $\rightarrow$ to denote the non-guarded move to the right encoded by the choice $(a+b)$, and use a weighted semantics in the semiring $(\mathbb{N} \cup \{\infty\}, +, \times, 0, 1)$: hence, expression $\rightarrow^* b \rightarrow^*$ *counts* the number of occurrences of letter b in the nested word. We now turn to the more complex task of counting the total number of occurrences of the letter b *inside a context with a call position labelled* a: more formally we want to sum over all possible call positions labelled a, the number of occurrences of b that appear strictly in-between this position and the matching return. For the nested word of Fig. 1, we must count 4 (in particular, position 7 must count for both call positions 5 and 6). In our formalism, we will achieve this task using expression:

$$E = \rightarrow^*(a? \wedge \mathsf{call}?)\, x!\big[\rightarrow^* x? \curvearrowright (\neg x? \leftarrow)^+ b? \rightarrow^*\big] \rightarrow^* .$$

First, we search for a call position labelled with a using expression $\rightarrow^*(a? \wedge \mathsf{call}?)$: tests such as $a?$ ot $\mathsf{call}?$ check the current position without moving. Then, we mark, with $x!-$, the call position of the interesting context with a pebble named x: this permits us to compute independently the subexpression between brackets on the nested word, restarting from the first position. The latter subexpression first searches for the pebble with $\rightarrow^* x?$, follows the call-return edge and then moves backward inside the context with $(\neg x? \leftarrow)^+$ to pick non-deterministically a position carrying letter b.

We turn to the formal syntax of our expressions. We let $\mathrm{Peb} = \{x, y, \ldots\}$ be an infinite set of pebble names. Weighted expressions are built upon simple Boolean tests from a set Test. The syntax of these basic tests is given as follows:

$$\alpha ::= a? \mid \tau? \mid x? \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha$$

where $a \in A$, $\tau \in \mathcal{T}$ and $x \in \mathrm{Peb}$. Thus, a test is a Boolean combination of atomic checks allowing one to verify whether a given position in a nested word has label a; whether it is the first or last position (which is useful since we deal with 2-way expressions); whether it is a call, a return or an internal action; or whether it carries a pebble with name x, respectively. Given a nested word $W = (a_0 \cdots a_{n-1}, \curvearrowright)$, a position $i \in \overline{\mathrm{pos}}(W)$ and an assignment of free pebble names given by a partial mapping $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(W)$, we denote $W, i, \sigma \models \alpha$ if test α is verified over the given model: the semantics of atomic tests is defined in Table 1, and the semantics of Boolean operators is defined as usual. Next, we present weighted regular expressions.

Definition 1. *The set* pebWE *of* weighted expressions with pebbles *from* Peb *is given by the following grammar:*

$$E ::= s \mid \alpha \mid \rightarrow \mid \leftarrow \mid \curvearrowright \mid \curvearrowleft \mid x!E \mid E + E \mid E \cdot E \mid E^+$$

Table 1. Semantics of atomic tests in Test

$$W, i, \sigma \models a? \text{ if } i < |W| \text{ and } a_i = a$$
$$W, i, \sigma \models \tau? \text{ if } \tau \in \tau(i)$$
$$W, i, \sigma \models x? \text{ if } \sigma(x) = i$$

Table 2. Semantics of pebWE

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

$$[\![d]\!](W, \sigma, i, j) = \begin{cases} 1 & \text{if } i \, d \, j \\ 0 & \text{otherwise} \end{cases} \qquad (\text{with } d \in \{\leftarrow, \rightarrow, \curvearrowright, \curvearrowleft\})$$

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

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

$$[\![E + F]\!] = [\![E]\!] + [\![F]\!] \qquad [\![E^+]\!] = \sum_{n>0} [\![E^n]\!]$$

where $s \in \mathbb{S}$, $\alpha \in$ Test, *and* $x \in$ Peb.

We get the classical Kleene star as an abbreviation: $E^* \stackrel{\text{def}}{=} 1 + E^+$. It is also convenient to introduce macros for "check-and-move": $a \stackrel{\text{def}}{=} a? \cdot \rightarrow$. This allows us to use common syntax such as $(ab)^+abc$, or to write $\rightarrow^* abba \leftarrow^+ \mathsf{first}? \rightarrow^* baab \rightarrow^*$ to identify words having both $abba$ and $baab$ as subwords.

A pebWE is interpreted over a nested word W with a marked initial position i and a marked final position j (as is the case in rational expressions or path expressions from XPath) and an assignment of free pebbles (as is the case in logics with free variables), given as a partial mapping $\sigma \colon \text{Peb} \to \text{pos}(W)$. The atomic expressions $\rightarrow$, $\leftarrow$, $\curvearrowright$, $\curvearrowleft$ have their natural interpretation as a binary relation R and are evaluated 1 or 0 depending on whether or not $(i, j) \in R$. On the contrary, formulae s, α, $x!E$ are non-progressing and require $i = j$. In particular, $x!E$ evaluates E in W with the current position marked with pebble x. The formal semantics of pebWE is given in Table 2: notice that the semantics of E^+ is well-defined since the semiring is complete. By default, i and j are the first and the last position of W, i.e., 0 and $|W|$, so that we use $[\![E]\!](W, \sigma)$ as a shortcut for $[\![E]\!](W, \sigma, 0, |W|)$. In the following, we call *pebble-depth* of an expression in pebWE its maximal number of nested $x!E$ operators.

Example 2. Over $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, consider the pebWE

$$E = \big((1 \, \mathsf{call}? \rightarrow \neg \mathsf{ret}?) + (\mathsf{int}? \rightarrow \neg \mathsf{ret}?) + \curvearrowright + (\mathsf{ret}? \rightarrow \neg \mathsf{ret}?)\big)^* x? \rightarrow^* .$$

Notice the use of $1 \in \mathbb{N}$ which is not the unit of the semiring. Moreover, operations $+$ are resolved by the max operator, whereas concatenation implies the use of addition in $\mathbb{N} \cup \{-\infty\}$. For every nested word W, and every position $i \in \mathrm{pos}(W)$, $[\![E]\!](W, [x \mapsto i])$ computes the call-depth of position i: indeed the first Kleene star is unambiguous, meaning that only one path starting from position 0 will lead to x in this iteration; along this path – the shortest one – we only count the number of times we enter inside the context of a call position. Hence, the call-depth of W can be computed with expression $E' = {\to}^*(x!E){\to}^*$.

4 Weighted Automata with Pebbles

We define an automaton that walks in a nested word W. Whether a transition is applicable depends on the current control state and the current position i in W, i.e., its letter and type $\tau(i)$. A transition then either moves to a successor/predecessor position (following the linear order or the nesting relation), or drops/lifts a pebble whose name is taken from Peb. The effect of a transition is described by a move from the set $\mathrm{Move} = \{\to, \leftarrow, \curvearrowright, \curvearrowleft, \uparrow\} \cup \{\downarrow_x \mid x \in \mathrm{Peb}\}$.

Definition 3. *A* pebble weighted automaton *(or shortly* pebWA*) is a tuple* $\mathcal{A} = (Q, A, I, \delta, T)$ *where* Q *is a finite set of* states*,* A *is the input alphabet,* $I \in \mathbb{S}^Q$ *is the* vector of initial weights*,* $T \in \mathbb{S}^Q$ *is the* vector of final weights*, and* $\delta \colon Q \times \mathrm{Test} \times \mathrm{Move} \times Q \to \mathbb{S}$ *is a transition function with finite support*[1]*.*

Informally, I assigns to any state $q \in Q$ the weight I_q of entering a run in q. Similarly, T determines the exit weight T_q at q. Finally, $\delta(p, \alpha, m, q)$, determines the weight of going from state p to state q depending on the move $m \in \mathrm{Move}$ and on the outcome of a test $\alpha \in \mathrm{Test}$. The set $\mathrm{Peb}(\mathcal{A})$ of *pebble names of* $\mathcal{A}$ is defined to be the set of pebble names that appear either in drop transitions $\downarrow_x$ or in tests $x?$ of $\mathcal{A}$.

Let us turn to the formal semantics of a pebWA $\mathcal{A} = (Q, A, I, \delta, T)$. A run is described as a sequence of *configurations* of $\mathcal{A}$. A configuration is a tuple (W, σ, q, π, i). Here, W is the nested word at hand, $\sigma \colon \mathrm{Peb} \to \mathrm{pos}(W)$ is a valuation, $q \in Q$ indicates the current state, $i \in \overline{\mathrm{pos}}(W)$ the current position, and $\pi \in (\mathrm{Peb} \times \mathrm{pos}(W))^*$. The valuation σ indicates the position of *free pebbles*, which may be tested using $x?$ even before being dropped with $\downarrow_x$. It can be omitted when there are no free pebbles. The string π may be interpreted as the contents of a stack (its top being the rightmost symbol of π) that keeps track of the positions where pebbles have been dropped, and in which order. Pebbles are reusable (or invisible as introduced in [11]): this means that we may use several pebbles having a same name from Peb. If several pebbles with name x have been dropped, only the last dropped is *visible* in the configuration. However, when the latter will be lifted, the previous occurrence of pebble x will become visible again. Formally, this means that a pebble name can occur at several places in π, but only its topmost occurrence is visible. Having this in mind, we define,

[1] The support of δ is the set of tuples (q, α, m, q') such that $\delta(q, \alpha, m, q') \neq 0$.

given σ and π, a new valuation $\sigma_\pi\colon \mathrm{Peb} \to \mathrm{pos}(W)$ by $\sigma_\varepsilon = \sigma$ and $\sigma_{\pi(x,i)}(x) = i$, $\sigma_{\pi(x,i)}(y) = \sigma_\pi(y)$ if $y \neq x$.

Any two configurations with fixed W and σ give rise to a *concrete transition* $(W, \sigma, p, \pi, i) \rightsquigarrow (W, \sigma, q, \pi', j)$. Its *weight* is defined by

$$\begin{cases} \displaystyle\sum_{\substack{\alpha \in \mathrm{Test}, d \in \{\rightarrow, \leftarrow, \curvearrowright, \curvearrowleft\} \\ W, \sigma_\pi, i \models \alpha \wedge i\, d\, j}} \delta(p, \alpha, d, q) & \text{if } \pi' = \pi \\ \displaystyle\sum_{\alpha \in \mathrm{Test} \mid W, \sigma_\pi, i \models \alpha} \delta(p, \alpha, \downarrow_x, q) & \text{if } j = 0,\ i < |W| \text{ and } \pi' = \pi(x, i) \\ \displaystyle\sum_{\alpha \in \mathrm{Test} \mid W, \sigma_\pi, i \models \alpha} \delta(p, \alpha, \uparrow, q) & \text{if } \pi = \pi'(y, j) \text{ for some } y \in \mathrm{Peb} \end{cases}$$

and 0, otherwise. In particular, this implies that a pebble cannot be dropped on position $|W|$ in agreement with the convention adopted for weighted expressions.

A *run* of $\mathcal{A}$ is a sequence of consecutive transitions. Its weight is the product of transition weights, multiplied from left to right. We are interested in runs that start at some position i, in state p, and end in some configuration with position j and state q. So, let $[\![\mathcal{A}_{p,q}]\!](W, \sigma, i, j)$ be defined as the sum of the weights of all runs from $(W, \sigma, p, \varepsilon, i)$ to $(W, \sigma, q, \varepsilon, j)$. Since the semiring is continuous, $[\![\mathcal{A}_{p,q}]\!](W, \sigma, i, j)$ is well defined.

The semantics of $\mathcal{A}$ wrt. the nested word W and the initial assignment σ includes the initial and terminal weights, and we let

$$[\![\mathcal{A}]\!](W, \sigma) = \sum_{p,q \in Q} I_p \times [\![\mathcal{A}_{p,q}]\!](W, \sigma, 0, |W|) \times T_q\,.$$

In order to evaluate automata, or prove some expressiveness results, we consider the natural subclass of pebWA that cannot drop an unbounded number of pebbles. We will hence identify *K-layered* automata, for $K \geq 0$, where a state contains information about the number $n \in \{0, \ldots, K\}$ of currently available pebbles. Formally, a pebWA $\mathcal{A} = (Q, A, I, M, T)$ is *K-layered* if there is a mapping $\ell\colon Q \to \{0, \ldots, K\}$ satisfying, for all $p, q \in Q$,

- if $I_q \neq 0$ or $T_q \neq 0$ then $\ell(q) = K$;
- if there is $\alpha \in \mathrm{Test}$ and $d \in \{\leftarrow, \rightarrow, \curvearrowright, \curvearrowleft\}$ such that $\delta(p, \alpha, d, q) \neq 0$ then $\ell(q) = \ell(p)$;
- if there is $\alpha \in \mathrm{Test}$ such that $\delta(p, \alpha, \uparrow, q) \neq 0$ then $\ell(q) = \ell(p) + 1$;
- if there is $\alpha \in \mathrm{Test}$ and $x \in \mathrm{Peb}$ such that $\delta(p, \alpha, \downarrow_x, q) \neq 0$ then $\ell(q) = \ell(p) - 1$.

Fig. 2 schematizes a 2-layered pebWA.

Example 4. We depict in Fig. 3 a pebWA which computes the call-depth of a nested word: it has the same semantics as expression E' of Example 2. Notice that this automaton is 1-layered.

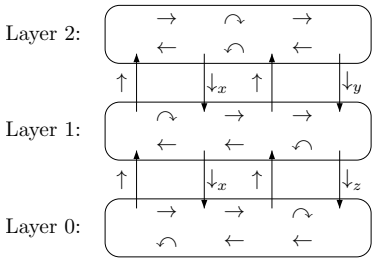


Fig. 2. A 2-layered pebWA

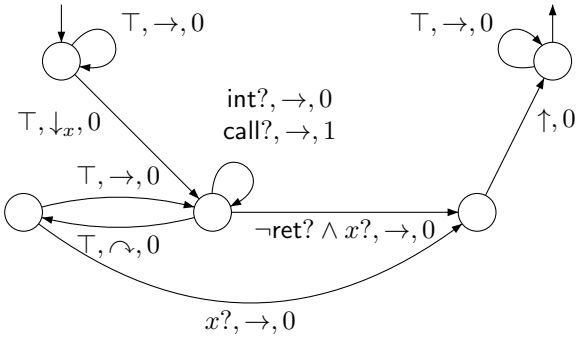


Fig. 3. A pebWA computing the call-depth

4.1 Kleene Theorem

We now extend the Kleene theorem to our setting. In order to express the complexity of the construction, we define the *literal-length* $\ell\ell(E)$ of an expression as the number of occurrences of moves in $\{\rightarrow, \leftarrow, \curvearrowright, \curvearrowleft\}$ plus twice the number of occurrences of ! (in $x!-$).

Theorem 5. *For each* pebWE E*, we can construct a* layered pebWA $\mathcal{A}(E)$ *equivalent to* E*, i.e., for all nested words* W *and for all assignments* σ *we have:* $[\![\mathcal{A}(E)]\!](W, \sigma) = [\![E]\!](W, \sigma)$*. Moreover, the number of layers in* $\mathcal{A}(E)$ *is the pebble-depth of* E*, and its number of states is* $1 + \ell\ell(E)$*. Conversely, for each* layered pebWA *we can construct an equivalent* pebWE*.*

Such extensions of Kleene's theorem have been proved for various weighted models. In [18], Sakarovitch gives a survey about different constructions establishing Schützenberger's theorem, namely Kleene's theorem for weighted one-way automata over finite words. An efficient algorithm constructing an automaton from an expression uses standard automata (which has as variants Berry-Sethi algorithm, or Glushkov algorithm). In [12], we extended this algorithm to deal with pebbles and two-way navigation in (classical) words. It is not difficult – and not surprising – to see that this extension holds in the context of nested words too. For the converse translation, weighted versions of the state elimination

method of Brzozowski-McCluskey, or the procedure of McNaughton-Yamada can easily be applied to our two-way/pebble setting as previously stated in [12] over (non-nested) words.

4.2 Evaluation

We now study the evaluation problem of a K-layered pebWA $\mathcal{A}$: given a nested word W over A and a valuation $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(W)$, compute $[\![\mathcal{A}]\!](W,\sigma)$. The problem is non-trivial since, even if the nested word is fixed, the number of accepting runs may be infinite.

Our evaluation algorithm requires the computation of the matrix N^* given a square matrix $N \in \mathbb{S}^{n\times n}$. By definition, N^* is defined as the infinite sum $\sum_{k\geq 0} N^k$, which is well defined since the semiring is continuous, but may seem difficult to compute. However, using Conway's decomposition of the star of a matrix (see [6] for more details), we can compute N^* with $\mathcal{O}(n)$ scalar star operations and $\mathcal{O}(n^3)$ scalar sum and product operations.

Theorem 6. *Given a layered* pebWA *with ρ pebble names and a nested word W, we can compute with $\mathcal{O}((\rho+1)|Q|^3|W|^{\rho+1})$ scalar operations (sum, product, star) the values $[\![\mathcal{A}_{p,q}]\!](W,\sigma)$ for all states $p,q \in Q$ and valuations $\sigma\colon \mathrm{Peb} \to \mathrm{pos}(W)$.*

Proof. In the whole proof, we fix a nested word $W = (w, \curvearrowright)$. We follow the same basic idea used to evaluate weighted automata over words, namely computing matrices of weights for partial runs [12]. The 2-way navigation is resolved by computing simultaneously matrices of weights of the back and forth loops, whereas we deal with layers inductively. Finally, we deal with call-return edges by using a hierarchical order based on the call-depth to compute the different matrices. Hence, for every position $j \in \overline{\mathrm{pos}}(W)$ we consider the pair $(\mathrm{start}(j), \mathrm{end}(j))$ of start and end positions as follows

$$\mathrm{start}(j) = \min\{i \in \mathrm{pos}(W) \mid i \leq j \wedge \forall k \quad i \leq k \leq j \implies \text{c-d}(k) \geq \text{c-d}(j)\}$$
$$\mathrm{end}(j) = \max\{i \in \overline{\mathrm{pos}}(W) \mid j \leq i \wedge \forall k \quad j \leq k \leq i \implies \text{c-d}(k) \geq \text{c-d}(j)\}$$

For positions of call-depth 0, the start position is 0 whereas the end position is $|W|$. For positions of call-depth at least 1 (see Fig. 4), the start position is the linear successor of the closest call in the past such that its matched return is after position j, whereas its end position is the linear predecessor of this return position.

Let $\mathcal{A} = (Q, A, I, M, T)$ be a K-layered pebWA. For $k \leq K$, we let $Q^{(k)} = \ell^{-1}(k)$ be the set of states in layer k. For every layer $k \in \{0,\dots,K\}$ and every states $p,q \in Q^{(k)}$ we denote by $B^{(k)}_{\sigma,p,q}$ the sum of weights of the runs from configuration $(W,\sigma,p,\varepsilon,0)$ to configuration $(W,\sigma,q,\varepsilon,|W|)$: observe that the stack of pebbles is empty at the beginning of these runs, hence they stay in layers $k, k-1,\dots,0$. Notice that $B^{(k)}_{\sigma,p,q} = [\![\mathcal{A}_{p,q}]\!](W,\sigma)$. In the following, these coefficients (and others that will be defined later) will be grouped into matrices: for

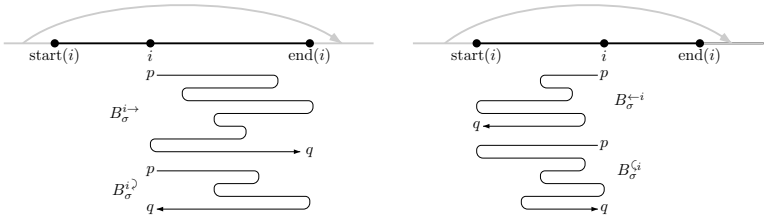


Fig. 4. Representation of the four types of matrices

example, we denote by $B_\sigma^{(k)}$ the $(Q^{(k)} \times Q^{(k)})$-matrix containing all coefficients $(B_{\sigma,p,q}^{(k)})_{p,q \in Q^{(k)}}$.

Fix a layer $k \in \{0, \dots, K\}$ of the automaton. Suppose by induction that we have already computed matrices $B_\sigma^{(k-1)}$ for every valuation $\sigma : \text{Peb} \to \text{pos}(W)$. For a valuation $\sigma : \text{Peb} \to \text{pos}(W)$, the matrix $B_\sigma^{(k)}$ will be obtained by the computation of four types of matrices for every position (see Fig. 4). For example, $B_{\sigma,p,q}^{i\rightarrow}$ (resp. $B_{\sigma,p,q}^{i\curvearrowright}$) is the sum of weights of the runs from configuration $(W, \sigma, p, \varepsilon, i)$ to $(W, \sigma, q, \varepsilon, \text{end}(i))$ (resp. $(W, \sigma, q, \varepsilon, i)$) with intermediary configurations of the form (W, σ, r, π, j) with $\pi \neq \varepsilon$ or $i \leq j \leq \text{end}(i)$. These runs are those which stay on the right of their starting position in positions with (at least) deeper call-depth (except when they drop pebbles, where then the automaton can read the whole nested word) stopping at the corresponding end position (resp. their starting position).

We compute these four types of matrices for every position i by decreasing value of call-depth. Suppose this has been done for every position of call-depth greater than d. We describe how to compute matrices $B_\sigma^{i\rightarrow}$ and $B_\sigma^{i\curvearrowright}$ for every position i of call-depth d, by decreasing values of i. [2] Similarly, matrices $B_\sigma^{\leftarrow i}$ and $B_\sigma^{\curvearrowleft i}$ can be computed by increasing values of positions i having call-depth d.

We let $M_{\sigma,i}^d$ the matrix with p,q-coefficient $\sum_{\alpha \in \text{Test} | W,\sigma,i \models \alpha} \delta(p, \alpha, d, q)$ for $d \in \{\leftarrow, \rightarrow, \curvearrowright, \curvearrowleft\}$: this coefficient denotes the weight of taking a transition with move d from state p to state q on position i with current valuation σ. We similarly define matrices for drop, denoted $M_{\sigma,i}^{\downarrow_x}$, and lift moves: without loss of generality, we assume that lift moves only occur on position $|W|$, so that it may be denoted as $M^\uparrow$ for any (useless) valuation σ.

There are four distinct cases:

If $i = \text{end}(i)$**:** The only way to loop on the right of i, staying on the left of $\text{end}(i) = i$ is to drop a pebble a certain number of times. We introduce a macro for the drop-lift sequences: $N_{\sigma,i} = 0$ if $k = 0$ since no pebble may still

[2] These positions can be partitioned according to their associated start and end positions, and computed independently if wanted.

be dropped, and for $k > 0$ we let

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

Then, we have (notice that the star of the null matrix is the identity)

$$B_{\sigma}^{i\circlearrowright} = \left(N_{\sigma,i}\right)^* = B_{\sigma}^{i\rightarrow} .$$

If $i < \mathrm{end}(i)$ and i is not a call: Then, looping on the right of i either starts by dropping of a pebble over i, or starts with a right move, followed by a loop on the right of $i+1$ and a left move. All of this may be iterated using a star operation:

$$B_{\sigma}^{i\circlearrowright} = \left(N_{\sigma,i} + M_{\sigma,i}^{\rightarrow} \times B_{\sigma}^{i+1\circlearrowright} \times M_{\sigma,i+1}^{\leftarrow}\right)^* .$$

Moving to the right of position i, until $\mathrm{end}(i)$, can be decomposed as a loop on the right of i, followed by a right move (from that point, we will not reach position i anymore) and a run from $i+1$ to $\mathrm{end}(i+1) = \mathrm{end}(i)$:

$$B_{\sigma}^{i\rightarrow} = B_{\sigma}^{i\circlearrowright} \times M_{\sigma,i}^{\rightarrow} \times B_{\sigma}^{i+1\rightarrow} .$$

If $i < \mathrm{end}(i)$ and $i \curvearrowright i+1$: The situation is very similar except that there are two moves leading from i to $i+1$ and two moves leading from $i+1$ to i.

$$\begin{aligned} B_{\sigma}^{i\circlearrowright} &= \left(N_{\sigma,i} + (M_{\sigma,i}^{\rightarrow} + M_{\sigma,i}^{\curvearrowright}) \times B_{\sigma}^{i+1\circlearrowright} \times (M_{\sigma,i+1}^{\leftarrow} + M_{\sigma,i+1}^{\curvearrowright})\right)^* \\ B_{\sigma}^{i\rightarrow} &= B_{\sigma}^{i\circlearrowright} \times (M_{\sigma,i}^{\rightarrow} + M_{\sigma,i}^{\curvearrowright}) \times B_{\sigma}^{i+1\rightarrow} . \end{aligned}$$

If $i < \mathrm{end}(i)$ and $i \curvearrowright j$ with $j \neq i+1$: Looping on the right of i consists of either (1) dropping a pebble over i, or (2) looping on the right of i without ever reaching j, or (3) going to j, looping on j staying between positions $i+1$ and $\mathrm{end}(i)$, and going back to i without reaching position j again. All of this is again possibly iterated using a star operation. Notice that positions $i+1$ and $j-1$ have a call-depth greater than d, hence their four types of matrices have been computed previously. We have $\mathrm{end}(i) = \mathrm{end}(j)$, $\mathrm{end}(i+1) = j-1$ and $\mathrm{start}(j-1) = i+1$. First we define a matrix for the runs going from i to j, another one for those looping over j, and a last one for those going from j to i:

$$\begin{aligned} \text{Goto-Return}_{\sigma,i} &= M_{\sigma,i}^{\curvearrowright} + M_{\sigma,i}^{\rightarrow} \times B_{\sigma}^{i+1\rightarrow} \times M_{\sigma,j-1}^{\rightarrow} \\ \text{Loop-Return}_{\sigma,j} &= \left(B_{\sigma}^{j\circlearrowright} \times M_{\sigma,j}^{\leftarrow} \times B_{\sigma}^{\circlearrowleft j-1} \times M_{\sigma,j-1}^{\rightarrow}\right)^* \\ \text{Goto-Call}_{\sigma,j} &= M_{\sigma,j}^{\curvearrowleft} + M_{\sigma,j}^{\leftarrow} \times B_{\sigma}^{\leftarrow j-1} \times M_{\sigma,i+1}^{\leftarrow} . \end{aligned}$$

Runs looping on the right of i are then computed by

$$\begin{aligned} B_{\sigma}^{i\circlearrowright} = [&N_{\sigma,i} + M_{\sigma,i}^{\rightarrow} \times B_{\sigma}^{i+1\circlearrowright} \times M_{\sigma,i+1}^{\leftarrow} \\ &+ \text{Goto-Return}_{\sigma,i} \times \text{Loop-Return}_{\sigma,j} \times B_{\sigma}^{j\circlearrowright} \times \text{Goto-Call}_{\sigma,j}]^* . \end{aligned}$$

Finally, to go from i to $\mathrm{end}(i)$, we split the runs considering the last time we reach position i and the last time we visit a position on the left of j:

$$B_\sigma^{i\to} = B_\sigma^{i\circlearrowright} \times \text{Goto-Return}_{\sigma,i} \times \text{Loop-Return}_{\sigma,j} \times B_\sigma^{j\to}\,.$$

If $i = 0$ then $\mathrm{end}(i) = |W|$ and we have $B_\sigma^{(k)} = B_\sigma^{0\to}$. Hence, we have computed the behavior of layer k.

To conclude this proof, it remains to count the number of matrix operations (sum, product and star) in the whole computation and then infer the number of scalar operations assuming standard algorithms on matrices (quadratic for sum, cubic for product and cubic for star as noticed previously).

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

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

Notice that if the nested word is in fact a word, our algorithm only needs to compute the two sets of matrices $B_\sigma^{i\to}$ and $B_\sigma^{i\circlearrowright}$ with a backward visit of the positions of the word. This is indeed a different algorithm than the one presented in [12] where the positions are visited in a forward manner.

4.3 Decidability of Emptiness

Classical decision problems over finite state automata have natural counterparts in the weighted setting. For example, the emptiness problem takes as input a pebWA $\mathcal{A}$, and asks whether there exists a nested word W such that $[\![\mathcal{A}]\!](W) \neq 0$.

Theorem 7. *The emptiness problem is decidable, with non-elementary complexity, for layered* pebWA *over a continuous semiring* $\mathbb{S}$ *with no zero divisor, i.e., such that* $s \times s' = 0$ *implies* $s = 0$ *or* $s' = 0$.

Proof. Let $\mathcal{A}$ be a pebWA and $\mathbb{S}$ be a continuous semiring with no zero divisor. The crucial fact is that the support of $\mathcal{A}$, i.e., the set of nested words W such that $[\![\mathcal{A}]\!](W) \neq 0$ is recognizable by a pebWA $\mathcal{B}$ over the *Boolean* semiring. Indeed, $\mathcal{B}$ is obtained from $\mathcal{A}$ by replacing nonzero weights by 1, i.e., keeping only transitions with nonzero weights and then ignoring weights. The correctness follows from the fact that in a continuous semiring, $0 = \sum_{i\in I} s_i$ implies $s_i = 0$ for all $i \in I$. Hence, $[\![\mathcal{A}]\!](W) \neq 0$ if and only if there is an accepting run of $\mathcal{A}$ on W whose weight is nonzero. Since $\mathbb{S}$ has no zero divisor, this is equivalent to the existence of an accepting run of $\mathcal{B}$ on W. Hence, the emptiness problem for $\mathcal{A}$ reduces to the emptiness problem for $\mathcal{B}$.

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

$$[\![\Phi + \Psi]\!] = [\![\Phi]\!] + [\![\Psi]\!] \qquad \left[\!\left[\textstyle\sum_x \Phi\right]\!\right](W,\sigma) = \sum_{u \in \mathrm{pos}(W)} [\![\Phi]\!](W, \sigma[x \mapsto u])$$

$$[\![\Phi \times \Psi]\!] = [\![\Phi]\!] \times [\![\Psi]\!] \qquad \left[\!\left[\textstyle\prod_x \Phi\right]\!\right](W,\sigma) = \prod_{u \in \mathrm{pos}(W)} [\![\Phi]\!](W, \sigma[x \mapsto u])$$

Table 3. Semantics of wFO

It should be no surprise that the emptiness problem for Boolean layered pebble automata over nested words is decidable. Indeed, mimicking the proof of [10], we may build from a Boolean layered pebble automaton $\mathcal{B}$ over nested words an equivalent MSO formula, for which decidability of satisfiability is known. Alternatively, we may use the fact that nested words can be encoded in trees and construct from a Boolean layered pebble automaton over nested words an equivalent pebble tree walking automaton, for which decidability of emptiness is known by [10]. □

5 Weighted Logical Specifications over Nested Words

5.1 Weighted First-Order Logic

We fix an infinite supply of first-order variables $\mathcal{V} = \{x, y, \ldots\}$. We suppose known the fragment of (Boolean) first-order formulae, denoted as FO, over nested words, defined by the grammar

$$\varphi ::= \top \mid P_a(x) \mid x \leq y \mid x \curvearrowright y \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x\, \varphi \mid \forall x\, \varphi$$

where $a \in A$ and $x, y \in \mathcal{V}$.

The weighted extension is based on sums and products as for FO with counting (see [15], e.g.). The class of weighted first-order formulae, denoted as wFO, and first introduced by authors of [7], is defined by:

$$\Phi ::= s \mid \varphi \mid \Phi + \Phi \mid \Phi \times \Phi \mid \textstyle\sum_x \Phi \mid \prod_x \Phi$$

where $s \in \mathbb{S}$, $\varphi \in \mathrm{FO}$, and $x \in \mathcal{V}$.

The semantics of a wFO formula is a map from nested words to the semiring. For the inductive definition, we need to consider formulae with free variables. So let $\Phi \in \mathrm{wFO}$. Then, $[\![\Phi]\!]$ maps to a value in $\mathbb{S}$ each pair (W, σ) where W is a nested word and $\sigma : V \to \mathrm{pos}(W)$ is a valuation of a subset $V \subseteq \mathcal{V}$ containing the free variables of Φ. The inductive definition is given in Table 3. It is possible to define a quantitative implication: given $\varphi \in \mathrm{FO}$ and $\Phi \in \mathrm{wFO}$, we let $\varphi \overset{+}{\Longrightarrow} \Phi$ be a macro for the formula $\neg\varphi + \varphi \times \Phi$. Then, its semantics coincides with the

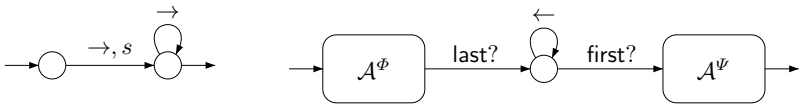


Fig. 5. Automata for $\Xi = s$ and $\Xi = \Phi \times \Psi$

semantics of Φ if φ holds, and its semantics is 1 (the unit of the semiring) if φ does not hold.

Example 8. In the semiring of natural numbers, the formula $\sum_y (x \leq y \wedge P_a(y))$ computes the number of a's after the position of variable x in the nested word.

In a probabilistic setting, the formula $\Phi(x) = \prod_y \left(y \leq x \overset{+}{\Longrightarrow} 1/2\right)$ maps a position i to $(1/2)^{i+1}$, hence defining a geometric probability subdistribution over the positions of the nested word. We can then compute the *expectation* of a formula $\Psi(x)$ by the formula $\sum_x \Phi(x) \times \Psi(x)$.

Finally, in the $(\mathbb{R}, \max, +, \infty, 0)$ semiring, the formula

$$\sum_x \prod_y \left[\left(\exists z\ (y \curvearrowright z \wedge y < x < z)\right) \overset{+}{\Longrightarrow} 1\right]$$

computes the call-depth of a nested word, as it was already presented for expressions and automata in Examples 2 and 4. Notice again that in the $(\mathbb{R}, \max, +, \infty, 0)$ semiring, sum stands for max, product for $+$, and the unit of the semiring is 0.

Theorem 9. *Let $\Phi \in$ wFO be a formula and $V \subseteq \mathcal{V}$ be a finite set containing the variables occurring in Φ (free or bound). We can effectively construct an equivalent layered* pebWA $\mathcal{A}^{\Phi}$ *with $\mathcal{O}(|\Phi|)$ states and set V of pebble names: $[\![\mathcal{A}^{\Phi}]\!](W, \sigma) = [\![\Phi]\!](W, \sigma)$ for every nested word W and valuation $\sigma \colon V \to \mathrm{pos}(W)$.*

Proof. This is achieved by structural induction on the formula. We deal with Boolean formulae below. For $\Xi = s \in \mathbb{S}$, the automaton is given on the left of Fig. 5. For the sum $\Xi = \Phi + \Psi$, we use a non-deterministic choice as usual. For the product $\Xi = \Phi \times \Psi$, the construction is described on the right of Fig. 5: the automaton first performs the computation of formula Φ, that must end in the last position of the nested word[3], before resetting the position to the first one, and then performs the computation of formula Ψ.

Constructions for $\Xi = \sum_x \Phi$ and $\Xi = \prod_x \Phi$ are schematized in Fig. 6. The idea is to use a pebble of name x that we drop – either using a non-deterministic

[3] This is checked by using a transition labelled by last?. Formally, this kind of transitions, which are not performing actions, is not allowed in our model of automata, however, we may easily transform these *only-testing* transitions by making them perform a little back-and-forth move around their position. For the current last? transition, it is even sufficient to make it move to the left using action ←, as this clearly does not change the overall semantics of the automaton.

choice, or systematically other all positions – before performing the computation of formula Φ using the automaton $\mathcal{A}^\Phi$ obtained by induction. Notice that automaton $\mathcal{A}^\Phi$ has x as a free pebble name, as x may be checked in its first layer, before being possibly dropped again.

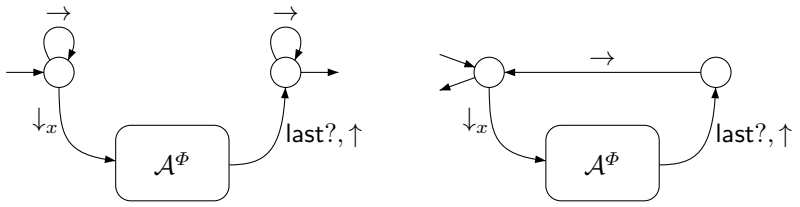


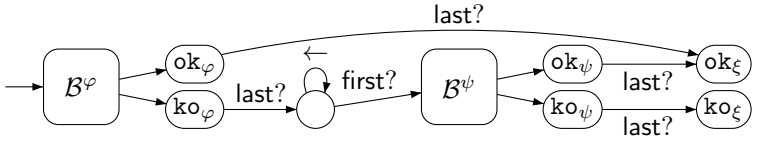
Fig. 6. Automata for $\sum_x \Phi$ and $\prod_x \Phi$

We explain now the construction of pebWA for Boolean formulae. The values computed by such automata should be in $\{0, 1\}$ so we cannot freely use nondeterminism. Also, to achieve the complexity stated in the theorem, we cannot use classical constructions yielding deterministic automata. Instead, we build unambiguous automata. Hence, for every $\varphi \in \mathrm{FO}$ and $V \subseteq \mathcal{V}$ containing the free variables of φ, we construct a pebWA $\mathcal{B}^\varphi$ having one initial state ι and two (final) states $\mathtt{ok}$ and $\mathtt{ko}$ such that for all nested words W and valuations $\sigma \colon V \to \mathrm{pos}(W)$, the quantitative semantics satisfies:

$$[\![\mathcal{B}^\varphi_{\iota,\mathtt{ok}}]\!](W, \sigma, 0, |W|) = \begin{cases} 1 & \text{if } W, \sigma \models \varphi \\ 0 & \text{otherwise,} \end{cases}$$

$$[\![\mathcal{B}^\varphi_{\iota,\mathtt{ko}}]\!](W, \sigma, 0, |W|) = \begin{cases} 0 & \text{if } W, \sigma \models \varphi \\ 1 & \text{otherwise.} \end{cases}$$

We obtain automaton $\mathcal{A}^\varphi$ by considering $\mathcal{B}^\varphi$ with ι (resp. $\mathtt{ok}$) having initial (resp. final) weight 1. To get an automaton for the negation of a formula, we simply exchange states $\mathtt{ok}$ and $\mathtt{ko}$. Automata for atoms $P_a(x)$ and $x \curvearrowright y$ are given in Fig. 7 and the automaton for $x \leq y$ is left to the reader.



The construction for disjunction $\xi = \varphi \vee \psi$ is described above. It is similar to the one used for the product: we start computing φ and stop if it is verified, otherwise, we reset to the beginning of the nested word and check formula ψ. The construction for conjunction is obtained dually.

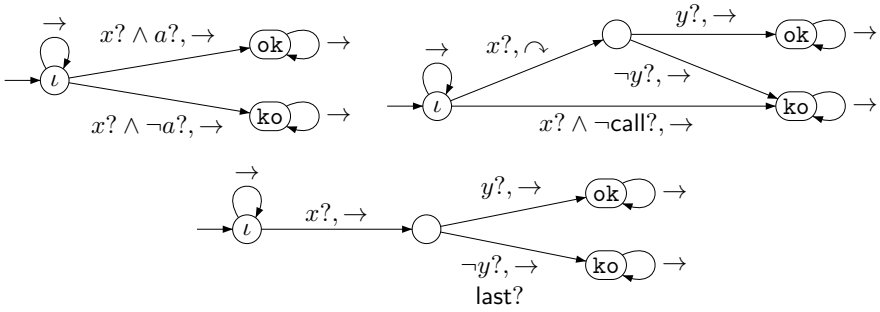
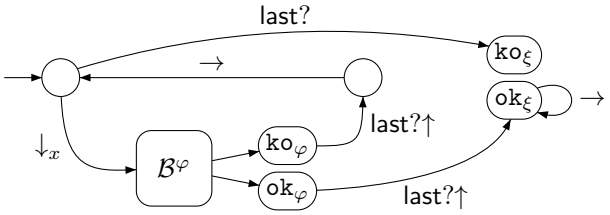


Fig. 7. Automata for $P_a(x)$, $x \curvearrowright y$ and $x \leq y$



Finally, the construction for existential quantification $\xi = \exists x\ \varphi$ is described above. During a run of this automaton, a pebble of name x is successively dropped over each position of the nested word, in order to simulate automaton $\mathcal{B}^\varphi$ over every position. If such a run ends in state $\mathtt{ok}_\xi$, it means that at some position i, formula φ has been positively verified over position i (the part of the run in lower layer ended in state $\mathtt{ok}_\varphi$). More precisely, as $\mathcal{B}^\varphi$ is unambiguous, in that case, all the runs of non-zero weight will end in state $\mathtt{ok}_\xi$ after dropping the pebble on position i, henceforth computing the weight 1, meaning that formula ξ is verified. In the contrary, if no position verifies formula φ, all the runs will end in state $\mathtt{ko}_\xi$, meaning that ξ is not verified. Again, the construction for universal quantification is dual. □

5.2 Weighted Temporal Logics

A temporal logic is usually based on modalities such as *next* and *until* where the until modality is a simple fixed point based on the next modality. When the structures (the models) are not linear, one may follow different paths which are in general based on elementary *steps*. For instance, in unranked trees, one may move vertically down to a child or up to the father, or horizontally to the right or left brother. Similarly, for nested words, several types of paths were introduced in [1] yielding various until modalities, some of them will be discussed below.

Here, we adopt a generic definition of temporal logics where until modalities are based on various elementary steps. Formally, an elementary step η is a regular

expression following the syntax:

$$\begin{aligned}\eta &::= s \mid \alpha \mid \to \mid \leftarrow \mid \curvearrowright \mid \curvearrowleft \mid \eta + \eta \mid \eta \cdot \eta \mid \eta^+ \\ \alpha &::= a? \mid \tau? \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \end{aligned} \tag{1}$$

with $s \in \mathbb{S}$, $a \in A$ and $\tau \in \mathcal{T}$. In some cases, elementary steps will naturally be *unambiguous*, meaning that the quantitative semantics $[\![\eta]\!]$ as defined in Section 3 coincides with the Boolean semantics: $[\![\eta]\!](W, i, j) \in \{0, 1\}$ for all nested words W and positions $i, j \in \overline{\mathrm{pos}}(W)$.

For instance, the (classical) linear until is based on the linear step $\eta = \to$. The *summary-up* until is based on the summary-up step σ^u defined as $\sigma^u = \curvearrowright + \neg\mathsf{call}? \cdot \to$ which may move directly from a call to the matching return, or go to the successor, but cannot "enter" a call. The *summary-down* until is based on the summary-down step defined as $\sigma^d = \curvearrowright + \to \cdot \neg\mathsf{ret}?$. Notice that σ^d is unambiguous, even though a call position may have two successors. An example of a non-unambiguous step, in the $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ semiring, is $\eta_{cd} = (1\,\mathsf{call}? \to \neg\mathsf{ret}?) + (\mathsf{int}? \to \neg\mathsf{ret}?) + \curvearrowright$, aiming at following a path of increasing call-depth, moreover computing 1 each time we enter inside a call.

The syntax of the *weighted* temporal logic wTL over nested words is defined by

$$\Phi ::= s \mid \alpha \mid \Phi + \Phi \mid \Phi \times \Phi \mid \Phi\, \mathsf{SU}^\eta\, \Phi$$

with $s \in \mathbb{S}$, α simple tests as defined in (1), and η elementary steps. Since a (weighted) temporal logic formula has an implicit free variable, the quantitative semantics $[\![\Phi]\!](W, i) \in \mathbb{S}$ maps a nested word W and a position $i \in \mathrm{pos}(W)$ to a value in the semiring. It is defined in Table 4.

This semantics may be computed in another way in case we only deal with unambiguous steps. In that case, given a nested word W, we say that two positions $i, j \in \mathrm{pos}(W)$ form an η-step if $[\![\eta]\!](W, i, j) = 1$. Moreover, an η-path is a sequence $i_0, \ldots, i_n \in \mathrm{pos}(W)$ such that (i_k, i_{k+1}) is an η-step for all $0 \leq k < n$. Then, $[\![\Phi\, \mathsf{SU}^\eta\, \Psi]\!](W, i)$ may be computed by the formula

$$\sum_{i=i_0,i_1,\ldots,i_n \eta\text{-path}} \left(\prod_{0<k<n} [\![\Phi]\!](W, i_k) \right) \times [\![\Psi]\!](W, i_n)\,.$$

As usual, we may use derived modalities such as the *non strict until* defined by $\varphi\, \mathsf{U}^\eta\, \psi \stackrel{\text{def}}{=} \psi + \varphi \times (\varphi\, \mathsf{SU}^\eta\, \psi)$ and η-*next* defined by $\mathsf{X}^\eta\, \varphi \stackrel{\text{def}}{=} \bot\, \mathsf{SU}^\eta\, \varphi$. As special cases, we get the *linear* next $\bot\, \mathsf{SU}^{\to}\, \varphi$ and the *jumping* next $\mathsf{X}^{\curvearrowright}\, \varphi = \bot\, \mathsf{SU}^{\curvearrowright}\, \varphi$. We also get *eventually* with $\mathsf{F}\, \varphi = \top\, \mathsf{U}^{\to}\, \varphi$, but notice that $\top\, \mathsf{SU}^{\curvearrowright}\, \varphi = \mathsf{X}^{\curvearrowright}\, \varphi$ since two consecutive $\curvearrowright$-steps are not possible.

As a concrete example, the call-depth of a nested word can be computed with the formula $\top\, \mathsf{SU}^{\eta_{cd}}\, \top$ as explained previously. Interestingly, we may also compute it with the unambiguous step σ^d with formula $(1 \times \neg\mathsf{ret}? \times \mathsf{X}^{\leftarrow}(\mathsf{call}?) + \mathsf{X}^{\leftarrow}(\neg\mathsf{call}?) + \mathsf{ret}? + \mathsf{first}?)\, \mathsf{U}^{\sigma^d}\, \top$: the idea is rather to compute 1 once the step has been performed, since we have a way to check if the previous step just entered a call position.

$$[\![s]\!](W,i) = s \qquad [\![\alpha]\!](W,i) = \begin{cases} 1 & \text{if } W,i \models \alpha \\ 0 & \text{otherwise} \end{cases}$$

$$[\![\Phi + \Psi]\!] = [\![\Phi]\!] + [\![\Psi]\!] \qquad [\![\Phi \times \Psi]\!] = [\![\Phi]\!] \times [\![\Psi]\!]$$

$$[\![\Phi \,\mathsf{SU}^\eta\, \Psi]\!](W,i) = \sum_{i=i_0,i_1,\ldots,i_n} \Big([\![\eta]\!](W,i_0,i_1) \times \prod_{0<k<n} \big([\![\eta]\!](W,i_k,i_{k+1}) \times [\![\Phi]\!](W,i_k) \big) \Big) \times [\![\Psi]\!](W,i_n)\,. \quad (2)$$

Table 4. Semantics of wTL

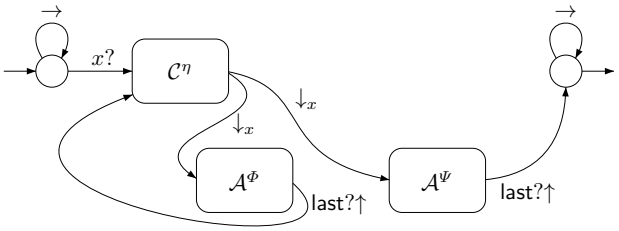


Fig. 8. Automaton for $\Xi = \Phi \,\mathsf{SU}^\eta\, \Psi$

Notice that the sum in (2) may be infinite for some step expressions such as $\eta = \leftarrow + \rightarrow$. On the other hand, if a step expression only moves forward (respectively, backward) then it defines a *future* (respectively, *past*) modality and the sum in (2) is finite. The following theorem shows that wTL formulae can be translated into equivalent layered pebWA.

Theorem 10. *For each* wTL *formula* Φ *we can effectively construct an equivalent layered* pebWA $\mathcal{A}^\Phi$ *with a single pebble name* x *and* $\mathcal{O}(|\Phi|)$ *states: for all nested words* W *and positions* $i \in \mathrm{pos}(W)$ *we have* $[\![\mathcal{A}^\Phi]\!](W, x \mapsto i) = [\![\Phi]\!](W,i)$.

Proof. We proceed by structural induction over the formula Φ. Automata for formula s and α are identical to those used in Theorem 9. Moreover, constructions of Theorem 9 for the sum $\Phi + \Psi$ and the product $\Phi \times \Psi$ work again in the case of temporal logics.

We detail the construction for SU^η where η is a step expression. Using Theorem 5, from the pebWE η we obtain an equivalent pebWA $\mathcal{C}^\eta$: $[\![\mathcal{C}^\eta]\!](W,i,j) = [\![\eta]\!](W,i,j)$ for all nested words W and positions $i,j \in \overline{\mathrm{pos}}(W)$.

Consider the wTL formula $\Xi = \Phi \,\mathsf{SU}^\eta\, \Psi$ and assume we have already constructed the pebWA $\mathcal{A}^\Phi$ and $\mathcal{A}^\Psi$. The pebWA $\mathcal{A}^\Xi$ is given in Fig. 8. Observe that we have added one layer and a constant number of states. The automaton starts by searching for pebble x, and then it enters in a loop. At every iteration, it enters in automaton $\mathcal{C}^\eta$ to follow the step η and reach the next position. When exiting $\mathcal{C}^\eta$ (having computed the weight $[\![\eta]\!](W,i,j)$), it chooses between two options: either it drops pebble x (overwriting the previous pebble x) and compute

formula Φ on the current position before lifting the pebble and continuing to the next iteration, or it decides that formula Ψ must now be checked, using a similar technique. In the overall, resolving the non-determinism by a sum in the semiring, the automaton computes exactly the semantics given in (2). □

Notice that we can extend the wTL in several ways. One option could be to allow arbitrary Boolean temporal formulae instead of simple (pebble-free) tests α. A more strict requirement over steps η should then be designed in order to be able to compute such Boolean temporal formulae with weighted automata, in the same way Boolean first-order formulae have been translated in Theorem 9.

6 Conclusion and Perspectives

We have presented a general framework to specify quantitative properties of nested words, and compile them into automata.

Several improvements can be considered. First, concerning our procedure for evaluation, [12] also presented an improved algorithm in the strongly layered case, namely when at every layer, only one pebble name can be dropped. We did not consider this case in this paper for lack of space, but we believe it can be adapted in the nested word case, and leave it for future work. Second, we would like to extend the decidability of emptiness in the general case, where the semiring may have zero divisors.

Finally, notice that, contrary to [1], wFO is strictly less expressive than our temporal logics. This is due to the power of η steps, which gives to wTL a flavor of weighted transitive closure (see [5]), and of regular LTL [14]. As other directions of research, we would like to study this transitive closure operator in order to find a logical fragment expressively equivalent to pebWA and pebWE.

References

1. Rajeev Alur, Marcelo Arenas, Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. First-order and temporal logics for nested words. *LMCS*, 4(4), 2008.
2. Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56:16:1–16:43, May 2009.
3. M. Bojańczyk. Tree-walking automata. In *LATA'08*, volume 5196 of *LNCS*, pages 1–17. Springer, 2008.
4. Benedikt Bollig, Paul Gastin, and Benjamin Monmege. Weighted specifications over nested words. In *Proceedings of FOSSACS'13*, 2013. Accepted paper.
5. Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. In *ICALP'10*, volume 6199 of *LNCS*, pages 587–598. Springer, 2010.
6. J.H. Conway. *Regular Algebra and Finite Machines*. Chapman & Hall, 1971.
7. M. Droste and P. Gastin. Weighted automata and weighted logics. In *Handbook of Weighted Automata*, chapter 5, pages 175–211. Springer, 2009.
8. M. Droste and H. Vogler. Weighted logics for unranked tree automata. *Theory Comput. Syst.*, 48(1):23–47, 2011.

9. Manfred Droste and Werner Kuich. Semirings and formal power series. In *Handbook of Weighted Automata*, chapter 1, pages 3–27. Springer, 2009.
10. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In *Jewels Are Forever*, pages 72–83. Springer, 1999.
11. Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML transformation by tree-walking transducers with invisible pebbles. In *Proceedings of PODS'07*, pages 63–72, 2007.
12. Paul Gastin and Benjamin Monmege. Adding pebbles to weighted automata. In *CIAA'12*, LNCS, pages 28–51. Springer-Verlag, 2012.
13. O. Gauwin, J. Niehren, and Y. Roos. Streaming tree automata. *Inf. Process. Lett.*, 109(1):13–17, 2008.
14. Martin Leucker and César Sánchez. Regular linear temporal logic. In *Proceedings of ICTAC'07*, volume 4711 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2007.
15. Leonid Libkin. *Elements of Finite Model Theory*. EATCS. Springer-Verlag, 2004.
16. P. Madhusudan and M. Viswanathan. Query automata for nested words. In *Proceedings of MFCS'09*, volume 5734 of *LNCS*, pages 561–573. Springer, 2009.
17. Christian Mathissen. Weighted logics for nested words and algebraic formal power series. *Logical Methods in Computer Science*, 6(1), 2010.
18. Jacques Sakarovitch. Automata and expressions. In *AutoMathA Handbook*. 2012. To appear.
19. M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.