Weighted Operator Precedence Languages*

Manfred Droste[1], Stefan Dück[1**], Dino Mandrioli[2], and Matteo Pradella[2,3]

[1] Institute of Computer Science, Leipzig University, D-04109 Leipzig, Germany
{droste,dueck}@informatik.uni-leipzig.de
[2] Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
{dino.mandrioli,matteo.pradella}@polimi.it
[3] IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133 Milano, Italy

Abstract. In the last years renewed investigation of operator precedence languages (OPL) led to discover important properties thereof: OPL are closed with respect to all major operations, are characterized, besides by the original grammar family, in terms of an automata family (OPA) and an MSO logic; furthermore they significantly generalize the well-known visibly pushdown languages (VPL). A different area of research investigates quantitive evaluations of formal languages by adding weights to strings. In this paper, we lay the foundation to marry these two research fields. We introduce weighted operator precedence automata and show how they are both strict extensions of OPA and weighted visibly pushdown automata. We prove a Nivat-like result which shows that quantitative OPL can be described by unweighted OPA and very particular weighted OPA. In a Büchi-like theorem, we show that weighted OPA are expressively equivalent to a weighted MSO-logic for OPL.

Keywords: quantitative automata, operator precedence languages, input-driven languages, visibly pushdown languages, quantitative logic

1 Introduction

In the long history of formal languages the family of regular languages (RL), those that are recognized by finite state machines (FSM) or are generated by regular grammars, has always played a major role: thanks to its simplicity and naturalness it enjoys properties that only partially extend to larger families. Among the many positive results that have been achieved for RL (e.g., expressiveness, decidability, minimization, ...), those of main interest in this paper are the following:

- RLs have been characterized in terms of various mathematical logics. The pioneering papers are due to Büchi, Elgot, and Trakhtenbrot [7, 23, 41] who

* Work partially supported by project AUTOVAM, funded by Fondazione Cariplo and Regione Lombardia.
** supported by Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg 1763 (QuantLA).

independently developed a monadic second order (MSO) logic defining exactly the RL family. This work too has been followed by many further results; in particular those that exploited weaker but simpler logics such as first-order, propositional, and temporal ones which culminated in the breakthrough of model checking to support automatic verification [35, 24, 8].
- Weighted RLs have been introduced by Schützenberger in his pioneering paper [39]: by assigning a weight in a suitable algebra to each language word, we may specify several attributes of the word, e.g., relevance, probability, etc. Much research then followed and extended Schützenberger's original work in various directions, cf. the books [22, 38, 30, 4, 15].

Unfortunately, all families with greater expressive power than RL – typically context-free languages (CFL), which are the most widely used family in practical applications – pay a price in terms of properties and, consequently, of possible tools supporting their automatic analysis. For instance, for CFL, the inclusion problem is undecidable and they are not closed under complement.

What was not possible for general CFL, however, has been possible for important subclasses of this family, which together we call *structured CFL*. Informally, with this term we denote those CFLs where the syntactic tree-structure of their words is immediately "visible" in the words themselves. A first historical example of such families is that of parenthesis languages, introduced by McNaughton in another seminal paper [34], which are generated by grammars whose right hand sides are enclosed within pairs of parentheses; not surprisingly an equivalent formalism of parenthesis grammars was soon defined, namely tree-automata which generalize the basics of FSM to tree-like structures instead of linear strings [40]. Among the many variations and generalizations of parenthesis languages the recent family of *input-driven languages (IDL)* [36, 6], alias *visibly pushdown languages (VPL)* [2], has received much attention in recent literature. For most of these structured CFL, including in particular IDL, the relevant algebraic properties of RL still hold [2]. One of the most noticeable results of this research field has been a characterization of IDL/VPL in terms of a MSO logic that is a fairly natural extension of Büchi's original one for RL [31, 2].

This fact has suggested to extend the investigation of weighted RL to various cases of structured languages. The result of such a fertile approach is a rich collection of *weighted logics*, first studied by Droste and Gastin [13], associated with *weighted tree automata* [20] and *weighted visibly pushdown automata (VPA)*, the automata recognizing VPL, also called *weighted nested word automata (NWA)* [11, 33, 19].

In an originally unrelated way *operator precedence languages (OPL)* have been defined and studied in two phases temporally separated by four decades. In his seminal work [26] Floyd was inspired by the precedence of multiplicative operations over additive ones in the execution of arithmetic expressions and extended such a relation to the whole input alphabet in such a way that it could drive a deterministic parsing algorithm that builds the syntax tree of any word that reflects the word's semantics; Figure 1 and Section 2 give an intuition of how an OP grammar generates arithmetic expressions and assigns them a natural

structure. After a few further studies [10], OPL's theoretical investigation has been abandoned due to the advent of LR grammars which, unlike OP grammars, generate all deterministic CFL.

OPL, however, enjoy a distinguishing property which we can intuitively describe as *"OPL are input driven but not visible"*. They can be claimed as *input-driven* since the parsing actions on their words –whether to push or to pop their stack– depend exclusively on the input alphabet and on the relation defined thereon, but their structure is *not visible* in their words: e.g, they can include unparenthesized arithmetic expressions where the precedence of multiplicative operators over additive ones is explicit in the syntax trees but hidden in their frontiers (see Figure 1). Furthermore, unlike other structured CFL, OPL include deterministic CFL that are not real-time [32].

This remark suggested to resume their investigation systematically at the light of the recent technological advances and related challenges. Such a renewed investigation led to prove their closure under all major language operations [9] and to characterize them, besides by Floyd's original grammars, in terms of an appropriate class of pushdown automata (OPA) and in terms of a MSO logic which is a fairly natural but not trivial extension of the previous ones defined to characterize RL and VPL [32]. Thus, OPL enjoy the same nice properties of RL and many structured CFL but considerably extend their applicability by breaking the barrier of visibility and real-time pushdown recognition.

In this paper we put together the two above research fields, namely we introduce *weighted OPL* and show that they are able to model system behaviors that cannot be specified by means of less powerful weighted formalisms such as weighted VPL. For instance, many events like interrupts in operating systems, exceptions in programming languages, errors during the transfer of complex web data, may have a different impact on the overall quality of the produced result. It is therefore important to evaluate how critically the occurrences of such events affect the normal system behavior, e.g., by counting the number of pending calls that have been preempted by an interrupt, or by weighing transmission errors depending on where they occur within a web page.

As an example consider a system logging all hierarchical calls and returns over words where this structural information is hidden. Depending on changing exterior factors like energy level, such a system could decide to log the above information in a selective way. In Section 3 we describe a few practical cases where system quality can be naturally modeled in terms of our new weighted OPL model. We will also show that VPL are not adequate to model such systems.

Our main contributions in this paper are the following, after introducing the necessary background on OPL (Section 2):

- The new model of *weighted OPA*, which have weights at their transitions, is introduced by adopting as the weight algebra a *valuation monoid*, i.e., an additive monoid equipped with a general *valuation function* [17] (Section 3). We also investigate semirings as a special and interesting case of valuation monoids (Section 8).

- Weighted OPA significantly increase the descriptive power of previous weighted extensions of VPA (Section 4), and have desired closure and robustness properties (Section 5).
- In general, there is a relevant difference in the expressive power of the model depending on whether it permits assigning weights to pop transitions or not. The difference in descriptive power is due to the fact that OPL may be non-real-time and therefore OPA may execute several pop moves without advancing their reading heads (Section 3). For commutative semirings, however, we show that weights on pop transitions do not increase the expressive power of the automata (Section 8).
- We extend the classical result of Nivat [37] to weighted OPL. This robustness result shows that the behaviors of weighted OPA without weights at pop transitions are exactly those that can be constructed from weighted OPA with only one state, intersected with OPL, and applying projections which preserve the structural information (Section 5).
- We propose a weighted MSO logic and a Büchi-Elgot-Trakhtenbrot-Theorem proving its expressive equivalence to weighted OPA without weights at pop transitions (Sections 6 and 7). This result holds for general weight structures, which include all semirings and valuation functions like average. As a corollary, for commutative semirings, this weighted logic is equivalent to weighted OPA including weights at pop transitions (Section 8).

An earlier, partial report on the above results appeared in [12]. The present version extends and completes the previous one in that it documents all technical details –mainly the proofs– that have been omitted there. Furthermore, the original formulation where the algebra of weights was a semiring is now generalized to valuation monoids, of which semirings are an important special case. Additionally, in contrast to the case of commutative semirings, we prove that even for commutative valuation monoids, the version of wOPA without pop weights is strictly weaker than the unrestricted version.

2 Operator Precedence Languages

We rely on the basic knowledge of the reader in the field of formal language theory, for concepts such as grammar, derivation and parsing, or refer them to any classic textbook of this discipline, e.g., [28].

Let Σ be an alphabet and ε the empty string. Let $G = (\Sigma, V_N, R, S)$ be a *context-free* (CF) grammar, where V_N is the nonterminal alphabet, R the rule (or production) set, and S the axiom. A rule is in *operator form* if its right hand side has no adjacent nonterminals; an *operator grammar* (OG) contains only such rules. The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $r, s, t, u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup V_N$. The strings may be empty, unless stated otherwise.

For an OG G and a nonterminal A, the *left and right terminal sets* are

$$\mathcal{L}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} Ba\alpha\} \qquad \mathcal{R}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} \alpha aB\}$$

where $B \in V_N \cup \{\varepsilon\}$ and $\overset{*}{\Rightarrow}$ denotes the derivation relation.
The following binary operator precedence (OP) relations are defined:

$$\begin{aligned}
\text{equal in precedence: } & a \doteq b \iff \exists A \to \alpha aBb\beta, B \in V_N \cup \{\varepsilon\} \\
\text{takes precedence: } & a \gtrdot b \iff \exists A \to \alpha Db\beta, D \in V_N \text{ and } a \in \mathcal{R}_G(D) \\
\text{yields precedence: } & a \lessdot b \iff \exists A \to \alpha aD\beta, D \in V_N \text{ and } b \in \mathcal{L}_G(D)
\end{aligned}$$

Following the custom of sequential parsers, we enclose the input string between two # special characters, and we assume that # yields precedence to any other character and any character takes precedence over #.

The *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma \cup \{\#\}| \times |\Sigma \cup \{\#\}|$ array that associates with any ordered pair (a, b) the set M_{ab} of OP relations holding between a and b. We define an *OP alphabet* as a pair (Σ, M).

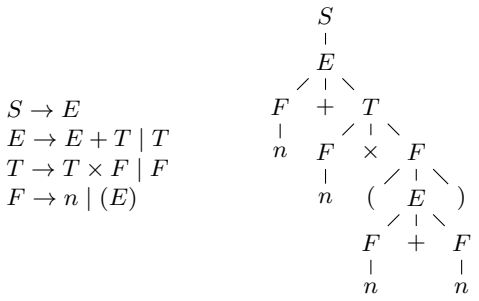
Definition 1. An OG G is an *operator precedence grammar* (OPG) if, and only if, $M = OPM(G)$ is a *conflict-free* matrix, i.e., $\forall a, b$, $|M_{ab}| \leq 1$.

Definition 2. An OPG is in *Fischer normal form* [25] if no two rules have the same right hand side (r.h.s.); no rule, possibly except one with the axiom S as the left hand side (l.h.s.), has ε as the r.h.s.; renaming rules, i.e., those with a single nonterminal character as the r.h.s., are those and only those with S as the l.h.s.

Example 3. Consider arithmetic expressions with two operators, an additive one and a multiplicative one that takes precedence over the other one, in the sense that, during the interpretation of the expression, multiplications must be executed before sums. Parentheses are used to force different precedence hierarchies. Figure 1 depicts a grammar generating arithmetic expressions (a), its precedence matrix (b), its version in Fischer Normal Form (c), and an example derivation tree of the expression $n + n \times (n + n)$ (d).

Notice that the structure of the syntax tree (uniquely) corresponding to the input expression reflects the precedence order which drives computing the value attributed to the expression. This structure, however, is not immediately visible in the expression; if we used a parenthesis grammar, it would produce the string $(n + (n \times (n + n)))$ instead of the previous one, and the structure of the corresponding tree would be immediately visible. For this reason we say that such grammars "hide" the structure associated with a sentence, whereas parenthesis grammars and other input-driven ones make the structure explicit in the sentences they generate.

The precedence relations are used to drive the parsing of a string in the following way. Preliminarily, the original OPG is transformed in Fischer Normal Form.



(d)

$S \to E$
$E \to E + T \mid T$
$T \to T \times F \mid F$
$F \to n \mid (E)$

(a)

	+	×	(	)	n
+	⋗	⋖	⋖	⋗	⋖
×	⋗	⋗	⋖	⋗	⋖
(	⋖	⋖	⋖	≐	⋖
)	⋗	⋗		⋗	
n	⋗	⋗		⋗	

(b)

$S \to E \mid T \mid F$
$E \to E + T \mid E + F \mid T + T \mid F + F \mid F + T \mid T + F$
$T \to T \times F \mid F \times F$
$F \to n \mid (E)$

(c)

Fig. 1. A grammar generating arithmetic expressions (a), its precedence matrix (b), its version in Fischer Normal Form (c), and an example derivation tree of the Fischer Normal Form (d).

The terminal part of the right hand side of a production is enclosed between a pair $\lessdot$, $\gtrdot$, such that the $\doteq$ holds between its adjacent terminals. For instance, with reference to Figure 1, the right hand side n is always enclosed between $\lessdot$, $\gtrdot$, in all its occurrences. Therefore a bottom-up parsing algorithm can reduce it to its left-hand side, i.e. F in this case. In a futher step, the right hand side $F + F$ is enclosed between (and), and since ($\lessdot +$ and $+ \gtrdot$) – notice that the nonterminals are "transparent" w.r.t. the parsing – it is reduced to E. Then the parsing goes on, until reaching S, and finally the syntaxt tree is built.

To model the hierarchical structure of a sentence and make it accessible without depicting the whole syntax tree, we introduce the *chain relation* $\curvearrowright$.

Let $w = (a_1 ... a_n) \in \Sigma^+$ be a non-empty word. We say $a_0 = a_{n+1} = \#$ and define a new relation $\curvearrowright$ on the set of all positions of $\#w\#$, inductively, as follows. Let $i, j \in \{0, 1, ..., n+1\}$, $i < j$. Then, we write $i \curvearrowright j$ if there exists a sequence of positions $k_1 ... k_m$, $m \geq 3$, such that $i = k_1 < ... < k_m = j$, $a_{k_1} \lessdot a_{k_2} \doteq ... \doteq a_{k_{m-1}} \gtrdot a_{k_m}$, and either $k_s + 1 = k_{s+1}$ or $k_s \curvearrowright k_{s+1}$ for each $s \in \{1, ..., m-1\}$. In particular, $i \curvearrowright j$ holds if $a_i \lessdot a_{i+1} \doteq ... \doteq a_{j-1} \gtrdot a_j$. Thus, by looking again at Figure 1, we can see that relation $i \curvearrowright j$ and the corresponding pair $(\lessdot, \gtrdot)$ embrace any subtree of the derivation tree of a string generated by a given grammar.

We say that a string w is *compatible* with the OPM M if for $\#w\#$ we have $0 \curvearrowright n+1$. In particular, this forces $M_{a_i a_j} \neq \emptyset$ for all $i + 1 = j$ and for all $i \curvearrowright j$. We denote by (Σ^+, M) the set of all non-empty words over Σ which are

compatible with M. For a *complete* OPM M, i.e. one without empty entries, this is Σ^+. In fact, for any $\#w\#$ it is possible to build a (unique) syntax tree whose frontier is w (see, e.g., [32]). An example of the $\curvearrowright$ relation will be given in Figure 11.

This new relation can be compared with the *nesting* or *matching relation* of [2], as it also is a non-crossing relation, going always forward and originating from additional information on the alphabet. However, it also features significant differences: instead of adding unary information to symbols, which partition the alphabet into three disjoint parts (calls, internals, and returns), we use the precedence relations between terminal symbols. Therefore, in contrast to the nesting relation, the same symbol can be either call or return depending on its context, and the same position can be part of multiple chain relations.

We now recall the definition of an operator precedence automaton from [32].

Definition 4. A *(nondeterministic) operator precedence automaton (OPA)* $\mathcal{A}$ over an OP alphabet (Σ, M) is a tuple $\mathcal{A} = (Q, I, F, \delta)$, where $\delta = (\delta_{\text{push}}, \delta_{\text{shift}}, \delta_{\text{pop}})$, consisting of

- Q, a finite set of states,
- $I \subseteq Q$, the set of initial states,
- $F \subseteq Q$, a set of final states, and
- the transition relations $\delta_{\text{push}}, \delta_{\text{shift}} \subseteq Q \times \Sigma \times Q$, and $\delta_{\text{pop}} \subseteq Q \times Q \times Q$.

Let $\Gamma = \Sigma \times Q$. A *configuration* of $\mathcal{A}$ is a triple $C = \langle \Pi, q, w\# \rangle$, where $\Pi \in \bot\Gamma^*$ represents a stack, $q \in Q$ the current state, and w the remaining input to read.

A *run* of $\mathcal{A}$ on $w = a_1 ... a_n$ is a finite sequence of configurations $C_0 \vdash ... \vdash C_m$ such that every transition $C_i \vdash C_{i+1}$ has one of the following forms, where a is the first component of the topmost symbol of the stack Π, or $\#$ if the stack is $\bot$, and b is the next symbol of the input to read:

$$\begin{array}{rll} \textit{push move}: & \langle \Pi, q, bx \rangle \vdash \langle \Pi[b, q], r, x \rangle & \text{if } a \lessdot b \text{ and } (q, b, r) \in \delta_{\text{push}}, \\ \textit{shift move}: & \langle \Pi[a, p], q, bx \rangle \vdash \langle \Pi[b, p], r, x \rangle & \text{if } a \doteq b \text{ and } (q, b, r) \in \delta_{\text{shift}}, \\ \textit{pop move}: & \langle \Pi[a, p], q, bx \rangle \vdash \langle \Pi, r, bx \rangle & \text{if } a \gtrdot b \text{ and } (q, p, r) \in \delta_{\text{pop}}. \end{array}$$

An *accepting run* of $\mathcal{A}$ on w is a run from $\langle \bot, q_I, w\# \rangle$ to $\langle \bot, q_F, \# \rangle$, where $q_I \in I$ and $q_F \in F$. The *language accepted by* $\mathcal{A}$, denoted $L(\mathcal{A})$, consists of all words of (Σ^+, M) over which $\mathcal{A}$ has an accepting run. We say that $L \subseteq (\Sigma^+, M)$ is an *OPL* if L is accepted by an OPA over (Σ, M). As proven in [32], the deterministic variant of an OPA, using a single initial state instead of I and transition functions instead of relations, is equally expressive to nondeterministic OPA.

An example automaton is depicted in Figure 2: with the OPM of Figure 1 (right), it accepts the same language as the grammar of Figure 1 (left).

In addition to the above description of OPL in terms of grammars and automata, the following second order logic has been defined to further characterize them [32].

Definition 5. The logic $\text{MSO}(\Sigma, M)$, short MSO, is defined as

$$\beta ::= \text{Lab}_a(x) \mid x \leq y \mid x \curvearrowright y \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta$$

where $a \in \Sigma \cup \{\#\}$, x, y are first-order variables; and X is a second order variable.

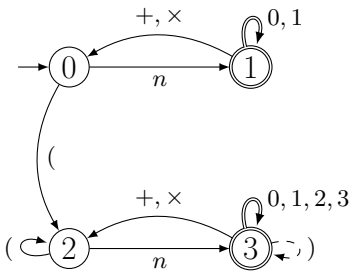


Fig. 2. Automaton for the language of the grammar of Figure 1. Shift, push and pop transitions are denoted by dashed, normal and double arrows, respectively.

We define the natural semantics for this (unweighted) logic as in [32]. The predicate $\mathrm{Lab}_a(x)$ asserts that position x is labeled a. The semantics of the formula $\curvearrowright$ is defined by the chain relation introduced above.

We use the following usual abbreviations:

$$\begin{aligned}
(\beta \wedge \varphi) &= \neg(\neg\beta \vee \neg\varphi),\\
(\beta \to \varphi) &= (\neg\beta \vee \varphi),\\
(\beta \leftrightarrow \varphi) &= (\beta \to \varphi) \wedge (\varphi \to \beta),\\
(\forall x.\varphi) &= \neg(\exists x.\neg\varphi),\\
(y = x) &= (x \le y) \wedge (y \le x),\\
(y = x + 1) &= (x \le y) \wedge \neg(y \le x) \wedge \forall z.(z \le x \vee y \le z),\\
\min(x) &= \forall y.(x \le y),\\
\max(x) &= \forall y.(y \le x)\ .
\end{aligned}$$

To summarize, the main previous results on OPL are the following:

- A language L over (Σ, M) is an OPL iff it is [32]
  - generated by an OP grammar (as defined in [32]);
  - recognized by an OPA;
  - MSO-definable.
- OPL sharing the same OPM are a boolean algebra [10] and are closed with respect to concatenation, Kleene * and other classical language operations [9].
- OPL strictly include VPL and other structured CFL [9].

3 Weighted Operator Precedence Languages

In this section, we introduce weighted extensions of operator precedence automata and their respective weighted languages and give examples showing how these weighted automata can express behaviors which were not expressible before.

As weight structures, we will employ *valuation monoids* and *semirings* as an important special case thereof. Valuation monoids are a very general weight structure that, besides covering classical semirings, are also able to model computations like average or discounting as demonstrated in the following.

Definition 6. [Droste, Meinecke [17]] A *valuation monoid* $\mathbb{D}$ is a tuple $\mathbb{D} = (D, +, \mathrm{Val}, 0)$ that consists of a commutative monoid $(D, +, 0)$ and is equipped with a *valuation function* $\mathrm{Val} : D^+ \to D$ with $\mathrm{Val}(d_1, ..., d_n) = 0$ if $d_i = 0$ for some $i \in \{1, .., n\}$.

$\mathbb{D}$ is called *commutative* if for all $n \in \mathbb{N}$ and all $d_1, ..., d_n \in D$, we have $\mathrm{Val}(d_1, .., d_n) = \mathrm{Val}(d_{\pi(1)}, .., d_{\pi(n)})$ for all permutations π.

Example 7. We set $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$.

1. We set $\mathbb{D}_1 = (\bar{\mathbb{R}}, \sup, \mathrm{avg}, -\infty)$, where
$$\mathrm{avg}(d_1, ..., d_n) = \frac{1}{n} \sum_{i=1}^{n} d_i \ .$$
2. Let $0 < \lambda < 1$ and $\bar{\mathbb{R}}_+ = \{x \in \mathbb{R} \mid x \geq 0\} \cup \{-\infty\}$. We set $\mathbb{D}_2 = (\bar{\mathbb{R}}_+, \sup, \mathrm{disc}_\lambda, -\infty)$, where
$$\mathrm{disc}_\lambda(d_1, .., d_n) = \sum_{i=1}^{n} \lambda^{i-1} d_i \ .$$

Then $\mathbb{D}_1$ and $\mathbb{D}_2$ are valuation monoids. Observe that $\mathbb{D}_1$ is commutative, while $\mathbb{D}_2$ is not commutative.

Definition 8. A *semiring* $\mathbb{K}$ is a tuple $\mathbb{K} = (K, +, \cdot, 0, 1)$ consisting of a commutative monoid $(K, +, 0)$, and a monoid $(K, \cdot, 1)$ such that $(x + y) \cdot z = x \cdot z + y \cdot z$, $x \cdot (y + z) = x \cdot y + x \cdot z$, and $0 \cdot x = x \cdot 0 = 0$ for all $x, y, z \in K$. $\mathbb{K}$ is called *commutative* if $(K, \cdot, 1)$ is commutative.

Then, every semiring is a valuation monoid where the product is employed as valuation function.

Important examples of commutative semirings cover the Boolean semiring $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$, the semiring of the natural numbers $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$, or the tropical semirings $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $\mathbb{R}_{\min} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$. Non-commutative semirings are given by $n \times n$-matrices over semirings $\mathbb{K}$ with matrix addition and multiplication as usual ($n \geq 2$), or the semiring $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ of languages over Σ.

Definition 9. A *weighted OPA (wOPA)* $\mathcal{A}$ over an OP alphabet (Σ, M) and a valuation monoid $\mathbb{D}$ is a tuple $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$, where $\mathrm{wt} = (\mathrm{wt}_{\mathrm{push}}, \mathrm{wt}_{\mathrm{shift}}, \mathrm{wt}_{\mathrm{pop}})$, consisting of

- an OPA $\mathcal{A}' = (Q, I, F, \delta)$ over (Σ, M) and
- the weight functions $\mathrm{wt}_{op} : \delta_{op} \to D$, for $op \in \{\mathrm{push}, \mathrm{shift}, \mathrm{pop}\}$.

A *restricted weighted OPA (rwOPA)* is defined similarly but without weights at pop transitions, i.e. with $\mathrm{wt} = (\mathrm{wt}_{\mathrm{push}}, \mathrm{wt}_{\mathrm{shift}})$.

A *configuration* of a wOPA or an rwOPA $\mathcal{A}$ is a tuple $C = \langle \Pi, q, w\#, \bar{d} \rangle$, where $(\Pi, q, w\#)$ is a configuration of the underlying OPA $\mathcal{A}'$ and $\bar{d}$ is a sequence of weights of D. Given a sequence $\bar{d} = (d_1, ..., d_i)$, we denote by $()$ the empty sequence and by $(\bar{d}, d)$ the sequence $(d_1, ..., d_i, d)$.

A *run* of a wOPA is a sequence of configurations $C_0 \vdash C_1 \vdash \ldots \vdash C_m$ satisfying the previous conditions and, additionally, we add every encountered weight to the sequence of weights as follows. As before, we denote with a the first component of the topmost symbol of the stack Π, or $\#$ if the stack is $\bot$, and with b the next symbol of the input to read:

$$\begin{aligned}
\langle \Pi, q, bx, \bar{d} \rangle &\vdash \langle \Pi[b, q], r, x, (\bar{d}, \mathrm{wt}_{\mathrm{push}}(q, b, r)) \rangle && \text{if } a \lessdot b \text{ and } (q, b, r) \in \delta_{\mathrm{push}}, \\
\langle \Pi[a, p], q, bx, \bar{d} \rangle &\vdash \langle \Pi[b, p], r, x, (\bar{d}, \mathrm{wt}_{\mathrm{shift}}(q, b, r)) \rangle && \text{if } a \doteq b \text{ and } (q, b, r) \in \delta_{\mathrm{shift}}, \\
\langle \Pi[a, p], q, bx, \bar{d} \rangle &\vdash \langle \Pi, r, bx, (\bar{d}, \mathrm{wt}_{\mathrm{pop}}(q, p, r)) \rangle && \text{if } a \gtrdot b \text{ and } (q, p, r) \in \delta_{\mathrm{pop}}.
\end{aligned}$$

A *run* of an rwOPA is defined analogously to a run of a wOPA but we do not have pop weights, so the third line above becomes

$$\langle \Pi[a, p], q, bx, \bar{d} \rangle \vdash \langle \Pi, r, bx, \bar{d} \rangle \text{ if } a \gtrdot b \text{ and } (q, p, r) \in \delta_{\mathrm{pop}}.$$

We call a run ρ *accepting* if it leads from $\langle \bot, q_I, w\#, () \rangle$ to $\langle \bot, q_F, \#, \bar{d} \rangle$, where $q_I \in I$ and $q_F \in F$. For such an accepting run, the *weight of* ρ is defined as

$$\mathrm{wt}(\rho) = \mathrm{Val}(\bar{d}) \ .$$

Note that in the case of a semiring, $\mathrm{wt}(\rho)$ equals the product of all encountered weights.

We denote by $\mathrm{acc}(\mathcal{A}, w)$ the set of all accepting runs of $\mathcal{A}$ on w. Finally, the *behavior of* $\mathcal{A}$ is a function $[\![\mathcal{A}]\!] : (\Sigma^+, M) \to D$, defined as

$$[\![\mathcal{A}]\!](w) = \sum_{\rho \in \mathrm{acc}(\mathcal{A}, w)} \mathrm{wt}(\rho) \ .$$

Every function $S : (\Sigma^+, M) \to D$ is called a *weighted operator precedence language* (short: *weighted language*, also *series*). A wOPA $\mathcal{A}$ *recognizes* or *accepts* a weighted language S if $[\![\mathcal{A}]\!] = S$. A weighted language S is called *recognizable* or a *wOPL* if there exists a wOPA $\mathcal{A}$ accepting it. S is *strictly recognizable* or an *rwOPL* if there exists an rwOPA $\mathcal{A}$ accepting it.

Remark 10. Since an rwOPA applies no weights at pop operations, strictly speaking rwOPA are not a special case of wOPA, due to the fact that the sequence of weights for the valuation gets shorter. This is in contrast with the semiring case, where applying no weight is equivalent to applying the neutral element of the multiplication (the '1').

Since semirings are a main instance of valuation monoids and, additionally, one could extend valuation monoids with a neutral element for the valuation function, we call rwOPL a "restricted" version of wOPL.

Example 11. Consider the source code of an HTML web page. Web pages are subject to errors, sometimes because of transmission, but often because they are written by inexperienced people. A typical error is forgetting to close a tag; for this reason new standards (e.g. HTML5) admit some of these situations. "Up to a certain point" these errors should be tolerated and should not prevent displaying the page by the browser, though not in a perfect shape. To give a precise meaning to the above still tolerable "certain point" we can define a suitable wOPA that exploits an average valuation function.

The automaton is based on the idea that deeper levels of nesting of HTML tags mark smaller and/or less relevant portions of the web page; therefore errors occurring within deeper levels of nesting should weigh less than those occurring at the external levels. Thus, we build a wOPA that counts the level of nesting up to a threshold K by means of its finite state memory: at any push corresponding to the opening of a new scope the finite memory counter is increased by 1 until level K is reached. At that point the wOPA remembers the reaching of the threshold by pushing the state q_K onto the stack and moves to a "forgetful state" which does not count anymore the nesting levels until the q_K is popped from the stack. From that point on, it restarts counting and de-counting up to value K.

During such operations the wOPA weighs possible errors in the following way: the valuation monoid is $(D, +, \mathrm{Val}, 0) = (\mathbb{R} \cup \{+\infty, -\infty\}, \sup, \mathrm{avg}_T, -\infty)$, i.e., the set of reals, augmented with $+\infty$ and $-\infty$ which is the neutral element; the valuation function is the average augmented with a threshold T defined as follows. For $d \in D^+$, we set $\mathrm{avg}_T(d) := +\infty$ if $\mathrm{avg}(d) > T$, and $\mathrm{avg}(d)$, otherwise. At start the weight is 0 which represents "no error" and therefore the best weight for the page. During the computation, if an error occurs when the automaton is at a nesting level h strictly less than K then the automaton assigns a weight $K - h$ to the transition; otherwise the weight is 1. When the automaton reaches the end of the string it computes the average of the weights and, if it is above T it produces an overall weight $+\infty$, which is considered intolerable.

Figure 3(a) shows a weighted OPA which implements the above policy in the simple case where $K = 2$. The automaton's alphabet is OT, which means *open tag*, CT, which means *closed tag*, e which means *error*; the figure adopts the same graphical notation as in Figure 2 with the addition that weights are given in brackets at transitions. The OPM is depicted in Figure 3(b). Notice that pop transitions carry no weight, thus the automaton is restricted. Also, the input is accepted only if at most one open tag remains unmatched: in such a case the automaton ends in q_0' instead of q_0.

We leave to the reader's imagination the many variations of the above basic version of the wOPA; e.g., making the weights depending on the type of the error, possibly including unrecoverable errors producing either an immediate $+\infty$ weight or just the rejection of the input. Notice also that the above automaton is deterministic, which makes the additive operation of the monoid useless; several nondeterministic versions of the automaton could be designed, e.g., to represent unpredictable weights of the errors or unpredictable reactions by the system to

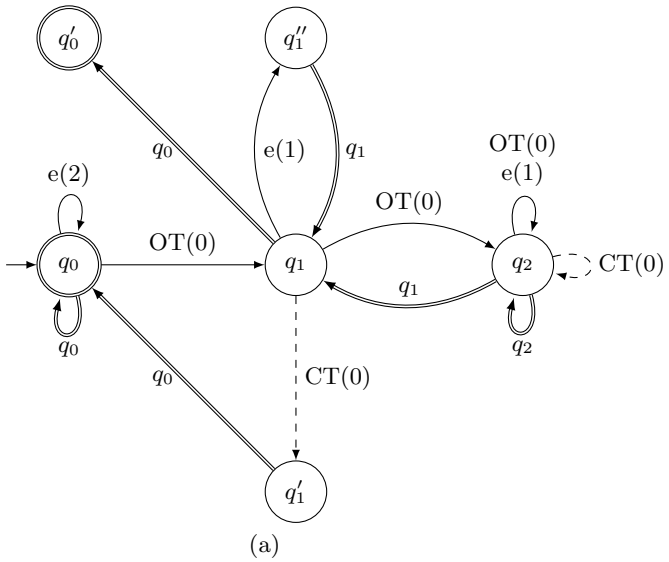


Fig. 3. A rwOPA processing tagged strings with errors.

the occurrence of some error. In such a case the sup monoid operation would formalize a worst-case evaluation among the accepting runs.

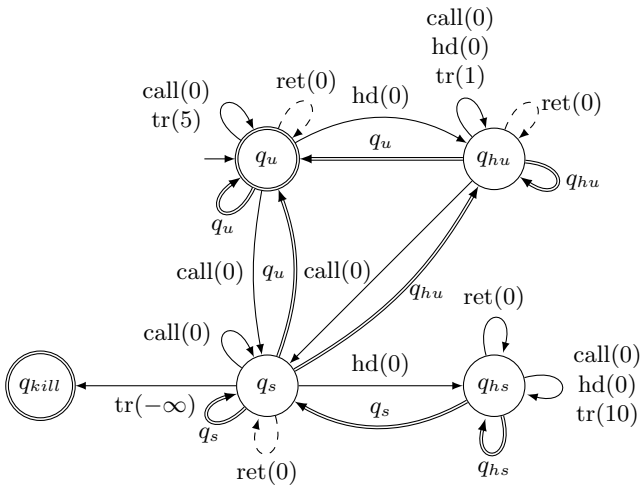
Example 12. We consider now the modeling of procedure calls with exception handling. The symbols call and ret are used to represent the calling of and the return from a procedure, respectively; hd stands for the installation of an exception *handler*, while tr is used for *throwing* exceptions. The automaton and its matrix are depicted in Figure 4: state q_u stands for the application running in *user mode*, while q_{hu} stands for *user mode with exception handler installed*; analogously, q_s stands for *supervisor mode*, and q_{hs} *supervisor mode with exception handler installed*; q_{kill} is an error state, reached when a throw is issued in supervisor mode, with weight $-\infty$: in that case the application gets killed.

Only *throws* have weights; unhandled throws have weights that are greater than handled throws; exceptions in supervisor mode have greater weights. With a valuation monoid with discounting, i.e. $(\overline{\mathbb{R}}_+, \sup, \mathrm{disc}_\lambda, -\infty)$, greater weights are associated with the first exceptions, if present, while subsequent exceptions are deemed less and less important.

Now we move from general valuation functions to the special case of semirings.

Example 13. We model a system that manages calls and returns in a traditional LIFO policy but discards all pending calls if an interrupt occurs[4]. The automaton

[4] A similar motivation inspired the extension of VPL as colored nested words by [1].



	call	ret	hd	tr
call	⋖	≐	⋖	⋗
ret	⋗	⋗	⋖	⋗
hd	⋖	⋗	⋖	⋖
tr	⋗	⋗	⋗	⋗

Fig. 4. A rwOPA modeling exception handling.

$\mathcal{A}_{\text{policy}}$ given in Figure 5 formalizes a system where the penalties for unmatched calls may change nondeterministically within intervals delimited by the special symbol \$. Precisely, the symbol \$ marks intervals during which sequences of calls, returns, and interrupts occur; "normally" unmatched calls are not penalized, but there is a special, nondeterministically chosen interval during which they are penalized; the global weight assigned to an input sequence is the maximum over all nondeterministic runs that are possible when recognizing the sequence.

Here, the alphabet is $\Sigma = \{\text{call}, \text{ret}, \text{itr}, \$\}$, and the OPM M is reported in Figure 5. As semiring, we take $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$.

Then, $[\![\mathcal{A}_{\text{policy}}]\!](w)$ equals the maximal number of pending calls between two consecutive \$.

$\mathcal{A}_{\text{policy}}$ can be easily modified/enriched to formalize several variations of its policy: e.g., different policies could be associated with different intervals, different weights could be assigned to different types of calls and/or interrupts, different policies could also be defined by choosing different semirings, etc.

Example 14. The next automaton $\mathcal{A}_{\log}$, depicted in Figure 6, chooses non-deterministically between logging everything and logging only 'important' information, e.g., only interrupts (this could be a system dependent on energy, WiFi, ...). Notice that, unlike the previous examples, in this case assigning nontrivial weights to pop transitions is crucial.

Let $\Sigma = \{\text{call}, \text{ret}, \text{itr}\}$, and define M as in Figure 5. We employ the semiring $(\text{Fin}_{\Sigma'}, \cup, \circ, \emptyset, \{\varepsilon\})$ of all finite languages over $\Sigma' = \{c, r, p, i\}$. Then, $[\![\mathcal{A}_{\log}]\!](w)$ yields all possible logs on w.

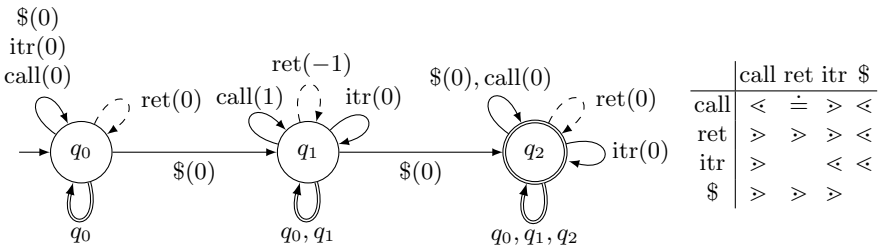


Fig. 5. The rwOPA $\mathcal{A}_{\text{policy}}$ penalizing unmatched calls.

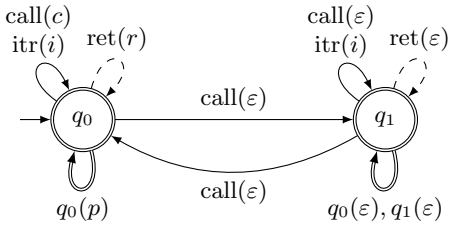


Fig. 6. The wOPA $\mathcal{A}_{\log}$ nondeterministically writes logs at different levels of detail.

As hinted at by our last example, the following two propositions show that in general, wOPA are more expressive than rwOPA.

Proposition 15. *There exists an OP alphabet (Σ, M), a semiring $\mathbb{K}$, and a weighted language $S : (\Sigma^+, M) \to K$ such that S is recognizable but not strictly recognizable.*

Proof. Let $\Sigma = \{c, r\}$, $c \lessdot c$, and $c \doteq r$. Consider the semiring $\text{Fin}_{\{a,b\}}$ of all finite languages over $\{a, b\}$ together with union and concatenation. Let $S : (\Sigma^+, M) \to \text{Fin}_{\{a,b\}}$ be the following weighted language

$$S(w) = \begin{cases} \{a^n b a^n\} \text{ , if } w = c^n r & \text{for some } n \in \mathbb{N} \\ \emptyset \text{ , otherwise} \end{cases} .$$

Then, we can define a wOPA which only reads $c^n r$, assigns the weight $\{a\}$ to every push and pop, and the weight $\{b\}$ to the one shift, and therefore accepts S, as in Figure 7.

Now, we show with a pumping argument that there exists no rwOPA which recognizes S. Assume there is an rwOPA $\mathcal{A}$ with $[\![\mathcal{A}]\!] = S$. Note that for all $n \in \mathbb{N}$, the structure of $c^n r$ is fixed as $c \lessdot c \lessdot ... \lessdot c \doteq r$. Let ρ be an accepting run of $\mathcal{A}$ on $c^n r$ with $\text{wt}(\rho) = \{a^n b a^n\}$. Then, the transitions of ρ consist of n pushes, followed by a shift, followed by n pops and can be written as

$$q_0 \xrightarrow{c} q_1 \xrightarrow{c} ... \xrightarrow{c} q_{n-1} \xrightarrow{c} q_n \dashrightarrow^{r} q_{n+1} \overset{q_{n-1}}{\Longrightarrow} q_{n+2} \overset{q_{n-2}}{\Longrightarrow} ... \overset{q_1}{\Longrightarrow} q_{2n} \overset{q_0}{\Longrightarrow} q_{2n+1} .$$

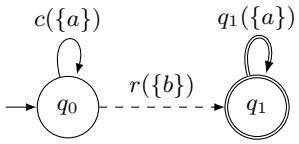


Fig. 7. The wOPA recognizing S with $S(w) = \{a^n b a^n\}$, if $w = c^n r$, $n \in \mathbb{N}$, and $S(w) = 0$, otherwise.

Both the number of states and the amount of pairs of states are bounded. If n is sufficiently large, there exist two pop transitions $\mathrm{pop}(q, p, r)$ and $\mathrm{pop}(q', p', r')$ in this sequence such that $q = q'$ and $p = p'$. This means that we have a loop in the pop transitions going from state q to $q' = q$. Furthermore, the corresponding push to the first transition of this loop was invoked when the automaton was in state p', while the corresponding push to the last pop was invoked in state p. Since $p = p'$, we also have a loop at the corresponding pushes. Then, the run where we skip both loops in the pops and in the pushes is an accepting run for $c^{n-k}r$, for some $k \in \mathbb{N} \setminus \{0\}$.

Since the weight of all pops is trivial, the weight of the pop-loop is ε. If the weight of the push-loop is also ε, then we have an accepting run for $c^{n-k}r$ of weight $\{a^n b a^n\}$, a contradiction. If the weight of the push-loop is not trivial, then by a simple case distinction it has to be either $\{a^i\}$ for some $i \in \mathbb{N} \setminus \{0\}$ or it has to contain the b. In the first case, the run without both loops has weight $\{a^{n-i} b a^n\}$ or $\{a^n b a^{n-i}\}$, in the second case it has weight $\{a^j\}$, for some $j \in \mathbb{N}$. The weights of all these runs are not of the form $a^{n-k} b a^{n-k}$, a contradiction. □

Proposition 15 uses a semiring as a special instance of a valuation monoid to show that in general and for a fixed weight structure, wOPA are more expressive than rwOPA. However, the proof relies on the non-commutativity of the semiring $\mathrm{Fin}_{\{a,b\}}$. On the other hand, in Section 8, we will show that for commutative semirings, rwOPA are equally expressive as wOPA. Notably, the same is not true for commutative valuation monoids, i.e. valuation monoids that do not depend on the order of the input, as the following result shows.

Proposition 16. *There exists an OP alphabet (Σ, M), a commutative valuation monoid $\mathbb{D}$, and a weighted language $S : (\Sigma^+, M) \to K$ such that S is recognizable but not strictly recognizable.*

Proof. Let $\mathbb{D} = (\mathbb{N}, \sup, \mathrm{Val}, -\infty)$, with $\mathrm{Val}(d_1, .., d_n) = mz$, where m is the minimum of d_1 to d_n and z is the number of occurrences of m in d_1 to d_n. This valuation function is based upon an idea of Andreas Maletti.

Let $\Sigma = \{c, r\}$, $c \lessdot c$, and $c \doteq r$. Let $S : (\Sigma^+, M) \to \mathbb{N}$ be the following weighted language

$$S(w) = \begin{cases} 2n+1 \ , & \text{if } w = c^n r \quad \text{for some } n \in \mathbb{N} \\ 0 \ , & \text{otherwise} \end{cases} .$$

We can define a wOPA which only reads $c^n r$, assigns the weight $\{1\}$ to every push, shift, or pop, and therefore accepts S, as in Figure 8. Note that every run of an wOPA on $c^n r$ consists of n pushes, one shift and n pops.

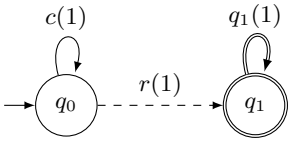


Fig. 8. The wOPA recognizing S with $S(w) = 2n+1$, if $w = c^n r$, $n \in \mathbb{N}$, and $S(w) = 0$, otherwise.

Next, we show that there exists no rwOPA which recognizes S. Let $\mathcal{A}$ be an rwOPA with $[\![\mathcal{A}]\!] = S$, in particular $[\![\mathcal{A}]\!](c^n r) = 2n+1$. Since we take the supremum of weights of all runs, at least one run ρ for $c^n r$ has to yield the weight $2n+1$ and no run should yield a larger weight. Also, note that every run of $\mathcal{A}$ on $c^n r$ uses $n+1$ weights. We denote the weights of ρ by d_{ρ_1} to $d_{\rho_{n+1}}$.

Let $W_{\max}$ be the maximal weight that occurs in $\mathcal{A}$ and consider $n = W_{\max}!$. Then, by the definition of $\mathbb{D}$, we have

$$2n+1 = [\![\mathcal{A}]\!](c^n r) = \mathrm{Val}(d_{\rho_1}, ..., d_{\rho_n}, d_{\rho_{n+1}})$$

By the definition of Val, we know that

$$2n+1 = \mathrm{Val}(d_{\rho_1}, ..., d_{\rho_n}, d_{\rho_{n+1}}) = m_\rho \cdot z_\rho$$

where, m_ρ and z_ρ are as above. Since $z_\rho \leq n+1$, we get $m_\rho \geq 2$. Furthermore, since $m_\rho \leq W_{\max}$, we know that $m_\rho \,|\, n$ due to the choice of n. But then m_ρ cannot divide $2n+1$, a contradiction. □

We note that in general, we assume a fixed weight structure. If on the other hand, we are allowed to modify the valuation monoid, it is possible to encode parts of the automaton in the valuation monoid. This leads to the fact that given a wOPA $\mathcal{A}$ over a valuation monoid $\mathbb{D}$, we can construct a valuation monoid $\mathbb{D}'$ such that there exists a rwOPA over $\mathbb{D}'$ with the same behavior as $\mathcal{A}$.

In the following section, we will study the connection between weighted nested word languages (weighted VPL) and weighted OPL.

4 Weighted OPL strictly include Weighted VPL

In this section, we show that restricted weighted OPL are a generalization of weighted *visibly pushdown languages*. We shortly recall the important definitions.

Let in the following Σ_{call}, Σ_{int}, Σ_{ret} be three disjoint alphabets, and $\hat{\Sigma} = \Sigma_{\mathrm{call}} \cup \Sigma_{\mathrm{int}} \cup \Sigma_{\mathrm{ret}}$ be a *visibly pushdown alphabet*. A *visibly pushdown automaton*

(VPA) over $\hat{\Sigma}$ is a pushdown automaton which pushes or pops exactly one symbol whenever it reads a call or return symbol, respectively, or uses no operation on the stack, otherwise. In [2], words over $\hat{\Sigma}$ are interpreted as *nested words* and their automata models are *nested word automata (NWA)*.

In [9], it was shown that for every VPA, there exists an equivalent operator precedence grammar which in turn can be transformed into an equivalent OPA. This proof uses the complete OPM of Figure 9.

	Σ_{call}	Σ_{ret}	Σ_{int}
Σ_{call}	$\lessdot$	$\doteq$	$\lessdot$
Σ_{ret}	$\gtrdot$	$\gtrdot$	$\gtrdot$
Σ_{int}	$\gtrdot$	$\gtrdot$	$\gtrdot$

Fig. 9. OPM for VPL

In [33] and [19] weighted extensions of NWA were introduced. These add semiring weights at every transition again depending on the information what symbols are calls, internals, or returns. In [11], we find a respective extension to infinite nested words that is using weights from a valuation monoid. We can apply these concepts in our setting as follows.

We say a *weighted nested word automaton (wNWA)* over $\hat{\Sigma}$ and over a valuation monoid $\mathbb{D}$ is an NWA that at every transition applies a weight of $\mathbb{D}$. Then, the weight of a run is computed as previously by the valuation function of $\mathbb{D}$. Finally, the behavior of a wNWA is a function assigning to every non-empty word over $\hat{\Sigma}$ the sum over the weights of all accepting runs of this word. We denote by *wVPL* the class of all such behaviors.

Note that every non-empty nested word has a representation as a word over a visibly pushdown alphabet $\hat{\Sigma}$ and can be interpreted as a compatible word of (Σ^+, M), where M is the OPM of Figure 9. Therefore, we can interpret the behavior of a wNWA over $\hat{\Sigma}$ as a weighted language $(\Sigma^+, M) \to \mathbb{D}$.

Theorem 17. *Let $\mathbb{D}$ be a valuation monoid, $\hat{\Sigma}$ be a visibly pushdown alphabet, and M be the OPM of Figure 9. Then for every wNWA $\mathcal{A}$, there exists an rwOPA $\mathcal{B}$ with $[\![\mathcal{A}]\!](w) = [\![\mathcal{B}]\!](w)$ for all $w \in (\Sigma^+, M)$.*

We give an intuition for this result as follows. Note that although sharing some similarities, pushes, shifts, and pops for OPA are not the same as calls, internals, and returns for NWA. Indeed, a return forces a pop of the NWA that also 'consumes' the current symbol, while a pop of an OPA just pops the stack and leaves the current symbol untouched. This observation remains true for (r)wOPA and wNWA.

After studying Figure 9, this leads to the important observation that every symbol of Σ_{ret} and therefore every pop transition of a NWA is simulated not by a pop, but by a shift transition of an OPA followed by a pop.

We give a short demonstrating example: Let $\Sigma_{\text{int}} = \{a\}$, $\Sigma_{\text{call}} = \{\langle c\}$, $\Sigma_{\text{ret}} = \{r\rangle\}$, $w = a\langle car\rangle$. Then every run of an NWA for this word looks like this

$$q_0 \xrightarrow{\quad a \quad} q_1 \xrightarrow{\langle c} q_2 \xrightarrow{\quad a \quad} q_3 \xrightarrow{\quad r\rangle \quad} q_4 \ .$$

Every run of an OPA (using the OPM of Figure 9) looks as follows:

$$q_0 \xrightarrow{a} q_1' \Rightarrow q_1 \xrightarrow{\langle c} q_2 \xrightarrow{a} q_3' \Rightarrow q_3 \overset{r\rangle}{\dashrightarrow} q_4' \Rightarrow q_4 \ ,$$

where the return was substituted (by the OPM, not by a choice of ours) by a shift followed by a pop.

It follows that we can simulate a weighted call by a weighted push, a weighted internal by a weighted push together with a pop, and a weighted return by a weighted shift together with a pop. Therefore, we may indeed omit weights at pop transitions.

Proof (of Theorem 17). Let $\mathcal{A} = (Q, I, F, (\delta_{\text{call}}, \delta_{\text{int}}, \delta_{\text{ret}}), (\text{wt}_{\text{call}}, \text{wt}_{\text{int}}, \text{wt}_{\text{ret}}))$ be a wNWA over $\hat{\Sigma}$ and a valuation monoid $\mathbb{D}$. We construct an rwOPA $\mathcal{B} = (Q', I', F', (\delta_{\text{push}}, \delta_{\text{shift}}, \delta_{\text{pop}}), (\text{wt}'_{\text{push}}, \text{wt}'_{\text{shift}})$ over (Σ, M) and $\mathbb{D}$. We set $Q' = Q \cup (Q \times Q)$, $I' = I$, and $F' = F$. We define the relations δ_{push}, δ_{shift}, δ_{pop}, and the functions wt'_{push} and $\text{wt}'_{\text{shift}}$ as follows.

We let δ_{push} contain all triples (q, a, r) with $(q, a, r) \in \delta_{\text{call}}$, and all triples $(q, a, (q, r))$ with $(q, a, r) \in \delta_{\text{int}}$. We set $\text{wt}'_{\text{push}}(q, a, r) = \text{wt}_{\text{call}}(q, a, r)$ and $\text{wt}'_{\text{push}}(q, a, (q, r)) = \text{wt}_{\text{int}}(q, a, r)$. Moreover, we let δ_{shift} contain all triples $(q, a, (p, r))$ with $(q, p, a, r) \in \delta_{\text{ret}}$ and set $\text{wt}'_{\text{shift}}(q, a, (p, r)) = \text{wt}_{\text{ret}}(q, p, a, r)$. Furthermore, we let δ_{pop} contain all triples $((q, r), q, r)$ with $(q, a, r) \in \delta_{\text{int}}$, and all triples $((p, r), p, r)$ with $(q, p, a, r) \in \delta_{\text{ret}}$.

Then, a run analysis of $\mathcal{A}$ and $\mathcal{B}$ shows that $[\![\mathcal{B}]\!] = [\![\mathcal{A}]\!]$. □

Together with the result that OPA are strictly more expressive than VPAs [9], this shows that

$$\text{wVPL} \subsetneq \text{rwOPL} \ .$$

In the semiring case, we get a complete picture of the expressive power of these three classes of weighted languages:

$$\text{wVPL} \subsetneq \text{rwOPL} \subsetneq \text{wOPL} \ .$$

We also note that in the context of formal power series for semirings, wVPL strictly contain recognizable power series and wOPL form a proper subset of the class of algebraic power series, i.e., series recognized by weighted pushdown automata [30].

5 Closure Properties and a Nivat Theorem

In this section, we study closure properties of weighted OPA and restricted weighted OPA. Then, we establish a connection between strictly recognizable weighted languages and unweighted languages. We show that rwOPL are exactly those weighted languages which can be derived from a restricted weighted OPA with only one state, intersected with an unweighted OPL, and using an OPM-preserving projection of the alphabet.

As usual, we define the sum $S+T$ of two weighted languages $S, T : (\Sigma^+, M) \to D$ by means of a pointwise definition as follows:

$$(S+T)(w) = S(w) + T(w) \text{ for each } w \in (\Sigma^+, M) \ .$$

Furthermore, for a weighted language $S : (\Sigma^+, M) \to D$ and an OPL $L \subseteq (\Sigma^+, M)$, we define a weighted language $S \cap L : (\Sigma^+, M) \to D$ by

$$(S \cap L)(w) = \begin{cases} S(w) \ , & \text{if } w \in L \\ 0 \ , & \text{otherwise} \end{cases} \ .$$

Proposition 18. *The sum of two recognizable (resp. strictly recognizable) weighted languages over (Σ, M) is again recognizable (resp. strictly recognizable).*

Proof. We use a standard disjoint union of two (r)wOPA accepting the given weighted language to obtain a (r)wOPA for the sum as follows.

Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ and $\mathcal{B} = (Q', I', F', \delta', \mathrm{wt}')$ be two wOPA over (Σ, M) and $\mathbb{D}$, and assume without loss of generality that Q and Q' are disjoint. We construct a wOPA $\mathcal{C} = (Q'', I'', F'', \delta'', \mathrm{wt}'')$ over (Σ, M) and $\mathbb{D}$ by defining $Q'' = Q \cup Q'$, $I'' = I \cup I'$, $F'' = F \cup F'$, $\delta'' = \delta \cup \delta'$. The weight function is defined by

$$\mathrm{wt}''(t) = \begin{cases} \mathrm{wt}(t) \ , & \text{if } t \in \delta \\ \mathrm{wt}'(t) \ , & \text{if } t \in \delta' \end{cases} \ .$$

Note that in the case of rwOPA, the weight functions wt, wt′, and wt″ are defined only for pushes and shifts. Then, $[\![\mathcal{C}]\!] = [\![\mathcal{A}]\!] + [\![\mathcal{B}]\!]$. Furthermore, if $\mathcal{A}$ and $\mathcal{B}$ are restricted, then so is $\mathcal{C}$. □

Proposition 19. *Let $S : (\Sigma^+, M) \to D$ be a recognizable (resp. strictly recognizable) weighted language and $L \subseteq (\Sigma^+, M)$ an OPL. Then, the weighted language $S \cap L$ is recognizable (resp. strictly recognizable).*

Proof. We use a product construction of automata.

Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ be a wOPA over (Σ, M) and $\mathbb{D}$ with $[\![\mathcal{A}]\!] = S$ and let $\mathcal{B} = (Q', q_0', F', \delta')$ be a deterministic OPA over (Σ, M) with $L(\mathcal{B}) = L$. We construct a wOPA $\mathcal{C} = (Q'', I'', F'', \delta'', \mathrm{wt}'')$ over (Σ, M) and $\mathbb{D}$, with $[\![\mathcal{C}]\!] = S \cap L$, as follows. We let $Q'' = Q \times Q'$, $I'' = I \times \{q_0'\}$, $F'' = F \times F'$, and

$$\begin{aligned}
\delta''_{\mathrm{push}} &= \{((q,q'),a,(r,r')) \mid (q,a,r) \in \delta_{\mathrm{push}} \text{ and } \delta'_{\mathrm{push}}(q',a) = r'\} \ , \\
\delta''_{\mathrm{shift}} &= \{((q,q'),a,(r,r')) \mid (q,a,r) \in \delta_{\mathrm{shift}} \text{ and } \delta'_{\mathrm{shift}}(q',a) = r'\} \ , \\
\delta''_{\mathrm{pop}} &= \{((q,q'),(p,p'),(r,r')) \mid (q,p,r) \in \delta_{\mathrm{pop}} \text{ and } \delta'_{\mathrm{pop}}(q',p') = r'\} \ .
\end{aligned}$$

Then, we define the weights of $\mathcal{C}$ by letting

$$\begin{aligned}
\mathrm{wt}''_{\mathrm{push}}((q,q'),a,(r,r')) &= \mathrm{wt}_{\mathrm{push}}(q,a,r) \ , \\
\mathrm{wt}''_{\mathrm{shift}}((q,q'),a,(r,r')) &= \mathrm{wt}_{\mathrm{shift}}(q,a,r) \ , \\
\mathrm{wt}''_{\mathrm{pop}}((q,q'),(p,p'),(r,r')) &= \mathrm{wt}_{\mathrm{pop}}(q,p,r) \ .
\end{aligned}$$

Let ρ be a run of $\mathcal{C}$. Then, we refer with $\rho_{\restriction Q}$ (resp. $\rho_{\restriction Q'}$) to the run of $\mathcal{A}$ (resp. $\mathcal{B}$) that can be obtained from ρ by projecting each state occurrence to its first (resp. second) component.

Note that for given a word w, the automata $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ have to use pushes, shifts, and pops at the same positions. Hence, every accepting run ρ of $\mathcal{C}$ on w defines exactly one accepting run $\rho_{\restriction Q}$ of $\mathcal{A}$ and exactly one accepting run $\rho_{\restriction Q'}$ of $\mathcal{B}$ on w with matching weights, and vice versa. We obtain

$$\begin{aligned}
[\![\mathcal{C}]\!](w) &= \sum_{\rho \in \mathrm{acc}(\mathcal{C},w)} \mathrm{wt}''(\rho) \\
&= \sum_{\substack{\rho, \text{ such that} \\ \rho_{\restriction Q} \in \mathrm{acc}(\mathcal{A},w) \\ \rho_{\restriction Q'} \in \mathrm{acc}(\mathcal{B},w)}} \mathrm{wt}''(\rho) \\
&= \begin{cases} \sum_{\rho \in \mathrm{acc}(\mathcal{A},w)} \mathrm{wt}(\rho) \ , & \text{if the run of } \mathcal{B} \text{ on } w \text{ is accepting} \\ 0 \ , & \text{otherwise} \end{cases} \\
&= (S \cap L)(w) \ .
\end{aligned}$$

Hence, $[\![\mathcal{C}]\!] = S \cap L$. Furthermore, if $\mathcal{A}$ and $\mathcal{B}$ are restricted, then by using the same construction without $\mathrm{wt}''_{\mathrm{pop}}$, we get a restricted wOPA $\mathcal{C}$ with $[\![\mathcal{C}]\!] = S \cap L$. □

Next, we show that recognizable weighted languages are closed under projections which preserve the OPM. For two OP alphabets (Σ, M), (Γ, M') and a mapping $h : \Sigma \to \Gamma$, we write $h : (\Sigma, M) \to (\Gamma, M')$ and say h is *OPM-preserving* if for all $\bullet \in \{\lessdot, \doteq, \gtrdot\}$, we have $a \bullet b$ if and only if $h(a) \bullet h(b)$. We can extend such an h to a function $h : (\Sigma^+, M) \to (\Gamma^+, M')$ as follows. Given a word $w = (a_1 a_2 ... a_n) \in (\Sigma^+, M)$, we define $h(w) = h(a_1 a_2 ... a_n) = h(a_1)h(a_2)...h(a_n)$.

Let $\mathbb{D}$ be a valuation monoid and $S : (\Sigma^+, M) \to D$ be a weighted language. Then, we define $h(S) : (\Gamma^+, M') \to D$ for each $v \in (\Gamma^+, M')$ by

$$h(S)(v) = \sum_{\substack{w \in (\Sigma^+, M) \\ h(w) = v}} S(w) \ . \tag{1}$$

Proposition 20. *Let $\mathbb{D}$ be a valuation monoid, $S : (\Sigma^+, M) \to D$ a recognizable (resp. strictly recognizable) weighted language, and $h : \Sigma \to \Gamma$ an OPM-preserving projection. Then, $h(S) : (\Gamma^+, M') \to D$ is recognizable (resp. strictly recognizable).*

Proof. We follow an idea of [21] and its application in [19] and [11]. Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ be a wOPA over (Σ, M) and $\mathbb{D}$ with $[\![\mathcal{A}]\!] = S$. We construct the wOPA $\mathcal{B} = (Q', I', F', \delta', \mathrm{wt}')$ over (Γ, M') and $\mathbb{D}$ as follows.

The main idea is that $\mathcal{B}$ simulates all runs of $\mathcal{A}$ by always remembering the last symbol that was read in a run of $\mathcal{A}$. We set $Q' = Q \times \Sigma$, $I' = I \times \{a_0\}$ for some fixed $a_0 \in \Sigma$, and $F' = F \times \Sigma$. We define the transition relations $\delta' = (\delta'_{\mathrm{push}}, \delta'_{\mathrm{shift}}, \delta'_{\mathrm{pop}})$ as

$$\begin{aligned}
\delta'_{\mathrm{push}} &= \{((q,a), b, (q',a')) \mid (q,a',q') \in \delta_{\mathrm{push}} \text{ and } b = h(a')\} \ , \\
\delta'_{\mathrm{shift}} &= \{((q,a), b, (q',a')) \mid (q,a',q') \in \delta_{\mathrm{shift}} \text{ and } b = h(a')\} \ , \\
\delta'_{\mathrm{pop}} &= \{((q,a), (q',a'), (q'',a)) \mid (q,q',q'') \in \delta_{\mathrm{pop}}\} \ .
\end{aligned}$$

Then, the weight functions are defined by

$$\begin{aligned}
\mathrm{wt}'_{\mathrm{push}}((q,a), h(a'), (q',a')) &= \mathrm{wt}_{\mathrm{push}}(q,a',q') \ , \\
\mathrm{wt}'_{\mathrm{shift}}((q,a), h(a'), (q',a')) &= \mathrm{wt}_{\mathrm{shift}}(q,a',q') \ , \\
\mathrm{wt}'_{\mathrm{pop}}((q,a), (q',a'), (q'',a'')) &= \mathrm{wt}_{\mathrm{pop}}(q,q',q'') \ .
\end{aligned}$$

Analogously to [19] and [11], this implies that for every run ρ of $\mathcal{A}$ on w, there exists exactly one run ρ' of $\mathcal{B}$ on v with $h(w) = v$ and $\mathrm{wt}(\rho) = \mathrm{wt}'(\rho')$. One difference to previous works is that a pop of a wOPA is not consuming the symbol. Therefore, we have to make sure to not change the symbol that we are currently remembering while processing a pop.

It follows that $[\![\mathcal{B}]\!](v) = h([\![\mathcal{A}]\!])(v)$, so $h(S) = [\![\mathcal{B}]\!]$ is recognizable. Furthermore, if $\mathcal{A}$ is restricted, then so is $\mathcal{B}$. □

Let $h : \Sigma' \to \Sigma$ be a map between two alphabets. Given an OP alphabet (Σ, M), we define $h^{-1}(M)$ by setting $h^{-1}(M)_{a'b'} = M_{h(a')h(b')}$ for all $a', b' \in \Sigma'$. Then $h : (\Sigma', h^{-1}(M)) \to (\Sigma, M)$ is OPM-preserving and for every weighted language $S : (\Sigma'^{+}, h^{-1}(M)) \to D$, we get a weighted language $h(S) : (\Sigma^{+}, M) \to D$, as above.

Let $\mathcal{N}(\Sigma, M, \mathbb{D})$ comprise all weighted languages $S : (\Sigma^{+}, M) \to D$ for which there exist an alphabet Σ', a map $h : \Sigma' \to \Sigma$, a one-state rwOPA $\mathcal{B}$ over $(\Sigma', h^{-1}(M))$ and $\mathbb{D}$, and an OPL L over $(\Sigma', h^{-1}(M))$ such that $S = h([\![\mathcal{B}]\!] \cap L)$.

Now, we show that every strictly recognizable weighted language can be decomposed into the fragments introduced above.

Proposition 21. *Let $S : (\Sigma^{+}, M) \to D$ be a weighted language. If S is strictly recognizable, then S is in $\mathcal{N}(\Sigma, M, \mathbb{D})$.*

Proof. We follow some ideas of [16] and [18].

Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ be a rwOPA over (Σ, M) and $\mathbb{D}$ with $[\![\mathcal{A}]\!] = S$. We set $\Sigma' = Q \times \Sigma \times Q$ as the extended alphabet. The intuition is that Σ' consists of the push and the shift transitions of $\mathcal{A}$. Let h be the projection of Σ' to Σ and let $M' = h^{-1}(M)$.

Let $L \subseteq (\Sigma'^{+}, M')$ be the language consisting of all words w' over the extended alphabet such that $h(w')$ has an accepting run of $\mathcal{A}$ which uses at every position the push, resp. the shift transition defined by the symbol of Σ' at this position.

We construct the unweighted OPA $\mathcal{A}' = (Q', I', F', \delta')$ over (Σ', M'), accepting L, as follows. We set $Q' = Q$, $I' = I$, $F' = F$, and define δ' as follows

$$\begin{aligned}
\delta'_{\text{push}} &= \{\, (q, (q, a, p), p) \mid (q, a, p) \in \delta_{\text{push}} \,\} \ , \\
\delta'_{\text{shift}} &= \{\, (q, (q, a, p), p) \mid (q, a, p) \in \delta_{\text{shift}} \,\} \ , \\
\delta'_{\text{pop}} &= \delta_{\text{pop}} \ .
\end{aligned}$$

Hence, $\mathcal{A}'$ has an accepting run on a word $w' \in (\Sigma'^{+}, M')$ if and only if $\mathcal{A}$ has an accepting run on $h(w')$, using the push and shift transitions defined by w'.

We construct the one-state rwOPA $\mathcal{B} = (Q'', I'', F'', \delta'', \mathrm{wt}'')$ over (Σ', M') and $\mathbb{D}$ as follows. We set $Q'' = I'' = F'' = \{q\}$, $\delta''_{\text{push}} = \delta''_{\text{shift}} = \{(q, a', q) \mid a' \in \Sigma'\}$, and $\delta''_{\text{pop}} = \{(q, q, q)\}$. We define the weight function wt'' for $a' \in \Sigma'$ as follows

$$\begin{aligned}
\mathrm{wt}''_{\text{push}}(q, a', q) &= \mathrm{wt}_{\text{push}}(a') \\
\mathrm{wt}''_{\text{shift}}(q, a', q) &= \mathrm{wt}_{\text{shift}}(a')
\end{aligned}$$

Now, let $w = a_1 ... a_n \in (\Sigma^{+}, M)$, let ρ be a run of $\mathcal{A}$ on w, and let $w' = h(w) \in (\Sigma'^{+}, M')$. Then, the rwOPA $\mathcal{B}$ has exactly one run on w'. In the following, we denote this run with $\rho_{w'}$.

We denote by $\mathrm{wt}_{\mathcal{A}}(\rho, w, i)$, resp. $\mathrm{wt}_{\mathcal{B}}(\rho_{w'}, w', i)$, the weight of the push or shift transition used by the run ρ, resp. $\rho_{w'}$, at position i. Since $\mathcal{A}$ and $\mathcal{B}$ are restricted, we have

$$\begin{aligned}
\mathrm{wt}(\rho) &= \mathrm{Val}((\mathrm{wt}_{\mathcal{A}}(\rho, w, i))_{i=1,\dots,|w|}) \\
\mathrm{wt}''(\rho_{w'}) &= \mathrm{Val}((\mathrm{wt}_{\mathcal{B}}(\rho_{w'}, w', i))_{i=1,\dots,|w|}) \ .
\end{aligned}$$

Furthermore, following the definition of $\mathcal{B}$, for all $h(w') = w$ and for all $i \in \{1, ..., n\}$, we have $\mathrm{wt}_{\mathcal{B}}(\rho_{w'}, w', i) = \mathrm{wt}_{\mathcal{A}}(\rho, w, i)$. It follows that

$$\begin{aligned}
h([\![\mathcal{B}]\!] \cap L)(w) &= \sum_{\substack{w' \in (\Sigma'^{+}, M') \\ h(w')=w}} ([\![\mathcal{B}]\!] \cap L)(w') \\
&= \sum_{\substack{w' \in L(\mathcal{A}') \\ h(w')=w}} [\![\mathcal{B}]\!](w') \\
&= \sum_{\substack{w' \in L(\mathcal{A}') \\ h(w')=w}} \mathrm{wt}''(\rho_{w'}) \\
&= \sum_{\substack{w' \in L(\mathcal{A}') \\ h(w')=w}} \mathrm{Val}((\mathrm{wt}_{\mathcal{B}}(\rho_{w'}, w', i))_{i=1,...,|w|}) \\
&= \sum_{\rho \in \mathrm{acc}(\mathcal{A}, w)} \mathrm{Val}((\mathrm{wt}_{\mathcal{A}}(\rho, w, i))_{i=1,...,|w|}) \\
&= \sum_{\rho \in \mathrm{acc}(\mathcal{A}, w)} \mathrm{wt}(\rho) \\
&= [\![\mathcal{A}]\!](w) = S(w) \ .
\end{aligned}$$

Hence, $S = h([\![\mathcal{B}]\!] \cap L)$, thus $S \in \mathcal{N}(\Sigma, M, \mathbb{D})$. □

Using this proposition and the above closure properties of recognizable weighted languages, we get the following Nivat-Theorem for weighted operator precedence automata.

Theorem 22. *Let $\mathbb{D}$ be a valuation monoid and $S : (\Sigma^+, M) \to D$ be a weighted language. Then S is strictly recognizable if and only if $S \in \mathcal{N}(\Sigma, M, \mathbb{D})$.*

Proof. The "only if"-part is immediate by Proposition 21.

For the converse, let Σ' be an alphabet, $h : \Sigma' \to \Sigma$, $L \subseteq (\Sigma'^{+}, h^{-1}(M))$ be an OPL, $\mathcal{B}$ a one-state rwOPA, and $S = h([\![\mathcal{B}]\!] \cap L)$. Then, Proposition 19 shows that $[\![\mathcal{B}]\!] \cap L$ is strictly recognizable. Now, Proposition 20 yields the result. □

6 Weighted MSO-Logic for OPL

In this section we will make use of the following theorem from [32].

Theorem 23. *A language L over (Σ, M) is an OPL iff L is MSO-definable.*

In the definition of our weighted MSO-logic, we will use modified ideas from Droste and Gastin [13], also incorporating the distinction between an unweighted (boolean) and a weighted part by Bollig and Gastin [5], and the introduction of a weighted "if-then-else" operator by Gastin and Monmege [27].

Definition 24. Given a valuation monoid $\mathbb{D}$, we define the weighted logic $\mathrm{MSO}(\mathbb{D}, (\Sigma, M))$, short $\mathrm{MSO}(\mathbb{D})$, as

$$\varphi ::= d \mid \varphi \oplus \varphi \mid \beta \,?\, \varphi : \varphi \mid \bigoplus_x \varphi \mid \bigoplus_X \varphi \mid \mathrm{Val}_x\, \varphi$$

where $d \in \mathbb{D}$, $\beta \in \mathrm{MSO}$, x, y are first-order variables; and X is a second order variable.

We call β boolean and φ weighted formulas. Let $w \in (\Sigma^+, M)$ and $\varphi \in \mathrm{MSO}(\mathbb{D})$. Following classical approaches for logics, we denote by $[w] = \{1, ..., |w|\}$ the set of all positions of w. Let $\mathrm{free}(\varphi)$ be the set of all free variables in φ, and let $\mathcal{V}$ be a finite set of variables containing $\mathrm{free}(\varphi)$. A $(\mathcal{V}, w)$-*assignment* σ is a function assigning to every first-order variable of $\mathcal{V}$ an element of $[w]$ and to every second order variable a subset of $[w]$. We define $\sigma[x \to i]$ as the $(\mathcal{V} \cup \{x\}, w)$-assignment mapping x to i and coinciding with σ on all variables different from x. The assignment $\sigma[X \to I]$ is defined analogously.

Consider the extended alphabet $\Sigma_\mathcal{V} = \Sigma \times \{0,1\}^\mathcal{V}$ together with its natural OPM $M_\mathcal{V}$ defined such that for all $(a, s), (b, t) \in \Sigma_\mathcal{V}$ and all $\bullet \in \{\lessdot, \doteq, \gtrdot\}$, we have $(a, s) \bullet (b, t)$ if and only if $a \bullet b$.

We identify a pair (w, σ) consisting of a word w and an assignment σ with a word (w, σ) over $(\Sigma_\mathcal{V}, M_\mathcal{V})$ as follows. For every position $p \in [w]$, the second component of the symbol of the word (w, σ) at position p is the vector that carries a 1 at the entry corresponding to variable x (resp. X) if $\sigma(x) = p$ (resp. $p \in \sigma(X)$).

A word over $\Sigma_\mathcal{V}$ is called *valid*, if every first-order variable is assigned to exactly one position. Being valid is a recognizable property which can be checked by an OPA.

We define the *semantics* of $\varphi \in \mathrm{MSO}(\mathbb{D})$ as a function $[\![\varphi]\!]_\mathcal{V} : (\Sigma_\mathcal{V}^+, M_\mathcal{V}) \to D$ inductively for all valid $(w, \sigma) \in (\Sigma_\mathcal{V}^+, M_\mathcal{V})$, as seen in Figure 10. If (w, σ) is not valid, we set $[\![\varphi]\!]_\mathcal{V}(w, \sigma) = 0$. We write $[\![\varphi]\!]$ for $[\![\varphi]\!]_{\mathrm{free}(\varphi)}$. If φ contains no free variables, φ is a *sentence* and $[\![\varphi]\!] : (\Sigma^+, M) \to D$.

$$\begin{aligned}
[\![k]\!]_\mathcal{V}(w, \sigma) &= k \quad \text{for all } k \in \mathbb{K} \\
[\![\varphi \oplus \psi]\!]_\mathcal{V}(w, \sigma) &= [\![\varphi]\!]_\mathcal{V}(w, \sigma) + [\![\psi]\!]_\mathcal{V}(w, \sigma) \\
[\![\beta \,?\, \varphi : \psi]\!]_\mathcal{V}(w, \sigma) &= \begin{cases} [\![\varphi]\!]_\mathcal{V}(w, \sigma)\ , \text{ if } (w, \sigma) \models \beta \\ [\![\psi]\!]_\mathcal{V}(w, \sigma)\ , \text{ otherwise} \end{cases} \\
[\![\textstyle\bigoplus_x \varphi]\!]_\mathcal{V}(w, \sigma) &= \sum_{i \in |w|} [\![\varphi]\!]_{\mathcal{V} \cup \{x\}}(w, \sigma[x \to i]) \\
[\![\textstyle\bigoplus_X \varphi]\!]_\mathcal{V}(w, \sigma) &= \sum_{I \subseteq |w|} [\![\varphi]\!]_{\mathcal{V} \cup \{X\}}(w, \sigma[X \to I]) \\
[\![\mathrm{Val}_x\, \varphi]\!]_\mathcal{V}(w, \sigma) &= \mathrm{Val}((([\![\varphi]\!]_{\mathcal{V} \cup \{x\}}(w, \sigma[x \to i]))_{i \in |w|})
\end{aligned}$$

Fig. 10. Semantics

Example 25. Let us return to the automaton $\mathcal{A}_{\text{policy}}$ over the semiring $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ as depicted in Figure 5. First, we define a boolean formula which states that if a return is present in a position x, then there must be a matching call:

$$\alpha(x) = \text{Lab}_{\text{ret}}(x) \to \exists y(y \curvearrowright x \wedge \text{Lab}_{\text{call}}(y)) \ .$$

The following boolean formula β defines three subsets of string positions, X_0, X_1, X_2, representing, respectively, the string portions where unmatched calls are not penalized, namely X_0, X_2, and the portion where they are, namely X_1.

$$\begin{aligned}\beta(x) = \quad & (x \in X_0 \leftrightarrow (\alpha(x) \wedge \exists y \exists z(y > x \wedge z > x \wedge y \neq z \wedge \text{Lab}_{\$}(y) \wedge \text{Lab}_{\$}(z)))) \\ & \wedge \left(x \in X_1 \leftrightarrow \left(\alpha(x) \wedge \exists y \exists z \begin{pmatrix} y \leq x \leq z \wedge y \neq z \wedge \text{Lab}_{\$}(y) \wedge \text{Lab}_{\$}(z) \\ \wedge((x \neq y \wedge x \neq z) \to \neg\, \text{Lab}_{\$}(x)) \end{pmatrix}\right)\right) \\ & \wedge (x \in X_2 \leftrightarrow (\alpha(x) \wedge \exists y \exists z(y < x \wedge z < x \wedge y \neq z \wedge \text{Lab}_{\$}(y) \wedge \text{Lab}_{\$}(z)))) \ .\end{aligned}$$

Further, we describe positions in X_0 or X_2 by the following boolean formula $\beta_{0,2}$

$$\beta_{0,2}(x) = (x \in X_0 \vee x \in X_2) \wedge (\text{Lab}_{\text{call}}(x) \vee \text{Lab}_{\text{ret}}(x) \vee \text{Lab}_{\text{itr}}(x)) \ .$$

Then, the weight assignment is formalized by

$$\begin{aligned}\varphi = \ & \beta_{0,2}(x)\,?\,0 : \\ & (x \in X_1 \wedge \text{Lab}_{\text{call}}(x))\,?\,1 : \\ & (x \in X_1 \wedge \text{Lab}_{\text{ret}}(x))\,?\,-1 : \\ & (x \in X_1 \wedge \text{Lab}_{\text{itr}}(x))\,?\,0 : \\ & \text{Lab}_{\$}(x)\,?\,0 : -\infty \ ,\end{aligned}$$

which assigns weight 0 to calls, returns, and interrupts outside portion X_1; and weights $1, -1, 0$ to calls, returns, and interrupts, respectively, within portion X_1.

Then, the formula

$$\psi = \text{Val}_x(\beta\,?\,\varphi : -\infty)$$

defines the weight assigned by $\mathcal{A}_{\text{policy}}$ to an input string through a single nondeterministic run. Finally, the formula

$$\chi = \bigoplus_{X_0} \bigoplus_{X_1} \bigoplus_{X_2} \psi$$

defines the global weight of every string in an equivalent way as the one defined by $\mathcal{A}_{\text{policy}}$.

Lemma 26. *Let $\varphi \in \text{MSO}(\mathbb{D})$ and let $\mathcal{V}$ be a finite set of variables with $\text{free}(\varphi) \subseteq \mathcal{V}$. Then, $[\![\varphi]\!]_{\mathcal{V}}(w, \sigma) = [\![\varphi]\!](w, \sigma|_{\text{free}(\varphi)})$ for each valid $(w, \sigma) \in (\Sigma^+_{\mathcal{V}}, M)$. In particular, $[\![\varphi]\!]$ is recognizable (resp. strictly recognizable) iff $[\![\varphi]\!]_{\mathcal{V}}$ is recognizable (resp. strictly recognizable).*

Proof. This is shown by means of Proposition 20 analogously to Proposition 3.3 of [13]. □

As shown by [13] in the case of words, the full weighted logic is strictly more powerful than weighted automata. A similar example also applies here. Therefore, in the following, we restrict our logic in an appropriate way. The main idea for this is to allow only functions with finitely many different values (step functions) after a Val-quantifier.

Definition 27. We define the *set of almost boolean formulas* ψ of MSO($\mathbb{D}$) by the following grammar

$$\psi ::= d \mid \psi \oplus \psi \mid \beta \,?\, \psi : \psi \ ,$$

where $d \in D$ and β is a boolean formula of MSO($\mathbb{D}$).

Definition 28. Let $\varphi \in$ MSO($\mathbb{D}$). We call φ *restricted* if for all subformulas $\mathrm{Val}_x\, \psi$ of φ, ψ is almost boolean.

In Example 25, the formula β is boolean, the formulas φ are almost boolean, and ψ and χ are restricted.

First, we study almost boolean formulas. The following propositions show that they are describing precisely a certain form of rwOPA's behaviors, which we call *OPL step functions*. We adapt ideas from [17] and [27].

Definition 29. For $d \in \mathbb{D}$, we denote by $d : (\Sigma^+, M) \to \mathbb{D}$ the weighted language assigning the weight d to every word w, i.e. $d(w) = d$.

Then, a weighted language S is called an *OPL step function*, if it has a representation

$$S = \sum_{i=1}^{n} d_i \cap L_i \ ,$$

where L_i are OPL forming a partition of (Σ^+, M) and $d_i \in \mathbb{D}$ for each $i \in \{1, ..., n\}$; so $S(w) = d_i$ iff $w \in L_i$, for each $i \in \{1, ..., n\}$.

Lemma 30. *The set of all OPL step functions is closed under* $+$.

Proof. Let $S = \sum_{i=1}^{k} d_i \cap L_i$ and $S' = \sum_{j=1}^{\ell} d'_j \cap L'_j$ be OPL step functions. Then the following holds

$$S + S' = \sum_{i=1}^{k} \sum_{j=1}^{\ell} (d_i + d'_j) \cap (L_i \cap L'_j) \ .$$

Since $(L_i \cap L'_j)$ are also OPL and form a partition of (Σ^+, M), it follows that $S + S'$ is also an OPL step function. □

Proposition 31. *(a) For every almost boolean formula* φ, $[\![\varphi]\!]$ *is an OPL step function.*

(b) If S is an OPL step function, then there exists an almost boolean sentence φ such that $S = [\![\varphi]\!]$.

Proof. (a) We show the first statement by structural induction on φ.

If $\varphi = d$, where $d \in \mathbb{K}$, then $[\![d]\!] = d \cap (\Sigma^+, M)$ is an OPL step function.

In the following, let φ_1 and φ_2 be almost boolean formulas such that $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$ are OPL step functions. Let $\mathcal{V} = \mathrm{free}(\varphi_1) \cup \mathrm{free}(\varphi_2)$. Then, it follows from Lemma 26 that $[\![\varphi_1]\!]_\mathcal{V}$ and $[\![\varphi_2]\!]_\mathcal{V}$ are also OPL step functions.

Thus, $[\![\varphi_1 \oplus \varphi_2]\!] = [\![\varphi_1]\!]_\mathcal{V} + [\![\varphi_2]\!]_\mathcal{V}$ is an OPL step function by Lemma 30.

Now, let $\varphi = \beta \,?\, \varphi_1 : \varphi_2$, where β is a boolean formula of $\mathrm{MSO}(\mathbb{D})$, $\mathcal{V} = \mathrm{free}(\beta) \cup \mathrm{free}(\varphi_1) \cup \mathrm{free}(\varphi_2)$ and $[\![\varphi_1]\!]_\mathcal{V} = \sum_{i=1}^{k} d_i \cap L_i$ and $[\![\varphi_2]\!]_\mathcal{V} = \sum_{j=1}^{\ell} d'_j \cap L'_j$ are again OPL step functions. Then

$$[\![\beta \,?\, \varphi_1 : \varphi_2]\!] = \sum_{i=1}^{k} d_i \cap (L_i \cap L_\mathcal{V}(\beta)) + \sum_{j=1}^{\ell} d'_j \cap (L'_j \cap L_\mathcal{V}(\neg\beta))$$

is also an OPL step function since the languages $L_\mathcal{V}(\beta)$ and $L_\mathcal{V}(\neg\beta)$ are OPL due to Theorem 23.

(b) Given an OPL step function $S = \sum_{i=1}^{n} d_i \cap L_i$, we use Theorem 23 to get sentences β_i with $L_\mathcal{V}(\beta_i) = L_i$. Then, the second statement follows from setting $\varphi = \bigoplus_{i=1}^{n} (\beta_i \,?\, d_i : 0)$ and the fact that the OPL $(L_i)_{1 \le i \le n}$ form a partition of (Σ^+, M). □

Proposition 32. *Let S be an OPL step function. Then S is strictly recognizable.*

Proof. Let $S = \sum_{i=1}^{n} d_i \cap L_i$, with $n \in \mathbb{N}$, $d_1, ..., d_n \in D$, and $L_1, ..., L_n$ OPL forming a partition of (Σ^+, M). It is easy to construct a two state rwOPA recognizing the constant weighted language $[\![d_i]\!]$ which assigns the weight d_i to every word. Hence, $[\![d_i]\!] \cap L_i$ is strictly recognizable by Proposition 19. Therefore, by Proposition 18, S is strictly recognizable. □

7 Characterization of Weighted OPL

In this section, we prove that recognizable weighted languages can be characterized by our restricted weighted logic. We start with necessary closure properties.

Lemma 33 (Closure under weighted disjunction). *Let φ and ψ be two formulas of $\mathrm{MSO}(\mathbb{D})$ such that $[\![\varphi]\!]$ and $[\![\psi]\!]$ are recognizable (resp. strictly recognizable). Then, $[\![\varphi \oplus \psi]\!]$ is recognizable (resp. strictly recognizable).*

Proof. We put $\mathcal{V} = \mathrm{free}(\varphi) \cup \mathrm{free}(\psi)$. Then, $[\![\varphi \oplus \psi]\!] = [\![\varphi]\!]_\mathcal{V} + [\![\psi]\!]_\mathcal{V}$ is recognizable (resp. strictly recognizable) by Lemma 26 and Proposition 18. □

Lemma 34 (Closure under $\bigoplus_x$, $\bigoplus_X$). *Let φ be a formula of $\mathrm{MSO}(\mathbb{D})$ such that $[\![\varphi]\!]$ is recognizable (resp. strictly recognizable). Then, $[\![\bigoplus_x \varphi]\!]$ and $[\![\bigoplus_X \varphi]\!]$ are recognizable (resp. strictly recognizable).*

Proof (Compare [13], Lemma 4.3). Let $\mathcal{X} \in \{x, X\}$ and $\mathcal{V} = \text{free}(\bigoplus_{\mathcal{X}} \varphi)$. We define $\pi : (\Sigma^+_{\mathcal{V}\cup\{\mathcal{X}\}}, M) \to (\Sigma^+_{\mathcal{V}}, M)$ by $\pi(w,\sigma) = (w, \sigma{\restriction_{\mathcal{V}}})$ for any $(w,\sigma) \in (\Sigma^+_{\mathcal{V}\cup\{\mathcal{X}\}}, M)$. Then, for $(w,\gamma) \in (\Sigma^+_{\mathcal{V}}, M)$, the following holds

$$\begin{aligned}[\![\bigoplus_X \varphi]\!](w,\gamma) &= \sum_{I\subseteq\{1,\dots,|w|\}} [\![\varphi]\!]_{\mathcal{V}\cup\{X\}}(w, \gamma[X \to I]) \\ &= \sum_{(w,\sigma)\in\pi^{-1}(w,\gamma)} [\![\varphi]\!]_{\mathcal{V}\cup\{X\}}(w,\sigma) \\ &= \pi([\![\varphi]\!]_{\mathcal{V}\cup\{X\}})(w,\gamma) \ .\end{aligned}$$

Analogously, we show that $[\![\bigoplus_x \varphi]\!](w,\gamma) = \pi([\![\varphi]\!]_{\mathcal{V}\cup\{x\}})(w,\gamma)$ for all $(w,\gamma) \in (\Sigma^+_{\mathcal{V}}, M)$. By Lemma 26, $[\![\varphi]\!]_{\mathcal{V}\cup\{\mathcal{X}\}}$ is recognizable because $\text{free}(\varphi) \subseteq \mathcal{V} \cup \{\mathcal{X}\}$. Then, $[\![\bigoplus_{\mathcal{X}} \varphi]\!]$ is recognizable by Proposition 20. □

Proposition 35 (Closure under restricted Val_x). *Let φ be an almost boolean formula of $\text{MSO}(\mathbb{D})$. Then, $[\![\text{Val}_x \varphi]\!]$ is strictly recognizable.*

Proof. We use ideas of [13] and the extensions in [19] and [11] with the following intuition.

In the first part, we write $[\![\varphi]\!]$ as OPL step function $\sum_{j=1}^m d_j \cap L_j$ and encode the information to which language $(w, \sigma[x \to i])$ belongs in a specially extended language $\tilde{L}$. Then we construct an MSO-formula for this language. Therefore, by Theorem 23, we get a deterministic OPA recognizing $\tilde{L}$. In the second part, we add the weights d_j to this automaton and return to our original alphabet.

More detailed, let $\varphi \in \text{MSO}(\mathbb{D}, (\Sigma, M))$ be almost boolean. We define $\mathcal{V} = \text{free}(\text{Val}_x \varphi)$ and $\mathcal{W} = \text{free}(\varphi) \cup \{x\}$. We consider the extended alphabets $\Sigma_{\mathcal{V}}$ and $\Sigma_{\mathcal{W}}$ together with their natural OPMs $M_{\mathcal{V}}$ and $M_{\mathcal{W}}$. By Proposition 31, $[\![\varphi]\!]$ is an OPL step function. Let $[\![\varphi]\!] = \sum_{j=1}^m d_j \cap L_j$ where L_j is an OPL over $(\Sigma_{\mathcal{W}}, M_{\mathcal{W}})$ for all $j \in \{1, ..., m\}$ and $(L_j)_j$ is a partition of $(\Sigma^+_{\mathcal{W}}, M_{\mathcal{W}})$. By the semantics of the valuation quantifier, we get

$$\begin{aligned}[\![\text{Val}_x\varphi]\!](w,\sigma) &= \text{Val}((\![\varphi]\!]_{\mathcal{W}}(w, \sigma[x \to i]))_{i\in[w]}) \\ &= \text{Val}((d_{g(i)})_{i\in[w]}), \\ \text{where} \qquad g(i) &= \begin{cases} 1 \ , \text{ if } (w, \sigma[x \to i]) \in L_1 \\ \dots \\ m \ , \text{ if } (w, \sigma[x \to i]) \in L_m \end{cases} \text{, for all } i \in [w] \ . \end{aligned} \tag{2}$$

Now, in the first part, we encode the information to which language $(w, \sigma[x \to i])$ belongs in a specially extended language $\tilde{L}$ and construct an MSO-formula for this language. We define the extended alphabet $\tilde{\Sigma} = \Sigma \times \{1, ..., m\}$, together with its natural OPM $\tilde{M}$ (which only refers to Σ), and we identify triples (w, g, σ) with words over $(\tilde{\Sigma}_{\mathcal{V}}, \tilde{M}_{\mathcal{V}})$, so:

$$(\tilde{\Sigma}^+_{\mathcal{V}}, \tilde{M}_{\mathcal{V}}) = \{(w, g, \sigma) \mid (w,\sigma) \in (\Sigma^+_{\mathcal{V}}, M_{\mathcal{V}}) \text{ and } g \in \{1, ..., m\}^{[w]}\} \ .$$

We define the languages $\tilde{L}, \tilde{L}_j, \tilde{L}'_j \subseteq (\tilde{\Sigma}^+_{\mathcal{V}}, \tilde{M}_{\mathcal{V}})$ as follows:

$$\tilde{L} = \left\{ (w,g,\sigma) \;\middle|\; \begin{array}{l} (w,\sigma) \in (\Sigma^+_{\mathcal{V}}, M_{\mathcal{V}}) \text{ is valid and} \\ \text{for all } i \in [w],\ j \in \{1,...,m\} :\ g(i) = j \Rightarrow (w, \sigma[x \to i]) \in L_j \end{array} \right\} ,$$

$$\tilde{L}_j = \left\{ (w,g,\sigma) \;\middle|\; \begin{array}{l} (w,\sigma) \in (\Sigma^+_{\mathcal{V}}, M_{\mathcal{V}}) \text{ is valid and} \\ \text{for all } i \in [w] :\ g(i) = j \Rightarrow (w, \sigma[x \to i]) \in L_j \end{array} \right\} ,$$

$$\tilde{L}'_j = \{\, (w,g,\sigma) \mid \text{for all } i \in [w] :\ g(i) = j \Rightarrow (w, \sigma[x \to i]) \in L_j \,\} \ .$$

Then, $\tilde{L} = \bigcap_{j=1}^{m} \tilde{L}_j$. Hence, in order to show that $\tilde{L}$ is an OPL, it suffices to show that each $\tilde{L}_j$ is an OPL. By a standard procedure, compare [13], we obtain a formula $\tilde{\varphi}_j \in \mathrm{MSO}(\tilde{\Sigma}_{\mathcal{V}}, \tilde{M}_{\mathcal{V}})$ with $L(\tilde{\varphi}_j) = \tilde{L}'_j$. Therefore, by Theorem 23, $\tilde{L}'_j$ is an OPL. It is straightforward to define an OPA accepting $\tilde{N}_{\mathcal{V}}$, the language of all valid words. By closure under intersection, $\tilde{L}_j = \tilde{L}'_j \cap \tilde{N}_{\mathcal{V}}$ is also an OPL and so is $\tilde{L}$. Hence, there exists a deterministic OPA $\tilde{\mathcal{A}} = (Q, q_0, F, \tilde{\delta})$ recognizing $\tilde{L}$.

In the second part, we add weights to $\tilde{\mathcal{A}}$ as follows. We construct the wOPA $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ over $(\Sigma_{\mathcal{V}}, M_{\mathcal{V}})$ and $\mathbb{D}$ by setting the weight of every transition of $\tilde{\mathcal{A}}$ which is labeled with j at the second coordinate to d_j.

That is, we keep the states, the initial state, and the accepting states, and for $\delta = (\delta_{\mathrm{push}}, \delta_{\mathrm{shift}}, \delta_{\mathrm{pop}})$ and all $q, q', p \in Q$ and $(a, j, s) \in \tilde{\Sigma}_{\mathcal{V}}$, we define

$$\delta_{\mathrm{push}}(q, (a,s), q') = \begin{cases} d_j \ , & \text{if } (q, (a,j,s), q') \in \tilde{\delta}_{\mathrm{push}} \\ 0 \ , & \text{otherwise} \end{cases}$$

$$\delta_{\mathrm{shift}}(q, (a,s), q') = \begin{cases} d_j \ , & \text{if } (q, (a,j,s), q') \in \tilde{\delta}_{\mathrm{shift}} \\ 0 \ , & \text{otherwise} \end{cases} \ .$$

Since $\tilde{\mathcal{A}}$ is deterministic, for every $(w,g,\sigma) \in \tilde{L}$, there exists exactly one accepting run $\tilde{r}$ of $\tilde{\mathcal{A}}$. On the other hand, for every $(w,g,\sigma) \notin \tilde{L}$, there is no accepting run of $\tilde{\mathcal{A}}$. Since (L_j) is a partition of $(\Sigma^+_{\mathcal{W}}, M_{\mathcal{W}})$, for every $(w,\sigma) \in (\Sigma_{\mathcal{V}}, M_{\mathcal{V}})$, there exists exactly one g with $(w,g,\sigma) \in \tilde{L}$. Thus, every $(w,\sigma) \in (\Sigma_{\mathcal{V}}, M_{\mathcal{V}})$ has exactly one run r of $\mathcal{A}$ determined by the run $\tilde{r}$ of (w,g,σ) of $\tilde{\mathcal{A}}$. We denote with $\mathrm{wt}_{\mathcal{A}}(r, (w,\sigma), i)$ the weight assigned by the run r of $\mathcal{A}$ on (w,σ) at position i, which is always the weight of the push or shift transition at this position. Then by definition of $\mathcal{A}$ and $\tilde{L}$, for all $i \in [w]$, we have

$$g(i) = j \Rightarrow \mathrm{wt}_{\mathcal{A}}(r, (w,\sigma), i) = d_j \wedge (w, \sigma[x \to i]) \in L_j \ .$$

By formula (2), we have

$$[\![\varphi]\!]_{\mathcal{W}}(w, \sigma[x \to i]) = d_j = \mathrm{wt}_{\mathcal{A}}(r, (w,\sigma), i) \ .$$

Hence, for the behavior of the automaton $\mathcal{A}$, we obtain

$$\begin{aligned}[\![\mathcal{A}]\!](w,\sigma) &= \sum_{r' \in \mathrm{acc}(\mathcal{A},w)} \mathrm{wt}(r') \\ &= \mathrm{Val}((\mathrm{wt}_{\mathcal{A}}(r,(w,\sigma),i))_{i=1,\dots,|w|}) \\ &= \mathrm{Val}(([\![\varphi]\!]_{\mathcal{W}}(w,\sigma[x \to i]))_{i=1,\dots,|w|}) \\ &= [\![\mathrm{Val}_x\, \varphi]\!](w,\sigma)\ .\end{aligned}$$

Thus, $\mathcal{A}$ recognizes $[\![\mathrm{Val}_x\, \varphi]\!]$. □

The following proposition is a summary of the previous results.

Proposition 36. *For every restricted* MSO($\mathbb{D}$)*-sentence φ, there exists an rwOPA $\mathcal{A}$ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.*

Proof. We use structural induction on φ. If φ is an almost boolean formula, then by Proposition 31, $[\![\varphi]\!]$ is an OPL step function. By Proposition 32, every OPL step function is strictly recognizable. Closure under $\oplus$ is dealt with by Lemma 33.

If $\varphi = \beta\,?\,\psi : \theta$ and $\mathcal{V} = \mathrm{free}(\beta) \cup \mathrm{free}(\psi) \cup \mathrm{free}(\theta)$, then we have $[\![\beta\,?\,\psi : \theta]\!] = [\![\psi]\!]_{\mathcal{V}} \cap L_{\mathcal{V}}(\beta) + [\![\theta]\!]_{\mathcal{V}} \cap L_{\mathcal{V}}(\neg\beta)$, which is strictly recognizable by Lemma 26 and Propositions 18 and 19.

The sum quantifications $\bigoplus_x$ and $\bigoplus_X$ are dealt with by Lemma 34. Since φ is restricted, we know that for every subformula $\mathrm{Val}_x\, \psi$, the formula ψ is an almost boolean formula. Therefore, we can apply Proposition 35 to obtain that $[\![\mathrm{Val}_x\, \psi]\!]$ is strictly recognizable.

The next proposition shows that the converse also holds.

Proposition 37. *For every rwOPA $\mathcal{A}$, there exists a restricted* MSO($\mathbb{D}$)*-sentence φ with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.*

Proof. The rationale adopted to build φ from $\mathcal{A}$ integrates the approach followed in [13, 19] with the one of [32]. On the one hand we need second order variables suitable to "carry" weights; on the other hand, unlike previous non-OP cases which are managed through real-time automata, an OPA can perform several transitions while remaining in the same position. Thus, we introduce the following second order variables: $X^{\mathrm{push}}_{p,a,q}$ represents the set of positions where $\mathcal{A}$ performs a push move from state p, reading symbol a and reaching state q; $X^{\mathrm{shift}}_{p,a,q}$ has the same meaning as $X^{\mathrm{push}}_{p,a,q}$ for a shift operation; $X^{\mathrm{pop}}_{p,q,r}$ represents the set of positions of the symbol that is on top of the stack when $\mathcal{A}$ performs a pop transition from state p, with q on top of the stack, reaching r.

Let $\mathcal{V}$ consist of all variables $X^{\mathrm{push}}_{p,a,q}$, $X^{\mathrm{shift}}_{p,a,q}$, and $X^{\mathrm{pop}}_{p,q,r}$ such that $a \in \Sigma$, $p,q,r \in Q$ and $(p,a,q) \in \delta_{\mathrm{push}}$ resp. δ_{shift}, resp. $(p,q,r) \in \delta_{\mathrm{pop}}$. Since Σ and Q are finite, there is an enumeration $\bar{X} = (X_1,..,X_m)$ of all variables of $\mathcal{V}$. We denote by $\bar{X}^{\mathrm{push}}$, $\bar{X}^{\mathrm{shift}}$, and $\bar{X}^{\mathrm{pop}}$ enumerations over only the respective set of second order variables.

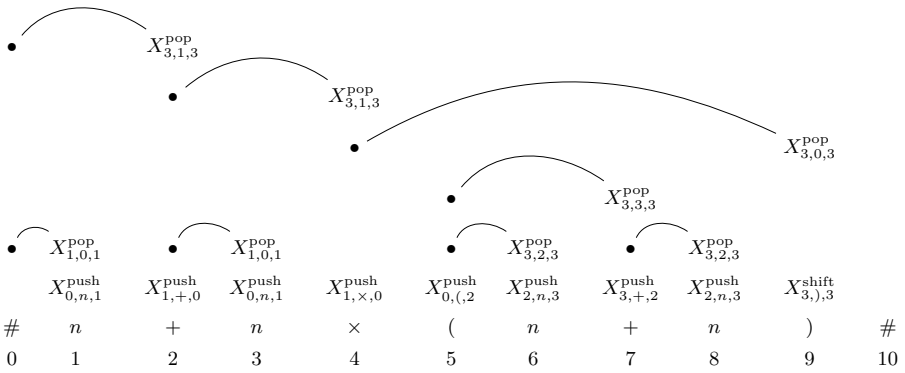


Fig. 11. The string of Figure 1 with the second order variables evidenced for the automaton of Figure 2. The symbol • marks the positions of the symbols that precede the push corresponding to the bound pop transition.

We introduce the shortcuts Tree, Next_i, Q_i, and $\text{Tree}_{p,q}$, originally defined in [32], reported and adapted here for convenience:

$$x \circ y := \bigvee_{a,b \in \Sigma, M_{a,b} = \circ} \text{Lab}_a(x) \wedge \text{Lab}_b(y), \text{ for } \circ \in \{\lessdot, \doteq, \gtrdot\}$$

$$\text{Tree}(x,z,v,y) := x \curvearrowright y \wedge \begin{pmatrix} (x+1 = z \ \vee \ x \curvearrowright z) \wedge \neg\exists t(z < t < y \wedge x \curvearrowright t) \\ \wedge \\ (v+1 = y \ \vee \ v \curvearrowright y) \wedge \neg\exists t(x < t < v \wedge t \curvearrowright y) \end{pmatrix}$$

In other words, Tree holds among the four positions (x,z,v,y) iff, at the time when a pop transition is executed: x (resp. y) is the rightmost leaf at the left (resp. the leftmost at the right) of the subtree whose scanning (and construction if used as a parser) is completed by the OPA through the current transition; z and y are the leftmost and rightmost terminal characters of the right hand side of the grammar production that is reduced by the pop transition of the OPA [32]. For instance, with reference to Figures 1 and 11, $\text{Tree}(5,7,7,9)$ and $\text{Tree}(4,5,9,10)$ hold.

$$\text{Succ}_q(x,y) := (x+1 = y) \wedge \bigvee_{p \in Q, a \in \Sigma} (x \in X^{\text{push}}_{p,a,q} \vee x \in X^{\text{shift}}_{p,a,q} \vee \min(x))$$

I.e., y is the position adjacent to x, $\text{Lab}_a(y)$ holds and, while reading a, the OPA reaches state q, either through a push or through a shift move.

$$\text{Next}_r(x,y) := \exists z \exists v. \left(\text{Tree}(x,z,v,y) \wedge \bigvee_{p,q \in Q} v \in X^{\text{pop}}_{p,q,r} \right)$$

I.e., $\mathrm{Next}_r(x,y)$ holds when a pop move reduces a subtree enclosed between positions x and y reaching state r.

$$Q_i(x,y) := \mathrm{Succ}_i(x,y) \vee \mathrm{Next}_i(x,y)$$

Finally,

$$\mathrm{Tree}_{i,j}(x,z,v,y) := \mathrm{Tree}(x,z,v,y) \wedge Q_i(v,y) \wedge Q_j(x,z)$$

refines the predicate Tree by making explicit that i and j are, respectively, the current state and the state on top of the stack when the pop move is executed.

We now define the unweighted formula ψ to characterize all accepting runs

$$\begin{aligned}\psi = \mathit{Partition}(\bar{X}^{\mathrm{push}}, \bar{X}^{\mathrm{shift}}) \wedge \mathit{Unique}(\bar{X}^{\mathrm{pop}}) \wedge \mathit{InitFinal} \\ \wedge\ \mathit{Trans}_{\mathrm{push}} \wedge \mathit{Trans}_{\mathrm{shift}} \wedge \mathit{Trans}_{\mathrm{pop}}\ .\end{aligned}$$

Here, the subformula *Partition* will enforce the push and shift sets to be (together) a partition of all positions. *InitFinal* controls the initial and the acceptance condition and Trans_{op} the transitions of the run together with the labels. For any number n of second order variables, the formulas below precisely define the above predicates.

$$\mathit{Partition}(X_1,...,X_n) = \forall x.\ \bigvee_{1\le i\le n} \big[(x \in X_i) \wedge \bigwedge_{\substack{1\le j\le n \\ i\neq j}} \neg(x \in X_j)\big]\ ,$$

$$\mathit{Unique}(X_1^{\mathrm{pop}},..,X_n^{\mathrm{pop}}) = \forall x.\ \bigwedge_{\substack{1\le i\le n \\ 1\le j\le n \\ i\neq j}} \neg(x \in X_i^{\mathrm{pop}} \wedge x \in X_j^{\mathrm{pop}})\ ,$$

$$\begin{aligned}\mathit{InitFinal} = \exists x \exists y \exists x' \exists y'.\big[\,&\min(x) \wedge \max(y) \wedge x+1 = x' \wedge y'+1 = y \\ &\wedge \bigvee_{\substack{i\in I,\, q\in Q \\ a\in\Sigma}} x' \in X_{i,a,q}^{\mathrm{push}} \\ &\wedge \bigvee_{\substack{f\in F,\, q\in Q \\ a\in\Sigma}} (y' \in X_{q,a,f}^{\mathrm{push}} \vee y' \in X_{q,a,f}^{\mathrm{shift}}) \\ &\wedge \bigvee_{f\in F} (\mathrm{Next}_f(x,y) \wedge \bigwedge_{j\neq f} \neg\,\mathrm{Next}_j(x,y))\big]\ ,\end{aligned}$$

$$\mathit{Trans}_{\mathrm{push}} = \forall x.\ \bigwedge_{p,q\in Q,a\in\Sigma} \big(x \in X_{p,a,q}^{\mathrm{push}} \to \big[\,\mathrm{Lab}_a(x) \wedge \exists z.(z \lessdot x \wedge Q_p(z,x))\big]\big)$$

$$\mathit{Trans}_{\mathrm{shift}} = \forall x.\ \bigwedge_{p,q\in Q,a\in\Sigma} \big(x \in X_{p,a,q}^{\mathrm{shift}} \to \big[\,\mathrm{Lab}_a(x) \wedge \exists z.(z \doteq x \wedge Q_p(z,x))\big]\big)\ .$$

I.e., if $x \in X_{p,a,q}^{\mathrm{push}}$ (resp. $X_{p,a,q}^{\mathrm{shift}}$), the formula holds in a run where, reading character a in position x, the automaton performs a push (resp. a shift) reaching state q from p; this may occur when $z \lessdot x$ (resp., $z \doteq x$) is immediately adjacent

to x or after a subtree between positions z and x has been built. Notice that the converse holds too of the above implications holds, due to the fact that the whole set of string positions is partitioned into the two disjoint sets of $\bar{X}^{\text{push}}$, $\bar{X}^{\text{shift}}$.

$$\mathit{Trans}_{\text{pop}} = \forall v. \bigwedge_{p,q \in Q} \left(\left[\bigvee_{r \in Q} v \in X^{\text{pop}}_{p,q,r} \right] \leftrightarrow \left[\exists x \exists y \exists z. (\text{Tree}_{p,q}(x,z,v,y)) \right] \right)$$

Thus, with arguments similar to Section 4.3 of [32], it can be shown that the sentences satisfying ψ are exactly those recognized by the unweighted OPA subjacent to $\mathcal{A}$.

Now, we add weights to every position x with the following restricted weighted formulas depending on our setting.

Now, we define

$$\begin{aligned} \theta'(x) = \bigoplus_{p,q \in Q} \Big(& \bigoplus_{a \in \Sigma} (x \in X^{\text{push}}_{p,a,q} \,?\, \text{wt}_{\text{push}}(p,a,q) : 0) \\ & \oplus \bigoplus_{a \in \Sigma} (x \in X^{\text{shift}}_{p,a,q} \,?\, \text{wt}_{\text{shift}}(p,a,q) : 0) \Big) \ . \end{aligned}$$

Then, we multiply up all weights of the encountered transitions using the valuation function as follows

$$\theta = \psi \,?\, \text{Val}_x \theta'(x) : 0 \ .$$

Since the subformulas of θ' are almost boolean, θ' is almost boolean. Furthermore, ψ is boolean, thus θ is a restricted formula.

Finally, we define

$$\varphi = \bigoplus_{X_1} \bigoplus_{X_2} \cdots \bigoplus_{X_m} \theta \ .$$

This implies $[\![\varphi]\!](w) = [\![\mathcal{A}]\!](w)$, for all $w \in (\Sigma^+, M)$. Therefore, φ is our required sentence with $[\![\mathcal{A}]\!] = [\![\varphi]\!]$. □

The following theorem summarizes the main results of this section.

Theorem 38. *Let $\mathbb{D}$ be a valuation monoid and $S : (\Sigma^+, M) \to D$ a weighted language. The following are equivalent:*

(i) $S = [\![\mathcal{A}]\!]$ for some rwOPA.
(ii) $S = [\![\varphi]\!]$ for some restricted sentence φ of MSO($\mathbb{D}$).

Theorem 38 documents a further step in the path of generalizing a series of results beyond the barrier of regular and structured –or visible– CFLs. Up to a few years ago, major properties of regular languages, such as closure with respect to all main language operations, decidability results, logic characterization, and, in this case, weighted language versions, could be extended to several classes of structured CFLs, among which the VPL one certainly obtained much attention. OPLs further generalize the above results not only in terms of strict inclusion, but mainly because they are not visible, in the sense explained in the introduction, nor

are they necessarily real-time: this allows them to cover important applications that could not be adequately modeled through more restricted classes.

Note that since every semiring is also a valuation monoid, Theorem 38 can also be applied to semirings. In this case, however, the product of the semiring is only taken into account as the valuation function. In the following section, we will take a closer look at the semiring case where we additionally include the binary product in the weighted logic.

8 The Semiring Case

In this section, we study semirings. Since they are a special case of valuation monoids, the previous results still hold for semirings. However, we can strengthen some of the results in this special case and, additionally, we can show that a weighted logic extended with the additional semiring-operation, the product, is still as expressive as (restricted) weighted OPA. In the following, let $\mathbb{K} = (K, +, \cdot, 0, 1)$ be a semiring.

We start by revisiting the comparison of the expressive power of wOPA (that have pop-weights) and rwOPA (that have no pop-weights). Since in the semiring case, applying no pop weights is the same as applying only trivial pop weights, we immediately see that rwOPL $\subset$ wOPL. We now study in which cases this relation is strict.

We already saw in Section 3, Proposition 15 that there exists a non-commutative semiring over which wOPA are strictly more expressive than rwOPA. We also proved in Proposition 16 that the same is true for commutative valuation monoids. However, in the special case of commutative semirings, we can show with the following result that rwOPA are as expressive as wOPA, and therefore can be seen as a kind of normal form for wOPA.

Theorem 39. *Let $\mathbb{K}$ be a commutative semiring and (Σ, M) an OP alphabet. Let $\mathcal{A}$ be a wOPA. Then, there exists an rwOPA $\mathcal{B}$ with $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$.*

Proof. Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ be a wOPA over (Σ, M) and $\mathbb{K}$. Note that for every pop transition of a wOPA, there exists exactly one push transition. We construct an rwOPA $\mathcal{B}$ over the state set $Q' = Q \times Q \times Q$ and with the same behavior as $\mathcal{A}$ with the following idea in mind. In the first state component of Q', $\mathcal{B}$ simulates $\mathcal{A}$. In the second and third state component of Q', the automaton $\mathcal{B}$ preemptively guesses the states q and r of the pop transition (q, p, r) of $\mathcal{A}$ which corresponds to the next push transition following after this configuration. This enables us to transfer the weight from the pop transition to the correct push transition.

The detailed construction of $\mathcal{B} = (Q', I', F', \delta', \mathrm{wt}')$ over (Σ, M) and $\mathbb{K}$ is the following. If Q is the empty set, then $[\![\mathcal{A}]\!] \equiv 0$ is trivially strictly recognizable. If Q is nonempty, let $q \in Q$ be a fixed state and set $Q' = Q \times Q \times Q$, $I' =$

$\{(q_1, q_2, q_3) \mid q_1 \in I, q_2, q_3 \in Q\}$, $F' = \{(q_1, q, q) \mid q_1 \in F\}$, and

$$\delta'_{\text{push}} = \{((q_1, q_2, q_3), a, (r_1, r_2, r_3)) \mid (q_1, a, r_1) \in \delta_{\text{push}} \text{ and } (q_2, q_1, q_3) \in \delta_{\text{pop}}\}$$
$$\delta'_{\text{shift}} = \{((q_1, q_2, q_3), a, (r_1, q_2, q_3)) \mid (q_1, a, r_1) \in \delta_{\text{shift}}\}$$
$$\delta'_{\text{pop}} = \{((q_1, q_2, q_3), (p_1, p_2, p_3), (r_1, q_2, q_3)) \mid (q_1, p_1, r_1) \in \delta_{\text{pop}}, p_2 = q_1, p_3 = r_1\}$$

Here, every push of $\mathcal{B}$ controls that the previously guessed q_2 and q_3 can be used by a pop transition of $\mathcal{A}$ going from q_2 to q_3 with q_1 on top of the stack. Every pop controls that the symbols on top of the stack are exactly the ones used at this pop. Since the second and third state component are guessed for the next push, they are passed on whenever we read a shift or pop. The second and third component pushed at the first position of a word are guessed by an initial state. At the last push, which therefore has no following push and will propagate the second and third component to the end of the run, the automaton $\mathcal{B}$ has to guess the distinguished state used in the final states.

Therefore, $\mathcal{B}$ has exactly one accepting run (of the same length) for every accepting run of $\mathcal{A}$, and vice versa. Finally, we define the transition weights as follows.

$$\text{wt}'_{\text{push}}((q_1, q_2, q_3), a, (r_1, r_2, r_3)) = \text{wt}_{\text{push}}(q_1, a, r_1) \cdot \text{wt}_{\text{pop}}(q_2, q_1, q_3)$$
$$\text{wt}'_{\text{shift}}((q_1, q_2, q_3), a, (r_1, r_2, r_3)) = \text{wt}_{\text{shift}}(q_1, a, r_1)$$

Then, the runs of $\mathcal{A}$ simulated by $\mathcal{B}$ have exactly the same weights but in a different order. Since $\mathbb{K}$ is commutative, it follows that $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$. □

In the following, we study additional closure results concerning the product of a semiring and we show how to strengthen our main result in the case of semirings to comprise logical sentences that include the product.

We define the Hadamard product $S \odot T$ of two weighted languages $S, T : (\Sigma^+, M) \to K$ by letting

$$(S \odot T)(w) = S(w) \cdot T(w) \text{ for each } w \in (\Sigma^+, M) \ .$$

We prove that restricted weighted languages and weighted languages are closed under multiplication with weights and, in the case of commutative semirings, are closed under the product in general, as follows.

Proposition 40. *Let $S : (\Sigma^+, M) \to K$ be a recognizable (resp. strictly recognizable) weighted language and $k \in \mathbb{K}$. Then $[\![k]\!] \odot S$ is recognizable (resp. strictly recognizable).*

Proof. Let $\mathcal{A} = (Q, I, F, \delta, \text{wt})$ be an (r)wOPA such that $[\![\mathcal{A}]\!] = S$. Then, we construct an (r)wOPA $\mathcal{B} = (Q', I', F, \delta', \text{wt}')$ as follows.

We set $Q = Q \cup I'$ and $I' = \{q'_I \mid q_I \in I\}$. The new transition relations δ' and weight functions wt' consist of all transitions of $\mathcal{A}$ with their respective weights and the following additional transitions: For every push transition (q_I, a, q) of δ_{push}, we add a push transition (q'_I, a, q) to δ'_{push} with $\text{wt}'_{\text{push}}(q'_I, a, q) = k \cdot \text{wt}_{\text{push}}(q_I, a, q)$.

Note that every run of an (w)OPA has to start with a push transition. Therefore, the two automata have the same respective runs, but $\mathcal{B}$ is exactly once in a state $q'_I \in I$. This together with the weight assignment ensures that $\mathcal{B}$ uses the same weights as $\mathcal{A}$ except at the very first transition of every run which is multiplied by k from the left. In particular, we do not change the weight of any pop transition. It follows that $[\![\mathcal{B}]\!] = [\![k]\!] \odot S$. Also, if $\mathcal{A}$ is restricted, so is $\mathcal{B}$. □

Proposition 41. *Let $\mathbb{K}$ be a semiring and $\mathcal{A}$ and $\mathcal{B}$ two wOPA such that all weights of $\mathcal{A}$ commute with all weights of $\mathcal{B}$. Then, $[\![\mathcal{A}]\!] \odot [\![\mathcal{B}]\!]$ is recognizable. If, additionally, $\mathcal{A}$ and $\mathcal{B}$ are restricted, then $[\![\mathcal{A}]\!] \odot [\![\mathcal{B}]\!]$ is strictly recognizable.*

Proof. We use a standard product construction over two wOPA as follows. Note that in contrast to the case of classical product constructions, the pop transitions of a wOPA do not consume a symbol.

Let $\mathcal{A} = (Q, I, F, \delta, \mathrm{wt})$ and $\mathcal{B} = (Q', I', F', \delta', \mathrm{wt}')$ be two wOPA. We construct a wOPA $\mathcal{P}$ as $\mathcal{P} = (Q \times Q', I \times I', F \times F', \delta^{\mathcal{P}}, \mathrm{wt}^{\mathcal{P}})$ where $\delta^{\mathcal{P}} = (\delta^{\mathcal{P}}_{\mathrm{push}}, \delta^{\mathcal{P}}_{\mathrm{shift}}, \delta^{\mathcal{P}}_{\mathrm{pop}})$ and set

$$\begin{aligned}
\delta^{\mathcal{P}}_{\mathrm{push}} &= \{((q,q'),a,(r,r')) \mid (q,a,r) \in \delta_{\mathrm{push}} \text{ and } (q',a,r') \in \delta'_{\mathrm{push}}\} \ , \\
\delta^{\mathcal{P}}_{\mathrm{shift}} &= \{((q,q'),a,(r,r')) \mid (q,a,r) \in \delta_{\mathrm{shift}} \text{ and } (q',a,r') \in \delta'_{\mathrm{shift}}\} \ , \\
\delta^{\mathcal{P}}_{\mathrm{pop}} &= \{((q,q'),(p,p'),(r,r')) \mid (q,p,r) \in \delta_{\mathrm{pop}} \text{ and } (q',p',r') \in \delta'_{\mathrm{pop}}\} \ ,
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{wt}^{\mathcal{P}}_{\mathrm{push}}((q,q'),a,(r,r')) &= \mathrm{wt}'_{\mathrm{push}}(q,a,r) \cdot \mathrm{wt}''_{\mathrm{push}}(q',a,r') \ , \\
\mathrm{wt}^{\mathcal{P}}_{\mathrm{shift}}((q,q'),a,(r,r')) &= \mathrm{wt}'_{\mathrm{shift}}(q,a,r) \cdot \mathrm{wt}''_{\mathrm{shift}}(q',a,r') \ , \\
\mathrm{wt}^{\mathcal{P}}_{\mathrm{pop}}((q,q'),(p,p'),(r,r')) &= \mathrm{wt}'_{\mathrm{pop}}(q,p,r) \cdot \mathrm{wt}''_{\mathrm{pop}}(q',p',r') \ .
\end{aligned}$$

Then with the same arguments as in the proof of Proposition 19 and using the commutativity of the weights of $\mathcal{A}$ and $\mathcal{B}$ and the distributivity of $\mathbb{K}$, it follows for all $w \in (\Sigma^+, M)$

$$\begin{aligned}
[\![\mathcal{P}]\!](w) &= \sum_{\rho \in \mathrm{acc}(\mathcal{P},w)} \mathrm{wt}^{\mathcal{P}}(\rho) \\
&= \sum_{\substack{\rho, \text{ such that} \\ \rho_{\restriction Q} \in \mathrm{acc}(\mathcal{A},w) \\ \rho_{\restriction Q'} \in \mathrm{acc}(\mathcal{B},w)}} \mathrm{wt}^{\mathcal{P}}(\rho) \\
&= \Big(\sum_{\rho_{\restriction Q} \in \mathrm{acc}(\mathcal{A},w)} \mathrm{wt}(\rho_{\restriction Q}) \Big) \cdot \Big(\sum_{\rho_{\restriction Q'} \in \mathrm{acc}(\mathcal{B},w)} \mathrm{wt}'(\rho_{\restriction Q'}) \Big) \\
&= [\![\mathcal{A}]\!](w) \cdot [\![\mathcal{B}]\!](w) \ .
\end{aligned}$$

Thus, $[\![\mathcal{P}]\!] = [\![\mathcal{A}]\!] \odot [\![\mathcal{B}]\!]$. Further, if $\mathcal{A}$ and $\mathcal{B}$ are restricted, then by leaving out $\mathrm{wt}^{\mathcal{P}}_{\mathrm{pop}}$, the wOPA $\mathcal{P}$ is also restricted. □

Note that Proposition 41 can be sharpened by saying that the product of two recognizable (strictly recognizable) series recognized by the automata $[\![\mathcal{A}]\!]$ and $[\![\mathcal{B}]\!]$ is again recognizable (strictly recognizable) over the subsemiring generated by the weights of $[\![\mathcal{A}]\!]$ and $[\![\mathcal{B}]\!]$.

To include the product into the weighted logic, we make the following adjustments to MSO($\mathbb{K}$).

Definition 42. Given a semiring $\mathbb{K} = (K, +, \cdot, 0, 1)$, we add the following two weighted formulas to get the logic $\mathrm{MSO}_{\otimes}(\mathbb{K})$; $\varphi ::= \beta$ and $\varphi ::= \varphi \otimes \varphi$, where β is a boolean formula. They have the following semantics

$$[\![\beta]\!]_{\mathcal{V}}(w, \sigma) = \begin{cases} 1\ , & \text{if } (w, \sigma) \models \beta \\ 0\ , & \text{otherwise} \end{cases}$$

$$[\![\varphi \otimes \psi]\!]_{\mathcal{V}}(w, \sigma) = [\![\varphi]\!]_{\mathcal{V}}(w, \sigma) \cdot [\![\psi]\!]_{\mathcal{V}}(w, \sigma)\ .$$

Note that $[\![\beta]\!] = [\![\beta \,?\, 1 : 0]\!]$ and $[\![\beta \,?\, \psi : \theta]\!] = [\![(\psi \otimes \beta) \oplus (\theta \otimes \neg\beta)]\!]$. Then, a respective version of Lemma 26 for $\mathrm{MSO}_{\otimes}(\mathbb{K})$ can be shown analogously.

Furthermore, we adjust the necessary restrictions on our logic as follows.

Definition 43. Let $\varphi \in \mathrm{MSO}_{\otimes}(\mathbb{K})$. We denote by $\mathrm{const}(\varphi)$ all weights of $\mathbb{K}$ occurring in φ.

We call φ *$\otimes$-restricted* if for all subformulas $\psi \otimes \theta$ of φ either ψ is almost boolean or $\mathrm{const}(\psi)$ and $\mathrm{const}(\theta)$ commute elementwise.

We call φ *restricted* if it is $\otimes$-restricted and for all subformulas $\mathrm{Val}_x\, \psi$ of φ, ψ is almost boolean.

Notice that the formulas ψ and χ of Example 25 can be interpreted as formulas of $\mathrm{MSO}_{\otimes}(\mathbb{K})$. In this case, they are restricted even if $\mathbb{K}$ is not commutative.

Proposition 44 (Closure under restricted weighted conjunction). *Let $\psi \otimes \theta$ be a subformula of a $\otimes$-restricted formula φ of $\mathrm{MSO}_{\otimes}(\mathbb{K})$ such that $[\![\psi]\!]$ and $[\![\theta]\!]$ are recognizable (resp. strictly recognizable). Then, $[\![\psi \otimes \theta]\!]$ is recognizable (resp. strictly recognizable) over the subsemiring of $\mathbb{K}$ generated by the constants occurring in ψ and θ.*

Proof. Since φ is $\otimes$-restricted, either ψ is almost boolean or the constants of both formulas commute.

Case 1: Let us assume ψ is almost boolean. Then, we can write $[\![\psi]\!]$ as OPL step function, i.e., $[\![\psi]\!] = \sum_{i=1}^{n} k_i \cap L_i$, where L_i are OPL. So, the weighted language $[\![\psi \otimes \theta]\!]$ equals a sum of weighted languages of the form $([\![k_i \otimes \theta]\!] \cap L_i)$. Then, by Proposition 40, $[\![k_i \otimes \theta]\!]$ is a recognizable (resp. strictly recognizable) weighted language. Therefore, $([\![k_i \otimes \theta]\!] \cap L_i)$ is recognizable (resp. strictly recognizable) by Proposition 19. Hence, $[\![\psi \otimes \theta]\!]$ is (strictly) recognizable by Proposition 18.

Case 2: Let us assume that the constants of ψ and θ commute. Then, Proposition 41 yields the claim. □

The formulas and operators of MSO($\mathbb{D}$) are part of $\mathrm{MSO}_{\otimes}(\mathbb{K})$ and the previous closure results Lemma 33, 34, and Proposition 35 can be shown for $\mathrm{MSO}_{\otimes}(\mathbb{K})$ analogously to the proofs for MSO($\mathbb{D}$). Therefore, we are now ready to prove the following.

Proposition 45. *For every restricted* $\mathrm{MSO}_{\otimes}(\mathbb{K})$*-sentence* φ*, there exists an rwOPA* $\mathcal{A}$ *with* $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.

Proof. We follow the proof of Proposition 36 and only have to additionally prove closure under $\otimes$, which is dealt with by Proposition 44. □

Proposition 46. *Let* $\mathbb{K}$ *be a commutative semiring. Then for every wOPA* $\mathcal{A}$*, there exists a restricted* $\mathrm{MSO}_{\otimes}(\mathbb{K})$*-sentence* φ *with* $[\![\mathcal{A}]\!] = [\![\varphi]\!]$.

Proof. We follow the proof of Proposition 37, but instead of using θ, we define

$$\begin{aligned}\theta'(x) = \bigotimes_{p,q\in Q} \Big(& \bigotimes_{a\in\Sigma} (x \in X^{\mathrm{push}}_{p,a,q} \,?\, \mathrm{wt}_{\mathrm{push}}(p,a,q) : 1) \\ & \otimes \bigotimes_{a\in\Sigma} (x \in X^{\mathrm{shift}}_{p,a,q} \,?\, \mathrm{wt}_{\mathrm{shift}}(p,a,q) : 1) \\ & \otimes \bigotimes_{r\in Q} (x \in X^{\mathrm{pop}}_{p,q,r} \,?\, \mathrm{wt}_{\mathrm{pop}}(p,q,r) : 1)\Big) \ .\end{aligned}$$

Note that in this case the valuation is a commutative product and this is indeed crucial because the valuation quantifier (i.e. the product quantifier) of θ' assigns the pop weight at a different position than the occurrence of the respective pop transition in the automaton. Using only one such quantifier, this is unavoidable, since the number of pops at a given position is only bounded by the word length. □

We observe that both in the case of commutative and non-commutative semirings, it follows directly from Proposition 37 that for every rwOPA $\mathcal{A}$, there is a restricted sentence of $\mathrm{MSO}_{\otimes}(\mathbb{K})$ with the same behavior as $\mathcal{A}$. Together with Proposition 45 and Proposition 46, this allows us to strengthen Theorem 38 in the case of semirings as follows.

Theorem 47. *Let* $\mathbb{K}$ *be a semiring and* $S : (\Sigma^+, M) \to K$ *a weighted language.*

1. *Then, the following are equivalent:*
   (i) $S = [\![\mathcal{A}]\!]$ *for some rwOPA.*
   (ii) $S = [\![\varphi]\!]$ *for some restricted sentence* φ *of* $\mathrm{MSO}_{\otimes}(\mathbb{K})$.
2. *Let* $\mathbb{K}$ *be commutative. Then, the following are equivalent:*
   (i) $S = [\![\mathcal{A}]\!]$ *for some wOPA.*
   (ii) $S = [\![\varphi]\!]$ *for some restricted sentence* φ *of* $\mathrm{MSO}_{\otimes}(\mathbb{K})$.

Theorem 47 also shows that the typical logical characterization of weighted languages does not generalize in the same way to the whole class wOPL: for non-rwOPL we need the extra hypothesis that $\mathbb{K}$ is commutative. This is due to the fact that pop transitions are applied in the reverse order than that of positions to which they refer (position v in formula $\mathit{Trans}_{\mathrm{pop}}$). Notice, however, that rwOPL allow for pop sequences whose length is not bounded by a constant; thus, they too include languages that are neither real-time nor visible. This remark naturally raises new intriguing questions which we will briefly address in the conclusion.

9 Conclusion

We introduced and investigated weighted operator precedence automata and a corresponding weighted MSO logic. We employ weights from *valuation monoids* that form a very general weight structure which not only includes all semirings but also computations like average and discounting.

In our main results we show, for any valuation monoid, that wOPA without pop weights generalize wVPA, they have the same expressive power as a restricted weighted MSO logic; and, their behaviors can also be described as homomorphic images of the behaviors of particularly simple wOPA reduced to arbitrary unweighted OPA.

In the special case of commutative semirings, we are able to consolidate these results to apply also to wOPA with arbitrary pop weights. Additionally, we also prove that the same is not possible for commutative valuation monoids.

This raises the problems to find, for arbitrary semirings and for wOPA with pop weights, both an expressively equivalent weighted MSO logic and a Nivat-type result. In [20], very similar problems arose for weighted automata on unranked trees and weighted MSO logic. In [14], the authors showed that with another definition of the behavior of weighted unranked tree automata, an equivalence result for the restricted weighted MSO logic could be derived. Is there another definition of the behavior of wOPA (with pop weights) making them expressively equivalent to our restricted weighted MSO logic?

In [32], operator precedence languages of infinite words were investigated and shown to be practically important. Therefore, the problem arises to develop a theory of wOPA on infinite words. In order to define their infinitary quantitative behaviors, in previous works [17, 11], valuation monoids proved most helpful.

Finally, a new investigation field can be opened by exploiting the natural suitability of OPL towards parallel elaboration [3]. Computing weights, in fact, can be seen as a special case of semantic elaboration which can be performed hand-in-hand with parsing. In this case too, we can expect different challenges depending on whether the weight structure is a semiring, it is commutative, or not and/or weights are attached to pop transitions too, which would be the natural way to follow the traditional semantic evaluation through synthesized attributes [29].

References

1. Alur, R., Fisman, D.: Colored nested words. In: Dediu, A.H., Janousek, J., Martín-Vide, C., Truthe, B. (eds.) Language and Automata Theory and Applications, LATA 2016. LNCS, vol. 9618, pp. 143–155. Springer (2016)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3), 16:1–16:43 (2009)
3. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Panella, F., Pradella, M.: Parallel parsing made practical. Sci. Comput. Program. 112(3), 195–226 (2015)
4. Berstel, J., Reutenauer, C.: Rational Series and Their Languages, EATCS Monographs in Theoretical Computer Science, vol. 12. Springer (1988)

5. Bollig, B., Gastin, P.: Weighted versus probabilistic logics. In: Diekert, V., Nowotka, D. (eds.) Developments in Language Theory, DLT 2009. LNCS, vol. 5583, pp. 18–38. Springer (2009)
6. von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in log n space. In: Proceedings of the Symposium on Fundamentals of Computation Theory. LNCS, vol. 158, pp. 40–51. Springer (1983)
7. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik und Grundlagen Math. 6, 66–92 (1960)
8. Choffrut, C., Malcher, A., Mereghetti, C., Palano, B.: First-order logics: some characterizations and closure properties. Acta Inf. 49(4), 225–248 (2012)
9. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. J. Comput. Syst. Sci. 78(6), 1837–1867 (2012)
10. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Information and Control 37(2), 115–133 (1978)
11. Droste, M., Dück, S.: Weighted automata and logics for infinite nested words. Inf. Comput. 253, 448–466 (2017)
12. Droste, M., Dück, S., Mandrioli, D., Pradella, M.: Weighted operator precedence languages. In: Larsen, K.G., Bodlaender, H.L., Raskin, J. (eds.) Mathematical Foundations of Computer Science, MFCS 2015. LIPIcs, vol. 83, pp. 31:1–31:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
13. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1-2), 69–86 (2007), extended abstract in ICALP 2005
14. Droste, M., Heusel, D., Vogler, H.: Weighted unranked tree automata over tree valuation monoids and their characterization by weighted logics. In: Maletti, A. (ed.) Conference Algebraic Informatics, CAI 2015. LNCS, vol. 9270, pp. 90–102. Springer (2015)
15. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. EATCS Monographs in Theoretical Computer Science, Springer (2009)
16. Droste, M., Kuske, D.: Weighted automata. In: Pin, J.E. (ed.) Handbook: "Automata: from Mathematics to Applications". Europ. Mathematical Soc. (to appear)
17. Droste, M., Meinecke, I.: Weighted automata and weighted MSO logics for average and long-time behaviors. Inf. Comput. 220, 44–59 (2012)
18. Droste, M., Perevoshchikov, V.: A Nivat theorem for weighted timed automata and weighted relative distance logic. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) International Colloquium on Automata, Languages, and Programming, ICALP 2014, Part II. LNCS, vol. 8573, pp. 171–182. Springer (2014)
19. Droste, M., Pibaljommee, B.: Weighted nested word automata and logics over strong bimonoids. Int. J. Found. Comput. Sci. 25(5), 641–666 (2014)
20. Droste, M., Vogler, H.: Weighted tree automata and weighted logics. Theor. Comput. Sci. 366(3), 228–247 (2006)
21. Droste, M., Vogler, H.: Weighted automata and multi-valued logics over arbitrary bounded lattices. Theor. Comput. Sci. 418, 14–36 (2012)
22. Eilenberg, S.: Automata, Languages, and Machines, Pure and Applied Mathematics, vol. 59-A. Academic Press (1974)
23. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Am. Math. Soc. 98(1), 21–52 (1961)
24. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, Volume B, pp. 995–1072. MIT Press (1990)
25. Fischer, M.J.: Some properties of precedence languages. In: STOC '69: Proc. first annual ACM Symp. on Theory of Computing. pp. 181–190. ACM, New York, NY, USA (1969)

26. Floyd, R.W.: Syntactic analysis and operator precedence. J. ACM 10(3), 316–333 (1963)
27. Gastin, P., Monmege, B.: A unifying survey on weighted logics and weighted automata. Soft Comput. 22, 1047–1065 (2018), http://dx.doi.org/10.1007/s00500-015-1952-6
28. Harrison, M.A.: Introduction to Formal Language Theory. Addison Wesley (1978)
29. Knuth, D.E.: Semantics of context-free languages. Mathematical Systems Theory 2(2), 127–145 (1968)
30. Kuich, W., Salomaa, A.: Semirings, Automata, Languages, EATCS Monographs in Theoretical Computer Science, vol. 6. Springer (1986)
31. Lautemann, C., Schwentick, T., Thérien, D.: Logics for context-free languages. In: Pacholski, L., Tiuryn, J. (eds.) Computer Science Logic, Selected Papers. LNCS, vol. 933, pp. 205–216. Springer (1994)
32. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: Operator precedence languages: Their automata-theoretic and logic characterization. SIAM J. Comput. 44(4), 1026–1088 (2015)
33. Mathissen, C.: Weighted logics for nested words and algebraic formal power series. Logical Methods in Computer Science 6(1) (2010), selected papers of ICALP 2008
34. McNaughton, R.: Parenthesis grammars. J. ACM 14(3), 490–500 (1967)
35. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge, USA (1971)
36. Mehlhorn, K.: Pebbling mountain ranges and its application of DCFL-recognition. In: Automata, Languages and Programming, ICALP 1980. LNCS, vol. 85, pp. 422–435 (1980)
37. Nivat, M.: Transductions des langages de Chomsky. Ann. de l'Inst. Fourier 18, 339–455 (1968)
38. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science, Springer (1978)
39. Schützenberger, M.P.: On the definition of a family of automata. Inf. Control 4(2-3), 245–270 (1961)
40. Thatcher, J.: Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. Journ. of Comp. and Syst.Sc. 1, 317–322 (1967)
41. Trakhtenbrot, B.A.: Finite automata and logic of monadic predicates (in Russian). Doklady Akademii Nauk SSR 140, 326–329 (1961)