Derived-term Automata for Extended Weighted Rational Expressions

Akim Demaille akim@lrde.epita.fr

EPITA Research and Development Laboratory (LRDE)
14-16, rue Voltaire, 94276 Le Kremlin-Bicêtre, France

Abstract. We present an algorithm to build an automaton from a rational expression. This approach introduces support for extended weighted expressions. Inspired by derived-term based algorithms, its core relies on a different construct, *rational expansions*. We introduce an inductive algorithm to compute the expansion of an expression from which the automaton follows. This algorithm is independent of the size of the alphabet, and actually even supports infinite alphabets. It can easily be accommodated to generate deterministic (weighted) automata. These constructs are implemented in Vcsn, a free-software platform dedicated to weighted automata and rational expressions.

1 Introduction

Foundational to Automata Theory, the Kleene Theorem (and its weighted extension, the Kleene–Schützenberger Theorem) states the equivalence of *recognizability* —the property of being accepted by an automaton— and *rationality* —the property of being defined by a *rational*, or *regular*, expression. Numerous constructive proofs (read *algorithms*) have been proposed to go from rational expressions to automata, and vice versa. This paper focuses on building an automaton from an expression.

In 1961 Glushkov [12] provides an algorithm to build a nondeterministic automaton (without spontaneous transitions) now often called the standard (or position, or Glushkov) automaton. Earlier (1960), McNaughton and Yamada [15] proposed the same construct for *extended* rational expressions (i.e., including intersection and complement operators), but performed the now usual subset-automaton construction on-the-fly, thus yielding a deterministic automaton. A key ingredient of these algorithms is that they build an automaton whose states represent positions in the rational expression, and computations on these automata actually represent "executions" of the rational expression.

In 1964 Brzozowski [4] shows that *extended* expressions can be used directly as acceptors: following a transition corresponds to computing the left-quotient of the current expression by the current letter. With a proper equivalence relation between expressions (namely ACI: associativity, commutativity, and idempotence of the addition), Brzozowski shows that there is a finite number of equivalence classes of such quotients, called *derivatives*. This leads to a very natural construction of a *deterministic* automaton whose states are these derivatives. A rather

discreet sentence [4, last line of p. 484] introduces the concept of *expansion*, which is not further developed.

In 1996 Antimirov [3] introduces a novel idea: do not apply ACI equivalence globally; rather, when computing the derivative of an expression which is a sum, split it in a set of *partial derivatives* (or *derived terms*) — which amounts to limiting ACI to the sums that are at the very upper level of the expression. A key feature of the built automaton is that it is non-deterministic; as a result the worst-case size of the resulting automaton (i.e., its number of states) is linear in the size of the expression, instead of exponential with Brzozowski's construct. Antimirov also suggests *not* to rely on derivation in implementations, but on so called *linear forms*, which are closely related to Brzozowski's expansions; derivation is used to prove correctness.

In 2005 Lombardy and Sakarovitch [14] generalize the computation of the derivation and derived-term automaton to support weights. Since, as is well-known, not all weighted non-deterministic automata can be determinized, their construction relies on a generalization of Antimirov's derived-term that generates a *non-deterministic* automaton. In their formalization, Antimirov's sets of derived terms naturally turn into *weighted* sets —each term is associated with a weight— that they name *polynomials* (of expressions). However, linear forms completely disappear, and the construction of the derived-term automaton relies on derivatives. Independently Rutten [18] proposes a similar construction.

In 2011, Caron et al. [5] complete Antimirov's construct to support extended expressions. This is at the price of a new definition of derivatives: sets of sets of expressions, interpreted as disjunctions of conjunctions of expressions.

The contributions of this paper are threefold. Firstly, we introduce *expansions*, which generalize Brzozowski's expansions and Antimirov's linear forms to support weighted expressions; they bind together the derivatives, the constant terms and the *firsts* of an expression (letters with which words of the language/series start). They make the computation of the derived-term automaton independent of the size of the alphabet, and actually completely eliminate the need for the alphabet to be finite. Secondly, we provide support for extended weighted rational expressions, which generalizes both Lombardy and Sakarovitch [14] and Caron et al. [5]. And thirdly, we introduce a variation of this algorithm to build *deterministic* (weighted) automata.

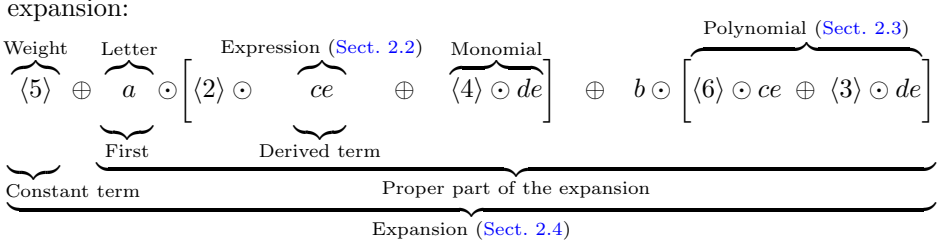
We first settle the notations in Sect. 2, provide an algorithm to compute the expansion of an expression in Sect. 3, which is used in Sect. 4 to propose an alternative construction of the derived-term automaton. In Sect. 5 we expose related work and conclude in Sect. 6.

The concepts introduced here are implemented in Vcsn. Vcsn is a free-software platform dedicated to weighted automata and rational expressions [10]. It supports both derivations and expansions, as exposed in this paper, and the corresponding constructions of the derived-term automaton[1].

[1] See the interactive environment, http://vcsn-sandbox.lrde.epita.fr, or http://vcsn.lrde.epita.fr/dload/2.3/notebooks/expression.derived_term.html, its

2 Notations

Our purpose is to define, compute, and use *rational expansions*. They intend to be to the differentiation (derivation) of rational expressions what differential forms are to the differentiation of functions. Defining expansions requires several concepts, defined bottom-up in this section. The following figure should help understanding these different entities, how they relate to each other, and where we are heading to: given a weighted rational expression $\mathsf{E}_1 = \langle 5\rangle 1 + \langle 2\rangle ace + \langle 6\rangle bce + \langle 4\rangle ade + \langle 3\rangle bde$ (weights are written in angle brackets), compute its expansion:



It is helpful to think of expansions as a normal form for expressions.

2.1 Rational Series

Series are to weighted automata what languages are to Boolean automata. Not all languages are rational (denoted by an expression), and similarly, not all series are rational (denoted by a weighted expression). We follow Sakarovitch [19].

Let A be a (finite) alphabet, and $\langle \mathbb{K}, +, \cdot, 0_{\mathbb{K}}, 1_{\mathbb{K}}\rangle$ a semiring whose (possibly non commutative) multiplication will be denoted by implicit concatenation. A (formal power) *series* over A^* with *weights* (or *multiplicities*) in $\mathbb{K}$ is any map from A^* to $\mathbb{K}$. The weight of a word m in a series s is denoted $s(m)$. The *support* of a series s is the language of words that have a non-zero weight in s. The *empty* series, $m \mapsto 0_{\mathbb{K}}$, is denoted 0; for any word u (including ε), u denotes the series $m \mapsto 1_{\mathbb{K}}$ if $m = u, 0_{\mathbb{K}}$ otherwise. Equipped with the pointwise addition ($s + t \coloneqq m \mapsto s(m) + t(m)$) and the Cauchy product ($s \cdot t \coloneqq m \mapsto \sum_{u,v\in A^*|u\cdot v=m} s(u)\cdot t(v)$) as multiplication, the set of these series forms a semiring denoted $\langle \mathbb{K}\langle\langle A^*\rangle\rangle, +, \cdot, 0, \varepsilon\rangle$.

The *constant term* of a series s, denoted s_ε, is $s(\varepsilon)$, the weight of the empty word. A series s is *proper* if $s_\varepsilon = 0_{\mathbb{K}}$. The *proper part* of s, denoted s_p, is the proper series which coincides with s on non empty words: $s = s_\varepsilon + s_p$.

A weight $k \in \mathbb{K}$ is *starrable* if its *star*, $k^* \coloneqq \sum_{n\in\mathbb{N}} k^n$, is defined. To ensure semantic soundness, we suppose that $\mathbb{K}$ is a *topological semiring*, i.e., it is equipped with a topology, and both addition and multiplication are continuous. Besides, it is supposed to be *strong*, i.e., the product of two summable families is summable. This ensures that $\mathbb{K}\langle\langle A^*\rangle\rangle$, equipped with the product topology derived from the topology on $\mathbb{K}$, is also a strong topological semiring. The *star* of a series is an infinite sum: $s^* \coloneqq \sum_{n\in\mathbb{N}} s^n$.

documentation, or this paper's companion notebook, http://vcsn.lrde.epita.fr/dload/2.3/notebooks/ICTAC-2016.html.

Proposition 1. *Let $\mathbb{K}$ be a strong topological semiring. Let $s \in \mathbb{K}\langle\langle A^* \rangle\rangle$, s^* is defined iff s_ε^* is defined and then $s^* = s_\varepsilon^* + s_\varepsilon^* s_p s^*$.*

Proof. By [19, Prop. 2.6, p. 396] s^* is defined iff s_ε^* is defined and then $s^* = (s_\varepsilon^* s_p)^* s_\varepsilon^* = s_\varepsilon^* (s_p s_\varepsilon^*)^*$. The result then follows directly from $s^* = \varepsilon + ss^*$: $s^* = s_\varepsilon^*(s_p s_\varepsilon^*)^* = s_\varepsilon^*(\varepsilon + (s_p s_\varepsilon^*)(s_p s_\varepsilon^*)^*) = s_\varepsilon^* + s_\varepsilon^* s_p (s_\varepsilon^*(s_p s_\varepsilon^*)^*) = s_\varepsilon^* + s_\varepsilon^* s_p s^*$. □

Rational languages are closed under intersection. When the semiring is commutative, series support a natural generalization of intersection, the Hadamard product, which we name *conjunction* and denote &. The conjunction of series s and t is defined as $s \,\&\, t \coloneqq m \mapsto s(m) \cdot t(m)$.

Rational languages are also closed under complement, but there is no unique consensus for a generalization for series. In the sequel, we will rely on the following definition: "s^c is the characteristic series of the complement of the support of s." More precisely, $s^c(m) \coloneqq s(m)^c$ where $\forall k \in \mathbb{K}, k^c \coloneqq 1_\mathbb{K}$ if $k = 0_\mathbb{K}$, $0_\mathbb{K}$ otherwise.

Proposition 2. *For series $s, s', t, t', s_a, t_a \in \mathbb{K}\langle\langle A^* \rangle\rangle$ with $a \in A$, for $S, T \subseteq A$, and weights $k, h, s_\varepsilon, t_\varepsilon \in \mathbb{K}$:*

$$(s + s') \,\&\, t = s \,\&\, t + s' \,\&\, t \qquad s \,\&\, (t + t') = s \,\&\, t + s \,\&\, t' \tag{1}$$

$$(ks) \,\&\, (ht) = (kh)(s \,\&\, t) \qquad \textit{when } \mathbb{K} \textit{ is commutative} \tag{2}$$

$$\Big(s_\varepsilon + \sum_{a \in S} a \cdot s_a\Big) \,\&\, \Big(t_\varepsilon + \sum_{a \in T} a \cdot t_a\Big) = s_\varepsilon t_\varepsilon + \sum_{a \in S \cap T} a \cdot (s_a \,\&\, t_a) \tag{3}$$

$$\Big(s_\varepsilon + \sum_{a \in S} a \cdot s_a\Big)^c = s_\varepsilon^c + \sum_{a \in S} a \cdot s_a^c + \sum_{a \in A \setminus S} a \cdot 0^c \tag{4}$$

From now on, when conjunction is used, we implicitly assume that the semiring is commutative.

2.2 Extended Weighted Rational Expressions

Several definitions of the weighted rational expressions compete, for instance (i) depending where the weights are expressions by themselves [8], or only appear as left- and right-exterior products [14, 1], and (ii) on the representation of the empty word: as a special expression "1" [14], or as a simple label "ε" belonging to $\Sigma \cup \{\varepsilon\}$ [13, Sec. 3.1], or finally, as a simple instance of a weight-as-expression for $1_\mathbb{K}$ [8]. We follow Sakarovitch [19, Def. III.2.3 p. 399].

Definition 1 (Extended Weighted Rational Expression). *A* rational (or regular) expression E *is a term built from the following grammar, where $a \in A$ is a letter, and $k \in \mathbb{K}$ a weight:* $\mathsf{E} ::= 0 \mid 1 \mid a \mid \mathsf{E}{+}\mathsf{E} \mid \langle k \rangle \mathsf{E} \mid \mathsf{E}\langle k \rangle \mid \mathsf{E}{\cdot}\mathsf{E} \mid \mathsf{E}^* \mid \mathsf{E}\&\mathsf{E} \mid \mathsf{E}^c$.

Since the product of $\mathbb{K}$ does not need to be commutative (unless conjunction is used) there are two exterior products: $\langle k \rangle \mathsf{E}$ and $\mathsf{E}\langle k \rangle$. The *size* (aka *length*) of an expression E, $|\mathsf{E}|$, is its number of symbols, excluding parentheses; its *width* (aka *literal length*), $\|\mathsf{E}\|$, is the number of occurrences of letters.

Rational expressions are syntactic objects; they provide a finite notation for (some) series, which are semantic objects.

Definition 2 (Series Denoted by an Expression). *Let* E *be an expression. The series denoted by* E*, noted* $[\![\mathsf{E}]\!]$*, is defined by induction on* E*:*

$$[\![0]\!] \coloneqq 0 \qquad [\![1]\!] \coloneqq \varepsilon \qquad [\![a]\!] \coloneqq a$$
$$[\![\mathsf{E}+\mathsf{F}]\!] \coloneqq [\![\mathsf{E}]\!] + [\![\mathsf{F}]\!] \qquad [\![\langle k\rangle\mathsf{E}]\!] \coloneqq k[\![\mathsf{E}]\!] \qquad [\![\mathsf{E}\langle k\rangle]\!] \coloneqq [\![\mathsf{E}]\!]k$$
$$[\![\mathsf{E}\cdot\mathsf{F}]\!] \coloneqq [\![\mathsf{E}]\!]\cdot[\![\mathsf{F}]\!] \quad [\![\mathsf{E}^*]\!] \coloneqq [\![\mathsf{E}]\!]^* \quad [\![\mathsf{E}\,\&\,\mathsf{F}]\!] \coloneqq [\![\mathsf{E}]\!]\,\&\,[\![\mathsf{F}]\!] \quad [\![\mathsf{E}^c]\!] \coloneqq [\![\mathsf{E}]\!]^c$$

An expression is *valid* if it denotes a series. More specifically, this requires that $[\![\mathsf{F}]\!]^*$ is well defined for each subexpression of the form F^*, i.e., that the constant term of $[\![\mathsf{F}]\!]$ is *starrable* in $\mathbb{K}$ (Proposition 1). So for instance, $1_{\mathbb{K}}^*$ and $(a^*)^*$ are valid in $\mathbb{B}$, but invalid in $\mathbb{Q}$. This definition of validity, which involves series (semantics) to define a property of expressions (syntax), will be made effective (syntactic) with the appropriate definition of the constant term $c(\mathsf{E})$ *of an expression* E (Definition 8).

Two expressions E and F are *equivalent* iff $[\![\mathsf{E}]\!] = [\![\mathsf{F}]\!]$. Some expressions are "trivially equivalent"; any candidate expression will be rewritten via the following *trivial identities*. Any subexpression of a form listed to the left of a '$\Rightarrow$' is rewritten as indicated on the right.

$$\mathsf{E}+0 \Rightarrow \mathsf{E} \qquad 0+\mathsf{E} \Rightarrow \mathsf{E}$$
$$\langle 0_{\mathbb{K}}\rangle\mathsf{E} \Rightarrow 0 \quad \langle 1_{\mathbb{K}}\rangle\mathsf{E} \Rightarrow \mathsf{E} \quad \langle k\rangle 0 \Rightarrow 0 \quad \langle k\rangle\langle h\rangle\mathsf{E} \Rightarrow \langle kh\rangle\mathsf{E}$$
$$\mathsf{E}\langle 0_{\mathbb{K}}\rangle \Rightarrow 0 \quad \mathsf{E}\langle 1_{\mathbb{K}}\rangle \Rightarrow \mathsf{E} \quad 0\langle k\rangle \Rightarrow 0 \quad \mathsf{E}\langle k\rangle\langle h\rangle \Rightarrow \mathsf{E}\langle kh\rangle$$
$$(\langle k\rangle\mathsf{E})\langle h\rangle \Rightarrow \langle k\rangle(\mathsf{E}\langle h\rangle) \qquad \ell\langle k\rangle \Rightarrow \langle k\rangle\ell$$
$$\mathsf{E}\cdot 0 \Rightarrow 0 \qquad 0\cdot\mathsf{E} \Rightarrow 0$$
$$(\langle k\rangle^?1)\cdot\mathsf{E} \Rightarrow \langle k\rangle\mathsf{E} \qquad \mathsf{E}\cdot(\langle k\rangle^?1) \Rightarrow \mathsf{E}\langle k\rangle$$
$$0^\star \Rightarrow 1$$
$$\mathsf{E}\,\&\,0 \Rightarrow 0 \qquad 0\,\&\,\mathsf{E} \Rightarrow 0 \qquad \mathsf{E}\,\&\,0^c \Rightarrow \mathsf{E} \qquad 0^c\,\&\,\mathsf{E} \Rightarrow \mathsf{E}$$
$$\langle k\rangle^?\ell\,\&\,\langle h\rangle^?\ell \Rightarrow \langle kh\rangle\ell \qquad \langle k\rangle^?\ell\,\&\,\langle h\rangle^?\ell' \Rightarrow 0$$
$$(\langle k\rangle\mathsf{E})^c \Rightarrow \mathsf{E}^c \qquad (\mathsf{E}\langle k\rangle)^c \Rightarrow \mathsf{E}^c$$

where E stands for a rational expression, $a \in A$ is a letter, $\ell, \ell' \in A \cup \{1\}$ denote two different labels, $k, h \in \mathbb{K}$ are weights, and $\langle k\rangle^?\ell$ denotes either $\langle k\rangle\ell$, or ℓ in which case $k = 1_{\mathbb{K}}$ in the right-hand side of $\Rightarrow$. The choice of these identities is beyond the scope of this paper (see Lombardy and Sakarovitch [14, p. 149]), however note that, with the exception of the last line, they are limited to trivial properties; in particular *linearity* ("weighted ACI": associativity, commutativity, and $\langle k\rangle\mathsf{E} + \langle h\rangle\mathsf{E} \Rightarrow \langle k+h\rangle\mathsf{E}$) is not enforced — polynomials will take care of it (Sect. 2.3). In practice, additional identities help reducing the number of derived terms [17], hence the final automaton size. The last two rules, about complement, will be discussed in Sect. 4.3; they are disabled when $\mathbb{K}$ has zero divisors for cases such as $(\langle x\rangle(a + \langle y\rangle b))^c$ with $xy = 0_{\mathbb{K}}$.

Example 1. Conjunction and complement can be combined to define new operators which are convenient syntactic sugar. For instance, $\mathsf{E} \lessdot \mathsf{F} \coloneqq \mathsf{E} + (\mathsf{E}^c\,\&\,\mathsf{F})$

allows to define a left-biased + operator: $[\![\mathsf{E} \lhd \mathsf{F}]\!](u) = [\![\mathsf{E}]\!](u)$ if $[\![\mathsf{E}]\!](u) \neq 0_{\mathbb{K}}$, $[\![\mathsf{F}]\!](u)$ otherwise. The following example mocks Lex-like scanners: identifiers are non-empty sequences of letters of $\{a, b\}$ that are not reserved keywords. The expression $\mathsf{E}_3 \coloneqq \langle 2\rangle ab \lhd \langle 3\rangle(a+b)^+$, with weights in $\mathbb{Z}$, maps the "keyword" ab to 2, and "identifiers" to 3. Once desugared and simplified by the trivial identities, we have $\mathsf{E}_3 = \langle 2\rangle ab + ((ab)^c \,\&\, \langle 3\rangle((a+b)(a+b)^*))$.

2.3 Rational Polynomials

At the core of the idea of "partial derivatives" introduced by Antimirov [3], is that of *sets* of rational expressions, later generalized in *weighted sets* by Lombardy and Sakarovitch [14], i.e., functions (partial, with finite domain) from the set of rational expressions into $\mathbb{K} \setminus \{0_{\mathbb{K}}\}$. It proves useful to view such structures as "polynomials of rational expressions". In essence, they capture the linearity of addition.

Definition 3 (Rational Polynomial). *A* polynomial *(of rational expressions) is a finite (left) linear combination of rational expressions. Syntactically it is represented by a term built from the grammar* $\mathsf{P} ::= 0 \mid \langle k_1\rangle \odot \mathsf{E}_1 \oplus \cdots \oplus \langle k_n\rangle \odot \mathsf{E}_n$ *where* $k_i \in \mathbb{K} \setminus \{0_{\mathbb{K}}\}$ *denote* non-null *weights, and* E_i *denote* non-null *expressions. Expressions may not appear more than once in a polynomial. A* monomial *is a pair* $\langle k_i\rangle \odot \mathsf{E}_i$. *The* terms *of* P *is the set* $\mathsf{exprs}(\mathsf{P}) \coloneqq \{\mathsf{E}_1, \ldots, \mathsf{E}_n\}$.

We use specific symbols ($\odot$ and $\oplus$) to clearly separate the outer polynomial layer from the inner expression layer. A polynomial P of expressions can be "projected" as a rational expression $\mathsf{expr}(\mathsf{P})$ by mapping its sum and left-multiplication by a weight onto the corresponding operators on rational expressions. This operation is performed on a canonical form of the polynomial (expressions are sorted in a well defined order). Polynomials denote series: $[\![\mathsf{P}]\!] \coloneqq [\![\mathsf{expr}(\mathsf{P})]\!]$.

Example 2. Let $\mathsf{E}_1 \coloneqq \langle 5\rangle 1 + \langle 2\rangle ace + \langle 6\rangle bce + \langle 4\rangle ade + \langle 3\rangle bde$. The polynomial '$\mathsf{P}_{1a} \coloneqq \langle 2\rangle \odot ce \oplus \langle 4\rangle \odot de$' has two monomials: '$\langle 2\rangle \odot ce$' and '$\langle 4\rangle \odot de$'. It denotes the (left) quotient of $[\![\mathsf{E}_1]\!]$ by a, and '$\mathsf{P}_{1b} \coloneqq \langle 6\rangle \odot ce \oplus \langle 3\rangle \odot de$' the quotient by b.

Let $\mathsf{P} = \langle k_1\rangle \odot \mathsf{E}_1 \oplus \cdots \oplus \langle k_n\rangle \odot \mathsf{E}_n$ be a polynomial, k a weight (possibly null) and F an expression (possibly null), we introduce the following operations:

$$\begin{aligned}
\mathsf{P} \cdot \mathsf{F} &\coloneqq \langle k_1\rangle \odot (\mathsf{E}_1 \cdot \mathsf{F}) \oplus \cdots \oplus \langle k_n\rangle \odot (\mathsf{E}_n \cdot \mathsf{F}) \\
\langle k\rangle \mathsf{P} &\coloneqq \langle kk_1\rangle \odot \mathsf{E}_1 \oplus \cdots \oplus \langle kk_n\rangle \odot \mathsf{E}_n \\
\mathsf{P}\langle k\rangle &\coloneqq \langle k_1\rangle \odot (\mathsf{E}_1\langle k\rangle) \oplus \cdots \oplus \langle k_n\rangle \odot (\mathsf{E}_n\langle k\rangle) \\
\mathsf{P}_1 \,\&\, \mathsf{P}_2 &\coloneqq \bigoplus_{\substack{\langle k_1\rangle \odot \mathsf{E}_1 \in \mathsf{P}_1 \\ \langle k_2\rangle \odot \mathsf{E}_2 \in \mathsf{P}_2}} \langle k_1 k_2\rangle \odot (\mathsf{E}_1 \,\&\, \mathsf{E}_2) \qquad \mathsf{P}^c \coloneqq \langle 1_{\mathbb{K}}\rangle \odot \mathsf{expr}(\mathsf{P})^c
\end{aligned} \tag{5}$$

Trivial identities might simplify the result, e.g., $(\langle 1_{\mathbb{K}}\rangle \odot a) \,\&\, (\langle 1_{\mathbb{K}}\rangle \odot b) = \langle 1_{\mathbb{K}}\rangle \odot (a \,\&\, b) = 0$. Note the asymmetry between left and right exterior products. The addition of polynomials is commutative, multiplication by zero (be it an expression or a weight) evaluates to the null polynomial, and the left-multiplication by a weight is distributive.

Lemma 1. $[\![\mathsf{P} \cdot \mathsf{F}]\!] = [\![\mathsf{P}]\!] \cdot [\![\mathsf{F}]\!]$ $\quad [\![\langle k\rangle \mathsf{P}]\!] = \langle k\rangle [\![\mathsf{P}]\!]$ $\quad [\![\mathsf{P}\langle k\rangle]\!] = [\![\mathsf{P}]\!]\langle k\rangle$
$[\![\mathsf{P}_1 \,\&\, \mathsf{P}_2]\!] = [\![\mathsf{P}_1]\!] \,\&\, [\![\mathsf{P}_2]\!]$ $\quad [\![\mathsf{P}^c]\!] = [\![\mathsf{P}]\!]^c$.

Proof. The first three are trivial. The case of & follows from (1) and (2). Complement follows from its definition: $[\![\mathsf{P}^c]\!] = [\![\langle 1_{\mathbb{K}}\rangle \odot \mathsf{expr}(\mathsf{P})^c]\!] = [\![\mathsf{expr}(\mathsf{P})^c]\!] = [\![\mathsf{expr}(\mathsf{P})]\!]^c = [\![\mathsf{P}]\!]^c$. □

2.4 Rational Expansions

Expansions group together a distinguished weight, and, for each letter, its associated polynomial. Let $[n]$ denote $\{1, \dots, n\}$.

Definition 4 (Rational Expansion). *A* rational expansion X *is a term built from the grammar* $\mathsf{X} ::= \langle k\rangle \oplus a_1 \odot [\mathsf{P}_1] \oplus \cdots \oplus a_n \odot [\mathsf{P}_n]$ *where* $k \in \mathbb{K}$ *is a weight (possibly null),* $a_i \in A$ *letters (occurring at most once), and* P_i *non-null polynomials. The* constant term *is* k*, the* proper part *is* $a_1 \odot [\mathsf{P}_1] \oplus \cdots \oplus a_n \odot [\mathsf{P}_n]$*, the* firsts *is* $\{a_1, \dots, a_n\}$ *(possibly empty), and the* terms *are* $\mathsf{exprs}(\mathsf{X}) \coloneqq \bigcup_{i\in[n]} \mathsf{exprs}(\mathsf{P}_i)$.

To ease reading, polynomials are written in square brackets. Contrary to expressions and polynomials, there is no specific term for the empty expansion: it is represented by $\langle 0_{\mathbb{K}}\rangle$, the null weight. Except for this case, null constant terms are left implicit. Besides their support for weights, expansions differ from Antimirov's linear forms in that they integrate the constant term, which gives them a flavor of series. Given an expansion X, we denote by X_ε (or $\mathsf{X}(\varepsilon)$) its constant term, by $f(\mathsf{X})$ its firsts, by X_p its proper part, and by X_a (or $\mathsf{X}(a)$) the polynomial corresponding to a in X, or the null polynomial if $a \notin f(\mathsf{X})$. Expansions will thus be written: $\mathsf{X} = \langle \mathsf{X}_\varepsilon\rangle \oplus \bigoplus_{a\in f(\mathsf{X})} a \odot [\mathsf{X}_a]$.

An expansion whose polynomials are monomials is said to be *deterministic*. An expansion X can be "projected" as a rational expression $\mathsf{expr}(\mathsf{X})$ by mapping weights, letters and polynomials to their corresponding rational expressions, and $\oplus/\odot$ to the sum/concatenation of rational expressions. Again, this is performed on a canonical form of the expansion: letters and polynomials are sorted. Expansions also denote series: $[\![\mathsf{X}]\!] \coloneqq [\![\mathsf{expr}(\mathsf{X})]\!]$. An expansion X is said to be *equivalent* to an expression E iff $[\![\mathsf{X}]\!] = [\![\mathsf{E}]\!]$.

Example 3 (Example 2 continued). Expansion $\mathsf{X}_1 \coloneqq \langle 5\rangle \oplus a \odot [\mathsf{P}_{1a}] \oplus b \odot [\mathsf{P}_{1b}]$ has $\mathsf{X}_1(\varepsilon) = \langle 5\rangle$ as constant term, and maps the letter a (resp. b) to the polynomial $\mathsf{X}_1(a) = \mathsf{P}_{1a}$ (resp. $\mathsf{X}_1(b) = \mathsf{P}_{1b}$). X_1 can be proved to be equivalent to E_1.

Let X, Y be expansions, k a weight, and E an expression (all possibly null):

$$\mathsf{X} \oplus \mathsf{Y} \coloneqq \langle \mathsf{X}_\varepsilon + \mathsf{Y}_\varepsilon\rangle \oplus \bigoplus_{a\in f(\mathsf{X})\cup f(\mathsf{Y})} a \odot [\mathsf{X}_a \oplus \mathsf{Y}_a] \tag{6}$$

$$\langle k\rangle \mathsf{X} \coloneqq \langle k\mathsf{X}_\varepsilon\rangle \oplus \bigoplus_{a\in f(\mathsf{X})} a \odot [\langle k\rangle \mathsf{X}_a] \qquad \mathsf{X}\langle k\rangle \coloneqq \langle \mathsf{X}_\varepsilon k\rangle \oplus \bigoplus_{a\in f(\mathsf{X})} a \odot [\mathsf{X}_a\langle k\rangle] \tag{7}$$

$$\mathsf{X}\cdot\mathsf{E} \coloneqq \bigoplus_{a\in f(\mathsf{X})} a\odot[\mathsf{X}_a\cdot\mathsf{E}] \qquad \text{with } \mathsf{X} \text{ proper: } \mathsf{X}_\varepsilon = 0_{\mathbb{K}} \tag{8}$$

$$\mathsf{X}\,\&\,\mathsf{Y} \coloneqq \langle \mathsf{X}_\varepsilon \mathsf{Y}_\varepsilon\rangle \oplus \bigoplus_{a\in f(\mathsf{X})\cap f(\mathsf{Y})} a\odot[\mathsf{X}_a\,\&\,\mathsf{Y}_a] \tag{9}$$

$$\mathsf{X}^c \coloneqq \langle \mathsf{X}^c_\varepsilon\rangle \oplus \bigoplus_{a\in f(X)} a\odot[\mathsf{X}^c_a] \oplus \bigoplus_{a\in A\setminus f(X)} a\odot[\mathsf{0}^c] \tag{10}$$

Since by definition expansions never map to null polynomials, some firsts might be smaller sets than suggested by these equations. For instance in $\mathbb{Z}$ the sum of $\langle 1\rangle\oplus a\odot[\langle 1\rangle\odot b]$ and $\langle 1\rangle\oplus a\odot[\langle -1\rangle\odot b]$ is $\langle 2\rangle$, and $\big(a\odot[\langle 1\rangle\odot b]\big)\,\&\,\big(a\odot[\langle 1\rangle\odot c]\big)$ is $\langle 0\rangle$ since $b\,\&\,c \Rightarrow \mathsf{0}$. Note that X^c is a deterministic expansion.

The following lemma is simple to establish: lift semantic equivalences, such as those of Proposition 2, to syntax, using Lemma 1.

Lemma 2. $[\![\mathsf{X}\oplus\mathsf{Y}]\!] = [\![\mathsf{X}]\!] + [\![\mathsf{Y}]\!]$ $\quad [\![\langle k\rangle\mathsf{X}]\!] = \langle k\rangle[\![\mathsf{X}]\!]$ $\quad [\![\mathsf{X}\langle k\rangle]\!] = [\![\mathsf{X}]\!]\langle k\rangle$
$[\![\mathsf{X}\cdot\mathsf{E}]\!] = [\![\mathsf{X}]\!]\cdot[\![\mathsf{E}]\!]$ $\quad [\![\mathsf{X}\,\&\,\mathsf{Y}]\!] = [\![\mathsf{X}]\!]\,\&\,[\![\mathsf{Y}]\!]$ $\quad [\![\mathsf{X}^c]\!] = [\![\mathsf{X}]\!]^c$.

2.5 Weighted Automata

Definition 5 (Automaton). *A* weighted automaton $\mathcal{A}$ *is a tuple* $\langle A, \mathbb{K}, Q, E, I, T\rangle$ *where:*

- A *(the set of labels) is an alphabet (usually finite), and* $\mathbb{K}$ *(the set of weights) is a semiring,*
- Q *is a set of states,* I *and* T *are the* initial *and* final *functions from* Q *into* $\mathbb{K}$*,*
- E *is a (partial) function from* $Q\times A\times Q$ *into* $\mathbb{K}\setminus\{0_{\mathbb{K}}\}$*;*
  its domain represents the transitions*:* $(source, label, destination)$*.*

An automaton is *locally finite* if each state has a finite number of outgoing transitions ($\forall s\in Q, \{s\}\times A\times Q\cap E$ is finite). A *finite automaton* has a finite number of states. A *path* p in an automaton is a sequence of transitions $(q_0,a_0,q_1)(q_1,a_1,q_2)\cdots(q_n,a_n,q_{n+1})$ where the source of each is the destination of the previous one; its *label* is the word $a_0a_1\cdots a_n$, its *weight* is $I(q_0)\otimes E(q_0,a_0,q_1)\otimes\cdots\otimes E(q_n,a_n,q_{n+1})\otimes T(q_{n+1})$. The *evaluation* of word u by a locally finite automaton $\mathcal{A}$, $\mathcal{A}(u)$, is the (finite) sum of the weights of all the paths labeled by u, or $0_{\mathbb{K}}$ if there are no such path. The *behavior* of such an automaton $\mathcal{A}$ is the series $[\![\mathcal{A}]\!] \coloneqq u\mapsto\mathcal{A}(u)$. A state q is *initial* if $I(q)\neq 0_{\mathbb{K}}$. A state q is *accessible* if there is a path from an initial state to q. The *accessible* part of an automaton $\mathcal{A}$ is the subautomaton whose states are the accessible states of $\mathcal{A}$. The size of a finite automaton, $|\mathcal{A}|$, is its number of states.

Definition 6 (Semantics of a State). *Given a weighted automaton* $\mathcal{A} = \langle A, \mathbb{K}, Q, E, I, T\rangle$*, inductively*[2] *define a semantic mapping* $[\![-]\!] : Q\to\mathbb{K}\langle\langle A^*\rangle\rangle$ *as follows:*

[2] The induction is on the length of the word u in $[\![q]\!](u)$, which is defined for all q and all words of the given length simultaneously.

- *For all* $q \in Q$, $[\![q]\!](\varepsilon) \coloneqq T(q)$.
- *For all* $q \in Q$, $a \in A$, *and* $u \in A^*$, $[\![q]\!](au) \coloneqq \sum_{q' \in Q} E(q, a, q')\Big([\![q']\!](u)\Big)$.

It follows by a simple inductive proof from this definition, that for all $u = a_1 \dots a_n$,

$$[\![q]\!](u) = \sum_{q_1 \in Q} \cdots \sum_{q_{n+1} \in Q} E(q, a_0, q_1) \cdots E(q_n, a_n, q_{n+1}) T(q_{n+1})$$

and from here it follows directly from the definition of $[\![\mathcal{A}]\!]$ that:

$$[\![\mathcal{A}]\!](u) = \sum_{q \in Q} I(q)\big([\![q]\!](u)\big) \tag{11}$$

We are interested, given an expression E, in an algorithm to compute an automaton $\mathcal{A}_\mathsf{E}$ such that $[\![\mathcal{A}_\mathsf{E}]\!] = [\![\mathsf{E}]\!]$ (Sect. 4). To this end, we first introduce a simple recursive procedure to compute *the* expansion of an expression.

3 Computing Expansions of Expressions

3.1 Expansion of a Rational Expression

Definition 7 (Expansion of a Rational Expression). *The* expansion of a rational expression E, *written* $d(\mathsf{E})$, *is the expansion defined inductively as follows:*

$$d(\mathsf{0}) \coloneqq \langle 0_\mathbb{K} \rangle \qquad d(\mathsf{1}) \coloneqq \langle 1_\mathbb{K} \rangle \qquad d(a) \coloneqq a \odot [\langle 1_\mathbb{K} \rangle \odot \mathsf{1}] \tag{12}$$

$$d(\mathsf{E} + \mathsf{F}) \coloneqq d(\mathsf{E}) \oplus d(\mathsf{F}) \qquad d(\langle k \rangle \mathsf{E}) \coloneqq \langle k \rangle d(\mathsf{E}) \qquad d(\mathsf{E}\langle k \rangle) \coloneqq d(\mathsf{E})\langle k \rangle \tag{13}$$

$$d(\mathsf{E} \cdot \mathsf{F}) \coloneqq d_p(\mathsf{E}) \cdot \mathsf{F} \oplus \big\langle d_\varepsilon(\mathsf{E}) \big\rangle d(\mathsf{F}) \tag{14}$$

$$d(\mathsf{E}^*) \coloneqq \big\langle d_\varepsilon(\mathsf{E})^* \big\rangle \oplus \big\langle d_\varepsilon(\mathsf{E})^* \big\rangle d_p(\mathsf{E}) \cdot \mathsf{E}^* \tag{15}$$

$$d(\mathsf{E} \,\&\, \mathsf{F}) \coloneqq d(\mathsf{E}) \,\&\, d(\mathsf{F}) \tag{16}$$

$$d(\mathsf{E}^c) \coloneqq d(\mathsf{E})^c \tag{17}$$

where $d_\varepsilon(\mathsf{E}) \coloneqq d(\mathsf{E})_\varepsilon, d_p(\mathsf{E}) \coloneqq d(\mathsf{E})_p$ *are the constant term/proper part of* $d(\mathsf{E})$.

The right-hand sides are indeed expansions. The computation trivially terminates: induction is performed on strictly smaller subexpressions. These formulas are enough to compute the expansion of an expression; there is no secondary process for the firsts — indeed $d(a) \coloneqq a \odot [\langle 1_\mathbb{K} \rangle \odot \mathsf{1}]$ suffices and every other case simply propagates or assembles the firsts — or the constant terms. In an implementation a single recursive call to $d(\mathsf{E})$ is performed for (14) and (15), from which $d_\varepsilon(\mathsf{E})$ and $d_p(\mathsf{E})$ are obtained. So for instance (15) should rather be written: $d(\mathsf{E}^*) \coloneqq \texttt{let}\ \mathsf{X} = d(\mathsf{E})\ \texttt{in}\ \langle \mathsf{X}^*_\varepsilon \rangle \oplus \langle \mathsf{X}^*_\varepsilon \rangle \mathsf{X}_p \cdot \mathsf{E}^*$. Besides, existing expressions should be referenced to, not duplicated: in the previous piece of code, E^* is not built again, the input argument is reused.

Lemma 3. *For any expression* E, $[\![d(\mathsf{E})]\!] = [\![\mathsf{E}]\!]$.

Proof. Proved by induction over E. The proof is straightforward for (12), (13), (16) and (17), using Lemma 2. The case of multiplication, (14), follows from:

$$\begin{aligned}\llbracket d(\mathsf{E}\cdot\mathsf{F})\rrbracket &= \Big\llbracket d_p(\mathsf{E})\cdot\mathsf{F}\oplus\big\langle d_\varepsilon(\mathsf{E})\big\rangle\cdot d(\mathsf{F})\Big\rrbracket = \llbracket d_p(\mathsf{E})\rrbracket\cdot\llbracket\mathsf{F}\rrbracket + \big\langle d_\varepsilon(\mathsf{E})\big\rangle\cdot\llbracket d(\mathsf{F})\rrbracket\\ &= \llbracket d_p(\mathsf{E})\rrbracket\cdot\llbracket\mathsf{F}\rrbracket + \big\langle d_\varepsilon(\mathsf{E})\big\rangle\cdot\llbracket\mathsf{F}\rrbracket \text{ (by inductive hypothesis)}\\ &= \Big(\llbracket\langle d_\varepsilon(\mathsf{E})\rangle\rrbracket + \llbracket d_p(\mathsf{E})\rrbracket\Big)\cdot\llbracket\mathsf{F}\rrbracket = \Big\llbracket\big\langle d_\varepsilon(\mathsf{E})\big\rangle + d_p(\mathsf{E})\Big\rrbracket\cdot\llbracket\mathsf{F}\rrbracket\\ &= \llbracket d(\mathsf{E})\rrbracket\cdot\llbracket\mathsf{F}\rrbracket = \llbracket\mathsf{E}\rrbracket\cdot\llbracket\mathsf{F}\rrbracket \text{ (by inductive hypothesis)} = \llbracket\mathsf{E}\cdot\mathsf{F}\rrbracket\end{aligned}$$

It might seem more natural to exchange the two terms (i.e., $\big\langle d_\varepsilon(\mathsf{E})\big\rangle\cdot d(\mathsf{F})\oplus d_p(\mathsf{E})\cdot\mathsf{F}$), but an implementation first computes $d(\mathsf{E})$ and then computes $d(\mathsf{F})$ *only if* $d_\varepsilon(\mathsf{E})\neq 0_{\mathbb{K}}$. The case of Kleene star, (15), follows from Proposition 1. □

By Lemma 3, given an expression E and its expansion $d(\mathsf{E}) = \langle k\rangle\oplus a_1\odot[\mathsf{P}_1]\oplus\cdots\oplus a_n\odot[\mathsf{P}_n]$, we have $\llbracket\mathsf{E}\rrbracket = \llbracket\langle k\rangle\oplus a_1\odot[\mathsf{P}_1]\oplus\cdots\oplus a_n\odot[\mathsf{P}_n]\rrbracket$ and thus, by the definition of the semantics of an expansion we have: $\llbracket\mathsf{E}\rrbracket = k + a_1\cdot\llbracket\mathsf{P}_1\rrbracket + \ldots + a_n\cdot\llbracket\mathsf{P}_n\rrbracket$. Now, the following facts on the constant term and left quotients of $\llbracket\mathsf{E}\rrbracket$ immediately follow:

$$\llbracket\mathsf{E}\rrbracket_\varepsilon = k \qquad {a_i}^{-1}\llbracket\mathsf{E}\rrbracket = \llbracket\mathsf{P}_i\rrbracket \qquad \text{for } 1\le i\le n \tag{18}$$

3.2 Connection with Derivatives

We reproduce here the definition of constant terms and derivatives from Lombardy et al [14, p. 148 and Def. 2], with our notations and covering extended expressions. To facilitate reading, weights such as the constant term are written in angle brackets, although so far this was reserved to syntactic constructs.

Definition 8 (Constant Term and Derivative).

$$c(0) \coloneqq \langle 0_{\mathbb{K}}\rangle,\quad c(1)\coloneqq\langle 1_{\mathbb{K}}\rangle, \qquad \partial_a 0\coloneqq 0,\quad \partial_a 1\coloneqq 0, \tag{19}$$

$$c(a)\coloneqq\langle 0_{\mathbb{K}}\rangle,\forall a\in A, \qquad \partial_a b\coloneqq 1 \text{ if } b=a,\ 0 \text{ otherwise,} \tag{20}$$

$$c(\mathsf{E}+\mathsf{F})\coloneqq c(\mathsf{E})+c(\mathsf{F}), \qquad \partial_a(\mathsf{E}+\mathsf{F})\coloneqq\partial_a\mathsf{E}\oplus\partial_a\mathsf{F}, \tag{21}$$

$$c(\langle k\rangle\mathsf{E})\coloneqq\langle k\rangle c(\mathsf{E}), \qquad \partial_a(\langle k\rangle\mathsf{E})\coloneqq\langle k\rangle(\partial_a\mathsf{E}), \tag{22}$$

$$c(\mathsf{E}\langle k\rangle)\coloneqq c(\mathsf{E})\langle k\rangle, \qquad \partial_a(\mathsf{E}\langle k\rangle)\coloneqq(\partial_a\mathsf{E})\,\langle k\rangle, \tag{23}$$

$$c(\mathsf{E}\cdot\mathsf{F})\coloneqq c(\mathsf{E})\cdot c(\mathsf{F}), \qquad \partial_a(\mathsf{E}\cdot\mathsf{F})\coloneqq(\partial_a\mathsf{E})\cdot\mathsf{F}\oplus\big\langle c(\mathsf{E})\big\rangle\partial_a\mathsf{F}, \tag{24}$$

$$c(\mathsf{E}^*)\coloneqq c(\mathsf{E})^*, \qquad \partial_a\mathsf{E}^*\coloneqq\big\langle c(\mathsf{E})^*\big\rangle(\partial_a\mathsf{E})\cdot\mathsf{E}^* \tag{25}$$

$$c(\mathsf{E}\,\&\,\mathsf{F})\coloneqq c(\mathsf{E})\cdot c(\mathsf{F}), \qquad \partial_a(\mathsf{E}\,\&\,\mathsf{F})\coloneqq\partial_a\mathsf{E}\,\&\,\partial_a\mathsf{F}, \tag{26}$$

$$c(\mathsf{E}^c)\coloneqq c(\mathsf{E})^c, \qquad \partial_a\mathsf{E}^c\coloneqq(\partial_a\mathsf{E})^c \tag{27}$$

where (25) applies iff $c(\mathsf{E})^*$ is defined in $\mathbb{K}$.

The reader is invited to compare Definition 7 and Definition 8, which does not include the computation of the firsts.

The following lemma is a syntactic version of (18).

Lemma 4. *For any expression* E, $d(\mathsf{E})(\varepsilon) = c(\mathsf{E})$, *and* $d(\mathsf{E})(a) = \partial_a \mathsf{E}$.

Proof. A straightforward induction on E. The cases of constants and letters are immediate consequences of (19) and (20) on the one hand, and (12) on the other hand. Equation (13) matches (21) to (23). Multiplication (concatenation) is again barely a change of notation between (14) and (24), and likewise for the Kleene star ((15) and (25)). Conjunction, (26), follows from (9) and (16), and complement, (27), from (17) and (10). □

Lemma 4 states that expansions, like Antimirov's linear forms, offer a different means to compute the expression derivatives. However expansions seem to better capture the essence of the process, where the computations of constant terms are tightly coupled with that of the derivations. The formulas are more concise. Expansions are also "more complete" than derivations, viz., the expansion of an expression can be seen as a normal-form of this expression: $\mathsf{E} \equiv \mathsf{expr}\big(d(\mathsf{E})\big)$ and $d(\mathsf{E}) = d(\mathsf{expr}\big(d(\mathsf{E})\big))$. Expansions are more efficient to perform effective calculations, such as building an automaton (Sect. 4.4).

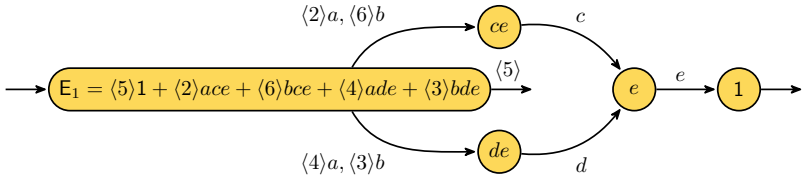
4 Expansion-Based Derived-Term Automaton

Definition 9 (Derived-Term Automaton). *The* derived-term automaton *of an expression* E *is the accessible part of the automaton* $\mathcal{A}_\mathsf{E} \coloneqq \langle A, \mathbb{K}, Q, E, I, T\rangle$ *defined as follows:*

- Q *is the set of rational expressions on alphabet* A *with weights in* $\mathbb{K}$,
- $E(\mathsf{F}, a, \mathsf{F}') = k$ *iff* $a \in f(d(\mathsf{F}))$ and $\langle k\rangle\mathsf{F}' \in d(\mathsf{F})(a)$,
- $I = \mathsf{E} \mapsto 1_\mathbb{K}$, $T(\mathsf{F}) = k$ *iff* $\langle k\rangle = d(\mathsf{F})(\varepsilon)$.

The resulting automaton is locally finite, and not necessarily deterministic.

Example 4 (Examples 2 and 3 continued). We have $d(\mathsf{E}_1) = \mathsf{X}_1$. $\mathcal{A}_{\mathsf{E}_1}$ is:



It is straightforward to extract an algorithm from Definition 9, using a work-list of states whose outgoing transitions to compute (see Algorithm 1). This approach admits a natural lazy implementation: the whole automaton is not computed at once, but rather, states and transitions are computed on-the-fly, on demand, for instance when evaluating a word.

Theorem 1. *Any (valid) expression* E *and its expansion-based derived-term automaton* $\mathcal{A}_\mathsf{E}$ *denote the same series, i.e.,* $[\![\mathcal{A}_\mathsf{E}]\!] = [\![\mathsf{E}]\!]$.

```
Input  : E, a rational expression
Output : ⟨E, I, T⟩ an automaton (simplified notation)
I(E) := 1_K ;                                   // Unique initial state
Q := Queue(E) ;                                 // A work list (queue) loaded with E
while Q is not empty do
  E := pop(Q) ;                                 // A new state/expression to complete
  X := d(E) ;                                   // The expansion of E
  T(E) := X(ε) ;                                // Final weight: the constant term
  foreach a ⊙ [P_a] ∈ X do                      // For each first/polynomial in X
    foreach ⟨k⟩ ⊙ F ∈ P_a do                     // For each monomial of P_a = X(a)
      E(E, a, F) := k ;                         // New transition
      if F ∉ Q then
        push(Q, F) ;                            // F is a new state, to complete later
      end
    end
  end
end
```

Algorithm 1: Building the derived-term automaton. The set of states is implicitly grown when transitions are added.

This theorem is a straightforward consequence of the following lemma. To make sure we do not get confused between, on the one hand, the semantics $[\![\mathsf{E}]\!]$ of expressions, and on the other hand the (as above defined) semantics of states in the derived term-automaton (which are also given by expressions), let the state associated with an expression E be denoted by q_{E}. Thus, the semantics of the state corresponding to E in the derived term automaton is written as $[\![q_{\mathsf{E}}]\!]$.

Lemma 5. *For all rational expressions* E, $[\![\mathsf{E}]\!] = [\![q_{\mathsf{E}}]\!]$.

Proof. We show by induction on the length of words u, that for all expressions E and all words u, $[\![\mathsf{E}]\!](u) = [\![q_{\mathsf{E}}]\!](u)$, from which the desired result follows.

Base case. Say we are given an expression E, with $d(\mathsf{E}) = \langle k\rangle \oplus a_1 \odot [\mathsf{P}_1] \oplus \cdots \oplus a_n \odot [\mathsf{P}_n]$. We have $[\![\mathsf{E}]\!](\varepsilon) = k$ by (18); similarly, by the definition of semantics of a state in an automaton and the definition of the derived term automaton, we have $[\![q_{\mathsf{E}}]\!](\varepsilon) = T(q_{\mathsf{E}}) = k$, as $k = d(\mathsf{E})(\varepsilon)$.

Inductive case. Assume the statement holds for all words u with $|u| < n$ and all expressions E. Now let $v = au$ be a word with $|v| = n$, and say, we have an expression E, again with $d(\mathsf{E}) = \langle k\rangle \oplus a_1 \odot [\mathsf{P}_1] \oplus \cdots \oplus a_n \odot [\mathsf{P}_n]$.

If $a = a_i$ for some i with $1 \le i \le n$, we then have $[\![\mathsf{E}]\!](au) = (a^{-1}[\![\mathsf{E}]\!])(u) = [\![\mathsf{P}_i]\!](u)$. Now let P_i be of the form

$$\mathsf{P}_i = \langle k_1\rangle \odot \mathsf{F}_1 \oplus \ldots \oplus \langle k_m\rangle \odot \mathsf{F}_m, \tag{28}$$

giving:

$[\![\mathsf{E}]\!](au) = [\![\mathsf{P}_i]\!](u)$

$$
\begin{aligned}
&= \llbracket \langle k_1 \rangle \odot \mathsf{F}_1 \oplus \ldots \oplus \langle k_m \rangle \odot \mathsf{F}_m \rrbracket(u) && \text{(def. of } \mathsf{P}_i\text{, (28))} \\
&= \big(k_1 \llbracket \mathsf{F}_1 \rrbracket + \ldots + k_m \llbracket \mathsf{F}_m \rrbracket\big)(u) && \text{(def. of } \mathsf{expr}\text{)} \\
&= k_1 \llbracket \mathsf{F}_1 \rrbracket(u) + \ldots + k_m \llbracket \mathsf{F}_m \rrbracket(u) && \text{(general fact about power series)} \\
&= k_1 \llbracket q_{\mathsf{F}_1} \rrbracket(u) + \ldots + k_m \llbracket q_{\mathsf{F}_m} \rrbracket(u) && \text{(inductive hypothesis)}
\end{aligned}
$$

We also have (again assuming that $a = a_i$, and thus that $a \in f(d(\mathsf{E}))$)

$$
\llbracket q_{\mathsf{E}} \rrbracket(au) = \sum_{q' \in Q} E(q_{\mathsf{E}}, a, q') \Big(\llbracket q' \rrbracket(u) \Big) \qquad \text{(Definition 6)}
$$

and, by the definition of E in the derived term automaton, we have $E(q_{\mathsf{E}}, a, q_{\mathsf{F}}) = k$ iff $\langle k \rangle \mathsf{F} \in d(\mathsf{E})(a) = \mathsf{P}_i$. This gives (using our expansion (28) of P_i)

$$
\llbracket q_{\mathsf{E}} \rrbracket(au) = k_1 \llbracket q_{\mathsf{F}_1} \rrbracket(u) + \ldots + k_m \llbracket q_{\mathsf{F}_m} \rrbracket(u)
$$

and completes the inductive case whenever $a = a_i$ for some i.

Finally, if $a \neq a_i$ for all i, it is easy to see that $\llbracket \mathsf{E} \rrbracket(au) = 0 = \llbracket q_{\mathsf{E}} \rrbracket(au)$, and the inductive step is now complete. □

Proof (Theorem 1). Follows from Lemma 5, and the fact, resulting from the definition of I in Definition 9 in combination with (11), that $\llbracket \mathcal{A}_{\mathsf{E}} \rrbracket = \llbracket q_{\mathsf{E}} \rrbracket$. □

4.1 Derived-Term Automaton Size

The smallness of the derived-term automaton for basic operators ($|\mathcal{A}_{\mathsf{E}}| \leq \|\mathsf{E}\| + 1$) [14, Theorem 2]) no longer applies with extended operators. Let m and n be coprime integers, $\mathsf{E} \coloneqq (a^m)^* \,\&\, (a^n)^*$ has width $\|\mathsf{E}\| = m + n$; it is easy to see that $|\mathcal{A}_{\mathsf{E}}| = mn$. It is also a classical result that the minimal (trim) automaton to recognize the language of $\mathsf{F}_n \coloneqq (a+b)^* a (a+b)^n$ has 2^{n+1} states; so $\|\mathsf{F}_n^c\| = 2n+3$, but $|\mathcal{A}_{\mathsf{F}_n^c}| = 2^{n+1}+1$ (the additional state is the sink state needed to get a *complete* deterministic automaton before complement). Actually, when complement is used on an infinite semiring, it is not even guaranteed that the automaton is finite (see Sect. 4.3).

Theorem 2. *If $\mathbb{K}$ is finite, or if E has no complement, then $\mathcal{A}_{\mathsf{E}}$ is finite.*

Proof. The proof goes in several steps. First introduce the *proper derived terms* of E, a set of expressions noted $\mathrm{PD}(\mathsf{E})$, and the *derived terms* of E, $\mathrm{D}(\mathsf{E}) \coloneqq \mathrm{PD}(\mathsf{E}) \cup \{\mathsf{E}\}$. $\mathrm{PD}(\mathsf{E})$ is defined inductively as in [14, Def. 3], to which we add

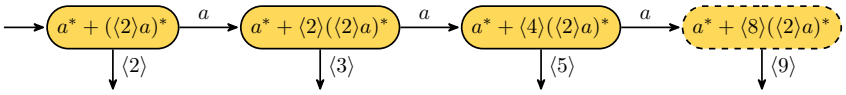
$$
\begin{aligned}
\mathrm{PD}(\mathsf{E} \,\&\, \mathsf{F}) &\coloneqq \{\mathsf{E}_i \,\&\, \mathsf{F}_j \mid \mathsf{E}_i \in \mathrm{PD}(\mathsf{E}), \mathsf{F}_j \in \mathrm{PD}(\mathsf{F})\} \\
\mathrm{PD}(\mathsf{E}^c) &\coloneqq \{(\langle k_1 \rangle \mathsf{E}_1 + \cdots + \langle k_n \rangle \mathsf{E}_n)^c \mid k_1, \ldots, k_n \in \mathbb{K}, \mathsf{E}_1, \ldots, \mathsf{E}_n \in \mathrm{PD}(\mathsf{E})\}
\end{aligned}
$$

Second, verify that $\mathrm{PD}(\mathsf{E})$ is finite (under the proper assumptions). Third, prove that $\mathrm{D}(\mathsf{E})$ is "stable by expansion", i.e., $\forall \mathsf{F} \in \mathrm{D}(\mathsf{E}), \mathsf{expr}\big(d(\mathsf{F})\big) \subseteq \mathrm{D}(\mathsf{E})$. Finally, observe that the states of $\mathcal{A}_{\mathsf{E}}$ are therefore members of $\mathrm{D}(\mathsf{E})$, which is finite. □

4.2 Deterministic Automata

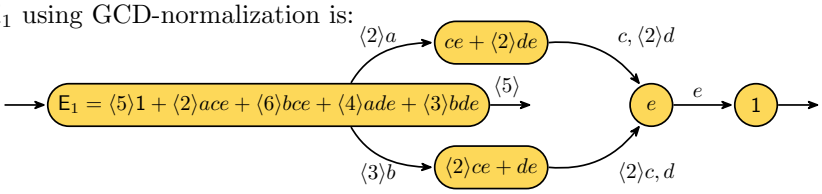
The exposed approach can be used to generate *deterministic* automata by *determinizing* the expansions: $\mathsf{det}(\mathsf{X}) \coloneqq \bigoplus_{a\in f(\mathsf{X})} \langle 1_{\mathbb{K}}\rangle \odot \mathsf{expr}(\mathsf{X}_a)$. The expr operator "consolidates" a polynomial into an expression that ensures this determinism. For instance the expansion $a \odot [\langle 1_{\mathbb{K}}\rangle \odot b \oplus \langle 1_{\mathbb{K}}\rangle \odot c]$, which would yield two transitions labeled by a (one to b and the other to c) is determinized into $a \odot [\langle 1_{\mathbb{K}}\rangle \odot (b + c)]$, yielding a single transition (to $b + c$).

It is well known that some nondeterministic *weighted* automata have no deterministic equivalent, in which case determinization loops. Our construct is subject to the same condition. The expression $\mathsf{E} \coloneqq a^* + (\langle 2\rangle a)^*$ on the alphabet $\{a\}$ admits an infinite number of derivatives: $\partial_{a^n}(\mathsf{E}) = a^* \oplus \langle 2^n\rangle(\langle 2\rangle a)^*$. Therefore our construction of *deterministic* automata would not terminate: the automaton is locally finite but infinite (and there is no finite deterministic automaton equivalent to E). However, a lazy implementation as available in Vcsn[1] would uncover the automaton on demand, for instance when evaluating a word.



To improve determinizability, when $\mathbb{K}$ features a left-division $\backslash$, we apply the usual technique used in the weighted determinization of automata: normalize the results to keep a unique representative of colinear polynomials. Concretely, when determinizing expansions, polynomials are first normalized: $\mathsf{det}(\mathsf{X}) \coloneqq \bigoplus_{a\in f(\mathsf{X})} \langle |\mathsf{X}_a|\rangle \odot \mathsf{expr}(|\mathsf{X}_a| \setminus \mathsf{X}_a)$ where, for a polynomial $\mathsf{P} = \bigoplus_{i\in I}\langle k_i\rangle \odot \mathsf{E}_i$, and a weight k, $k \setminus \mathsf{P} \coloneqq \bigoplus_{i\in I}\langle k \setminus k_i\rangle \odot \mathsf{E}_i$, and the weight $|\mathsf{P}|$ denotes some "norm" of (the coefficients of) P. For instance $|\mathsf{P}|$ can be the GCD of the k_i (so that the coefficients are coprime), or, in the case of a field, the first non null k_i (so that the first non null coefficient is $1_{\mathbb{K}}$), or the sum of the k_i provided it's not null (so that the sum of the coefficients is $1_{\mathbb{K}}$), etc.

Example 5 (Examples 2 to 4 cont.). The deterministic derived-term automaton of E_1 using GCD-normalization is:



4.3 The Case of Complement

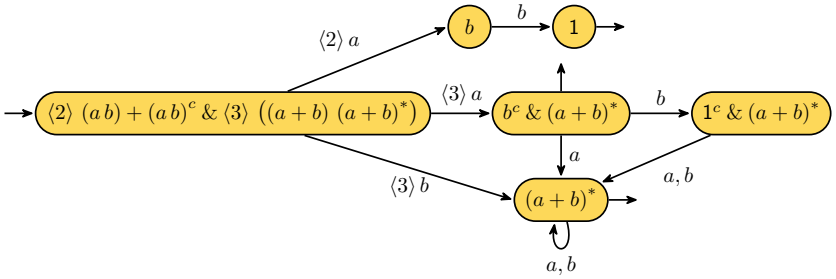
The classical algorithm to complement an (unweighted) automaton, by complementation of the set of the final states, requires a deterministic and complete automaton (which can lead to an exponential number of states). In our case "local" determinism (i.e., restricted to complemented subexpressions) is ensured by expr in the definition of the complement of an expansion in (5) and (10).

In the case of weighted expressions, we hit the same problems —and apply the same techniques— as in Sect. 4.2: not all expressions generate finite automata. A strict (non-lazy) implementation would not terminate on $\big(a^* + (\langle 2\rangle a)^*\big)^c$; a lazy implementation would uncover finite portions of the automaton, on demand. However, although $\mathsf{F} \coloneqq (\langle 2\rangle a)^* + (\langle 4\rangle aa)^*$ admits an infinite number of derivatives, F^c features only two: $\big(\langle 2\rangle(\langle 2\rangle a)^* + \langle 4\rangle(a(\langle 4\rangle aa)^*)\big)^c \Rightarrow \big((\langle 2\rangle a)^* + \langle 2\rangle(a(\langle 4\rangle aa)^*)\big)^c$ and itself. It is the trivial identity $(\langle k\rangle \mathsf{E})^c \Rightarrow \mathsf{E}^c$ that eliminates the common factor.

Example 6 (Example 1 continued). We have:

$$d(\mathsf{E}_3) = a \odot [\langle 2\rangle \odot b \oplus \langle 3\rangle \odot \big(b^c \,\&\, (a+b)^*\big)] \oplus b \odot [\langle 3\rangle \odot (a+b)^*]$$

The lower part of $\mathcal{A}_{\mathsf{E}_3}$ is characteristic of the complement of a complete deterministic automaton:



4.4 Complexity and Performances

We focus on basic expressions. Obviously, $\|\mathsf{E}\| \le |\mathsf{E}|$, and we know $|\mathcal{A}_{\mathsf{E}}| \le \|\mathsf{E}\| + 1$.

The complexity of Antimirov's algorithm is $O(\|E\|^3|E|^2)$ [7]: for each of the $|\mathcal{A}_{\mathsf{E}}|$ states, we may generate at most $|\mathcal{A}_{\mathsf{E}}|$ partial derivatives, each one to compare to the $|\mathcal{A}_{\mathsf{E}}|$ derived-terms. That's $O(|\mathcal{A}_{\mathsf{E}}|^3)$ comparisons to perform on objects of size $O(|\mathsf{E}|^2)$.

However, hash tables allow us to avoid these costly comparisons. For each of the $|\mathcal{A}_{\mathsf{E}}|$ states, we may generate at most $|\mathcal{A}_{\mathsf{E}}|$ partial derivatives and number them via a hash table. Computing an expansion builds an object of size $O(|\mathsf{E}|^2)$, however using references instead of deep copies allows to stay linear, so the complexity is $O(\|E\|^2|E|)$.

To build the derived-term automaton using derivation, one loops over the alphabet for each derived term. This incurs a performance penalty with large alphabets. Let a_i, b_i, for $i \in [m]$, be distinct letters. The following table reports the duration of the process, in milliseconds, for $\mathsf{E}_n^m \coloneqq \sum_{i=1}^m (a_i + b_i)^* a_i (a_i + b_i)^n$ (right associative) by Vcsn[3]. The parameter m controls the alphabet size: 2, 128, and 254 (Vcsn reserves two `chars`) corresponding to $m = 1, 64$, and 127. The parameter n makes the expression arbitrarily long.

[3] Vcsn 2.2 as of 2016-05-16, compiled with Clang 3.6 with options `-O3 -DNDEBUG`, and run on a Mac OS X 10.11.4, Intel Core i7 2.9GHz, 8GB of RAM. Best run out of five.

		n					
	m	1	10	50	100	500	1000
derivation	1	0.07	0.16	0.93	2.7	48.4	199
	64	17.1	84.0	548	1,471	23,608	87,137
	127	64.4	319	2,041	5,407	85,611	323,652
expansion	1	0.04	0.11	0.48	1.3	17.8	76
	64	1.82	6.40	39.19	100	1,368	4,951
	127	3.62	12.92	75.70	205	2,741	10,045

The expansion-based algorithm always performs better than the derivation-based one, dramatically on large alphabets on this benchmark.

One can optimize the derivation-based algorithm by computing the firsts globally [17] or locally, on-the-fly, and then derivating on this set. However, on sums such as $a_1 + \cdots + a_n$ (where a_i are distinct letters) the expansion requires a single traversal ($O(n)$) whereas one still needs n derivations, an $O(n^2)$ process.

Besides, the derivation-based algorithm computes the constant term of an expression several times: to check whether the current state is final, to compute the derivation of products and stars, and to compute the firsts of products. To fix this issue, these repeated computations can be cached.

Addressing both concerns (iteration over the alphabet, repeated computation of the constant term) for the derivation-based algorithm requires three tightly entangled algorithms (constant term, derivation, first). Expansions, on the other hand, keep them together, in a single construct, computed in a single traversal of the expression.

5 Related Work

Compared to Brzozowski [4] we introduced *weighted* expansions, and their direct computation, making them the core computation of the algorithm. This was partly done for basic Boolean expressions by Antimirov [3] as "linear forms."

Our work is deeply influenced by Lombardy and Sakarovitch [14] and shares many similarities. However, by carefully avoiding the derivatives with respect to words, we get simpler proofs for Theorems 1 and 2. Besides, these proofs free us from the requirement of a yielding a finite automaton, which is needed to establish the correctness of Vcsn's on-the-fly computation of infinite derived-term automata. We also introduced the construction of deterministic (weighted) derived-term automata. Fischer et al. [11] also make profit from laziness to address non rational languages (such as $a^n b^n c^n$); however their approach is quite different: their core construction is the Glushkov automaton, and they use named rational expressions to build "infinite rational expressions", gracefully handled thanks to Haskell's laziness.

There is a striking similarity between (non-weighted) expansions, and auxiliary tools used by Mirkin to define a "prebase" of an expression [16]. See for instance the proof of Proposition 1 as reproduced by Champarnaud and Ziadi [6].

Aside from our support for weighted expressions, our approach of extended operators is comparable to that of Caron et al. [5], but, we believe, using a simpler framework. Basically, their sets of sets of expressions correspond to polynomials of conjunctions: their $\{\{\mathsf{E},\mathsf{F}\},\{\mathsf{G},\mathsf{H}\}\}$ is our $\mathsf{E}\,\&\,\mathsf{F}\oplus\mathsf{G}\,\&\,\mathsf{H}$. Using our framework, the automaton of Fig. 3 [5] has one state less, since $\{\mathsf{E},\mathsf{F}\}$ and $\{\mathsf{E}\cap\mathsf{F}\}$ both are $\mathsf{E}\,\&\,\mathsf{F}$. Actually, the main point of sets of sets of expressions is captured by our *distributive* definition of the conjunction of polynomials, (5), which matches that of their ⋒ operator; indeed what they call the "natural extension" [5, Sect. 3.1] would correspond to $\mathsf{P}_1\,\&\,\mathsf{P}_2 \coloneqq \mathsf{expr}(\mathsf{P}_1)\,\&\,\mathsf{expr}(\mathsf{P}_2)$. Additional properties of & (e.g., associativity), can be enabled via new trivial identities. Like us, their ⊖ operator ensures that complemented expressions generate deterministic automata.

For basic (weighted) expressions, completely different approaches build the derived-term automaton with a quadratic complexity [1, 8]. However, the expansion-based algorithm features some unique properties. It supports a simple and natural on-the-fly implementation. It provides insight on the built automata by labeling states with the language/series they denote (e.g., Vcsn renders derived-term automata as in Examples 4 to 6). It is a flexible framework in which new operators can be easily supported (e.g., the shuffle and infiltration operators in Vcsn), and even multitape rational expressions to generate transducers [9]. It supports the direct construction of deterministic automata. And it copes easily with alternative derivation schemes, such as the "broken derived-terms" [2].

6 Conclusion

The construction of the derived-term automaton from a weighted rational expression is a powerful technique: states have a natural interpretation (they are identified by their future: the series they compute), extended rational expressions are easily supported, determinism can be requested, and it even offers a natural lazy, on-the-fly, implementation to handle infinite automata.

To build the derived-term automaton, we generalized Brzozowski's expansions to weighted expressions, and an inductive algorithm to compute the expansion of a rational expression. The formulas on which this algorithm is built reunite as a unique entity three facets that were kept separated in previous works: constant term, firsts, and derivatives. This results in a simpler set of equations, a proof that does not require the resulting automaton to be finite, and an implementation whose complexity is independent of the size of the alphabet and even applies when it is infinite (e.g., when labels are strings, integers, etc.). Building the derived-term automaton using expansions is straightforward, even when required to be deterministic. We have also shown that using proper techniques, the complexity of the algorithm is much better that previously reported.

The computation of expansions and derivations are implemented in Vcsn[1], together with their automaton construction procedures (possibly lazy, possibly deterministic). Our implementation actually supports for additional operators on rational expressions (e.g., shuffle and infiltration).

Acknowledgments Interactions with A. Duret-Lutz, S. Lombardy, L. Saiu and J. Sakarovitch resulted in this work. Anonymous reviewers made very helpful

comments. In particular, an anonymous reviewer of ICALP 2016 contributed the proof of Theorem 1, much simpler than the original one (which was still based on derivatives), and proposed the benchmark of Sect. 4.4.

References

1. C. Allauzen and M. Mohri. A unified construction of the Glushkov, follow, and Antimirov automata. In *Mathematical Foundations of Computer Science*, vol. 4162 of *LNCS*, pp. 110–121. Springer, 2006.
2. P.-Y. Angrand, S. Lombardy, and J. Sakarovitch. On the number of broken derived terms of a rational expression. *Journal of Automata, Languages and Combinatorics*, 15(1/2):27–51, 2010.
3. V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *TCS*, 155(2):291–319, 1996.
4. J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
5. P. Caron, J.-M. Champarnaud, and L. Mignot. Partial derivatives of an extended regular expression. In *Language and Automata Theory and Applications*, vol. 6638 of *LNCS*, pp. 179–191. Springer, 2011.
6. J.-M. Champarnaud and D. Ziadi. From Mirkin's prebases to Antimirov's word partial derivatives. *Fundam. Inf.*, 45(3):195–205, Jan. 2001.
7. J.-M. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *TCS*, 289(1):137–163, 2002.
8. J.-M. Champarnaud, F. Ouardi, and D. Ziadi. An efficient computation of the equation $\mathbb{K}$-automaton of a regular $\mathbb{K}$-expression. In *Developments in Language Theory*, vol. 4588 of *LNCS*, pp. 145–156. Springer, 2007.
9. A. Demaille. Derived-term automata of multitape rational expressions. In *CIAA'16*, vol. 9705 of *LNCS*, pp. 51–63, July 2016. Springer.
10. A. Demaille, A. Duret-Lutz, S. Lombardy, and J. Sakarovitch. Implementation concepts in Vaucanson 2. In *CIAA'13*, vol. 7982 of *LNCS*, pp. 122–133, July 2013. Springer.
11. S. Fischer, F. Huch, and T. Wilke. A play on regular expressions: Functional pearl. In *Proc. of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP'10, pp. 357–368, 2010. ACM.
12. V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16: 1–53, 1961.
13. R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Comput. Linguist.*, 20(3):331–378, Sept. 1994.
14. S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *TCS*, 332(1-3):141–177, 2005.
15. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–47, 1960.
16. B. G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5:110–116, 1966.
17. S. Owens, J. Reppy, and A. Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, Mar. 2009.
18. J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *TCS*, 308(1-3):1–53, 2003.
19. J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. Corrected English translation of *Éléments de théorie des automates*, Vuibert, 2003.