Regular Expressions and Transducers over Alphabet-Invariant and User-Defined Labels

Stavros Konstantinidis[1(✉)], Nelma Moreira[2], Rogério Reis[2], and Joshua Young[1]

[1] Saint Mary's University, Halifax, NS, Canada
s.konstantinidis@smu.ca, jyo04@hotmail.com
[2] CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal
{nam,rvr}@dcc.fc.up.pt

Abstract. We are interested in regular expressions and transducers that represent word relations in an alphabet-invariant way—for example, the set of all word pairs u, v where v is a prefix of u independently of what the alphabet is. Current software systems of formal language objects do not have a mechanism to define such objects. We define transducers in which transition labels involve what we call set specifications, some of which are alphabet invariant. In fact, we consider automata-type objects, called labelled graphs, where each transition label can be any string, as long as that string represents a subset of a certain monoid. Then, the behaviour of the labelled graph is a subset of that monoid. We do the same for regular expressions. We obtain extensions of known algorithmic constructions on ordinary regular expressions and transducers, including partial derivative based methods, at the broad level of labelled graphs such that the computational efficiency of the extended constructions is not sacrificed. Then, for regular expressions with set specs we obtain a direct partial derivative method for membership. For transducers with set specs we obtain further algorithms that can be applied to questions about independent regular languages, in particular the witness version of the property satisfaction question.

Keywords: Alphabet-invariant transducers · Regular expressions
Partial derivatives · Algorithms · Monoids

1 Introduction

We are interested in 2D regular expressions and transducers over alphabets whose cardinality is not fixed, or whose alphabet is even unknown. In particular,

Research supported by NSERC (Canada) and by FCT project UID/MAT/00144/2013 (Portugal). Reference [16] is a detailed version of this paper.

C. Câmpeanu (Ed.): CIAA 2018, LNCS 10977, pp. 4–27, 2018.
https://doi.org/10.1007/978-3-319-94812-6_2

assume that the alphabet is $\Gamma = \{0, 1, \ldots, n-1\}$ and consider the 2D regular expression

$$\big(0/0 + \cdots + (n-1)/(n-1)\big)^* \big(0/\boldsymbol{e} + \cdots + (n-1)/\boldsymbol{e}\big)^*,$$

where $\boldsymbol{e}$ is the symbol for the empty string. This 2D regular expression has $O(n)$ symbols and describes the prefix relation: all word pairs (u, v) such that v is a prefix of u. Similarly, consider the transducer in Fig. 1, which has $O(n^2)$ transitions. Current software systems of formal language objects require users to enter all these transitions in order to define and process the transducer. We want to be able to use special labels in transducers such as those in the transducer $\hat{t}_{\text{sub2}}$ in Fig. 2. In that figure, the label $(\forall/=)$ represents the set $\{(a, a) \mid a \in \Gamma\}$ and the label $(\forall/\forall\neq)$ represents the set $\{(a, a') \mid a, a' \in \Gamma, a \neq a'\}$. Moreover that transducer has only a fixed number of 5 transitions. Similarly, using these special labels, the above 2D regular expression can be written as $(\forall/=)^*(\forall/\boldsymbol{e})^*$. Note that the new regular expression as well as the new transducer in Fig. 2 are *alphabet invariant* as they contain no symbol of the intended alphabet Γ—precise definitions are provided in the next sections.

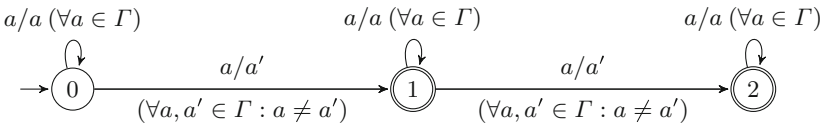

Fig. 1. The transducer realizes the relation of all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2.

We also want to be able to define algorithms that work *directly* on regular expressions and transducers with special labels, without of course having to expand these labels to ordinary ones. Thus, for example, we would like to have an efficient algorithm that computes whether a pair (u, v) of words is in the relation realized by the transducer in Fig. 2, and an efficient algorithm to compute the composition of two transducers with special labels.

We start off with the broad concept of a set B of labels, called *label set*, where each label $\beta \in B$ is simply a string that represents a subset $\mathcal{I}(\beta)$ of a monoid M. Then we define type B automata (called *labelled graphs*) in which every transition label is in B. Similarly we consider type B regular expressions whose base objects (again called labels) are elements of B and represent monoid subsets. Our first set of results apply to any user-defined set B and monoid M. Then, we consider further results specific to the cases of (i) 1D regular expressions and automata (monoid $M = \Gamma^*$), (ii) 2D regular expressions and transducers (monoid $M = \Gamma^* \times \Gamma^*$) with special labels (called *set specs*). We note that a concept of label set similar to the one defined here is considered in [12]. In particular, [12] considers label sets with weights, and the objectives of that work are different from the ones here.

We emphasize that we do not attempt to define regular expressions and automata outside of monoids; rather we use monoid-based regular expressions and automata as a foundation such that (i) one can define such objects with alphabet invariant labels or with a priori unknown label sets B, as long as each of the labels represents a subset of a known monoid; (ii) many known algorithms and constructions on monoid-based regular expressions and automata are extended to work directly and as efficiently on certain type B objects.

We also mention the framework of symbolic automata and transducers of [23, 24]. In that framework, a transition label is a logic predicate describing a set of domain elements (characters). The semantics of that framework is very broad and includes the semantics of label sets in this work. As such, the main algorithmic results in [23,24] do not include time complexity estimates. Moreover, outside of the logic predicates there is no provision to allow for user-defined labels and related algorithms working directly on these labels.

The paper is organized as follows. The next section makes some assumptions about *alphabets* Γ of non-fixed size. Section 3 defines two specific label sets: the set of *set specs*, in which each element represents a subset of Γ or the empty string, and the set of *pairing specs* that is used for transducer-type labelled graphs. Some of these label sets can be *alphabet invariant*. Section 4 discusses the general concept of a *label set* B, which has a behaviour $\mathcal{I}$ and refers to a monoid mon B; that is, $\mathcal{I}(\beta)$ is a subset of mon B for any label $\beta \in B$. Section 5 defines type B labelled graphs $\hat{g}$ and their behaviours $\mathcal{I}(\hat{g})$. When B is the set of pairing specs then $\hat{g}$ is a transducer-type graph and realizes a word relation. Section 6 defines regular expressions $\mathbf{r}$ over any label set B and their behaviour $\mathcal{I}(\mathbf{r})$, and establishes the equivalence of type B graphs and type B regular expressions (Theorem 1) as well as the partial derivative automaton corresponding to $\mathbf{r}$ via the concept of linear form of $\mathbf{r}$ (Theorem 3). Then, for a regular expression $\mathbf{r}$ over set specs it presents the partial derivative machinery for deciding directly if a word is in $\mathcal{L}(\mathbf{r})$ (Lemma 11). Section 7 considers the possibility of defining 'higher level' versions of product constructions that work on automata/transducers over known monoids. To this end, we consider the concept of *polymorphic operation* '$\odot$' that is partially defined between two elements of some labels sets B, B', returning an element of some label set C, and also partially defined on the elements of the monoids mon B and mon B', returning an element of the monoid mon C. In this case, if $\odot$ is known to work on automata/transducers over mon B, mon B' then it would also work on type B, B' graphs (Theorem 4). Section 8 presents some basic algorithms on automata with set specs and transducers with set specs. Section 9 defines the composition of two transducers with set specs such that the complexity of this operation is consistent with the case of ordinary transducers (Theorem 5). Section 10 considers the questions of whether a transducer with set specs realizes an identity and whether it realizes a function. It is shown that both questions can be answered with a time complexity consistent with that in the case of ordinary transducers (Theorems 6 and 7). Section 11 shows that, like ordinary transducers, transducers with set specs that define independent language properties can be processed directly (without expanding them) and

efficiently to answer the witness version of the property satisfaction question for regular languages (Corollary 2 and Example 12). Finally, the last section contains a few concluding remarks and directions for future research.

2 Terminology and Alphabets of Non-fixed Size

The set of positive integers is denoted by $\mathbb{N}$. Then, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Let S be a set. We denote the *cardinality* of S by $|S|$ and the set of all subsets of S by 2^S. To indicate that ϕ is a *partial mapping* of a set S into a set T we shall use the notation $\phi : S \dashrightarrow T$. We shall write $\phi(s) = \perp$ to indicate that ϕ is not defined on $s \in S$.

An *alphabet space* Ω is an infinite and totally ordered set whose elements are called *symbols*. We shall assume that Ω is fixed and contains the digits $0, 1, \ldots, 9$, which are ordered as usual, as well as the *special symbols*

$$\forall \quad \exists \quad \nexists \quad = \quad \neq \quad / \quad \boldsymbol{e} \quad \oplus \quad \oslash$$

We shall denote by '$<$' the total order of Ω. As usual we use the term *string* or *word* to refer to any finite sequence of symbols. The *empty string* is denoted by ε. For any string w we say that w is *sorted* if the symbols contained in w occur in the left to right direction according to the total order of Ω. For example, the word 012 is sorted, but 021 is not sorted. For any set of symbols S, we use the notation $\mathrm{wo}(S) =$ the sorted word consisting of the symbols in S. For example, if $S = \{0, 1, 2\}$, then $\mathrm{wo}(S) = 012$ and $\mathrm{wo}(\{2, 0\}) = 02$.

Let $g \in \Omega$ and w be a string. The expression $|w|_g$ denotes the number of occurrences of g in w, and the expression alph w denotes the set $\{g \in \Omega : |w|_g > 0\}$, that is, the set of symbols that occur in w. For example,

$$\mathrm{alph}\,(1122010) = \{0, 1, 2\}.$$

An *alphabet* is any finite nonempty subset of Ω. In the following definitions we consider an alphabet Γ, called the alphabet of *reference*, and we assume that *Γ contains at least two symbols and no special symbols.*

Algorithmic Convention About Alphabet Symbols. We shall consider algorithms on automata and transducers where the alphabets Γ involved are not of fixed size and, therefore, $|\Gamma| \to \infty$; thus, the alphabet size $|\Gamma|$ is accounted for in time complexity estimates. Moreover, we assume that each Γ-symbol is of size $O(1)$. This approach is also used in related literature (e.g., [1]), where it is assumed implicitly that the cost of comparing two Γ-symbols is $O(1)$.

In the algorithms presented below, we need operations that require to access only a part of Γ or some information about Γ such as $|\Gamma|$. We assume that Γ has been preprocessed such that the value of $|\Gamma|$ is available and is $O(\log|\Gamma|)$ bits long and the *minimum symbol* $\min\Gamma$ of Γ is also available. In particular, we assume that we have available a *sorted array* ARR_Γ consisting of all Γ-symbols. While this is a convenient assumption, if in fact it is not applicable then one can make the array from Γ in time $O\big(|\Gamma| \log|\Gamma|\big)$. Then, the minimum symbol of Γ is simply $\mathrm{ARR}_\Gamma[0]$. Moreover, we have available an *algorithm* notIn(w), which

returns a symbol in Γ that is not in alph w, where w is a *sorted word* in Γ^* with $0 < |w| < |\Gamma|$. Next we explain that the desired algorithm

notIn(w) can be made to work in time $O(|w|)$.

The algorithm notIn(w) works by using an index i, initially $i = 0$, and incrementing i until $\mathrm{ARR}_\Gamma[i] \neq w[i]$, in which case the algorithm returns $\mathrm{ARR}_\Gamma[i]$.

3 Set Specifications and Pairing Specifications

Here we define expressions, called set specs, that are used to represent subsets of the alphabet Γ or the empty string. These can be used as labels in automata-type objects (labelled graphs) and regular expressions defined in subsequent sections.

Definition 1. *A* set specification*, or* set spec *for short, is any string of one of the four forms*

$$\boldsymbol{e} \qquad \forall \qquad \exists w \qquad \nexists w$$

where w is any sorted nonempty string containing no repeated symbols and no special symbols. The set of set specs is denoted by SSP.

Let $F, \exists u, \nexists u, \exists v, \nexists v$ be any set specs with $F \neq \boldsymbol{e}$. We define the partial operation $\cap : \mathrm{SSP} \times \mathrm{SSP} \dashrightarrow \mathrm{SSP}$ *as follows.*

$$\begin{array}{l}
\boldsymbol{e} \cap \boldsymbol{e} = \boldsymbol{e}, \quad \boldsymbol{e} \cap F = F \cap \boldsymbol{e} = \perp \\
\forall \cap F = F \cap \forall = F \\
\exists u \cap \exists v = \exists \,\mathrm{wo}\big(\mathrm{alph}\ u \cap \mathrm{alph}\ v\big), \quad \textit{if}\ \big(\mathrm{alph}\ u \cap \mathrm{alph}\ v\big) \neq \emptyset \\
\exists u \cap \exists v = \perp, \quad \textit{if}\ \big(\mathrm{alph}\ u \cap \mathrm{alph}\ v\big) = \emptyset \\
\nexists u \cap \nexists v = \nexists \,\mathrm{wo}\big(\mathrm{alph}\ u \cup \mathrm{alph}\ v\big) \\
\exists u \cap \nexists v = \exists \,\mathrm{wo}\big(\mathrm{alph}\ u \setminus \mathrm{alph}\ v\big), \quad \textit{if}\ \big(\mathrm{alph}\ u \setminus \mathrm{alph}\ v\big) \neq \emptyset \\
\exists u \cap \nexists v = \perp, \quad \textit{if}\ \big(\mathrm{alph}\ u \setminus \mathrm{alph}\ v\big) = \emptyset \\
\nexists u \cap \exists v = \exists v \cap \nexists u
\end{array}$$

Example 1. As any set spec X is a string, it has a length $|X|$. We have that $|\forall| = 1$ and $|\exists w| = 1 + |w|$. Also,

$$\exists 035 \cap \exists 1358 = \exists 35, \quad \nexists 035 \cap \exists 1358 = \exists 18, \quad \nexists 035 \cap \nexists 1358 = \nexists 01358.$$

Lemma 1. *For any given set specs G and F, $G \cap F$ can be computed in time $O(|G| + |F|)$.*

Definition 2. *Let Γ be an alphabet of reference. We say that a set spec F* respects *Γ, if the following restrictions hold when F is of the form $\exists w$ or $\nexists w$:*

$$w \in \Gamma^* \ \textit{and}\ 0 < |w| < |\Gamma|.$$

In this case, the language *$\mathcal{L}(F)$ of F (*with respect *to Γ) is the subset of $\Gamma \cup \{\varepsilon\}$ defined as follows:*

$$\mathcal{L}(\boldsymbol{e}) = \{\varepsilon\}, \qquad \mathcal{L}(\forall) = \Gamma, \qquad \mathcal{L}(\exists w) = \mathrm{alph}\ w, \qquad \mathcal{L}(\nexists w) = \Gamma \setminus \mathrm{alph}\ w.$$

The set of set specs that respect Γ is denoted as follows

$$\mathrm{SSP}[\Gamma] = \{\alpha \in \mathrm{SSP} \mid \alpha \textit{ respects } \Gamma\}.$$

Remark 1. In the above definition, the requirement $|w| < |\Gamma|$ implies that there is at least one Γ-symbol that does not occur in w. Thus, to represent Γ we must use $\forall$ as opposed to the longer set spec $\exists\,\mathrm{wo}(\Gamma)$.

Lemma 2. *Let Γ be an alphabet of reference and let $F \neq \boldsymbol{e}$ be a set spec respecting Γ. The following statements hold true.*

1. *For given $g \in \Gamma$, testing whether $g \in \mathcal{L}(F)$ can be done in time $O(\log |F|)$.*
2. *For given $g \in \Gamma$, testing whether $\mathcal{L}(F) \setminus \{g\} = \emptyset$ can be done in time $O(|F|)$.*
3. *For any fixed $k \in \mathbb{N}$, testing whether $|\mathcal{L}(F)| \geq k$ can be done in time $O(|F| + \log |\Gamma|)$, assuming the number $|\Gamma|$ is given as input along with F.*
4. *Testing whether $|\mathcal{L}(F)| = 1$ and, in this case, computing the single element of $\mathcal{L}(F)$ can be done in time $O(|F|)$.*
5. *Computing an element of $\mathcal{L}(F)$ can be done in time $O(|F|)$.*
6. *If $|\mathcal{L}(F)| \geq 2$ then computing two different $\mathcal{L}(F)$-elements can be done in time $O(|F|)$.*

Now we define expressions for describing certain finite relations that are subsets of $(\Gamma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$.

Definition 3. *A* pairing specification*, or* pairing spec *for short, is a string of the form*

$$\boldsymbol{e}/\boldsymbol{e} \qquad \boldsymbol{e}/G \qquad F/\boldsymbol{e} \qquad F/G \qquad F/{=} \qquad F/G{\neq} \tag{1}$$

where F, G are set specs with $F, G \neq \boldsymbol{e}$. The set of pairing specs is denoted by PSP*. The* inverse p^{-1} *of a pairing spec* p *is defined as follows depending on the possible forms of* p *displayed in* (1)*:*

$$(\boldsymbol{e}/\boldsymbol{e})^{-1} = (\boldsymbol{e}/\boldsymbol{e}), \qquad (\boldsymbol{e}/G)^{-1} = (G/\boldsymbol{e}), \quad (F/\boldsymbol{e})^{-1} = (\boldsymbol{e}/F),$$
$$(F/G)^{-1} = (G/F), \quad (F/{=})^{-1} = (F/{=}), \quad (F/G{\neq})^{-1} = (G/F{\neq}).$$

Example 2. As a pairing spec p is a string, it has a length $|\mathsf{p}|$. We have that $|\forall/\boldsymbol{e}| = 3$ and $|\exists u/\nexists v| = 3 + |u| + |v|$. Also, $(\forall/\boldsymbol{e})^{-1} = (\boldsymbol{e}/\forall)$ and $(\exists u/\forall{\neq})^{-1} = (\forall/\exists u{\neq})$.

Definition 4. *A pairing spec is called* alphabet invariant *if it contains no set spec of the form $\exists w, \nexists w$. The set of alphabet invariant pairing specs is denoted by* $\mathrm{PSP}^{\mathrm{invar}}$*.*

Definition 5. *Let Γ be an alphabet of reference and let* p *be a pairing spec. We say that* p respects *Γ, if any set spec occurring in* p *respects Γ. The set of pairing specs that respect Γ is denoted as follows*

$$\mathrm{PSP}[\Gamma] = \{\mathsf{p} \in \mathrm{PSP} : \mathsf{p} \textit{ respects } \Gamma\}.$$

The relation $\mathcal{R}(\mathsf{p})$ *described by* p *(with respect to Γ) is the subset of $\Gamma^* \times \Gamma^*$ defined as follows.*

$$\mathcal{R}(\boldsymbol{e}/\boldsymbol{e}) = \{(\varepsilon, \varepsilon)\}; \quad \mathcal{R}(\boldsymbol{e}/G) = \{(\varepsilon, y) \mid y \in \mathcal{L}(G)\};$$

$\mathcal{R}(F/\boldsymbol{e}) = \{(x,\varepsilon) \mid x \in \mathcal{L}(F)\}$; $\mathcal{R}(F/G) = \{(x,y) \mid x \in \mathcal{L}(F), y \in \mathcal{L}(G)\}$;
$\mathcal{R}(F/{=}) = \{(x,x) \mid x \in \mathcal{L}(F)\}$;
$\mathcal{R}(F/G{\neq}) = \{(x,y) \mid x \in \mathcal{L}(F), y \in \mathcal{L}(G), x \neq y\}$.

Remark 2. All the alphabet invariant pairing specs are

$$\boldsymbol{e}/\boldsymbol{e} \qquad \boldsymbol{e}/\forall \qquad \forall/\boldsymbol{e} \qquad \forall/\forall \qquad \forall/{=} \qquad \forall/\forall{\neq}$$

Any alphabet invariant pairing spec p respects all alphabets of reference Γ, as p contains no set specs of the form $\exists w$ or $\nexists w$.

4 Label Sets and Their Behaviours

We are interested in automata-type objects (labelled graphs) $\hat{g}$ in which every transition label β represents a subset $\mathcal{I}(\beta)$ of some monoid M. These subsets are the behaviours of the labels and are used to define the behaviour of $\hat{g}$ as a subset of M—see next section for labelled graphs. We shall use the notation

ε_M for the neutral element of the monoid M.

If S, S' are any two subsets of M then, as usual, we define

$$SS' = \{mm' \mid m \in S,\ m' \in S'\} \quad \text{and} \quad S^i = S^{i-1}S \quad \text{and} \quad S^* = \cup_{i=0}^{\infty} S^i,$$

where $S^0 = \{\varepsilon_M\}$ and the monoid operation is denoted by simply concatenating elements. We shall only consider *finitely generated* monoids M where each $m \in M$ has a unique *canonical* (string) representation $\underline{m}$. Then, we write $\underline{M} = \{\underline{m} \mid m \in M\}$.

Example 3. We shall consider two standard monoids. First, the free monoid Γ^* (or Σ^*) whose neutral element is ε. The canonical representation of a nonempty word w is w itself and that of ε is $\boldsymbol{e}$: $\underline{\varepsilon} = \boldsymbol{e}$. Second, the monoid $\Sigma^* \times \Delta^*$ (or $\Gamma^* \times \Gamma^*$) whose neutral element is $(\varepsilon,\varepsilon)$. The canonical representation of a word pair (u,v) is $\underline{u}/\underline{v}$. In particular, $\underline{(\varepsilon,\varepsilon)} = \boldsymbol{e}/\boldsymbol{e}$.

A *label set* B is a nonempty set of nonempty strings (over Ω). A *label behaviour* is a mapping $\mathcal{I} : B \to 2^M$, where M is a monoid. Thus, the behaviour $\mathcal{I}(\beta)$ of a label $\beta \in B$ is a subset of M. We shall consider label sets B with *fixed behaviours*, so we shall

denote by mon B the *monoid of* B via its fixed behaviour.

Notational Convention. We shall make the *convention* that for any label sets B_1, B_2 with fixed behaviours $\mathcal{I}_1, \mathcal{I}_2$, we have:

if mon B_1 = mon B_2 then $\mathcal{I}_1(\beta) = \mathcal{I}_2(\beta)$, for all $\beta \in B_1 \cap B_2$.

With this convention we can simply use a single behaviour notation $\mathcal{I}$ for all label sets with the same behaviour monoid, that is, we shall use $\mathcal{I}$ for any B_1, B_2 with mon B_1 = mon B_2. This convention is applied in the example below: we use $\mathcal{L}$ for the behaviour of both the label sets $\Sigma_{\boldsymbol{e}}$ and SSP$[\Gamma]$.

Example 4. We shall use some of the following label sets and their fixed label behaviours.

1. $\Sigma_{\boldsymbol{e}} = \Sigma \cup \{\boldsymbol{e}\}$ with behaviour $\mathcal{L} : \Sigma_{\boldsymbol{e}} \to 2^{\Sigma^*}$ such that $\mathcal{L}(g) = \{g\}$, if $g \in \Sigma$, and $\mathcal{L}(\boldsymbol{e}) = \{\varepsilon\}$. Thus, mon $\Sigma_{\boldsymbol{e}} = \Sigma^*$.
2. Σ with behaviour $\mathcal{L} : \Sigma \to 2^{\Sigma^*}$ such that $\mathcal{L}(g) = \{g\}$, for $g \in \Sigma$. Thus, mon $\Sigma = \Sigma^*$.
3. SSP$[\Gamma]$ with behaviour $\mathcal{L} : \mathrm{SSP}[\Gamma] \to 2^{\Gamma^*}$, as specified in Definition 2. Thus, mon SSP$[\Gamma] = \Gamma^*$.
4. REG Σ = REG $\Sigma_{\boldsymbol{e}}$ = all regular expressions over Σ with behaviour $\mathcal{L} :$ REG $\Sigma \to 2^{\Sigma^*}$ such that $\mathcal{L}(r)$ is the language of the regular expression r. Thus, mon (REG Σ) $= \Sigma^*$.
5. $[\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}] = \{x/y \mid x \in \Sigma_{\boldsymbol{e}}, y \in \Delta_{\boldsymbol{e}}\}$ with behaviour $\mathcal{R} : [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}] \to 2^{\Sigma^* \times \Delta^*}$ such that $\mathcal{R}(\boldsymbol{e}/\boldsymbol{e}) = \{(\varepsilon, \varepsilon)\}$, $\mathcal{R}(x/\boldsymbol{e}) = \{(x, \varepsilon)\}$, $\mathcal{R}(\boldsymbol{e}/y) = \{(\varepsilon, y)\}$, $\mathcal{R}(x/y) = \{(x, y)\}$, for any $x \in \Sigma$ and $y \in \Delta$. Thus, mon $[\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}] = \Sigma^* \times \Delta^*$.
6. PSP$[\Gamma]$ with behaviour $\mathcal{R} : \mathrm{PSP}[\Gamma] \to 2^{\Gamma^* \times \Gamma^*}$ as specified in Definition 5. Thus, mon PSP$[\Gamma] = \Gamma^* \times \Gamma^*$.
7. $\mathrm{PSP}^{\mathrm{invar}}$ with behaviour $\mathcal{R}_\perp : \mathrm{PSP}^{\mathrm{invar}} \to \{\emptyset\}$. Thus, $\mathcal{I}(\beta) = \emptyset$, for any $\beta \in \mathrm{PSP}^{\mathrm{invar}}$.
8. If B_1, B_2 are label sets with behaviours $\mathcal{I}_1, \mathcal{I}_2$, respectively, then $[B_1, B_2]$ is the label set $\{\beta_1/\beta_2 \mid \beta_1 \in B_1, \beta_2 \in B_2\}$ with behaviour and monoid such that $\mathcal{I}(\beta_1/\beta_2) = \mathcal{I}_1(\beta_1) \times \mathcal{I}_2(\beta_2)$ and mon $[B_1, B_2] =$ mon $B_1 \times$ mon B_2.
9. $[\mathrm{REG}\,\Sigma, \mathrm{REG}\,\Delta]$ with behaviour $\mathcal{R}$ in the monoid $\Sigma^* \times \Delta^*$ such that $\mathcal{R}(\mathbf{r}/\mathbf{s}) = \mathcal{L}(\mathbf{r}) \times \mathcal{L}(\mathbf{s})$, for any $\mathbf{r} \in \mathrm{REG}\,\Sigma$ and $\mathbf{s} \in \mathrm{REG}\,\Delta$.

For any monoid of interest M, $\underline{M}$ is a label set such that

$$\text{mon } \underline{M} = M \quad \text{and} \quad \mathcal{I}(\underline{m}) = \{m\}.$$

Thus for example, as mon PSP$[\Gamma] =$ mon $\underline{\Gamma^* \times \Gamma^*} = \Gamma^* \times \Gamma^*$ and the behaviour of PSP is denoted by $\mathcal{R}$, we have $\mathcal{R}(\underline{(0,1)}) = \mathcal{R}(0/1) = \{(0,1)\} = \mathcal{R}(\exists 0/\exists 1)$.

Remark 3. We shall not attempt to define the set of all labels. We limit ourselves to those of interest in this paper. Of course one can define new label sets X at will, depending on the application; and in doing so, one would also define concepts related to those label sets, such as the mon X.

5 Labelled Graphs, Automata, Transducers

Let B be a label set with behaviour $\mathcal{I}$. A *type B graph* is a quintuple

$$\hat{g} = \big(Q, B, \delta, I, F\big)$$

such that Q is a nonempty set whose elements are called *states*; $I \subseteq Q$ is the nonempty set of initial, or start states; $F \subseteq Q$ is the set of final states; δ is a set, called the set of *edges* or *transitions*, consisting of triples (p, β, q) such that

$p, q \in Q$ and β is a nonempty string of Ω-symbols; the set of *labels* $\text{Labels}(\hat{g}) = \{\beta \mid (p, \beta, q) \in \delta\}$ is a subset of B.

We shall use the term *labelled graph* to mean a type B graph as defined above, for some label set B. The labelled graph is called *finite* if Q and δ are both finite. *Unless otherwise specified, a labelled graph, or type B graph, will be assumed to be finite.*

As a label β is a string, the length $|\beta|$ is well-defined. Then, the *size* $|e|$ of an edge $e = (p, \beta, q)$ is the quantity $1 + |\beta|$ and the size of δ is $\|\delta\| = \sum_{e \in \delta} |e|$. Then the *graph size* of $\hat{g}$ is the $|\hat{g}| = |Q| + \|\delta\|$. A *path* P of $\hat{g}$ is a sequence of consecutive transitions, that is, $P = \langle q_{i-1}, \beta_i, q_i \rangle_{i=1}^{\ell}$ such that each (q_{i-1}, β_i, q_i) is in δ. The path P is called *accepting*, if $q_0 \in I$ and $q_\ell \in F$. If $\ell = 0$ then P is empty and it is an accepting path if $I \cap F \neq \emptyset$. A state is called *isolated*, if it does not occur in any transition of $\hat{g}$. A state is called *useful*, if it occurs in some accepting path. Note that any state in $I \cap F$ is useful and can be isolated. The labelled graph $\hat{g}$ is called *trim*, if

every state of $\hat{g}$ is useful, and $\hat{g}$ has at most one isolated state in $I \cap F$.

Definition 6. *Let $\hat{g} = \big(Q, B, \delta, I, F\big)$ be a labelled graph, for some label set B with behaviour $\mathcal{I}$. We define the* behaviour *$\mathcal{I}(\hat{g})$ of $\hat{g}$ as the set of all $m \in \text{mon } B$ such that there is an accepting path $\langle q_{i-1}, \beta_i, q_i \rangle_{i=1}^{\ell}$ of $\hat{g}$ with*

$$m \in \mathcal{I}(\beta_1) \cdots \mathcal{I}(\beta_\ell).$$

The expansion *$\exp \hat{g}$ of $\hat{g}$ is the labelled graph $\big(Q, \underline{\text{mon } B}, \delta_{\exp}, I, F\big)$ such that*

$$\delta_{\exp} = \{(p, \underline{m}, q) \mid \exists\, (p, \beta, q) \in \delta : m \in \mathcal{I}(\beta)\}.$$

In some cases it is useful to modify $\hat{g}$ by adding the transition $(q, \underline{\varepsilon_{\text{mon } B}}, q)$ (a self loop) for each state q of $\hat{g}$. The resulting labelled graph is denoted *by $\hat{g}^{\varepsilon}$.*

Remark 4. The above definition remains valid with no change if the labelled graph, or its expansion, is not finite. The expansion graph of $\hat{g}$ can have infinitely many transitions—for example if $\hat{g}$ is of type $\text{REG}\, \Sigma$.

Lemma 3. *For each type B graph $\hat{g} = (Q, B, \delta, I, F)$, we have that*

$$\mathcal{I}(\hat{g}) = \mathcal{I}(\exp \hat{g}) \quad \textit{and} \quad \mathcal{I}(\hat{g}) = \mathcal{I}(\hat{g}^{\varepsilon}).$$

Definition 7. *Let Σ, Δ, Γ be alphabets.*

1. *A nondeterministic finite* automaton *with empty transitions, or* ε-NFA *for short, is a labelled graph $\hat{a} = (Q, \Sigma_{\boldsymbol{e}}, \delta, I, F)$. If $\text{Labels}(\hat{a}) \subseteq \Sigma$ then $\hat{a}$ is called an* NFA. *The* language *$\mathcal{L}(\hat{a})$ accepted by $\hat{a}$ is the behaviour of $\hat{a}$ with respect to the label set $\Sigma_{\boldsymbol{e}}$.*
2. *An* automaton with set specs *is a labelled graph $\hat{b} = (Q, \text{SSP}[\Gamma], \delta, I, F)$. The* language *$\mathcal{L}(\hat{b})$ accepted by $\hat{b}$ is the behaviour of $\hat{b}$ with respect to $\text{SSP}[\Gamma]$.*
3. *A* transducer (in standard form) *is a labelled graph $\hat{t} = (Q, [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}], \delta, I, F)$. The* relation *$\mathcal{R}(\hat{t})$* realized *by $\hat{t}$ is the behaviour of $\hat{t}$ with respect to $[\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}]$.*

4. *A* transducer with set specs *is a labelled graph* $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$. *The* relation $\mathcal{R}(\hat{s})$ realized *by* $\hat{s}$ *is the behaviour of* $\hat{s}$ *with respect to* $\mathrm{PSP}[\Gamma]$.
5. *An* alphabet invariant transducer *is a labelled graph* $\hat{i} = (Q, \mathrm{PSP}^{\mathrm{invar}}, \delta, I, F)$. *If* Γ *is an alphabet then the* Γ-version *of* $\hat{i}$ *is the transducer with set specs* $\hat{i}[\Gamma] = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$.

Remark 5. The above definitions about automata and transducers are equivalent to the standard ones. The only slight deviation is that, instead of using the empty word ε in transition labels, here we use the empty word symbol $\boldsymbol{e}$. This has two advantages: (i) it allows us to make a uniform presentation of definitions and results and (ii) it is consistent with the use of a symbol for the empty word in regular expressions. As usual about transducers $\hat{t}$, we denote by $\hat{t}(w)$ the *set of outputs of* $\hat{t}$ on input w, that is,

$$\hat{t}(w) = \{u \mid (w, u) \in \mathcal{R}(\hat{t})\}.$$

Moreover, for any language L, we have that $\hat{t}(L) = \cup_{w \in L} \hat{t}(w)$.

Lemma 4. *If* $\hat{b}$ *is an automaton with set specs then* $\exp \hat{b}$ *is an* ε*-NFA. If* $\hat{s}$ *is a transducer with set specs then* $\exp \hat{s}$ *is a transducer (in standard form).*

Convention. Let $\Phi(\hat{u})$ be any statement about the behaviour of an automaton or transducer $\hat{u}$. If $\hat{v}$ is an automaton or transducer with set specs then we make the convention that the statement $\Phi(\hat{v})$ means $\Phi(\exp \hat{v})$. For example, "$\hat{s}$ is an input-altering transducer" means that "$\exp \hat{s}$ is an input-altering transducer"—a transducer $\hat{t}$ is *input-altering* if $u \in \hat{t}(w)$ implies $u \neq w$, or equivalently $(w, w) \notin \mathcal{R}(\hat{t})$, for any word w.

Example 5. The transducers in Fig. 2 are alphabet invariant. Both transducers are much more succinct compared to their expanded Γ-versions, as $|\Gamma| \to \infty$:

$$|\exp \hat{t}_{\mathrm{sub2}}[\Gamma]| = O(|\Gamma|^2) \quad \text{and} \quad |\exp \hat{t}_{\mathrm{px}}[\Gamma]| = O(|\Gamma|).$$

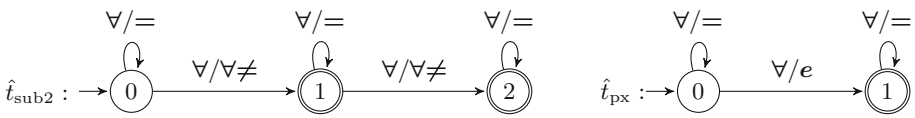

Fig. 2. The left transducer realizes the relation of all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2. The right transducer realizes the relation of all (u, v) such that v is a proper prefix of u.

Following [25], if $\hat{t} = (Q, [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}], \delta, I, F)$ is a transducer then $\hat{t}^{-1}$ is the transducer $(Q, [\Delta_{\boldsymbol{e}}, \Sigma_{\boldsymbol{e}}], \delta', I, F)$, where $\delta' = \{(p, y/x, q) \mid (p, x/y, q) \in \delta\}$, such that $\mathcal{R}(\hat{t}^{-1}) = \mathcal{R}(\hat{t})^{-1}$.

Lemma 5. *For each transducer* $\hat{s}$ *with set specs we have that* $\mathcal{R}(\hat{s}^{-1}) = \mathcal{R}(\hat{s})^{-1}$ *and* $\exp(\hat{s}^{-1}) = (\exp \hat{s})^{-1}$.

6 Regular Expressions over Label Sets

We extend the definitions of regular and 2D regular expressions to include set specs and pairing specs, respectively. We start off with a definition that would work with any label set (called set of atomic formulas in [20]).

Definition 8. *Let B be a label set with behaviour $\mathcal{I}$ such that no $\beta \in B$ contains the special symbol $\oslash$. The set* $\mathrm{REG}\, B$ *of* type B regular expressions *is the set of strings consisting of the 1-symbol string $\oslash$ and the strings in the set Z that is defined inductively as follows.*

- $\underline{\varepsilon_{\mathrm{mon}\ B}}$ *is in Z, and every $\beta \in B$ is in Z.*
- *If $\mathbf{r}, \mathbf{s} \in Z$ then $(\mathbf{r}+\mathbf{s}), (\mathbf{r}\cdot\mathbf{s}), (\mathbf{r}^*)$ are in Z.*

The behaviour $\mathcal{I}(\mathbf{r})$ of a type B regular expression $\mathbf{r}$ is defined inductively as follows.

- $\mathcal{I}(\oslash) = \emptyset$ *and* $\mathcal{I}(\underline{\varepsilon_{\mathrm{mon}\ B}}) = \varepsilon_{\mathrm{mon}\ B}$*;*
- $\mathcal{I}(\beta)$ *is the subset of* $\mathrm{mon}\ B$ *already defined by the behaviour $\mathcal{I}$ on B;*
- $\mathcal{I}(\mathbf{r}+\mathbf{s}) = \mathcal{I}(\mathbf{r}) \cup \mathcal{I}(\mathbf{s})$; $\quad \mathcal{I}(\mathbf{r}\cdot\mathbf{s}) = \mathcal{I}(\mathbf{r})\mathcal{I}(\mathbf{s})$; $\quad \mathcal{I}(\mathbf{r}^*) = \mathcal{I}(\mathbf{r})^*$.

Example 6. Let Σ, Δ be alphabets. Using Σ as a label set, we have that $\mathrm{REG}\, \Sigma$ is the set of ordinary regular expressions over Σ. For the label set $[\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}]$, we have that $\mathrm{REG}[\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}]$ is the set of rational expressions over $\Sigma^* \times \Delta^*$ in the sense of [20].

Example 7. Consider the UNIX utility `tr`. For any strings u, v of length $\ell > 0$, the command `tr` u v can be 'simulated' by the following regular expression of type PSP[ASCII]

$$\Big((\nexists u/=) \ + \ (\exists u[0]/\exists v[0]) \ + \cdots + \ \big(\exists u[\ell-1]/v[\ell-1]\big) \Big)^*$$

where ASCII is the alphabet of standard ASCII characters. Similarly, the command `tr -d` u can be 'simulated' by the type PSP[ASCII] regular expression $\big(\exists u/\boldsymbol{e} + \nexists u/=\big)^*$.

The Thompson method, [22], of converting an ordinary regular expression over Σ—a type $\Sigma_{\boldsymbol{e}}$ regular expression in the present terminology—to an ε-NFA can be extended without complications to work with type B regular expressions, for any label set B. Similarly, the state elimination method of automata, [6], can be extended to labelled graphs of any type B.

Theorem 1. *Let B be a label set with behaviour $\mathcal{I}$. For each type B regular expression $\mathbf{r}$, there is a type B graph $\hat{g}(\mathbf{r})$ such that*

$$\mathcal{I}(\mathbf{r}) = \mathcal{I}\big(\hat{g}(\mathbf{r})\big) \quad \textit{and} \quad |\hat{g}(\mathbf{r})| = O(|\mathbf{r}|).$$

Conversely, for each type B graph $\hat{g}$ there is a type B regular expression $\mathbf{r}$ such that $\mathcal{I}(\hat{g}) = \mathcal{I}(\mathbf{r})$.

Derivatives based methods for the manipulation of regular expressions have been widely studied [2,5,7,8,10,17,18]. In recent years, partial derivative automata were defined and characterised for several kinds of expressions. Not only they are in general more succinct than other equivalent constructions but also for several operators they are easily defined (e.g. for intersection [3] or tuples [11]). The partial derivative automaton of a regular expression over Σ^* was introduced independently by Mirkin [18] and Antimirov [2]. Champarnaud and Ziadi [9] proved that the two formulations are equivalent. Lombardy and Sakarovitch [17] generalised these constructions to weighted regular expressions, and recently Demaille [11] defined derivative automata for multitape weighted regular expressions.

Here we define the partial derivative automaton for a regular expressions of a type B. Given a finite set S of expressions we define its behaviour as $\mathcal{I}(S) = \bigcup_{\mathbf{s}\in S} \mathcal{I}(\mathbf{s})$. We say that two regular expressions $\mathbf{r}, \mathbf{s}$ of a type B are *equivalent*, $\mathbf{r} \sim \mathbf{s}$, if $\mathcal{I}(\mathbf{r}) = \mathcal{I}(\mathbf{s})$. Let the *set of labels* of an expression $\mathbf{r}$ be the set $\mathsf{SS}(\mathbf{r}) = \{\, \beta \mid \beta \in B \text{ and } \beta \text{ occurs in } \mathbf{r} \,\}$. The *size* of an expressions $\mathbf{r}$ is $\|\mathbf{r}\| = |\,\mathsf{SS}(\mathbf{r})|$; it can be inductively defined as follows:

$$\begin{aligned} \|\oslash\| = 0, \quad \|\underline{\varepsilon_{\text{mon } B}}\| = 0, \quad \|\beta\| = 1 \\ \|\mathbf{r} + \mathbf{s}\| = \|\mathbf{r}\| + \|\mathbf{s}\| \\ \|\mathbf{rs}\| = \|\mathbf{r}\| + \|\mathbf{s}\| \\ \|\mathbf{r}^*\| = \|\mathbf{r}\|. \end{aligned}$$

We define the *constant part* $\mathsf{c} : \mathrm{REG}\, B \to \{\underline{\varepsilon_{\text{mon } B}}, \oslash\}$ by $\mathsf{c}(\mathbf{r}) = \underline{\varepsilon_{\text{mon } B}}$ if $\varepsilon_{\text{mon } B} \in \mathcal{I}(\mathbf{r})$, and $\mathsf{c}(\mathbf{r}) = \oslash$ otherwise. This function is extended to sets of expressions by $\mathsf{c}(S) = \underline{\varepsilon_{\text{mon } B}}$ if and only if exists $\mathbf{r} \in S$ such that $\mathsf{c}(\mathbf{r}) = \underline{\varepsilon_{\text{mon } B}}$.

The *linear form* of a regular expression $\mathbf{r}$, $\mathsf{n}(\mathbf{r})$, is given by the following inductive definition:

$$\begin{aligned} \mathsf{n}(\oslash) = \mathsf{n}(\underline{\varepsilon_{\text{mon } B}}) &= \emptyset, \\ \mathsf{n}(\beta) &= \{(\beta, \underline{\varepsilon_{\text{mon } B}})\}, \\ \mathsf{n}(\mathbf{r} + \mathbf{r}') &= \mathsf{n}(\mathbf{r}) \cup \mathsf{n}(\mathbf{r}'), \\ \mathsf{n}(\mathbf{r}\mathbf{r}') &= \begin{cases} \mathsf{n}(\mathbf{r})\mathbf{r}' \cup \mathsf{n}(\mathbf{r}') & \text{if } \mathsf{c}(\mathbf{r}) = \underline{\varepsilon_{\text{mon } B}}, \\ \mathsf{n}(\mathbf{r})\mathbf{r}' & \text{otherwise,} \end{cases} \\ \mathsf{n}(\mathbf{r}^*) &= \mathsf{n}(\mathbf{r})\mathbf{r}^*, \end{aligned}$$

where for any $S \subseteq B \times \mathrm{REG}\, B$, we define $S\underline{\varepsilon_{\text{mon } B}} = \underline{\varepsilon_{\text{mon } B}} S = S$, $S\oslash = \oslash S = \oslash$, and $S\mathbf{s} = \{\, (\beta, \mathbf{rs}) \mid (\beta, \mathbf{r}) \in S \,\}$ if $\mathbf{s} \neq \underline{\varepsilon_{\text{mon } B}}$ (and analogously for $\mathbf{s}S$). Let $\mathcal{I}(\mathsf{n}(\mathbf{r})) = \bigcup_{(\beta,\mathbf{s})\in \mathsf{n}(\mathbf{r})} \mathcal{I}(\beta)\mathcal{I}(\mathbf{s})$.

Lemma 6. *For all* $\mathbf{r} \in \mathrm{REG}\, B$, $\mathbf{r} \sim \mathsf{c}(\mathbf{r}) \cup \mathsf{n}(\mathbf{r})$.

Proof. The proof is trivial proceeding by induction on $\mathbf{r}$. □

For a regular expression $\mathbf{r}$ and $\beta \in \mathsf{SS}(\mathbf{r})$, the *set of partial derivatives* of $\mathbf{r}$ w.r.t. β is

$$\hat{\partial}_\beta(\mathbf{r}) = \{\, \mathbf{s} \mid (\beta, \mathbf{s}) \in \mathsf{n}(\mathbf{r}) \,\}.$$

It is clear that we can iteratively compute the linear form of an expression $\mathbf{s} \in \hat{\partial}_\beta(\mathbf{r})$, for $\beta \in \mathsf{SS}(\mathbf{r})$. The set of all the resulting expressions is denoted by $\pi(\mathbf{r})$, and $\mathsf{PD}(\mathbf{r}) = \pi(\mathbf{r}) \cup \{\mathbf{r}\}$ is the set of partial derivatives of $\mathbf{r}$.

The *partial derivative graph* of $\mathbf{r}$ is the labeled graph

$$\hat{a}_{\mathrm{PD}}(\mathbf{r}) = \big(\mathsf{PD}(\mathbf{r}), B, \delta_{\mathrm{PD}}, \{\mathbf{r}\}, F\big),$$

where $F = \{\, \mathbf{r}_1 \in \mathsf{PD}(\mathbf{r}) \mid \mathsf{c}(\mathbf{r}_1) = \underline{\varepsilon_{\mathrm{mon}\ B}} \,\}$, $\delta_{\mathrm{PD}} = \varphi(\mathbf{r}) \cup \mathsf{F}(\mathbf{r})$ with $\varphi(\mathbf{r}) = \{(\mathbf{r}, \beta, \mathbf{r}') \mid (\beta, \mathbf{r}') \in \mathsf{n}(\mathbf{r})\}$ and $\mathsf{F}(\mathbf{r}) = \{\, (\mathbf{r}_1, \beta, \mathbf{r}_2) \mid \mathbf{r}_1 \in \pi(\mathbf{r}) \wedge \beta \in \mathsf{SS}[\mathbf{r}] \wedge \mathbf{r}_2 \in \hat{\partial}_\beta(\mathbf{r}_1) \,\}$.

The following lemma generalizes from ordinary regular expressions [5,9,18], and shows that the set of (strict) partial derivatives is finite.

Lemma 7. *π satisfies the following:*

$$\begin{aligned} \pi(\oslash) = \pi(\underline{\varepsilon_{\mathrm{mon}\ B}}) &= \emptyset, & \pi(\mathbf{r}_1 + \mathbf{r}_2) &= \pi(\mathbf{r}_1) \cup \pi(\mathbf{r}_2), \\ \pi(\beta) &= \{\underline{\varepsilon_{\mathrm{mon}\ B}}\}, & \pi(\mathbf{r}_1\mathbf{r}_2) &= \pi(\mathbf{r}_1)\mathbf{r}_2 \cup \pi(\mathbf{r}_2), \\ \pi(\mathbf{r}^*) &= \pi(\mathbf{r})\mathbf{r}^*. \end{aligned}$$

Theorem 2. *We have that $|\pi(\mathbf{r})| \leq \|\boldsymbol{r}\|$ and $|\mathsf{PD}(\boldsymbol{r})| \leq \|\boldsymbol{r}\| + 1$.*

Proof. Direct consequence of Lemma 7 using induction on $\mathbf{r}$. □

The proof of the following result is analogous to the ones for ordinary regular expressions [2,18].

Theorem 3. $\mathcal{I}(\hat{a}_{\mathrm{PD}}(\mathbf{r})) = \mathcal{I}(\mathbf{r})$.

A direct algorithm to decide if an element of mon B belongs to $\mathcal{I}(\mathbf{r})$ depends on the behaviour $\mathcal{I}$ of the particular label set B.

6.1 Regular Expressions with Set Specifications

Here we consider regular expressions of type $\mathrm{SSP}[\Sigma]$ whose fixed behaviours are languages over the alphabet Σ. We want a direct algorithm to decide if a word belongs to the language represented by the expression. Given $L_1, L_2 \subseteq \Sigma^*$ and $x \in \Sigma$, the *quotient* of a language[1] w.r.t x satisfies the following relations

$$\begin{aligned} &x^{-1}(L_1 \cup L_2) = x^{-1}L_1 \cup x^{-1}L_2, \quad x^{-1}L_1^* = (x^{-1}L_1)L_1^*, \\ &x^{-1}(L_1\ L_2) = (x^{-1}L_1)L_2 \text{ if } \varepsilon \notin L_1 \text{ or } (x^{-1}L_1)L_2 \cup x^{-1}L_2 \text{ if } \varepsilon \in L_1. \end{aligned}$$

Quotients can be extended to words and languages: ${\varepsilon^{-1}}L = L$, ${(wx)^{-1}}L = {x^{-1}}({w^{-1}}L)$ and ${L_1}^{-1}L = \bigcup_{w \in L_1} {w^{-1}}L$. If $L_1 \subseteq L_2 \subseteq \Sigma^*$ then ${L_1}^{-1}L \subseteq {L_2}^{-1}L$ and $L^{-1}L_1 \subseteq L^{-1}L_2$.

[1] It is customary to use $x^{-1}L$ for denoting quotients; this should not be confused with the inverse p^{-1} of a pairing spec.

Given two set specifications $F, G \in \mathrm{SSP}[\Sigma] \setminus \{\boldsymbol{e}\}$ we extend the notion of partial derivative to the set of partial derivatives of F w.r.t G with possible $F \neq G$, by

$$\partial_F(G) = \begin{cases} \{\boldsymbol{e}\} & \text{if } F \cap G \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}$$

and $\partial_F(\mathbf{r}) = \hat{\partial}_F(\mathbf{r})$ for all other cases of $\mathbf{r}$.

The set of partial derivatives of $\mathbf{r} \in \mathrm{REG}\,\mathrm{SSP}[\Sigma]$ w.r.t. a word $x \in (\mathrm{SSP}[\Sigma] \setminus \{\boldsymbol{e}\})^\star$ is inductively defined by $\partial_\varepsilon(\mathbf{r}) = \{\mathbf{r}\}$ and $\partial_{xF}(\mathbf{r}) = \partial_F(\partial_x(\mathbf{r}))$, where, given a set $S \subseteq \mathrm{REG}\,\mathrm{SSP}[\Sigma]$, $\partial_F(S) = \bigcup_{\mathbf{r} \in S} \partial_F(\mathbf{r})$. Moreover one has $\mathcal{L}(\partial_x(\mathbf{r})) = \bigcup_{\mathbf{r}_1 \in \partial_x(\mathbf{r})} \mathcal{L}(\mathbf{r}_1)$.

Lemma 8. *For two set specifications $F, G \in \mathrm{SSP}[\Sigma]$, $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \{\varepsilon\}$ if $F \cap G \neq \bot$, and $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \emptyset$ otherwise.*

For instance, if $\exists w \cap \nexists u \neq \bot$ then

$$\mathcal{L}(\exists w)^{-1}\mathcal{L}(\nexists u) = \bigcup_{x \in \mathrm{alph}\ w} x^{-1}(\Sigma \setminus \mathrm{alph}\ u) = \{\varepsilon\}.$$

Lemma 9. *For all $\boldsymbol{r} \in \mathrm{REG}\,\mathrm{SSP}[\Sigma]$ and $F \in \mathrm{SSP}[\Sigma]$, $\mathcal{L}(F)^{-1}\mathcal{L}(\boldsymbol{r}) = \mathcal{L}(\partial_F(\boldsymbol{r}))$.*

Proof. For $\mathbf{r} = \emptyset$ and $\mathbf{r} = \boldsymbol{e}$ it is obvious. For $\mathbf{r} = G$ the result follows from Lemma 8. In fact, if $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \{\varepsilon\}$ then $\partial_F(G) = \{\boldsymbol{e}\}$ and thus $\mathcal{L}(\partial_F(\mathbf{r})) = \{\varepsilon\}$; otherwise if $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \emptyset$ then $\partial_F(G) = \emptyset$, and also $\mathcal{L}(\partial_F(\mathbf{r})) = \emptyset$. The remaining cases follow by induction as with ordinary regular expressions. □

Lemma 10. *For all $g \in (\mathrm{SSP}[\Sigma] \setminus \{\boldsymbol{e}\})^*$, $\mathcal{L}(g)^{-1}\mathcal{L}(\boldsymbol{r}) = \mathcal{L}(\partial_g(\boldsymbol{r}))$.*

Proof. By induction on $|g|$ using Lemma 9. □

Lemma 11. *For all $w \in \Sigma^*$, the following propositions are equivalent:*

1. $w \in \mathcal{L}(\boldsymbol{r})$
2. *$w = x_1 \cdots x_n$ and there exists $\mathsf{s}(w) = F_1 \cdots F_n$ with $F_i \in \mathsf{SS}(\boldsymbol{r})$, $\exists x_i \cap F_i \neq \bot$ and $\mathsf{c}(\partial_{\mathsf{s}(w)}(\boldsymbol{r})) = \varepsilon$.*

7 Label Operations and the Product Construction

We shall consider partial operations $\odot$ on label sets B, B' such that, when defined, the product $\beta \odot \beta'$ of two labels belongs to a certain label set C. Moreover, we shall assume that $\odot$ is also a partial operation on mon B, mon B' such that, when defined, the product $m \odot m'$ of two monoid elements belongs to mon C. We shall call $\odot$ a *polymorphic* operation (in analogy to polymorphic operations in programming languages) when $\mathcal{I}(\beta \odot \beta') = \mathcal{I}_1(\beta) \odot \mathcal{I}_1(\beta')$ where $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}$ are the behaviours of B, B', C. This concept shall allow us to also use $\odot$ as the name of the product construction on labelled graphs that respects the behaviours of the two graphs.

Example 8. We shall consider the following monoidal operations, which are better known when applied to subsets of the monoid.

- $\cap : \Sigma^* \times \Sigma^* \dashrightarrow \Sigma^*$ such that $u \cap v = u$ if $u = v$; else, $u \cap v = \perp$. Of course, for any two languages $K, L \subseteq \Sigma^*$, $K \cap L$ is the usual intersection of K, L.
- $\circ : (\Sigma_1^* \times \Delta^*) \times (\Delta^* \times \Sigma_2^*) \dashrightarrow (\Sigma_1^* \times \Sigma_2^*)$ such that $(u, v) \circ (w, z) = (u, z)$ if $v = w$; else, $(u, v) \circ (w, z) = \perp$. For any two relations R, S, $R \circ S$ is the usual composition of R, S.
- $\downarrow : (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$ such that $(u, v) \downarrow w = (u, v)$ if $u = w$; else, $(u, v) \downarrow w = \perp$. For a relation R and language L,

$$R \downarrow L = R \cap (L \times \Delta^*). \tag{2}$$

- $\uparrow : (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$ such that $(u, v) \uparrow w = (u, v)$ if $v = w$; else, $(u, v) \downarrow w = \perp$. For a relation R and language L,

$$R \uparrow L = R \cap (\Sigma^* \times L). \tag{3}$$

Definition 9. *Let B, B', C be label sets with behaviours $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}$, respectively. A polymorphic operation $\odot$ over B, B', C, denoted as "$\odot : B \times B' \Rightarrow C$", is defined as follows.*

- *It is a partial mapping:* $\odot : B \times B' \dashrightarrow C$.
- *It is a partial mapping:* $\odot : \text{mon } B \times \text{mon } B' \dashrightarrow \text{mon } C$.
- *For all $\beta \in B$ and $\beta' \in B'$ we have*

$$\mathcal{I}(\beta \odot \beta') = \mathcal{I}_1(\beta) \odot \mathcal{I}_2(\beta'),$$

where we assume that $\mathcal{I}(\beta \odot \beta') = \emptyset$, if $\beta \odot \beta' = \perp$; and we have used the notation $S \odot S' = \{m \odot m' \mid m \in S, m' \in S', m \odot m' \neq \perp\}$. for any $S \subseteq \text{mon } B$ and $S' \subseteq \text{mon } B'$.

Example 9. The following polymorphic operations are based on label sets of standard automata and transducers using the monoidal operations in Example 8.

- "$\cap : \Sigma_e \times \Sigma_e \Rightarrow \Sigma_e$" is defined by
  - the partial operation $\cap : \Sigma_e \times \Sigma_e \dashrightarrow \Sigma_e$ such that $x \cap y = x$, if $x = y$, else $x \cap y = \perp$; and
  - the partial operation $\cap : \Sigma^* \times \Sigma^* \dashrightarrow \Sigma^*$.

  Obviously, $\mathcal{L}(x \cap y) = \mathcal{L}(x) \cap \mathcal{L}(y)$.
- "$\circ : [\Sigma_e, \Delta_e] \times [\Delta_e, \Sigma'_e] \Rightarrow [\Sigma_e, \Sigma'_e]$" is defined by
  - the operation $\circ : [\Sigma_e, \Delta_e] \times [\Delta_e, \Sigma'_e] \dashrightarrow [\Sigma_e, \Sigma'_e]$ such that $(x/y_1) \circ (y_2/z) = (x/z)$ if $y_1 = y_2$, else $(x/y_1) \circ (y_2/z) = \perp$; and
  - the operation $\circ : (\Sigma^* \times \Delta^*) \times (\Delta^* \times \Sigma'^*) \dashrightarrow (\Sigma^* \times \Sigma'^*)$.

  Obviously, $\mathcal{R}((x, y_1) \circ (y_2, z)) = \mathcal{R}((x, y_1)) \circ \mathcal{R}((y_2, z))$.
- "$\downarrow : [\Sigma_e, \Delta_e] \times \Sigma_e \Rightarrow [\Sigma_e, \Delta_e]$" is defined by
  - the operation $\downarrow : [\Sigma_e, \Delta_e] \times \Sigma_e \dashrightarrow [\Sigma_e, \Delta_e]$ such that $(x/y) \downarrow z = (x/y)$ if $x = z$, else $(x/y) \downarrow z = \perp$; and
  - the operation $\downarrow : (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$.

Obviously, $\mathcal{R}((x/y) \downarrow z) = \mathcal{R}(x/y) \downarrow \mathcal{L}(z)$.
- "$\uparrow\colon [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}] \times \Delta_{\boldsymbol{e}} \Rightarrow [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}]$" is defined by
  - the operation $\uparrow\colon [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}] \times \Delta_{\boldsymbol{e}} \dashrightarrow [\Sigma_{\boldsymbol{e}}, \Delta_{\boldsymbol{e}}]$ such that $(x/y) \uparrow z = (x/y)$ if $y = z$, else $(x/y) \uparrow z = \bot$; and
  - the operation $\uparrow\colon (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$.

  Obviously, $\mathcal{R}((x/y) \uparrow z) = \mathcal{R}(x/y) \uparrow \mathcal{L}(z)$.

Example 10. The following polymorphic operations are based on label sets of automata and transducers with set specs.

- "$\cap$: $\mathrm{SSP}[\Gamma] \times \mathrm{SSP}[\Gamma] \Rightarrow \mathrm{SSP}[\Gamma]$" is defined by the partial operation $\cap$: $\mathrm{SSP}[\Gamma] \times \mathrm{SSP}[\Gamma] \dashrightarrow \mathrm{SSP}[\Gamma]$, according to Definition 1, and the partial operation $\cap$: $\Gamma^* \times \Gamma^* \dashrightarrow \Gamma^*$. For any $B, F \in \mathrm{SSP}[\Gamma]$, we have $\mathcal{L}(B \cap F) = \mathcal{L}(B) \cap \mathcal{L}(F)$.
- "$\downarrow\colon \mathrm{PSP}[\Gamma] \times \Gamma_{\boldsymbol{e}} \Rightarrow \mathrm{PSP}[\Gamma]$" is defined as follows. First, by the partial operation $\downarrow\colon \mathrm{PSP}[\Gamma] \times \Gamma_{\boldsymbol{e}} \dashrightarrow \mathrm{PSP}[\Gamma]$ such that

$$\mathsf{p} \downarrow x = \begin{cases} \boldsymbol{e}/\operatorname{right}\mathsf{p}, & \text{if } x = \boldsymbol{e} \text{ and } \operatorname{left}\mathsf{p} = \boldsymbol{e}; \\ \exists x/\operatorname{right}\mathsf{p}, & \text{if } x, \operatorname{left}\mathsf{p} \neq \boldsymbol{e} \text{ and } x \in \mathcal{L}(\operatorname{left}\mathsf{p}); \\ \bot, & \text{otherwise.} \end{cases}$$

  Second, by the partial operation $\downarrow\colon (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$. We have that $\mathcal{R}(\mathsf{p} \downarrow x) = \mathcal{R}(\mathsf{p}) \downarrow \mathcal{L}(x)$. Moreover we have that $\mathsf{p} \downarrow x$ can be computed from p and x in time $O(|\mathsf{p}|)$.
- "$\uparrow\colon \mathrm{PSP}[\Gamma] \times \Delta_{\boldsymbol{e}} \Rightarrow \mathrm{PSP}[\Gamma]$" is defined as follows. First, by the partial operation $\uparrow\colon \mathrm{PSP}[\Gamma] \times \Delta_{\boldsymbol{e}} \dashrightarrow \mathrm{PSP}[\Gamma]$ such that $\mathsf{p} \uparrow x = (\mathsf{p}^{-1} \downarrow x)^{-1}$. Second, by the partial operation $\uparrow\colon (\Sigma^* \times \Delta^*) \times \Delta^* \dashrightarrow (\Sigma^* \times \Delta^*)$. We have that $\mathcal{R}(\mathsf{p} \uparrow x) = \mathcal{R}(\mathsf{p}) \uparrow \mathcal{L}(x)$. Moreover we have that $\mathsf{p} \uparrow x$ can be computed from p and x in time $O(|\mathsf{p}|)$.

Further below, in Sect. 9, we define the polymorphic operation '$\circ$' between pairing specs.

Definition 10. *Let $\hat{g} = (Q, B, \delta, I, F)$ and $\hat{g}' = (Q', B', \delta', I', F')$ be type B and B', respectively, graphs and let "$\odot : B \times B' \Rightarrow C$" be a polymorphic operation. The* product *$\hat{g} \odot \hat{g}'$ is the type C graph*

$$\big(P, C, \delta \odot \delta', I \times I', F \times F'\big)$$

defined as follows. First make the following two possible modifications on $\hat{g}, \hat{g}'$: if there is a label β in $\hat{g}$ such that $\varepsilon_{\mathrm{mon}\ B} \in \mathcal{I}(\beta)$ then modify $\hat{g}'$ to $\hat{g}'^{\varepsilon}$; and if there is a label β' in $\hat{g}'$ (before being modified) such that $\varepsilon_{\mathrm{mon}\ B'} \in \mathcal{I}(\beta')$ then modify $\hat{g}'$ to $\hat{g}'^{\varepsilon}$. In any case, use the same names $\hat{g}$ and $\hat{g}'$ independently of whether they were modified. Then P and $\delta \odot \delta'$ are defined inductively as follows:

1. *$I \times I' \subseteq P$.*
2. *If $(p, p') \in P$ and there are $(p, \beta, q) \in \delta$ and $(p', \beta', q') \in \delta'$ with $\beta \odot \beta' \neq \bot$ then $(q, q') \in P$ and $\big((p, p'), \beta \odot \beta', (q, q')\big) \in \delta \odot \delta'$.*

Example 11. Here we recall three known examples of product constructions involving automata and transducers.

1. For two ε-NFAs $\hat{a}, \hat{a}'$, using the polymorphic operation "$\cap : \Sigma_e \times \Sigma_e \Rightarrow \Sigma_e$", the product construction produces the ε-NFA $\hat{a} \cap \hat{a}'$ such that $\mathcal{L}(\hat{a} \cap \hat{a}') = \mathcal{L}(\hat{a}) \cap \mathcal{L}(\hat{a}')$. Note that if $\hat{a}, \hat{a}'$ are NFAs then also $\hat{a} \cap \hat{a}'$ is an NFA.
2. For two transducers $\hat{t}, \hat{t}'$, using the polymorphic operation "$\circ : [\Sigma_e, \Delta_e] \times [\Delta_e, \Sigma'_e] \Rightarrow [\Sigma_e, \Sigma'_e]$", the product construction produces the transducer $\hat{t} \circ \hat{t}'$ such that $\mathcal{R}(\hat{t} \circ \hat{t}') = \mathcal{R}(\hat{t}) \circ \mathcal{R}(\hat{t}')$.
3. For a transducer $\hat{t}$ and an ε-NFA $\hat{a}$, using the polymorphic operation "$\downarrow : [\Sigma_e, \Delta_e] \times \Sigma_e \Rightarrow [\Sigma_e, \Delta_e]$", the product construction produces the transducer $\hat{t} \downarrow \hat{a}$ such that $\mathcal{R}(\hat{t} \downarrow \hat{a}) = \mathcal{R}(\hat{t}) \downarrow \mathcal{L}(\hat{a})$. Similarly, using the polymorphic operation "$\uparrow : [\Sigma_e, \Delta_e] \times \Delta_e \Rightarrow [\Sigma_e, \Delta_e]$", the product construction produces the transducer $\hat{t} \uparrow \hat{a}$ such that $\mathcal{R}(\hat{t} \uparrow \hat{a}) = \mathcal{R}(\hat{t}) \uparrow \mathcal{L}(\hat{a})$. These product constructions were used in [14] to answer algorithmic questions about independent languages—see Sect. 11.

Lemma 12. *The following statements hold true about the product graph* $\hat{g} \odot \hat{g}' = (P, C, \delta \odot \delta', I \times I', F \times F')$ *of two trim labelled graphs* $\hat{g}, \hat{g}'$ *as defined in Definition 10.*

1. $|P| = O(|\delta||\delta'|)$ *and* $|\delta \odot \delta'| \leq |\delta||\delta'|$.
2. *If the value* $\beta \odot \beta'$ *can be computed from the labels* β *and* β' *in time, and is of size,* $O(|\beta| + |\beta'|)$*, then* $\|\delta \odot \delta'\|$ *is of magnitude* $O(|\delta|\|\delta'\| + |\delta'|\|\delta\|)$ *and* $\delta \odot \delta'$ *can be computed within time of the same order of magnitude.*

Theorem 4. *If "*$\odot : B \times B' \Rightarrow C$*" is a polymorphic operation and* $\hat{g}, \hat{g}'$ *are type* B, B'*, respectively, graphs then* $\hat{g} \odot \hat{g}'$ *is a type* C *graph such that*

$$\mathcal{I}(\hat{g} \odot \hat{g}') = \mathcal{I}(\exp \hat{g} \odot \exp \hat{g}').$$

How to Apply the Above Theorem. We can apply the theorem when we have a known product construction $\odot$ on labelled graphs $\hat{u}, \hat{u}'$ over monoids M, M' (see Example 11) and we wish to apply a 'higher level' version of $\odot$; that is, apply $\odot$ on labelled graphs $\hat{g}, \hat{g}'$ with behaviours in the monoids M, M'. This would avoid expanding $\hat{g}$ and $\hat{g}'$. We apply the theorem in Lemma 13.2, in Theorem 5 and in Corollary 1.

8 Automata and Transducers with Set Specifications

Here we present some basic algorithms on automata and transducers with set specs. These can be applied to answer the satisfaction question about independent languages (see Sect. 11).

Lemma 13. *Let* $\hat{b} = (Q, \mathrm{SSP}[\Gamma], \delta, I, F)$ *and* $\hat{b}' = (Q', \mathrm{SSP}[\Gamma], \delta', I', F')$ *be trim automata with set specs and let* w *be a string.*

1. *There is a* $O(|\hat{b}|)$ *algorithm* $\texttt{nonEmptyW}(\hat{b})$ *returning either a word in* $\mathcal{L}(\hat{b})$*, or* `None` *if* $\mathcal{L}(\hat{b}) = \emptyset$*. The decision version of this algorithm,* $\texttt{emptyP}(\hat{b})$*, simply returns whether* $\mathcal{L}(\hat{b})$ *is empty.*
2. *There is a* $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$ *algorithm returning the automaton with set specs* $\hat{b} \cap \hat{b}'$ *such that* $\mathcal{L}(\hat{b} \cap \hat{b}') = \mathcal{L}(\hat{b}) \cap \mathcal{L}(\hat{b}')$*.*
3. *There is a* $O(|w||\hat{b}|)$ *algorithm returning whether* $w \in \mathcal{L}(\hat{b})$*.*

Proof. (Partial) For the second statement, we compute the product $\hat{b} \cap \hat{b}'$. As the value $\beta \cap \beta'$ of two labels can be computed in linear time, Lemma 12 implies that $\hat{b} \cap \hat{b}'$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Now we have

$$\begin{aligned} \mathcal{L}(\hat{b} \cap \hat{b}') &= \mathcal{L}(\exp \hat{b} \cap \exp \hat{b}') && (4)\\ &= \mathcal{L}(\exp \hat{b}) \cap \mathcal{L}(\exp \hat{b}') && (5)\\ &= \mathcal{L}(\hat{b}) \cap \mathcal{L}(\hat{b}') && (6) \end{aligned}$$

Equation (4) follows from the fact that "$\cap : \mathrm{SSP}[\Gamma] \times \mathrm{SSP}[\Gamma] \Rightarrow \mathrm{SSP}[\Gamma]$" is a polymorphic operation—see Theorem 4 and Example 10. Equation (5) follows from the fact that each $\exp \hat{b}, \exp \hat{b}'$ is an ε-NFA and the operation $\cap$ is well-defined on these objects—see Lemma 4 and Example 11. For the third statement, one makes an automaton with set specs $\hat{b}_w$ accepting $\{w\}$, then computes $\hat{a} = \hat{b}_w \cap \hat{b}$, and then uses $\texttt{emptyP}(\hat{a})$ to get the desired answer.

Lemma 14. *Let,* $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ *be a trim transducer with set specs and let* $\hat{a} = (Q', \Gamma_e, \delta', I', F')$ *be a trim* ε*-NFA, and let* (u, v) *be a pair of words.*

1. *There is a* $O(|\hat{s}|)$ *algorithm* $\texttt{nonEmptyW}(\hat{s})$ *returning either a word pair in* $\mathcal{R}(\hat{s})$*, or* `None` *if* $\mathcal{R}(\hat{s}) = \emptyset$*. The decision version of this algorithm,* $\texttt{emptyP}(\hat{s})$*, simply returns whether* $\mathcal{R}(\hat{s})$ *is empty.*
2. *There is a* $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$ *algorithm returning the transducer with set specs* $\hat{s} \downarrow \hat{a}$ *such that* $\mathcal{R}(\hat{s} \downarrow \hat{a}) = \mathcal{R}(\hat{s}) \downarrow \mathcal{L}(\hat{a})$*.*
3. *There is a* $O(|u||v||\hat{s}|)$ *algorithm returning whether* $(u, v) \in \mathcal{R}(\hat{s})$*.*

9 Composition of Transducers with Set Specifications

Next we are interested in defining the composition $\mathsf{p}_1 \circ \mathsf{p}_2$ of two pairing specs in a way that $\mathcal{R}(\mathsf{p}_1) \circ \mathcal{R}(\mathsf{p}_2)$ is equal to $\mathcal{R}(\mathsf{p}_1 \circ \mathsf{p}_2)$. It turns out that, for a particular subcase about the structure of $\mathsf{p}_1, \mathsf{p}_2$, the operation $\mathsf{p}_1 \circ \mathsf{p}_2$ can produce two or three pairing specs. To account for this, we define a new label set:

$\mathrm{PSP}_+[\Gamma]$ consists of strings $\mathsf{p}_1 \oplus \cdots \oplus \mathsf{p}_\ell$,

where $\ell \in \mathbb{N}$ and each $\mathsf{p}_i \in \mathrm{PSP}[\Gamma]$. Moreover we have the (fixed) label behaviour $\mathcal{R} : \mathrm{PSP}_+[\Gamma] \to 2^{\Gamma^* \times \Gamma^*}$ such that

$$\mathcal{R}(\mathsf{p}_1 \oplus \cdots \oplus \mathsf{p}_\ell) = \mathcal{R}(\mathsf{p}_1) \cup \cdots \cup \mathcal{R}(\mathsf{p}_\ell).$$

Definition 11. *Let Γ be an alphabet of reference. The partial operation*

$$\circ : \mathrm{PSP}[\Gamma] \times \mathrm{PSP}[\Gamma] \dashrightarrow \mathrm{PSP}_+[\Gamma]$$

is defined between any two pairing specs $\mathsf{p}_1, \mathsf{p}_2$ respecting Γ as follows.

$$\mathsf{p}_1 \circ \mathsf{p}_2 = \bot, \quad \text{if } \mathcal{L}(\mathrm{rset}\,\mathsf{p}_1) \cap \mathcal{L}(\mathrm{left}\,\mathsf{p}_2) = \emptyset.$$

Now we assume that the above condition is not true and we consider the structure of p_1 and p_2 according to Definition 3(1) using A, B, F, G, W, X, Y, Z as set specs, where $A, B, F, G \neq \boldsymbol{e}$—thus, we assume below that $\mathcal{L}(B) \cap \mathcal{L}(F) \neq \emptyset$ and $\mathcal{L}(X) \cap \mathcal{L}(Y) \neq \emptyset$.

$$(W/X) \circ (Y/Z) = W/Z \qquad (W/B) \circ (F/{=}) = W/B \cap F$$

$$(W/B) \circ \big(F/G{\neq}\big) = \begin{cases} W/G, & \text{if } |\mathcal{L}(B \cap F)| \geq 2 \\ W/\,G \cap \not\ni b, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(G) \setminus \{b\} \neq \emptyset \\ \bot, & \text{otherwise.} \end{cases}$$

$$(B/{=}) \circ (F/Z) \;=\; B \cap F/Z \qquad (B/{=}) \circ (F/{=}) \;=\; B \cap F/{=}$$

$$(B/{=}) \circ (F/G{\neq}) = \begin{cases} \bot, & \text{if } \mathcal{L}(G) = \mathcal{L}(B \cap F) = \{g\} \\ B \cap F/G{\neq}, & \text{otherwise.} \end{cases}$$

$$(A/B{\neq}) \circ (F/Z) = \begin{cases} A/Z, & \text{if } |\mathcal{L}(B \cap F)| \geq 2 \\ A \cap \not\ni b/Z, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(A) \setminus \{b\} \neq \emptyset \\ \bot & \text{otherwise.} \end{cases}$$

$$(A/B{\neq}) \circ (F/{=}) = \begin{cases} \bot, & \text{if } \mathcal{L}(A) = \mathcal{L}(B \cap F) = \{a\} \\ A/B \cap F{\neq}, & \text{otherwise.} \end{cases}$$

$$(A/B{\neq}) \circ (F/G{\neq}) = \begin{cases} A/G, & \text{if } |\mathcal{L}(B \cap F)| \geq 3 \\ A \cap \not\ni b/G \cap \not\ni b, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(A) \setminus \{b\} \\ & \quad \neq \emptyset \text{ and } \mathcal{L}(G) \setminus \{b\} \neq \emptyset \\ \boldsymbol{D}, & \text{if } \mathcal{L}(B \cap F) = \{b_1, b_2\} \\ \bot, & \text{otherwise.} \end{cases}$$

where $\boldsymbol{D}$ consists of up to three $\oplus$-terms as follows: $\boldsymbol{D}$ includes $A \cap \not\ni b_1 b_2/G$, if $\mathcal{L}(A) \setminus \{b_1, b_2\} \neq \emptyset$; $\boldsymbol{D}$ includes $\exists b_1/G \cap \not\ni b_2$, if $b_1 \in \mathcal{L}(A)$ and $\mathcal{L}(G) \setminus \{b_2\} \neq \emptyset$; $\boldsymbol{D}$ includes $\exists b_2/G \cap \not\ni b_1$, if $b_2 \in \mathcal{L}(A)$ and $\mathcal{L}(G) \setminus \{b_1\} \neq \emptyset$; $\boldsymbol{D} = \bot$ if none of the previous three conditions is true.

Remark 6. In the above definition, we have omitted cases where $\mathsf{p}_1 \circ \mathsf{p}_2$ is obviously undefined. For example, as $F/{=}$ and $F/G{\neq}$ are only defined when $F, G \neq \boldsymbol{e}$, we omit the case $(W/\boldsymbol{e}) \circ (F/{=})$.

Remark 7. If we allowed $\bot$ to be a pairing spec, then the set $\mathrm{PSP}[\Gamma]$ with the composition operation '$\circ$' would be 'nearly' a semigroup: the subcase

"$(A/B{\neq}) \circ (F/G{\neq})$ with $\mathcal{L}(B \cap F) = \{b_1, b_2\}$" in the above definition is the only one where the result of the composition is not necessarily a single pairing spec. For example, let the alphabet Γ be $\{0, 1, 2\}$ and $A = \exists 01$, $B = F = \exists 12$, and $G = \exists 012$. Then,

$$\mathcal{R}(A/B{\neq}) \circ \mathcal{R}(F/G{\neq}) = \{(0,0), (0,1), (0,2), (1,0), (1,1)\},$$

which is equal to $\mathcal{R}(\{\exists 0/\exists 012,\ \exists 1/\exists 01\})$. This relation is not equal to $\mathcal{R}(\mathsf{p})$, for any pairing spec p.

Lemma 15. *The relation $\mathcal{R}(\mathsf{p}_1 \circ \mathsf{p}_2)$ is equal to $\mathcal{R}(\mathsf{p}_1) \circ \mathcal{R}(\mathsf{p}_2)$, for any pairing specs $\mathsf{p}_1, \mathsf{p}_2$ respecting Γ.*

Remark 8. The polymorphic operation "$\circ : \mathrm{PSP}[\Gamma] \times \mathrm{PSP}[\Gamma] \Rightarrow \mathrm{PSP}_+[\Gamma]$" is well-defined by the partial operations $\circ$ in Definition 11 and in Example 8.

Definition 12. *Let $\hat{t} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ and $\hat{s} = (Q', \mathrm{PSP}[\Gamma], \delta', I', F')$ be transducers with set specs. The transducer $\hat{t} \circledcirc \hat{s}$ with set specs is defined as follows. First compute the transducer $\hat{t} \circ \hat{s}$ with labels in $\mathrm{PSP}_+[\Gamma]$. Then, $\hat{t} \circledcirc \hat{s}$ results when each transition $(p, \mathsf{p}_1 \oplus \cdots \oplus \mathsf{p}_\ell, q)$ of $\hat{t} \circ \hat{s}$, with $\ell > 1$, is replaced with the ℓ transitions (p, p_i, q).*

Theorem 5. *For any two trim transducers $\hat{t} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ and $\hat{s} = (Q', \mathrm{PSP}[\Gamma], \delta', I', F')$ with set specs, $\hat{t} \circledcirc \hat{s}$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Moreover, $\mathcal{R}(\hat{t} \circledcirc \hat{s}) = \mathcal{R}(\hat{t}) \circ \mathcal{R}(\hat{s})$.*

10 Transducer Identity and Functionality

The question of whether a given transducer is functional is of central importance in the theory of rational relations [19]. Also important is the question of whether a given transducer $\hat{t}$ realizes an *identity*, that is, whether $\hat{t}(w) = \{w\}$, when $|\hat{t}(w)| > 0$. In [1], the authors present an algorithm $\texttt{identityP}(\hat{t})$ that works in time $O(|\delta| + |Q||\Delta|)$ and tells whether $\hat{t} = (Q, \Sigma, \Delta, \delta, I, F)$ realizes an identity. Here we have that

$$\text{for trim}\,\hat{t},\ \texttt{identityP}(\hat{t})\ \text{ works in time }\ O(|\delta||\Delta|). \tag{7}$$

The algorithm $\texttt{functionalityP}(\hat{s})$ deciding functionality of a transducer $\hat{t} = (Q, \Gamma, \delta, I, F)$ first constructs the *square transducer* $\hat{u}$, [4], in which the set of transitions $\delta_{\hat{u}}$ consists of tuples $((p, p'), y/y', (q, q'))$ such that $(p, x/y, q)$ and $(p', x/y', q')$ are any transitions in $\hat{t}^{\varepsilon}$. Then, it follows that $\hat{t}$ is functional if and only if $\hat{u}$ realizes an identity. Note that $\hat{u}$ has $O(|\delta|^2)$ transitions and its graph size is $O(|\hat{t}|^2)$. Thus, we have that

$$\text{for trim}\,\hat{t},\ \texttt{functionalityP}(\hat{t})\ \text{ works in time }\ O(|\delta|^2|\Delta|). \tag{8}$$

Theorem 6. *The question of whether a trim transducer $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ with set specs realizes an identity can be answered in time $O\big(|\delta||\Gamma|\big)$.*

Remark 9. Consider the trim transducer $\hat{s}$ with set specs in Theorem 6. Of course one can test whether it realizes an identity by simply using `identityP`$(\exp \hat{s})$, which would work in time $O(|\delta_{\exp}||\Gamma|)$ according to (7). This time complexity is clearly higher than the time $O(|\delta||\Gamma|)$ in the above theorem when $|\delta_{\exp}|$ is of order $|\delta||\Gamma|$ or $|\delta||\Gamma|^2$ (for example if $\hat{s}$ involves labels $\forall/=$ or $\forall/\forall$).

Theorem 7. *The question of whether a trim transducer $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ with set specs is functional can be answered in time $O(|\delta|^2|\Gamma|)$.*

Remark 10. Consider the trim transducer $\hat{s}$ with set specs in Theorem 7. Of course one can simply use `functionalityP`$(\exp \hat{s})$ to test whether $\hat{s}$ is functional, which would work in time $O(|\delta_{\exp}|^2|\Gamma|)$ according to (8). This time complexity is clearly higher than the time $O(|\delta|^2|\Gamma|)$ in the above theorem when $|\delta_{\exp}|$ is of order $|\delta||\Gamma|$ or $|\delta||\Gamma|^2$ (for example if $\hat{s}$ involves labels $\forall/=$ or $\forall/\forall$).

11 Transducers and Independent Languages

Let $\hat{t}$ be a transducer. A language L is called $\hat{t}$-*independent*, [21], if

$$u, v \in L \text{ and } v \in \hat{t}(u) \quad \text{implies} \quad u = v. \tag{9}$$

If $\hat{t}$ is input-altering then, [15], the above condition is equivalent to

$$\hat{t}(L) \cap L = \emptyset. \tag{10}$$

The *property described* by $\hat{t}$ is the set of all $\hat{t}$-independent languages. Main examples of such properties are code-related properties. For example, the transducer $\hat{t}_{\mathrm{sub2}}$ describes all the 1-substitution error-detecting languages and $\hat{t}_{\mathrm{px}}$ describes all prefix codes. The *property satisfaction* question is whether, for given transducer $\hat{t}$ and regular language L, the language L is $\hat{t}$-independent. The *witness* version of this question is to compute a pair (u, v) of different L-words (if exists) violating condition (9).

Remark 11. The witness version of the property satisfaction question for input-altering transducers $\hat{s}$ (see Eq. (10)) can be answered in time $O(|\hat{s}| \cdot |\hat{a}|^2)$, where $\hat{a}$ is the given ε-NFA accepting L (see [15]). This can be done using the function call `nonEmptyW`$(\hat{s} \downarrow \hat{a} \uparrow \hat{a})$. Further below we show that the same question can be answered even when $\hat{s}$ has set specs, and this could lead to time savings.

Corollary 1. *Let $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ be a transducer with set specs and let $\hat{b} = (Q', \Gamma_e, \delta', I', F')$ be an ε-NFA. Each transducer $\hat{s} \downarrow \hat{b}$ and $\hat{s} \uparrow \hat{b}$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Moreover, we have that*

$$\mathcal{R}(\hat{s} \downarrow \hat{b}) = \mathcal{R}(\hat{s}) \downarrow \mathcal{L}(\hat{b}) \quad \text{and} \quad \mathcal{R}(\hat{s} \uparrow \hat{b}) = \mathcal{R}(\hat{s}) \uparrow \mathcal{L}(\hat{b}).$$

Corollary 2. *Consider the witness version of the property satisfaction question for input-altering transducers $\hat{s}$. The question can be answered in time $O(|\hat{s}| \cdot |\hat{a}|^2)$ even when the transducer $\hat{s}$ involved has set specs.*

Example 12. We can apply the above corollary to the transducer $\hat{t}_{\text{sub2}}[\Gamma]$ of Example 5, where Γ is the alphabet of $\hat{b}$, so that we can decide whether a regular language is 1-substitution error-detecting in time $O(|\hat{b}|^2)$. On the other hand, if we used the ordinary transducer $\exp \hat{t}_{\text{sub2}}[\Gamma]$ to decide the question, the required time would be $O(|\Gamma|^2 \cdot |\hat{b}|^2)$.

12 Concluding Remarks

Regular expressions and transducers over pairing specs allow us to describe many independence properties in a simple, alphabet invariant, way and such that these alphabet invariant objects can be processed as efficiently as their ordinary (alphabet dependent) counterparts. This is possible due to the efficiency of basic algorithms on these objects presented here. A direction for further research is to investigate how algorithms not considered here can be extended to regular expressions and transducers over pairing specs; for example, algorithms involving transducers that realize synchronous relations.

Algorithms on *deterministic* machines with set specs might not work as efficiently as their alphabet dependent counterparts. For example the question of whether $w \in \mathcal{L}(\hat{b})$, for given word w and DFA $\hat{b}$ with set specs, is probably not decidable efficiently within time $O(|w|)$. Despite this, it might be of interest to investigate this question further.

Label sets can have any format as long as one provides their behaviour. For example, a label can be a string representation of a FAdo automaton, [13], whose behaviour of course is a regular language. At this broad level, we were able to generalize a few results like the product construction and the partial derivative automaton. A research direction is to investigate whether more results can be obtained at this level, or even for label sets satisfying some constraint. For example, whether membership, or other decision problems, can be decided using partial derivatives for regular expressions involving labels other than set and pairing specs[2].

References

1. Allauzen, C., Mohri, M.: Efficient algorithms for testing the twins property. J. Autom. Lang. Comb. **8**(2), 117–144 (2003)
2. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2), 291–319 (1996)
3. Bastos, R., Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average complexity of partial derivative automata for semi-extended expressions. J. Autom. Lang. Comb. **22**(1–3), 5–28 (2017)

[2] While we have not obtained in this work the partial derivatives corresponding to a regular expression involving pairing specs, it is our immediate plan to do so—see [16].

4. Béal, M.-P., Carton, O., Prieur, C., Sakarovitch, J.: Squaring transducers: an efficient procedure for deciding functionality and sequentiality. Theor. Comput. Sci. **292**(1), 45–63 (2003)
5. Broda, S., Machiavelo, A., Moreira, N., Reis, R.: On the average state complexity of partial derivative automata: an analytic combinatorics approach. Int. J. Found. Comput. Sci. **22**(7), 1593–1606 (2011). MR2865339
6. Brzozowski, J.A., McCluskey, E.J.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Trans. Electron. Comput. **12**, 67–76 (1963)
7. Brzozowski, J.: Derivatives of regular expressions. J. Assoc. Comput. Mach. **11**, 481–494 (1964)
8. Caron, P., Champarnaud, J.-M., Mignot, L.: Partial derivatives of an extended regular expression. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 179–191. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21254-3_13
9. Champarnaud, J.M., Ziadi, D.: From Mirkin's prebases to Antimirov's word partial derivatives. Fundam. Inf. **45**(3), 195–205 (2001)
10. Champarnaud, J.M., Ziadi, D.: Canonical derivatives, partial derivatives and finite automaton constructions. Theor. Comput. Sci. **289**, 137–163 (2002)
11. Demaille, A.: Derived-term automata of multitape rational expressions. In: Han, Y.-S., Salomaa, K. (eds.) CIAA 2016. LNCS, vol. 9705, pp. 51–63. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40946-7_5
12. Demaille, A., Duret-Lutz, A., Lombardy, S., Saiu, L., Sakarovitch, J.: A type system for weighted automata and rational expressions. In: Holzer, M., Kutrib, M. (eds.) CIAA 2014. LNCS, vol. 8587, pp. 162–175. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08846-4_12
13. FAdo: Tools for formal languages manipulation. http://fado.dcc.fc.up.pt/. Accessed Apr 2018
14. Konstantinidis, S.: Transducers and the properties of error-detection, error-correction and finite-delay decodability. J. Univ. Comput. Sci. **8**, 278–291 (2002)
15. Konstantinidis, S.: Applications of transducers in independent languages, word distances, codes. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 45–62. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60252-3_4
16. Konstantinidis, S. Moreira, N., Reis, R., Young, J.: Regular expressions and transducers over alphabet-invariant and user-defined labels. arXiv.org, arXiv:1805.01829 (2018)
17. Lombardy, S., Sakarovitch, J.: Derivatives of rational expressions with multiplicity. Theor. Comput. Sci. **332**(1–3), 141–177 (2005)
18. Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. Eng. Cybern. **5**, 51–57 (1966)
19. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, Berlin (2009)
20. Sakarovitch, J.: Automata and rational expressions. arXiv.org, arXiv:1502.03573 (2015)
21. Shyr, H.J., Thierrin, G.: Codes and binary relations. In: Malliavin, M.P. (ed.) Séminaire d'Algèbre Paul Dubreil Paris 1975–1976 (29ème Année). LNM, vol. 586, pp. 180–188. Springer, Heidelberg (1977). https://doi.org/10.1007/BFb0087133
22. Thompson, K.: Regular expression search algorithm. Commun. ACM (CACM) **11**, 419–422 (1968)

23. Veanes, M.: Applications of symbolic finite automata. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 16–23. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39274-0_3
24. Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjorner, N.: Symbolic finite state transducers: algorithms and applications. In: Field, J., Hicks, M. (eds.) Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, pp. 137–150 (2012)
25. Sheng, Y.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_2