

Regular Expressions and Transducers over Alphabet-invariant and User-defined Labels*

Stavros Konstantinidis¹, Nelma Moreira², Rogerio Reis², and Joshua Young¹

¹ Saint Mary's University, Halifax, Nova Scotia, 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 give a more broad definition of 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 a few classic algorithmic constructions on ordinary regular expressions and transducers at the broad level of labelled graphs and in such a way that the computational efficiency of the extended constructions is not sacrificed. For regular expressions with set specs we obtain the corresponding partial derivative automata. 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 independent 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, assume that the alphabet is

$$\Gamma = \{0, 1, \dots, n-1\}$$

and consider the 2D regular expression

$$(0/0 + \dots + (n-1)/(n-1))^* (0/e + \dots + (n-1)/e)^*,$$

* Research supported by NSERC (Canada) and by FCT project UID/MAT/00144/2013 (Portugal).

where e is the symbol for the empty string. This 2D regular expression has $O(n)$ symbols and describes the prefix relation, that is, 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 $\hat{t}_{\text{sub}2}$ in Fig. 2. In that figure, the label $(\forall/=)$ represents the set $\{(a, a) \mid a \in \Gamma\}$ and the label (\forall/\neq) represents the set $\{(a, a') \mid a, a' \in \Gamma, a \neq a'\}$ (these labels are called pairing specs). 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

$$(\forall/=)^*(\forall/e)^*.$$

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.

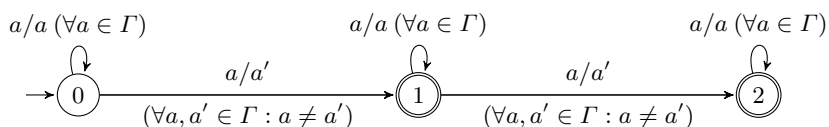


Fig. 1: The transducer realizes the relation of all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2.

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 special labels, called *label set*, where each special 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*). A labelled graph in

this work can possibly be considered to be a compact version of an automaton³ over the monoid M in the sense of [20].

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 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 efficiently on the type B objects.

We also mention the framework of symbolic automata and transducers of [24,23]. 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 [24,23] 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 role of a label set is similar to that of an alphabet, or of the set of regular expressions: it enables users to represent sets of interest. While some of our results apply to regular expressions and labelled graphs over any user-defined label set, the particular case where the label set is the set of pairing specs allows us to rewrite ordinary transducers, like the one in Fig. 1, in a simpler form such that algorithms can work directly on these simpler transducers. In particular, we can employ simple transducers like the one in Fig. 2 to answer the satisfaction question in the theory of independent formal languages. While it seems that pairing specs work well with nondeterministic automata and transducers, this might not be true when dealing with deterministic ones. We discuss this issue further in the last section of the paper.

The paper is organized as follows. The next section makes some assumptions about *alphabets* Γ of non-fixed size. These assumptions are needed in algorithms that process regular expressions and automata with labels involving Γ -symbols. Section 3 defines the set of *set specs*, a particular kind of a label set in which each element represents a subset of Γ or the empty string, and presents basic algorithms on set specs. Section 4 defines the set of *pairing specs*, a particular kind of a label set that is used for transducer-type labelled graphs. Some of these pairing specs are *alphabet invariant*. Section 5 discusses the general concept of a *label set*, with set specs and pairing specs being two specific examples of label sets. Each label set B has a behaviour \mathcal{I} and refers to a monoid, denoted by $\text{mon } B$; that is, $\mathcal{I}(\beta)$ is a subset of $\text{mon } B$ for any label $\beta \in B$. Section 6 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 7 establishes that the rational operations of union, catenation and Kleene star on ordinary automata and transducers work without complications on any labelled graphs.

³ While the labels of the automaton can possibly represent sets in compressed format, we have no intention to define any specific compression method, as the syntax of the labels is left to the application.

Section 8 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 (see Theorem 1 and Corollary 1). Section 9 considers the concept of linear form of a regular expression over a label set, which leads to the definition of its corresponding partial derivative graph. Then, for regular expressions over set specs it presents the development of the corresponding finite automaton that is equivalent to the regular expression (see Theorem 3). Section 10 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 $\text{mon } B$ and $\text{mon } B'$, returning an element of the monoid $\text{mon } C$. In this case, if \odot is known to work on automata/transducers over $\text{mon } B, \text{mon } B'$ then it would also work on type B, B' graphs (see Theorem 4). Section 11 presents some basic algorithms on automata with set specs and transducers with set specs. Section 12 defines the composition of two transducers with set specs such that the complexity of this operation is consistent with the case of ordinary transducers (see Theorem 5). Section 13 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 (see Theorem 6 and Theorem 7). Section 14 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 (see Corollary 4 and Example 14). 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 $0, 1, \dots, 9$, which are ordered as usual, as well as the *special symbols*

$$\forall \exists \nexists = \neq / e \oplus \circ$$

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

$$\text{wo}(S) = \text{the sorted word consisting of the symbols in } S.$$

For example, if $S = \{0, 1, 2\}$, then $\text{wo}(S) = 012$ and $\text{wo}(\{2, 0\}) = 02$.

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 $\text{alph } w$ denotes the set $\{g \in \Omega : |w|_g > 0\}$, that is, the set of symbols that occur in w . For example,

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

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, $|\Gamma| \rightarrow \infty$; 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., [2]), where it is assumed implicitly that the cost of comparing two Γ -symbols is $O(1)$. A similar assumption is made in graph algorithms where the size of a graph (V, E) is $|V| + |E| \rightarrow \infty$, but the size of each vertex is implicitly considered to be $O(1)$, [17]. We note that there are proposals to represent the elements of Γ using non-constant size objects—for instance, [1] represents each Γ -symbol as a binary word of length $O(\log |\Gamma|)$.

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 $O(|\Gamma| \log |\Gamma|)$. Then, the minimum symbol of Γ is simply $\text{ARR}_\Gamma[0]$.

Moreover, we have available an *algorithm* $\text{notIn}(w)$, which returns a symbol in Γ that is not in $\text{alph } w$, where w is a *sorted word* in Γ^* with $0 < |w| < |\Gamma|$. Next we explain that the desired algorithm

$$\text{notIn}(w) \text{ can be made to work in time } O(|w|) \tag{1}$$

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

3 Set 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. We also present some basic algorithms on set specs, which are needed for processing those regular expressions and labelled graphs.

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

$$e \quad \forall \quad \exists w \quad \not\exists w$$

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 $F, \exists u, \not\exists u, \exists v, \not\exists v$ be any set specs with $F \neq e$. We define the partial operation $\cap : \text{SSP} \times \text{SSP} \dashrightarrow \text{SSP}$ as follows.

$$\begin{aligned} e \cap e &= e, & e \cap F &= F \cap e = \perp \\ \forall \cap F &= F \cap \forall = F \\ \exists u \cap \exists v &= \exists \text{ wo } (\text{alph } u \cap \text{alph } v), & \text{ if } (\text{alph } u \cap \text{alph } v) &\neq \emptyset \\ \exists u \cap \exists v &= \perp, & \text{ if } (\text{alph } u \cap \text{alph } v) &= \emptyset \\ \not\exists u \cap \not\exists v &= \not\exists \text{ wo } (\text{alph } u \cup \text{alph } v) \\ \exists u \cap \not\exists v &= \exists \text{ wo } (\text{alph } u \setminus \text{alph } v), & \text{ if } (\text{alph } u \setminus \text{alph } v) &\neq \emptyset \\ \exists u \cap \not\exists v &= \perp, & \text{ if } (\text{alph } u \setminus \text{alph } v) &= \emptyset \\ \not\exists u \cap \exists v &= \exists v \cap \not\exists u \end{aligned}$$

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,

$$\exists 035 \cap \exists 1358 = \exists 35, \quad \not\exists 035 \cap \exists 1358 = \exists 18, \quad \not\exists 035 \cap \not\exists 1358 = \not\exists 01358.$$

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

Proof. The required algorithm works as follows. If either of G, F is e then $G \cap F$ is computed according to Def. 1. Else, if either of G, F is \forall , return the other one. Now suppose that $G = \exists u$ and $F = \exists v$. As u, v are sorted, the sorted word w consisting of their common symbols is computed by using two indices i and j , initially pointing to the first symbols of u and v , and then advancing them as follows: if $u[i] = v[j]$ then output $u[i]$ and increment both i, j by 1; if $u[i] < v[j]$ then increment only i ; else increment only j . So the output would be $\exists w$, if $|w| > 0$, or \perp if $|w| = 0$. In either case, each symbol of u and v is not accessed more than once, so the process works in time $O(|u| + |v|)$. Now suppose that $G = \not\exists u$ and $F = \not\exists v$. Then one can use a process similar to the above to compute the sorted word w consisting of the union of the symbols in u, v . So the output would be $\not\exists w$. Now suppose that $G = \exists u$ and $F = \not\exists v$. Again the process to compute the sorted word w consisting of the symbols in u that are not in v involves two indices i and j , initially pointing to the first symbols of u and v , and then advancing them as follows: if $u[i] = v[j]$ then increment both i, j by 1; if $u[i] < v[j]$ then output $u[i]$ and increment only i ; else increment only j . So the output would be $\exists w$, if $|w| > 0$, or \perp if $|w| = 0$. The last case about G, F is symmetric to the last one.

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

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

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

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

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

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

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 $\exists \text{wo}(\Gamma)$.

Lemma 2. Let Γ be an alphabet of reference and let G, F be set specs respecting Γ . The following statements hold true.

1. $\mathcal{L}(F) \neq \emptyset$; and $\mathcal{L}(F) = \Gamma$ if and only if $F = \forall$.
2. $\mathcal{L}(G \cap F) = \mathcal{L}(G) \cap \mathcal{L}(F)$, if $G \cap F \neq \perp$.
3. If $F = \exists w$ or $F = \nexists w$ then $|\mathcal{L}(F)| \leq |\Gamma| - 1$.
4. $|F| \leq |\Gamma|$.

Proof. The first statement follows from the above definition and the following: If $F = \exists w$ then $(\text{alph } w) \notin \{\emptyset, \Gamma\}$, as $0 < |w| < |\Gamma|$; and if $F = \nexists w$ then again $(\Gamma \setminus \text{alph } w) \notin \{\emptyset, \Gamma\}$. For the second statement, we consider the definition of the operation ‘ \cap ’ as well as the above definition. Clearly the statement holds, if $G = F = e$, or if one of G, F is \forall and the other one is not e . Then, one considers the six cases of Definition 1 where G, F contain \exists or \nexists . For example, if $G = \exists u$ and $F = \nexists v$, we have that $\mathcal{L}(F) = \Gamma \setminus \text{alph } v$, so $\mathcal{L}(G) \cap \mathcal{L}(F) = \text{alph } u \setminus \text{alph } v$, which is equal to $\mathcal{L}(G \cap F)$. The other cases can be shown analogously. The third and fourth statements follow from the restriction $0 < |w| < |\Gamma|$ in Definition 2.

The next lemma concerns simple algorithmic questions about set specs that are needed as basic tools in other algorithms further below.

Lemma 3. Let Γ be an alphabet of reference and let $F \neq e$ 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 $O(|F|)$.

Proof. For the first statement, we note that the condition to test is equivalent to one of “ $F = \forall$ ”, “ $F = \exists w$ and $|w|_g > 0$ ”, “ $F = \exists w$ and $|w|_g = 0$ ”; and that one can use binary search to test whether g occurs in w . For the second statement, we note that the condition to test is equivalent to one of “ $F = \exists g$ ”, “ $F = \exists w$ and $|w| = |I| - 1$ and $|w|_g = 0$ ”. For the third statement, we note that the condition to test is equivalent to one of $|I| \geq k$, $|w| \geq k$, $|I| - |w| \geq k$, depending on whether F is one of $\forall, \exists w, \exists w$. The last case requires time $O(|F|)$ to compute $|w|$ and then $O(\log |I| + \log |w|)$ time for arithmetic operations, which is $O(\log |I|)$ as $|w| < |I|$. For the fourth statement, we note that $|\mathcal{L}(F)| = 1$ is equivalent to whether “ $F = \exists g$ and $|g| = 1$ ” or “ $F = \exists w$ and $|w| = |I| - 1$ ”. In the former case, the algorithm returns g . In the latter case, we use the algorithm $\text{notIn}(w)$ to get the desired symbol in $I \setminus \text{alph } w$. The latter case is the worse of the two, and works in time $O(|F| + \log |I|)$ to compute $|w|$ and test whether $|w| = |I| - 1$, plus time $O(|F|)$ to execute $\text{notIn}(w)$ (see the bound in (1)). The total time is $O(|F|)$, as $|F| = |I|$. For the fifth statement, if $F = \forall$ or $F = \exists w$ the algorithm simply returns $\text{ARR}_I[0]$ or $w[0]$, respectively. The worst case is when $F = \exists w$, where, as before, the algorithm uses $\text{notIn}(w)$ requiring time $(|F|)$. For the sixth statement, the algorithm first finds any $g_1 \in \mathcal{L}(F)$, then computes the set $\text{spec } B = F \cap \exists g_1$ and then computes any $g_2 \in \mathcal{L}(B)$.

4 Pairing Specifications

Here we define expressions for describing certain finite relations that are subsets of $(I \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\})$. First, we define their syntax and then their semantics.

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

$$e/e \quad e/G \quad F/e \quad F/G \quad F/= \quad F/G \neq \quad (2)$$

where F, G are set specs with $F, G \neq e$. The set of pairing specs is denoted by PSP. The inverse \mathbf{p}^{-1} of a pairing spec \mathbf{p} is defined as follows depending on the possible forms of \mathbf{p} displayed in (2):

$$\begin{aligned} (e/e)^{-1} &= (e/e), & (e/G)^{-1} &= (G/e), & (F/e)^{-1} &= (e/F), \\ (F/G)^{-1} &= (G/F), & (F/=)^{-1} &= (F/=), & (F/G \neq)^{-1} &= (G/F \neq) \end{aligned}$$

Example 2. As a pairing spec \mathbf{p} is a string, it has a length $|\mathbf{p}|$. We have that $|\forall/e| = 3$ and $|\exists u/\exists v| = 3 + |u| + |v|$. Also, $(\forall/e)^{-1} = (e/\forall)$ 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 $\exists w, \exists w$. The set of alphabet invariant pairing specs is denoted by $\text{PSP}^{\text{invar}}$.

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

$$\text{PSP}[\Gamma] = \{\mathbf{p} \in \text{PSP} : \mathbf{p} \text{ respects } \Gamma\}.$$

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

$$\begin{aligned} \mathcal{R}(\mathbf{e}/\mathbf{e}) &= \{(\varepsilon, \varepsilon)\}; \\ \mathcal{R}(\mathbf{e}/G) &= \{(\varepsilon, y) \mid y \in \mathcal{L}(G)\}; \\ \mathcal{R}(F/\mathbf{e}) &= \{(x, \varepsilon) \mid x \in \mathcal{L}(F)\}; \\ \mathcal{R}(F/G) &= \{(x, y) \mid x \in \mathcal{L}(F), y \in \mathcal{L}(G)\}; \\ \mathcal{R}(F/=) &= \{(x, x) \mid x \in \mathcal{L}(F)\}; \\ \mathcal{R}(F/G\neq) &= \{(x, y) \mid x \in \mathcal{L}(F), y \in \mathcal{L}(G), x \neq y\}. \end{aligned}$$

Remark 2. All the alphabet invariant pairing specs are

$$\mathbf{e}/\mathbf{e} \quad \mathbf{e}/\forall \quad \forall/\mathbf{e} \quad \forall/\forall \quad \forall/= \quad \forall/\forall\neq$$

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

Lemma 4. Let $\mathbf{p} \in \text{PSP}[\Gamma]$. The following statements hold true.

1. $\mathcal{R}(\mathbf{p}) = \emptyset$ if and only if \mathbf{p} is of the form $F/G\neq$ and $\mathcal{L}(F) = \mathcal{L}(G) = \{g\}$ for some $g \in \Gamma$.
2. $\mathcal{R}(\mathbf{p}^{-1}) = \mathcal{R}(\mathbf{p})^{-1}$.
3. \mathbf{p}^{-1} can be computed from \mathbf{p} in time $O(|\mathbf{p}|)$.

Proof. The first statement follows from Definition 5 when we note that the set $\{(x, y) \mid x \in \mathcal{L}(F), y \in \mathcal{L}(G), x \neq y\}$ is empty if and only if $\mathcal{L}(F) = \mathcal{L}(G) = \{g\}$, for some $g \in \Gamma$. The last two statements follow from Definitions 3 and 5.

Some notation on pairing specs. Let \mathbf{p} be a pairing spec. Then the *left* part, $\text{left } \mathbf{p}$, of \mathbf{p} is the string on the left of the symbol ‘/’, and the *right* part, $\text{right } \mathbf{p}$, of \mathbf{p} is the string on the right of ‘/’. We have the following examples:

$$\text{left}(\exists w/\forall\neq) = \exists w \quad \text{right}(\exists w/\forall\neq) = \forall\neq \quad \text{left}(\forall/=) = \forall \quad \text{right}(\forall/=) = =$$

While the expression $\mathcal{L}(\text{left } \mathbf{p})$ makes sense when \mathbf{p} respects the alphabet of reference, this is not the case for $\mathcal{L}(\text{right } \mathbf{p})$. So we define $\text{rset } \mathbf{p}$ to be as follows, depending on the structure of \mathbf{p} according to (2)

$$\begin{aligned} \text{rset}(\mathbf{e}/\mathbf{e}) &= \mathbf{e}, & \text{rset}(\mathbf{e}/G) &= G, & \text{rset}(F/\mathbf{e}) &= \mathbf{e}, \\ \text{rset}(F/G) &= G, & \text{rset}(F/=) &= F, & \text{rset}(F/G\neq) &= G. \end{aligned}$$

The above notation implies

$$\mathcal{R}(\mathbf{p}) \subseteq \mathcal{L}(\text{left } \mathbf{p}) \times \mathcal{L}(\text{rset } \mathbf{p}). \quad (3)$$

Lemma 5. If $\mathbf{p} \in \text{PSP}[\Gamma]$ then $|\mathbf{p}| \leq 2|\Gamma| + 2$.

Proof. Follows from Lemma 2.

5 Label Sets and their Behaviours

We are interested in automata-type objects (labelled graphs) \hat{g} in which every transition label β represents a set $\mathcal{I}(\beta)$ of elements in some monoid M . The subsets $\mathcal{I}(\beta) \subseteq M$ are the behaviours of the labels and they are used to define the behaviour of \hat{g} as a subset of M . We focus on sets of labels in this section—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 *canonical* (string) representation \underline{m} . Then, we write

$$\underline{M} = \{\underline{m} \mid m \in M\}.$$

In the example below, we provide sample canonical representations for the two monoids of interest to this work.

Example 3. We shall consider two standard monoids.

1. 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 e : $\underline{\varepsilon} = e$.
2. 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, $(\varepsilon, \varepsilon) = e/e$.

A *label set* B is a nonempty set of nonempty strings (over Ω). A *label behaviour* is a mapping

$$\mathcal{I} : B \rightarrow 2^M,$$

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 $\text{mon } B$ 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 $\text{mon } B_1 = \text{mon } B_2$ 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 $\text{mon } B_1 = \text{mon } B_2$. This convention is applied in the example below: we use \mathcal{L} for the behaviour of both the label sets Σ_e and $\text{SSP}[T]$.

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

1. $\Sigma_e = \Sigma \cup \{e\}$ with behaviour $\mathcal{L} : \Sigma_e \rightarrow 2^{\Sigma^*}$ such that $\mathcal{L}(g) = \{g\}$, if $g \in \Sigma$, and $\mathcal{L}(e) = \{\varepsilon\}$. Thus, $\text{mon } \Sigma = \Sigma^*$.
2. Σ with behaviour $\mathcal{L} : \Sigma \rightarrow 2^{\Sigma^*}$ such that $\mathcal{L}(g) = \{g\}$, for $g \in \Sigma$. Thus, $\text{mon } \Sigma = \Sigma^*$.
3. $\text{SSP}[I]$ with behaviour $\mathcal{L} : \text{SSP}[I] \rightarrow 2^{I^*}$, as specified in Def. 2. Thus, $\text{mon } \text{SSP}[I] = I^*$.
4. $\text{REG } \Sigma = \text{REG } \Sigma_e =$ all regular expressions over Σ with behaviour $\mathcal{L} : \text{REG } \Sigma \rightarrow 2^{\Sigma^*}$ such that $\mathcal{L}(r)$ is the language of the regular expression r . Thus, $\text{mon}(\text{REG } \Sigma) = \Sigma^*$.
5. $[\Sigma_e, \Delta_e] = \{x/y \mid x \in \Sigma_e, y \in \Delta_e\}$ with behaviour

$$\mathcal{R} : [\Sigma_e, \Delta_e] \rightarrow 2^{\Sigma^* \times \Delta^*}$$

such that $\mathcal{R}(e/e) = \{(\varepsilon, \varepsilon)\}$, $\mathcal{R}(x/e) = \{(x, \varepsilon)\}$, $\mathcal{R}(e/y) = \{(\varepsilon, y)\}$, $\mathcal{R}(x/y) = \{(x, y)\}$, for any $x \in \Sigma$ and $y \in \Delta$. Thus, $\text{mon}[\Sigma_e, \Delta_e] = \Sigma^* \times \Delta^*$.

6. $\text{PSP}[I]$ with behaviour $\mathcal{R} : \text{PSP}[I] \rightarrow 2^{I^* \times I^*}$ as specified in Def. 5. Thus, $\text{mon } \text{PSP}[I] = I^* \times I^*$.
7. $\text{PSP}^{\text{invar}}$ with behaviour $\mathcal{R}_\perp : \text{PSP}^{\text{invar}} \rightarrow \{\emptyset\}$. Thus, $\mathcal{I}(\beta) = \emptyset$, for any $\beta \in \text{PSP}^{\text{invar}}$.
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

$$\mathcal{I}(\beta_1/\beta_2) = \mathcal{I}_1(\beta_1) \times \mathcal{I}_2(\beta_2) \quad \text{and} \quad \text{mon}[B_1, B_2] = \text{mon } B_1 \times \text{mon } B_2.$$

9. $[\text{REG } \Sigma, \text{REG } \Delta]$ 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 $\mathbf{r} \in \text{REG } \Sigma$ and $\mathbf{s} \in \text{REG } \Delta$.

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 $\text{mon } \text{PSP}[I] = \text{mon } \underline{I^* \times I^*} = I^* \times I^*$ 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 $\text{mon } X$.

6 Labelled Graphs, Automata, Transducers

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

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

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 quantity

$$|\hat{g}| = |Q| + \|\delta\|.$$

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 $I \cap F$.

Computing the trim part of \hat{g} means removing the non-useful states and keeping only one isolated state in $I \cap F$ (if such states exist), and can be computed in linear time $O(|\hat{g}|)$.

Lemma 6. *Let $\hat{g} = (Q, B, \delta, I, F)$ be a trim labelled graph. We have that*

$$|Q| \leq 2\|\delta\| + 1.$$

Proof. Q can be partitioned into three sets: Q_1 = the set of states having an outgoing edge but no incoming edge; Q_2 = the set of states having an incoming edge; and possibly a single isolated state in $I \cap F$. The claim follows from the fact that $|Q_1|, |Q_2| \leq \|\delta\|$.

Definition 6. *Let $\hat{g} = (Q, B, \delta, I, F)$ 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* $\text{exp } \hat{g}$ of \hat{g} is the labelled graph $(Q, \underline{\text{mon } B}, \delta_{\text{exp}}, I, F)$ such that

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

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}^ε .

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 REG Σ .

Lemma 7. *For each type B graph $\hat{g} = (Q, B, \delta, I, F)$, we have that*

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

Proof. Let $m \in \mathcal{I}(\exp \hat{g})$. Then there is an accepting path $\langle q_{i-1}, \underline{m}_i, q_i \rangle_{i=1}^\ell$ of $\exp \hat{g}$ such that $m \in \mathcal{I}(\underline{m}_1) \cdots \mathcal{I}(\underline{m}_\ell) = \{m_1\} \cdots \{m_\ell\}$; hence, $m = m_1 \cdots m_\ell$. By definition of δ_{exp} , for each $i = 1, \dots, \ell$, there is $(q_{i-1}, \beta_i, q_i) \in \delta$ such that $m_i \in \mathcal{I}(\beta_i)$, so $\langle q_{i-1}, \beta_i, q_i \rangle_{i=1}^\ell$ is an accepting path of \hat{g} . Then, $\mathcal{I}(\beta_1) \cdots \mathcal{I}(\beta_\ell) \subseteq \mathcal{I}(\hat{g})$, and $m \in \mathcal{I}(\hat{g})$. Conversely, for any $m \in \mathcal{I}(\hat{g})$, one uses similar arguments to show that $m \in \mathcal{I}(\exp \hat{g})$. Thus, $\mathcal{I}(\hat{g}) = \mathcal{I}(\exp \hat{g})$.

To show that $\mathcal{I}(\hat{g}) = \mathcal{I}(\hat{g}^\varepsilon)$, let δ^ε be the set of transitions in \hat{g}^ε . As $\delta \subseteq \delta^\varepsilon$, we have $\mathcal{I}(\hat{g}) \subseteq \mathcal{I}(\hat{g}^\varepsilon)$. For the converse, the main idea is that, if any accepting path of \hat{g}^ε contains transitions $(q_{i-1}, \varepsilon_{\text{mon } B}, q_i)$ with $q_{i-1} = q_i$, then these transitions can be omitted resulting into an accepting path of \hat{g} .

Remark 5. As stated before, our focus is on two kinds of monoids: Σ^* and $\Sigma^* \times \Delta^*$. Recall that, in those monoids, the neutral elements ε and $(\varepsilon, \varepsilon)$ have canonical representations e and e/e , which are of fixed length. Thus, we shall assume that $\varepsilon_{\text{mon } B} = O(1)$, for any label set B . This implies that

$$|\hat{g}^\varepsilon| = O(|\hat{g}|).$$

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

Remark 6. 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 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 $\hat{t}(L) = \cup_{w \in L} \hat{t}(w)$.

Remark 7. The size of an alphabet invariant transducer \hat{i} is of the same order of magnitude as $|Q| + |\delta|$.

Lemma 8. *If \hat{b} is an automaton with set specs then $\text{exp } \hat{b}$ is an ε -NFA. If \hat{s} is a transducer with set specs then $\text{exp } \hat{s}$ 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 $\Phi(\text{exp } \hat{v})$. For example, “ \hat{s} is an input-altering transducer” means that “ $\text{exp } \hat{s}$ 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 shown in Fig. 2 are alphabet invariant. Both transducers are much more succinct compared to their expanded Γ -versions, as $|\Gamma| \rightarrow \infty$:

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

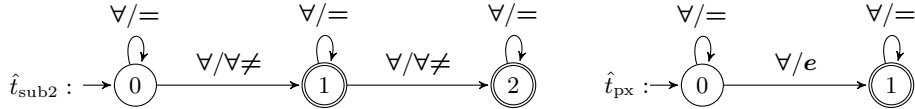


Fig. 2: The left transducer realizes the relation of all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2. The right transducer realizes the relation of all (u, v) such that v is a proper prefix of u .

Example 6. If expanded, the automaton with set specs in Fig. 3, will have $3n + 1$ transitions, as opposed to the current 7 ones.

Following [26], if $\hat{t} = (Q, [\Sigma_e, \Delta_e], \delta, I, F)$ is a transducer then \hat{t}^{-1} is the transducer $(Q, [\Delta_e, \Sigma_e], \delta', I, F)$, where $\delta' = \{(p, y/x, q) \mid (p, x/y, q) \in \delta\}$, such that

$$\mathcal{R}(\hat{t}^{-1}) = \mathcal{R}(\hat{t})^{-1}. \quad (4)$$

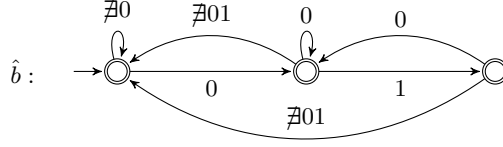


Fig. 3: The automaton accepts all strings over $\Gamma = \{0, \dots, n\}$ that do not contain 011.

Lemma 9. For each transducer \hat{s} with set specs we have that

$$\exp(\hat{s}^{-1}) = (\exp \hat{s})^{-1} \quad \text{and} \quad \mathcal{R}(\hat{s}^{-1}) = \mathcal{R}(\hat{s})^{-1}.$$

Proof. The first identity follows from two facts: (i) $\exp(\hat{s}^{-1})$ has transitions of the form $(p, y/x, q)$, where $y/x \in \mathcal{R}(\mathbf{p}^{-1})$ and (p, \mathbf{p}^{-1}, q) is a transition in \hat{s}^{-1} ; and (ii) $(\exp \hat{s})^{-1}$ has transitions of the form $(p, y/x, q)$, where $x/y \in \mathcal{R}(\mathbf{p})$ and (p, \mathbf{p}, q) is a transition in \hat{s} . The second identity follows from (4) and Definition 6.

Remark 8. Let $\hat{t} = (Q, \Gamma, \delta, I, F)$ be a transducer with set specs. By Lemma 5, we have that

$$\|\delta\| \leq (2|\Gamma| + 3)|\delta|.$$

7 Rational Operations

The three standard *rational operations* (union, catenation, star) on automata and transducers can be defined on labelled graphs with appropriate constraints on the monoids involved. Let $\hat{g} = (Q, B, \delta, I, F)$ and $\hat{g}' = (Q', B', \delta', I', F')$ be labelled graphs such that

$$\text{mon } B = \text{mon } B' \quad \text{and} \quad Q \cap Q' = \emptyset.$$

The graph $\hat{g} \cup \hat{g}'$ of type $C = B \cup B' \cup \{\underline{\varepsilon_{\text{mon } B}}\}$ is defined as follows

$$\hat{g} \cup \hat{g}' = (Q \cup Q' \cup \{s\}, C, \delta \cup \delta' \cup E, \{s\}, F \cup F'),$$

where s is a (new) state not in $Q \cup Q'$ and E is the set of transitions $(s, \underline{\varepsilon_{\text{mon } B}}, p)$, for all $p \in I \cup I'$.

The graph $\hat{g} \cdot \hat{g}'$ of type $C = B \cup B' \cup \{\underline{\varepsilon_{\text{mon } B}}\}$ is defined as follows

$$\hat{g} \cdot \hat{g}' = (Q \cup \{q\} \cup Q', C, \delta \cup \delta' \cup E \cup E', I, F'),$$

where q is a (new) state not in $Q \cup Q'$, E is the set of transitions $(f, \underline{\varepsilon_{\text{mon } B}}, q)$, for all $f \in F$, and E' is the set of transitions $(q, \underline{\varepsilon_{\text{mon } B}}, i')$, for all $i' \in I'$.

The graph \hat{g}^* of type $D = B \cup \{\underline{\varepsilon_{\text{mon } B}}\}$ is defined as follows

$$\hat{g}^* = (Q \cup \{s\}, D, \delta \cup E_1 \cup E_2, \{s\}, F \cup \{s\}),$$

where s is a (new) state not in $Q \cup Q'$, E_1 is the set of transitions $(s, \underline{\varepsilon_{\text{mon } B}}, i)$ for all $i \in I$, and E_2 is the set of transitions $(f, \underline{\varepsilon_{\text{mon } B}}, s)$ for all $f \in F$.

Lemma 10. *Let $\hat{g} = (Q, B, \delta, I, F)$ and $\hat{g}' = (Q', B', \delta', I', F')$ be trim labelled graphs such that $\text{mon } B = \text{mon } B'$.*

1. $\mathcal{I}(\hat{g} \cup \hat{g}') = \mathcal{I}(\hat{g}) \cup \mathcal{I}(\hat{g}')$ and $|\hat{g} \cup \hat{g}'| = O(|\hat{g}| + |\hat{g}'|)$.
2. $\mathcal{I}(\hat{g} \cdot \hat{g}') = \mathcal{I}(\hat{g})\mathcal{I}(\hat{g}')$ and $|\hat{g} \cdot \hat{g}'| = O(|\hat{g}| + |\hat{g}'|)$.
3. $\mathcal{I}(\hat{g}^*) = \mathcal{I}(\hat{g})^*$ and $|\hat{g}^*| = O(|\hat{g}|)$.

In the above lemma, the statements about the sizes of the graphs follow immediately from the definitions of their constructions. For the statements about the behaviours of the constructed graphs, it is sufficient to show the statements about their expansions. For example, for the third statement, one shows that

$$\mathcal{I}(\text{exp } \hat{g}^*) = \mathcal{I}(\text{exp } \hat{g})^*.$$

But then, one works at the level of the monoid $\text{mon } B$ and the proofs are essentially the same as the ones for the case of ε -NFAs (see e.g. [25]).

8 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 \emptyset . The set $\text{REG } B$ of type B regular expressions is the set of strings consisting of the 1-symbol string \emptyset and the strings in the set Z that is defined inductively as follows.*

- $\underline{\varepsilon_{\text{mon } B}}$ is in Z .
- 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.

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

Example 7. Let Σ, Δ be alphabets. Using Σ as a label set, we have that $\text{REG } \Sigma$ is the set of ordinary regular expressions over Σ . For the label set $[\Sigma_e, \Delta_e]$, we have that $\text{REG}[\Sigma_e, \Delta_e]$ is the set of rational expressions over $\Sigma^* \times \Delta^*$ in the sense of [20].

Example 8. Let $\Gamma = \{0, 1, \dots, n-1\}$. In type $\text{SSP}[\Gamma]$ regular expressions, the set specs $\forall, \exists w, \nexists w$ correspond to the following UNIX expressions, respectively: ‘.’, ‘[w]’, ‘[~w]’. So $\mathcal{L}(\forall) = \Gamma$. When the alphabet size n is a parameter rather than fixed, the savings when using expressions over label sets could be of order $O(n)$ or even $O(n^2)$. For example, the expression \forall is of size $O(1)$ but the corresponding (ordinary) regular expression of type Γ_e is $0 + \dots + (n-1)$, which is of size $O(n)$. Similarly, the following regular expression over $\text{PSP}[\Gamma]$

$$(\forall/=)^* (\forall/\forall\neq) (\forall/=)^* \quad (5)$$

is of size $O(1)$. It describes all word pairs (u, v) such that the Hamming distance of u, v is 1. The corresponding (ordinary) regular expression over $\text{ED}[\Gamma]$ is

$$(0/0 + \dots + (n-1)/(n-1))^* (\mathbf{r}_0 + \dots + \mathbf{r}_{n-1}) (0/0 + \dots + (n-1)/(n-1))^*$$

which is of size $O(n^2)$, where each \mathbf{r}_i is the sum of all expressions i/j with $j \neq i$ and $i, j \in \Gamma$.

Example 9. 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 $\text{PSP}[\text{ASCII}]$

$$\left((\nexists u/=) + (\exists u[0]/\exists v[0]) + \dots + (\exists u[\ell-1]/\exists v[\ell-1]) \right)^*$$

where ASCII is the alphabet of standard ASCII characters. Similarly, the command

`tr -d u`

can be ‘simulated’ by the following regular expression of type $\text{PSP}[\text{ASCII}]$

$$(\exists u/e + \nexists u/=)^*$$

For the command

`tr -s u`

it seems that any regular expression over $\text{PSP}[\text{ASCII}]$ cannot be of size $O(\ell^2)$. A related (ordinary) transducer is shown below in Fig. 4.

The Thompson method, [22], of converting an ordinary regular expression over Σ —a type Σ_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 , using Lemma 10.

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*

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

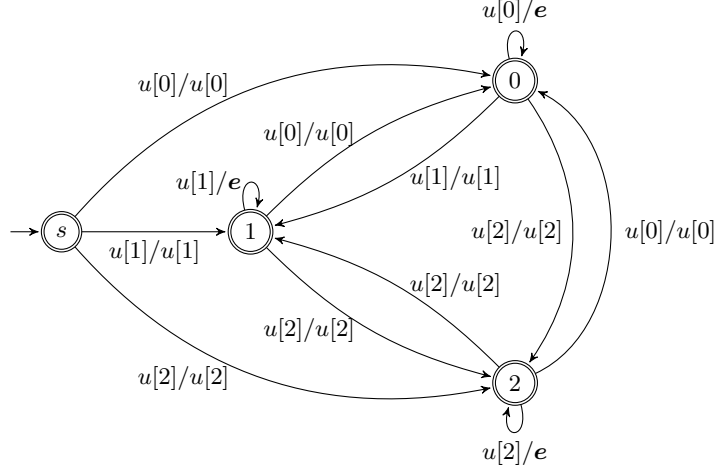


Fig. 4: Transducer realizing the command $\text{tr } -s \ u$ with $|u| = 3$.

For the converse of the above theorem, we shall extend the state elimination method of automata, [7], to labelled graphs.

Let $\hat{g} = (Q, B, \delta, \{s\}, \{f\})$ be a type REG B graph, where B is a label set with some behaviour \mathcal{I} . We say that \hat{g} is *non-returning*, if $s \neq f$, there are no transitions going into s , there are no transitions coming out of f , and there is at most one transition between any two states of \hat{g} . For any states $p, r \in Q$, let $B_{p,r} = \{\beta \mid (p, \beta, r) \in \delta\}$, and let $\mathbf{r}_{p,r} = \beta_1 + \dots + \beta_m$, where the β_i 's are the elements of $B_{p,r}$, if $B_{p,r} \neq \emptyset$. We define next the labelled graph \hat{h} that *results by eliminating* a state $q \in Q \setminus \{s, f\}$ from \hat{g} . It is the type REG B graph

$$\hat{h} = (Q \setminus \{q\}, B, \delta', \{s\}, \{f\}) \quad (6)$$

such that δ' is defined as follows. For any states $p, r \in Q \setminus \{q\}$:

- If $(p, \alpha, r) \in \delta$ then $(p, \alpha, r) \in \delta'$.
- If $(p, \alpha_1, q), (q, \alpha_2, r) \in \delta$ then either $(p, \alpha_1(\mathbf{r}_{q,q}^*)\alpha_2, r) \in \delta'$ if $B_{q,q} \neq \emptyset$, or $(p, \alpha_1\alpha_2, r) \in \delta'$ if $B_{q,q} = \emptyset$.

Lemma 11. *Let $\hat{g} = (Q, B, \delta, \{s\}, \{f\})$ be a non-returning labelled graph, where B is a label set with some behaviour \mathcal{I} . If \hat{h} is the type REG B graph that results by eliminating a state $q \in Q \setminus \{s, f\}$ from \hat{g} then $\mathcal{I}(\hat{h}) = \mathcal{I}(\hat{g})$.*

Proof. The main steps of the proof are analogous to those used in traditional proofs for the case of NFAs [25]. First, let $m \in \mathcal{I}(\hat{g})$. Then, there is an accepting path $P = \langle q_{i-1}, \beta_i, q_i \rangle_{i=1}^{\ell}$ of \hat{g} such that $\ell \geq 1$, $m = m_1 \dots m_{\ell}$ and each $m_i \in \mathcal{I}(\beta_i)$. By a q -block of transitions in P we mean a path $R = \langle q_{j-1}, \beta_j, q_j \rangle_{j=b}^{b+r}$ such that

$r \geq 1$, $q_{b-1} \neq q$, $q_b = \dots = q_{b+r-1} = q$, $q_{b+r} \neq q$.

As \hat{g} has at most one transition between any two states, we have that, if $r \geq 2$, then $\beta_{b+1} = \dots = \beta_{b+r-1} = \mathbf{r}_{q,q}$. Then,

$$e = (q_{b-1}, \beta_b(\mathbf{r}_{q,q})^* \beta_{b+r}, q_{b+r}) \quad \text{or} \quad e = (q_{b-1}, \beta_b \beta_{b+r}, q_{b+r})$$

is a transition in δ' —see (6). Moreover, $m_b \dots m_{b+r} \in \mathcal{I}(\beta_b) \mathcal{I}(\mathbf{r}_{q,q}^*) \mathcal{I}(\beta_{b+r})$. If we replace in P the q -block R with the transition e , and we repeat this with all q -blocks in P , then we get an accepting path of \hat{h} such that $m \in \mathcal{I}(\hat{h})$.

Conversely, let $m \in \mathcal{I}(\hat{h})$. Then there is an accepting path $P' = \langle q_{i-1}, \beta_i, q_i \rangle_{i=1}^\ell$ of \hat{h} such that $\ell \geq 1$, $m = m_1 \dots m_\ell$ and each $m_i \in \mathcal{I}(\beta_i)$. For each $i \in \{1, \dots, \ell\}$, we define a path P_i of \hat{g} as follows. If $(q_{i-1}, \beta_i, q_i) \in \delta$ then $P_i = \langle q_{i-1}, \beta_i, q_i \rangle$. Else, there are $(q_{i-1}, \alpha_1, q), (q, \alpha_2, q_i) \in \delta$ such that

$$\beta_i = \alpha_1(\mathbf{r}_{q,q})^* \alpha_2 \quad \text{or} \quad \beta_i = \alpha_1 \alpha_2.$$

As $m_i \in \mathcal{I}(\beta_i)$ and, if defined, $\mathcal{I}(\mathbf{r}_{q,q})^* = \cup_{r=0}^\infty \mathcal{I}(\mathbf{r}_{q,q})^r$, we have that there is $r \geq 0$ such that

$$m_i = k'_i k_{i,1} \dots k_{i,r} k''_i, \quad k'_i \in \mathcal{I}(\alpha_1), \quad k''_i \in \mathcal{I}(\alpha_2), \quad k_{i,1}, \dots, k_{i,r} \in \mathcal{I}(\mathbf{r}_{q,q}).$$

Then, the path P_i is

$$\langle (q_{i-1}, \alpha_1, q), (q, \mathbf{r}_{q,q}, q), \dots, (q, \mathbf{r}_{q,q}, q), (q, \alpha_2, q_i) \rangle,$$

which has r repetitions of $(q, \mathbf{r}_{q,q}, q)$. Now define the sequence P to be the concatenation of all paths P_i . This sequence is an accepting path of \hat{g} and this implies that $m \in \mathcal{I}(\hat{g})$.

As a type B graph \hat{g} is also a type REG B graph, and as \hat{g} can be modified to be non-returning, we can apply the above lemma repeatedly until we get a type REG B graph \hat{h} with set of states $\{s, f\}$ such that $\mathcal{I}(\hat{g}) = \mathcal{I}(\hat{h})$. Then, we have that $\mathcal{I}(\hat{h}) = \mathcal{I}(\mathbf{r}_{s,f})$. Thus, we have the following consequence of Lemma 11.

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

9 Partial Derivatives of Regular Expressions

Derivatives based methods for the manipulation of regular expressions have been widely studied [8,3,18,16,6,11,9]. 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 [4] or tuples [12]). The partial derivative automaton of a regular expression over Σ^* was introduced independently by Mirkin [18] and Antimirov [3]. Champarnaud and Ziadi [10] proved that the two formulations are equivalent. Lombardy and Sakarovitch [16]

generalised these constructions to weighted regular expressions, and recently Demaille [12] defined derivative automata for multitape weighted regular expressions.

Here we define the partial derivative automaton for regular expressions with set specifications. First however, we start with more general regular expressions over label sets.

Given a finite set S of expressions we define its behaviour as $\mathcal{I}(S) = \bigcup_{s \in S} \mathcal{I}(s)$. 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 label sets* of an expression \mathbf{r} be the set $\mathbb{SS}(\mathbf{r}) = \{ \beta \mid \beta \in B \text{ and } \beta \text{ occurs in } \mathbf{r} \}$. The *size* of an expressions \mathbf{r} is $\|\mathbf{r}\| = |\mathbb{SS}(\mathbf{r})|$; it can be inductively defined as follows:

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

Given a monoid M the (*left*) *quotient* of $P \subseteq M$ by an element $m \in M$ is defined by $m^{-1}P = \{ m' \in M \mid mm' \in P \}$. This can be extended additively to the quotient of P by another subset Q of M , $Q^{-1}P$.

From now on we assume that $\text{mon } B$ is either Σ^* or $\Sigma^* \times \Gamma^*$, and that $\mathcal{I}(\beta)$ is a subset of generators of M , for all $\beta \in B$.

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

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

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

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

Lemma 12. *For all $\mathbf{r} \in \text{REG } B$, $\mathbf{r} \sim c(\mathbf{r}) \cup n(\mathbf{r})$.*

Proof. Proof by induction on \mathbf{r} . □

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

$$\partial_\beta(\mathbf{r}) = \{ \mathbf{s} \mid (\beta, \mathbf{s}) \in \mathbf{n}(\mathbf{r}) \}.$$

It is clear that we can iteratively compute the linear form of an expression $\mathbf{s} \in \partial_\beta(\mathbf{r})$, for $\beta \in \text{SS}(\mathbf{r})$. The set containing \mathbf{r} and of all the resulting expressions is called the set of partial derivatives of $\mathbf{r} \in \text{REG } B$, $\text{PD}(\mathbf{r})$.

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

$$\hat{a}_{\text{PD}}(\mathbf{r}) = \langle \text{PD}(\mathbf{r}), \Sigma, \delta_{\text{PD}}, \mathbf{r}, F \rangle,$$

where $F = \{ \mathbf{r}_1 \in \text{PD}(\mathbf{r}) \mid \mathbf{c}(\mathbf{r}_1) = \underline{\varepsilon_{\text{mon}B}} \}$, and $\delta_{\text{PD}} = \{ (\mathbf{r}_1, \beta, \mathbf{r}_2) \mid \mathbf{r}_1 \in \text{PD}(\mathbf{r}) \wedge \beta \in \text{SS}[\mathbf{r}] \wedge \mathbf{r}_2 \in \partial_\beta(\mathbf{r}_1) \}$.

Whether this graph is finite and whether it is equivalent to \mathbf{r} (has the same behaviour, that is) depends on the behaviour \mathcal{I} of the label set B .

9.1 Regular Expressions with Set Specifications

Here we consider regular expressions of type $\text{SSP}[\Sigma]$ which fixed behaviours are languages over alphabet Σ . Given $L_1, L_2 \subseteq \Sigma^*$ and $x \in \Sigma$, the quotient of a language w.r.t x satisfies the following relations

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

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 $L^{-1}L_1 \subseteq L^{-1}L_2$.

Given two set specifications $F, G \in \text{SSP}[\Sigma] \setminus \{e\}$ 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

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

Note that this definition is coherent with the definition of partial derivatives of an expression w.r.t. a label given before. The set of partial derivatives of \mathbf{r} w.r.t. a word $x \in (B \setminus \{\varepsilon_{\text{mon}B}\})^*$ is inductively defined by $\partial_\varepsilon(\mathbf{r}) = \{\mathbf{r}\}$ and $\partial_{x\beta}(\mathbf{r}) = \partial_\beta(\partial_x(\mathbf{r}))$, where, given a set $S \subseteq \text{REG } B$, $\partial_\beta(S) = \bigcup_{\mathbf{r} \in S} \partial_\beta(\mathbf{r})$. Moreover one has $\mathcal{L}(\partial_x(\mathbf{r})) = \bigcup_{\mathbf{r}_1 \in \partial_x(\mathbf{r})} \mathcal{L}(\mathbf{r}_1)$. The following lemmas ensure that the partial derivative automaton has the same behaviour of the correspondent regular expression. These results generalize known results for ordinary regular expressions [3,18].

Lemma 13. For two set specifications $F, G \in \text{SSP}[\Sigma]$, $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \{\varepsilon\}$ if $F \cap G \neq \perp$, and $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \emptyset$ otherwise.

Proof. Given $L \subseteq \Sigma$ and $x \in \Sigma$ one has $x^{-1}L = \{\varepsilon\}$ if $x \in L$ and $x^{-1}L = \emptyset$, otherwise. Thus, the result follows. \square

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

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

Lemma 14. For all $\mathbf{r} \in \text{REG SSP}[\Sigma]$ and $F \in \text{SSP}[\Sigma]$, $\mathcal{L}(F)^{-1}\mathcal{L}(\mathbf{r}) = \mathcal{L}(\partial_F(\mathbf{r}))$.

Proof. For $\mathbf{r} = \emptyset$ and $\mathbf{r} = \mathbf{e}$ it is obvious. For $\mathbf{r} = G$ the result follows from Lemma 13. In fact, if $\mathcal{L}(F)^{-1}\mathcal{L}(G) = \{\varepsilon\}$ then $\partial_F(G) = \{\mathbf{e}\}$ 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. \square

Lemma 15. For all $g \in (\text{SSP}[\Sigma] \setminus \{\mathbf{e}\})^*$, $\mathcal{L}(g)^{-1}\mathcal{L}(\mathbf{r}) = \mathcal{L}(\partial_g(\mathbf{r}))$.

Proof. By induction on $|g|$ using Lemma 14. \square

Lemma 16. For all $w \in \Sigma^*$, the following propositions are equivalent:

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

Proof. For $w = \varepsilon$, $n = 0$ and $\mathbf{c}(\mathbf{r}) = \varepsilon$ if and only if $\mathbf{c}(\partial_\varepsilon(\mathbf{r})) = \varepsilon$. For $w \neq \varepsilon$ we prove by induction on the structure of \mathbf{r} .

If $\mathbf{r} = F \neq \mathbf{e}$ and $w \in \mathcal{L}(\mathbf{r})$, w is some letter x . Then $\exists x \cap F \neq \perp$ and also $\mathbf{c}(\partial_F(\mathbf{r})) = \varepsilon$. If 2. holds then $\partial_{\mathbf{s}(w)}(\mathbf{r}) \neq \emptyset$ if and only if $\mathbf{s}(w) = F$. Because $\exists x \cap F \neq \perp$ we have $w \in \mathcal{L}(\mathbf{r})$.

Suppose the result holds for \mathbf{r}_1 and \mathbf{r}_2 .

Let $\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2$. If $w \in \mathcal{L}(\mathbf{r})$ suppose without loss of generality that $w \in \mathcal{L}(\mathbf{r}_1)$. By the induction hypothesis there exists $\mathbf{s}(w) = F_1 \cdots F_n$ with $F_i \in \text{SS}(\mathbf{r}_1)$, $\exists x_i \cap F_i \neq \perp$ and $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r}_1)) = \varepsilon$. Using Lemma 15, $\varepsilon \in \mathcal{L}(\mathbf{s}(w)^{-1}\mathcal{L}(\mathbf{r}_1)) \subseteq \mathcal{L}(\mathbf{s}(w))^{-1}\mathcal{L}(\mathbf{r}_1 + \mathbf{r}_2) = \mathcal{L}(\mathbf{s}(w))^{-1}\mathcal{L}(\mathbf{r})$ and thus $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r})) = \varepsilon$. If 2. holds suppose without loss of generality that $\varepsilon \in \mathcal{L}(\partial_{\mathbf{s}(w)}(\mathbf{r}_1))$, i.e. $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r}_1)) = \varepsilon$. Then $w \in \mathcal{L}(\mathbf{r}_1) \subseteq \mathcal{L}(\mathbf{r})$.

For $\mathbf{r} = \mathbf{r}_1 \cdot \mathbf{r}_2$ and $w \in \mathcal{L}(\mathbf{r})$, $w = w_1 w_2$ with $w_i \in \mathcal{L}(\mathbf{r}_i)$, for $i = 1, 2$. If $w_1 = \varepsilon$ let $w_2 = x_1 \cdots x_n$. By the induction hypothesis there exists $\mathbf{s}(w) = F_1 \cdots F_n$ with $F_i \in \text{SS}(\mathbf{r}_2)$, $\exists x_i \cap F_i \neq \perp$ and $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r}_2)) = \varepsilon$. Using Lemma 15, $\varepsilon \in \mathcal{L}(\mathbf{s}(w))^{-1}\mathcal{L}(\mathbf{r}_2) \subseteq \mathcal{L}(\mathbf{s}(w))^{-1}\mathcal{L}(\mathbf{r}_1 \cdot \mathbf{r}_2) = \mathcal{L}(\mathbf{s}(w))^{-1}\mathcal{L}(\mathbf{r})$ and thus $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r})) = \varepsilon$. If $w_1 = x'_1 \cdots x'_m \neq \varepsilon$ then there also exists $\mathbf{s}(w') = F'_1 \cdots F'_m$ with $F'_i \in \text{SS}(\mathbf{r}_1)$, $\exists x'_i \cap F'_i \neq \perp$ and $\mathbf{c}(\partial_{w'_x}(\mathbf{r}_1)) = \varepsilon$. If $w_2 = \varepsilon$ then $\varepsilon \in \mathcal{L}(\mathbf{s}(w'))^{-1}\mathcal{L}(\mathbf{r}_1) \subseteq \mathcal{L}(\mathbf{s}(w'))^{-1}\mathcal{L}(\mathbf{r}_1 \cdot \mathbf{r}_2) = \mathcal{L}(\mathbf{s}(w'))^{-1}\mathcal{L}(\mathbf{r})$ and thus $\mathbf{c}(\partial_{\mathbf{s}(w')}(\mathbf{r})) = \varepsilon$. Otherwise, let

$\mathfrak{s}(w)$ be as in the case of $w_1 = \varepsilon$, and one concludes that $\varepsilon \in \mathcal{L}(\mathfrak{s}(w')\mathfrak{s}(w))^{-1}\mathcal{L}(\mathbf{r}) = \mathcal{L}(\mathfrak{s}(w))^{-1}(\mathcal{L}(\mathfrak{s}(w'))^{-1}\mathcal{L}(\mathbf{r}_1))\mathcal{L}(\mathbf{r}_2)$ and thus $\mathfrak{c}(\partial_{\mathfrak{s}(w')\mathfrak{s}(w)}(\mathbf{r})) = \varepsilon$.

If 2. holds then $\varepsilon \in \mathcal{L}(\mathfrak{s}(w))^{-1}(\mathcal{L}(\mathbf{r}_1)\mathcal{L}(\mathbf{r}_2)) = (\mathcal{L}(F_1) \cdots \mathcal{L}(F_n))^{-1}(\mathcal{L}(\mathbf{r}_1)\mathcal{L}(\mathbf{r}_2))$. We have three cases to consider:

- a) $\mathfrak{c}(\partial_{\mathfrak{s}(w)}(\mathbf{r}_1)) = \varepsilon$
- b) $\mathfrak{c}(\mathbf{r}_1) = \varepsilon$ and $\mathfrak{c}(\partial_{\mathfrak{s}(w)}(\mathbf{r}_2)) = \varepsilon$
- c) $\mathfrak{s}(w) = \mathfrak{s}(u)\mathfrak{s}(v)$ with $\mathfrak{s}(u) = B_1 \cdots B_j$ and $\mathfrak{s}(v) = B_{j+1} \cdots B_n$ and $\mathfrak{c}(\partial_{\mathfrak{s}(u)}(\mathbf{r}_1)) = \varepsilon$ and $\mathfrak{c}(\partial_{\mathfrak{s}(v)}(\mathbf{r}_2)) = \varepsilon$.

For $\mathbf{r} = \mathbf{r}_1^*$, if $w \in \mathcal{L}(\mathbf{r})$ there exists n such that $w \in \mathcal{L}(\mathbf{r}_1)^n$. Then the proof is similar to the case of concatenation. \square

The set of all partial derivatives of \mathbf{r} w.r.t. non-empty words is denoted by $\partial^+(\mathbf{r})$. Then $\text{PD}(\mathbf{r}) = \partial^+(\mathbf{r}) \cup \{\mathbf{r}\}$.

Lemma 17. *For $\mathbf{r} \in \text{REG SSP}[\Sigma]$, the following hold.*

1. If $\partial^+(\mathbf{r}) \neq \emptyset$, then there is $\mathbf{r}_0 \in \partial^+(\mathbf{r})$ with $\mathfrak{c}(\mathbf{r}_0) = \varepsilon$.
2. If $\partial^+(\mathbf{r}) = \emptyset$ and $\mathbf{r} \neq \emptyset$, then $\mathcal{L}(\mathbf{r}) = \{\varepsilon\}$ and $\mathfrak{c}(\mathbf{r}) = \varepsilon$.

Proof. 1. From the definition of regular expressions follows that \emptyset cannot appear as a subexpression of a larger term. Suppose that there is some $\mathbf{v} \in \partial^+(\mathbf{r})$ and $\mathcal{L}(\mathbf{v}) \neq \emptyset$. Then there is some word $w \in \Sigma^*$ such that $w \in \mathcal{L}(\mathbf{v})$. Then $\varepsilon \in \mathcal{L}(\partial_{\mathfrak{s}(w)}(\mathbf{v}))$ which means that there is some $\mathbf{r}_0 \in \partial_{\mathfrak{s}(w)}(\mathbf{v}) \subseteq \partial^+(\mathbf{r})$ such that $\mathfrak{c}(\mathbf{r}_0) = \varepsilon$.

2. $\partial^+(\mathbf{r}) = \emptyset$ implies that there is no word $z \in \Sigma^+$ in $\mathcal{L}(\mathbf{r})$. On the other hand, since \emptyset does not appear in \mathbf{r} , it follows that $\mathcal{L}(\mathbf{r}) \neq \emptyset$. Thus, $\mathcal{L}(\mathbf{r}) = \{\varepsilon\}$. \square

The following proposition generalizes from ordinary regular expressions [18,10,6], and shows that the set of partial derivatives is finite.

Lemma 18. *∂^+ satisfies the following:*

$$\begin{aligned} \partial^+(\emptyset) &= \partial^+(\mathbf{e}) = \emptyset, & \partial^+(\mathbf{r}_1 + \mathbf{r}_2) &= \partial^+(\mathbf{r}_1) \cup \partial^+(\mathbf{r}_2), \\ \partial^+(F) &= \{\varepsilon\} & \partial^+(\mathbf{r}_1 \mathbf{r}_2) &= \partial^+(\mathbf{r}_1)\mathbf{r}_2 \cup \partial^+(\mathbf{r}_2), \\ \partial^+(\mathbf{r}^*) &= \partial^+(\mathbf{r})\mathbf{r}^*. \end{aligned}$$

Theorem 2.

$$\begin{aligned} |\partial^+(\mathbf{r})| &\leq \|\mathbf{r}\|, \\ |\text{PD}(\mathbf{r})| &\leq \|\mathbf{r}\| + 1. \end{aligned}$$

Proof. Direct consequence of Lemma 18 using induction on \mathbf{r} . \square

The following proposition shows that the *partial derivative automaton* of \mathbf{r} with set specifications is equivalent to \mathbf{r} .

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

Proof. By induction on $|g|$ with $g \in \text{SS}(\mathbf{r})^*$, one can prove that there is a path from \mathbf{r}_1 to \mathbf{r}_2 labeled by g if and only if $\mathbf{r}_2 \in \partial_g(\mathbf{r}_1)$, for any $\mathbf{r}_1 \in \text{PD}(\mathbf{r})$. Now, we prove that for any $\mathbf{r}_1 \in \text{PD}(\mathbf{r})$ and $w \in \Sigma^*$,

$$w \in \mathcal{L}_{\mathbf{r}_1}(\hat{a}_{\text{PD}}(\mathbf{r})) \Leftrightarrow w \in \mathcal{L}(\mathbf{r}_1).$$

Let $w = x_1 \cdots x_n$. If $w \in \mathcal{L}(\mathbf{r}_1)$ applying Lemma 16 one concludes that there exists $\mathbf{s}(w) = F_1 \cdots F_n$ such that $\mathbf{c}(\partial_{\mathbf{s}(w)}(\mathbf{r}_1)) = \varepsilon$. Then there exists $\mathbf{r}_2 \in \partial_{\mathbf{s}(w)}(\mathbf{r}_1)$ such that $\mathbf{c}(\mathbf{r}_2) = \varepsilon$, and thus $w \in \mathcal{L}_{\mathbf{r}_1}(\hat{a}_{\text{PD}}(\mathbf{r}))$.

If $w \in \mathcal{L}_{\mathbf{r}_1}(\hat{a}_{\text{PD}}(\mathbf{r}))$, there is an accepting path from \mathbf{r}_1 to a state \mathbf{r}_2 labeled by $\mathbf{s}(w) = F_1 \cdots F_n$, $\mathbf{c}(\mathbf{r}_2) = \varepsilon$ and $w \in \mathcal{L}(F_1) \cdots \mathcal{L}(F_n)$. Then we conclude that $\exists x_i \cap F_i \neq \perp$ for $i = 1, \dots, n$, and again by Lemma 16, $w \in \mathcal{L}(\mathbf{r}_1)$. \square

10 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 $\text{mon } B, \text{mon } B'$ such that, when defined, the product $m \odot m'$ of two monoid elements belongs to $\text{mon } C$. 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 10. 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 ,

$$R \downarrow L = R \cap (L \times \Delta^*). \quad (7)$$

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

$$R \uparrow L = R \cap (\Sigma^* \times L). \quad (8)$$

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 11. The following polymorphic operations are based on label sets of standard automata and transducers using the monoidal operations in Ex. 10.

- “ $\cap : \Sigma_e \times \Sigma_e \Rightarrow \Sigma_e$ ” 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, $\mathcal{R}((x/y) \downarrow z) = \mathcal{R}(x/y) \downarrow \mathcal{L}(z)$.

- “ $\uparrow : [\Sigma_e, \Delta_e] \times \Delta_e \Rightarrow [\Sigma_e, \Delta_e]$ ” is defined by
 - the operation $\uparrow : [\Sigma_e, \Delta_e] \times \Delta_e \dashrightarrow [\Sigma_e, \Delta_e]$ such that $(x/y) \uparrow z = (x/y)$ if $x = z$, else $(x/y) \uparrow z = \perp$; and
 - the operation $\uparrow : (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$.

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

Example 12. The following polymorphic operations are based on label sets of automata and transducers with set specs.

- “ $\cap : \text{SSP}[\Gamma] \times \text{SSP}[\Gamma] \Rightarrow \text{SSP}[\Gamma]$ ” is defined by the partial operation $\cap : \text{SSP}[\Gamma] \times \text{SSP}[\Gamma] \dashrightarrow \text{SSP}[\Gamma]$, according to Def. 1, and the partial operation $\cap : \Gamma^* \times \Gamma^* \dashrightarrow \Gamma^*$. By Lemma 2, for any $B, F \in \text{SSP}[\Gamma]$, we have that

$$\mathcal{L}(B \cap F) = \mathcal{L}(B) \cap \mathcal{L}(F).$$

- “ $\downarrow: \text{PSP}[I] \times I_e \Rightarrow \text{PSP}[I]$ ” is defined as follows. First, by the partial operation $\downarrow: \text{PSP}[I] \times I_e \dashrightarrow \text{PSP}[I]$ such that

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

Second, by the partial operation $\downarrow: (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$. We have that

$$\mathcal{R}(\mathfrak{p} \downarrow x) = \mathcal{R}(\mathfrak{p}) \downarrow \mathcal{L}(x)$$

Moreover we have that $\mathfrak{p} \downarrow x$ can be computed from \mathfrak{p} and x in time $O(|\mathfrak{p}|)$.

- “ $\uparrow: \text{PSP}[I] \times \Delta_e \Rightarrow \text{PSP}[I]$ ” is defined as follows. First, by the partial operation $\uparrow: \text{PSP}[I] \times \Delta_e \dashrightarrow \text{PSP}[I]$ such that $\mathfrak{p} \uparrow x = (\mathfrak{p}^{-1} \downarrow x)^{-1}$. Second, by the partial operation $\uparrow: (\Sigma^* \times \Delta^*) \times \Delta^* \dashrightarrow (\Sigma^* \times \Delta^*)$. We have that

$$\mathcal{R}(\mathfrak{p} \uparrow x) = \mathcal{R}(\mathfrak{p}) \uparrow \mathcal{L}(x)$$

Moreover we have that $\mathfrak{p} \uparrow x$ can be computed from \mathfrak{p} and x in time $O(|\mathfrak{p}|)$.

Further below, in Sect. 12, 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

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

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 $\varepsilon_{\text{mon } B} \in \mathcal{I}(\beta)$ then modify \hat{g}' to \hat{g}'^ε ; and if there is a label β' in \hat{g}' (before being modified) such that $\varepsilon_{\text{mon } B'} \in \mathcal{I}(\beta')$ then modify \hat{g}' to \hat{g}'^ε . 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 $\beta \odot \beta' \neq \perp$ then $(q, q') \in P$ and $((p, p'), \beta \odot \beta', (q, q')) \in \delta \odot \delta'$.

Example 13. 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 automaton \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. 14.

Lemma 19. *The following statements hold true about the product graph $\hat{g} \circ \hat{g}' = (P, C, \delta \circ \delta', I \times I', F \times F')$ of two trim labelled graphs \hat{g}, \hat{g}' as defined in Def. 10.*

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

Proof. As $P \subseteq Q \times Q'$, Lemma 6 implies that $|P| \leq (2|\delta| + 1)(2|\delta'| + 1)$, so $|P| = O(|\delta||\delta'|)$. As we get at most one transition in $\delta \circ \delta'$ for each pair of transitions in δ and δ' , we have that $|\delta \circ \delta'| \leq |\delta||\delta'|$. For the second statement, we have that $\delta \circ \delta'$ can be computed in time

$$\sum_{(p, \beta, q) \in \delta} \sum_{(p', \beta', q') \in \delta'} C_{\beta, \beta'}$$

where $C_{\beta, \beta'}$ is the cost of computing the value $\beta \circ \beta'$ from the labels β and β' . Then, the statement follows using standard summation manipulations and the premise that $C_{\beta, \beta'}$ is of magnitude $O(|\beta| + |\beta'|)$.

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

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

Proof. Recall that each transition (p, \underline{m}, q) of $\exp \hat{g}$ comes from a corresponding transition (p, β, q) of \hat{g} such that $m \in \mathcal{I}_1(\beta)$; and similarly each transition (p', \underline{m}', q') of $\exp \hat{g}'$ comes from a corresponding transition (p', β', q') of \hat{g}' such that $m' \in \mathcal{I}_2(\beta')$; where we used $\mathcal{I}_1, \mathcal{I}_2$ for the behaviours of B, B' . Also, if $\beta \circ \beta' \neq \perp$ and $m \circ m' \neq \perp$ then

$$((p, p'), \beta \circ \beta', (q, q')) \text{ is a transition of } \hat{g} \circ \hat{g}' \text{ and}$$

$((p, p'), \underline{m \odot m'}, (q, q'))$ is a transition of $(\exp \hat{g} \odot \exp \hat{g}')$.

First consider any $m \in \mathcal{I}(\exp \hat{g} \odot \exp \hat{g}')$. Then $\exp \hat{g} \odot \exp \hat{g}'$ has an accepting path

$$\langle (q_{i-1}, q'_{i-1}), \underline{m_i \odot m'_i}, (q_i, q'_i) \rangle_{i=1}^{\ell} \text{ such that } m = (m_1 \odot m'_1) \cdots (m_{\ell} \odot m'_{\ell}).$$

Then, for each $i = 1, \dots, \ell$, there is a transition (q_{i-1}, β_i, q_i) of \hat{g} with $m_i \in \mathcal{I}_1(\beta_i)$; and similarly for \hat{g}' , we have $m'_i \in \mathcal{I}_2(\beta'_i)$. Then,

$$(m_i \odot m'_i) \in \mathcal{I}(\beta_i) \odot \mathcal{I}(\beta'_i) = \mathcal{I}(\beta_i \odot \beta'_i)$$

Moreover, $\hat{g} \odot \hat{g}'$ has the accepting path

$$\langle (q_{i-1}, q'_{i-1}), \beta_i \odot \beta'_i, (q_i, q'_i) \rangle_{i=1}^{\ell}$$

which implies that $\mathcal{I}(\beta_1 \odot \beta'_1) \cdots \mathcal{I}(\beta_{\ell} \odot \beta'_{\ell}) \subseteq \mathcal{I}(\hat{g} \odot \hat{g}')$. Hence, $m \in \mathcal{I}(\hat{g} \odot \hat{g}')$. Conversely, consider any $m \in \mathcal{I}(\hat{g} \odot \hat{g}')$. Then $\hat{g} \odot \hat{g}'$ has an accepting path

$$\langle (q_{i-1}, q'_{i-1}), \beta_i \odot \beta'_i, (q_i, q'_i) \rangle_{i=1}^{\ell} \text{ such that } m = m_1 \cdots m_{\ell}$$

and each $m_i \in \mathcal{I}(\beta_i \odot \beta'_i) = \mathcal{I}_1(\beta_i) \odot \mathcal{I}_2(\beta'_i)$, which implies that

$$\text{each } m_i = k_i \odot k'_i \text{ with } k_i \in \mathcal{I}_1(\beta_i) \text{ and } k'_i \in \mathcal{I}_2(\beta'_i).$$

Then, for each $i = 1, \dots, \ell$, there is a transition $(q_{i-1}, \underline{k_i}, q_i)$ of $\exp \hat{g}$ and similarly there is a transition $(q'_{i-1}, \underline{k'_i}, q'_i)$ of $\exp \hat{g}'$. Then $\exp \hat{g} \odot \exp \hat{g}'$ has the accepting path

$$\langle (q_{i-1}, q'_{i-1}), \underline{k_i \odot k'_i}, (q_i, q'_i) \rangle_{i=1}^{\ell}$$

which implies that $(k_1 \odot k'_1) \cdots (k_{\ell} \odot k'_{\ell}) \in \mathcal{I}(\exp \hat{g} \odot \exp \hat{g}')$. Hence, $m \in \mathcal{I}(\exp \hat{g} \odot \exp \hat{g}')$. \square

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 Ex. 13) 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 20.2, in Theorem 5 and in Corollary 3.

11 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 Section 14).

Remark 9. For every ε -NFA $\hat{a} = (Q, \Gamma_e, \delta, I, F)$, one can make in linear time an automaton with set specs $\hat{a}' = (Q, \text{SSP}[\Gamma], \delta', I, F)$ such that, δ' consists of all transitions $(p, e, q) \in \delta$ union all transitions $(p, \exists g, q)$ where $(p, g, q) \in \delta$ and $g \in \Gamma$.

Lemma 20. *In the statements below,*

$$\hat{b} = (Q, \text{SSP}[\Gamma], \delta, I, F) \quad \text{and} \quad \hat{b}' = (Q', \text{SSP}[\Gamma], \delta', I', F')$$

are trim automata with set specs and w is a string.

1. There is a $O(|\hat{b}|)$ algorithm `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, `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. For the first statement, we simply use a breadth-first search (BFS) algorithm, starting from any initial state $s \in I$, which is considered visited, and stopping, either when a final state is reached (trying if necessary all initial states), or all states have been visited. In the latter case the desired algorithm returns `None` (or `False`). For the algorithm `emptyP`(\hat{b}) nothing further is needed. For `nonEmptyW`(\hat{b}), when a non-visited state q is visited from a previously visited state p using a transition $e = (p, \beta, q)$, an element $x \in \mathcal{L}(\beta)$ is computed in time $O(|\beta|) = O(|e|)$, using Lemma 3. The algorithm also constructs a string graph G that will be used to find the desired word in $\mathcal{L}(\hat{b})$. When the above transition is accessed and x is computed then the edge (q, x, p) is added to G . If the algorithm stops because it reached a final state f , then there is a unique path in G from f to the initial state s , which can be used to find the desired word in $\mathcal{L}(\hat{b})$ (the path is unique as every state is visited only once). The cost of BFS is $O(|Q| + |\delta|)$, but here when an edge $e \in \delta$ is accessed the algorithm spends time $O(|e|)$, so the cost is

$$O(|Q| + \sum_{e \in \delta} |e|).$$

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 19 implies that $\hat{b} \cap \hat{b}'$ can be computed in time $O(|\Gamma| + |\delta||\delta'| + |\delta'||\delta|)$. Now we have

$$\mathcal{L}(\hat{b} \cap \hat{b}') = \mathcal{L}(\text{exp } \hat{b} \cap \text{exp } \hat{b}') \tag{9}$$

$$= \mathcal{L}(\text{exp } \hat{b}) \cap \mathcal{L}(\text{exp } \hat{b}') \tag{10}$$

$$= \mathcal{L}(\hat{b}) \cap \mathcal{L}(\hat{b}') \tag{11}$$

Statement (9) follows from the fact that “ $\cap : \text{SSP}[\Gamma] \times \text{SSP}[\Gamma] \Rightarrow \text{SSP}[\Gamma]$ ” is a polymorphic operation—see Theorem 4 and Ex. 12. Statement (10) follows from the fact that each $\text{exp } \hat{b}, \text{exp } \hat{b}'$ is an ε -NFA and the operation \cap is well-defined on these objects—see Lemma 8 and Ex. 13.

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 `emptyP`(\hat{a}) to get the desired answer.

Lemma 21. *In the statements below, $\hat{s} = (Q, \text{PSP}[\Gamma], \delta, I, F)$ is a trim transducer with set specs and $\hat{a} = (Q', \Gamma_e, \delta', I', F')$ is a trim ε -NFA and (u, v) is a pair of words.*

1. *There is a $O(|\hat{s}|)$ algorithm `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, `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 $(u, v) \in \mathcal{R}(\hat{s})$.*

Proof. The first statement is completely analogous to the first statement of Lemma 20. For the second statement, we compute the product $\hat{s} \downarrow \hat{a}$. As the product $\mathbf{p} \downarrow x$ of two labels can be computed in linear time, Lemma 19 implies that $\hat{s} \downarrow \hat{a}$ can be computed in time $O(|\Gamma| + |\delta||\delta'| + |\delta'||\delta|)$. Now we have

$$\mathcal{R}(\hat{s} \downarrow \hat{a}) = \mathcal{R}(\exp \hat{s} \downarrow \exp \hat{a}) \tag{12}$$

$$= \mathcal{R}(\exp \hat{s}) \downarrow \mathcal{L}(\exp \hat{a}) \tag{13}$$

$$= \mathcal{R}(\hat{s}) \downarrow \mathcal{L}(\hat{a}) \tag{14}$$

Statement (12) follows from the fact that ‘ $\downarrow: \text{PSP}[\Gamma] \times \Gamma_e \Rightarrow \text{PSP}[\Gamma]$ ’ is a polymorphic operation—see Theorem 4 and Ex. 12. Statement (13) follows from the fact that $\exp \hat{s}$ is a transducer and $\exp \hat{a}$ is an ε -NFA and the operation \downarrow is well-defined on these objects—see Lemma 8 and Ex. 13.

For the third statement, first make two automata with set specs \hat{b}_u and \hat{b}_v accepting $\{u\}$ and $\{v\}$ respectively, then compute $\hat{t} = \hat{s} \downarrow \hat{a}_u \uparrow \hat{a}_v$, and then use `emptyP`(\hat{t}) to get the desired answer.

12 Composition of Transducers with Set Specifications

Next we are interested in defining the composition $\mathbf{p}_1 \circ \mathbf{p}_2$ of two pairing specs in a way that $\mathcal{R}(\mathbf{p}_1) \circ \mathcal{R}(\mathbf{p}_2)$ is equal to $\mathcal{R}(\mathbf{p}_1 \circ \mathbf{p}_2)$. By Definition 5, the operator $\mathcal{R}()$ is defined with respect to an alphabet of reference Γ , so the value of $\mathbf{p}_1 \circ \mathbf{p}_2$ should depend on Γ . It turns out that, for a particular subcase about the structure of $\mathbf{p}_1, \mathbf{p}_2$, the operation $\mathbf{p}_1 \circ \mathbf{p}_2$ can produce two or three pairing specs. To account for this, we define a new label set:

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

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

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

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

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

is defined between any two pairing specs $\mathfrak{p}_1, \mathfrak{p}_2$ respecting Γ as follows, where again \perp means undefined.

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

Now we assume that the above condition is not true and we consider the structure of \mathfrak{p}_1 and \mathfrak{p}_2 according to Def. 3(2) using A, B, F, G, W, X, Y, Z as set specs, where $A, B, F, G \neq e$ —thus, we assume below that $\mathcal{L}(B) \cap \mathcal{L}(F) \neq \emptyset$ and $\mathcal{L}(X) \cap \mathcal{L}(Y) \neq \emptyset$.

$$(W/X) \circ (Y/Z) = W/Z$$

$$(W/B) \circ (F/=) = W/B \cap F$$

$$(W/B) \circ (F/G \neq) = \begin{cases} W/G, & \text{if } |\mathcal{L}(B \cap F)| \geq 2 \\ W/G \cap \nexists b, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(G) \setminus \{b\} \neq \emptyset \\ \perp, & \text{otherwise.} \end{cases}$$

$$(B/=) \circ (F/Z) = B \cap F/Z$$

$$(B/=) \circ (F/=) = B \cap F/=$$

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

$$(A/B \neq) \circ (F/Z) = \begin{cases} A/Z, & \text{if } |\mathcal{L}(B \cap F)| \geq 2 \\ A \cap \nexists b/Z, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(A) \setminus \{b\} \neq \emptyset \\ \perp & \text{otherwise.} \end{cases}$$

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

$$(A/B \neq) \circ (F/G \neq) = \begin{cases} A/G, & \text{if } |\mathcal{L}(B \cap F)| \geq 3 \\ A \cap \nexists b/G \cap \nexists b, & \text{if } \mathcal{L}(B \cap F) = \{b\} \text{ and } \mathcal{L}(A) \setminus \{b\} \\ & \neq \emptyset \text{ and } \mathcal{L}(G) \setminus \{b\} \neq \emptyset \\ \mathbf{D}, & \text{if } \mathcal{L}(B \cap F) = \{b_1, b_2\} \\ \perp, & \text{otherwise} \end{cases}$$

where \mathbf{D} consists of up to three \oplus -terms as follows:

\mathbf{D} includes $A \cap \nexists b_1 b_2/G$, if $\mathcal{L}(A) \setminus \{b_1, b_2\} \neq \emptyset$;

\mathbf{D} includes $\exists b_1/G \cap \nexists b_2$, if $b_1 \in \mathcal{L}(A)$ and $\mathcal{L}(G) \setminus \{b_2\} \neq \emptyset$;

\mathbf{D} includes $\exists b_2/G \cap \nexists b_1$, if $b_2 \in \mathcal{L}(A)$ and $\mathcal{L}(G) \setminus \{b_1\} \neq \emptyset$;

and $\mathbf{D} = \perp$ if none of the above three conditions is true.

Remark 10. In the above definition, we have omitted cases where $\mathfrak{p}_1 \circ \mathfrak{p}_2$ is obviously undefined. For example, as $F/=$ and $F/G \neq$ are only defined when $F, G \neq e$, we omit the case $(W/e) \circ (F/=)$.

Remark 11. If we allowed \perp to be a pairing spec, then the set $\text{PSP}[\Gamma]$ 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 $\mathcal{R}(\mathfrak{p})$, for any pairing spec \mathfrak{p} .

Lemma 22. *The relation $\mathcal{R}(\mathfrak{p}_1 \circ \mathfrak{p}_2)$ is equal to $\mathcal{R}(\mathfrak{p}_1) \circ \mathcal{R}(\mathfrak{p}_2)$, for any pairing specs $\mathfrak{p}_1, \mathfrak{p}_2$ respecting Γ .*

Proof. We shall use the following shorthand notation:

$$Q = \mathcal{R}(\mathfrak{p}_1) \circ \mathcal{R}(\mathfrak{p}_2) \text{ and } R = \mathcal{R}(\mathfrak{p}_1 \circ \mathfrak{p}_2) \text{ and } \mathcal{R}(\perp) = \emptyset.$$

We shall distinguish several cases about the form of $\mathfrak{p}_1 \circ \mathfrak{p}_2$ according to Def. 11. By looking at that definition and using (3), we have that

$$Q, R \subseteq \mathcal{L}(\text{left } \mathfrak{p}_1) \times \mathcal{L}(\text{rset } \mathfrak{p}_2) \quad (15)$$

Case $(W/X) \circ (Y/Z) = W/Z$. We have that $R = \mathcal{R}(W/Z) = \mathcal{L}(W) \times \mathcal{L}(Z)$. As Q consists of all pairs $(w, z) = (w, x) \circ (x, z)$ with $w \in \mathcal{L}(W), z \in \mathcal{L}(Z)$ and $x \in \mathcal{L}(X) \cap \mathcal{L}(Y)$, we have that $Q = R$.

Case $(W/B) \circ (F/=) = W/B \cap F$. We have that $R = \mathcal{R}(W/B \cap F) = \mathcal{L}(W) \times \mathcal{L}(B \cap F)$. As Q consists of all pairs $(w, f) = (w, b) \circ (f, f)$ with $w \in \mathcal{L}(W), b \in \mathcal{L}(B), f \in \mathcal{L}(F)$ and $b = f$, we have that $Q = R$.

Case $(W/B) \circ (F/G \neq)$. We have three subcases. First, when $|\mathcal{L}(B \cap F)| \geq 2$. Then, $R = \mathcal{L}(W) \times \mathcal{L}(G)$. By (15), $Q \subseteq R$. Now let $(w, g) \in R = \mathcal{L}(W) \times \mathcal{L}(G)$, and pick any $b \in \mathcal{L}(B \cap F) \setminus \{g\}$. Then, $(w, g) = (w, b) \circ (b, g) \in Q$. Hence, $R \subseteq Q$. In the second subcase, $\mathcal{L}(B \cap F) = \{b\}$, for some $b \in \Gamma$, and $\mathcal{L}(G) \setminus \{b\} \neq \emptyset$. Then, $R = \mathcal{L}(W) \times \mathcal{L}(G \cap \nabla b)$. The claim $R = Q$ follows by noting that Q consists of all pairs $(w, g) = (w, b) \circ (f, g)$ with $w \in \mathcal{L}(W), f \in \mathcal{L}(F), g \in \mathcal{L}(G)$ and $f = b$ and $f \neq g$. In the third subcase, $\mathcal{L}(G) = \mathcal{L}(B \cap F) = \{b\}$, so $Q = \emptyset$, so $Q = R$.

Case $(B/=) \circ (F/Z) = B \cap F/Z$. Analogous to case $(W/B) \circ (F/=) = W/B \cap F$.

Case $(B/=) \circ (F/=) = B \cap F/=$. Similar to case $(W/B) \circ (F/=) = W/B \cap F$, where here $R = \{(b, b) \mid b \in B \cap F\}$.

Case $(B/=) \circ (F/G \neq)$. First, note that $(b, g) \in Q$ iff “ $(b, g) = (b, b) \circ (f, g)$ with $g \neq f = b \in \mathcal{L}(B \cap F)$ ”. Thus, if $\mathcal{L}(G) = \mathcal{L}(B \cap F) = \{g\}$, for some $g \in \Gamma$, then $Q = \emptyset = R$. Otherwise, any $(b, g) \in Q$ must be in $\mathcal{R}(B \cap F/G \neq) = R$; and conversely, if $(b, g) \in R$ then $g \neq b$ and $b \in \mathcal{L}(B \cap F)$. As $(b, g) = (b, b) \circ (b, g)$ we have that $(b, g) \in \mathcal{R}(B/=) \circ \mathcal{R}(F/G \neq) = Q$.

Case $(A/B \neq) \circ (F/Z)$. Analogous to case $(W/B) \circ (F/G \neq)$.

Case $(A/B \neq) \circ (F/=)$. First note that $(a, f) \in Q$ iff $(a, f) = (a, b) \circ (b, f)$ with $a \neq b = f \in \mathcal{L}(B \cap F)$. Thus, if $\mathcal{L}(A) = \mathcal{L}(B \cap F) = \{a\}$, for some $a \in \Gamma$, then $Q = \emptyset = R$. Otherwise, any $(a, f) \in Q$ must be in $\mathcal{R}(A/B \neq \cap F/=) = R$; and conversely, if $(a, f) \in R$ then $a \neq f$ and $f \in \mathcal{L}(B \cap F)$. As $(a, f) = (a, f) \circ (f, f)$ we have that $(a, f) \in \mathcal{R}(A/B \neq) \circ \mathcal{R}(F/=) = Q$.

Case $(A/B \neq) \circ (F/G \neq)$. First note that, if $(a, g) \in Q$, then $(a, g) \in \mathcal{L}(A) \times \mathcal{L}(G)$ and

$$(a, g) = (a, b) \circ (f, g) \text{ with } a \neq b = f \neq g, b \in \mathcal{L}(B) \cap \mathcal{L}(F).$$

We have three subcases. First, $\mathcal{L}(B \cap F) \geq 3$. Then $R = \mathcal{R}(A/G)$ and, by (15), $Q \subseteq R$. Now let $(a, g) \in R$ and pick any $b \in \mathcal{L}(B \cap F) \setminus \{a, g\}$. Then, $(a, g) = (a, b) \circ (b, g) \in \mathcal{R}(A/B \neq) \circ \mathcal{R}(F/G \neq)$. Hence, $R \subseteq Q$. In the second subcase, $\mathcal{L}(B \cap F) = \{b\}$ and $\mathcal{L}(A) \setminus \{b\} \neq \emptyset$ and $\mathcal{L}(G) \setminus \{b\} \neq \emptyset$, for some $b \in \Gamma$. Then the claim $Q = R$ follows by simple inspection on the elements of Q, R . In the third subcase, $\mathcal{L}(B \cap F) = \{b_1, b_2\}$, for some $b_1, b_2 \in \Gamma$. The relation Q can be partitioned into three subsets:

$$\begin{aligned} Q_0 &= \{(a, g) \mid a \in \mathcal{L}(A) \setminus \{b_1, b_2\}, g \in \mathcal{L}(G)\} \\ Q_1 &= \mathcal{L}(A) \times \mathcal{L}(G) \cap \{(b_1, g) \mid g \notin \mathcal{L}(G) \setminus \{b_2\}\} \\ Q_2 &= \mathcal{L}(A) \times \mathcal{L}(G) \cap \{(b_2, g) \mid g \notin \mathcal{L}(G) \setminus \{b_1\}\} \end{aligned}$$

Then we have that

$$\begin{aligned} \mathcal{R}(A \cap \nexists b_1 b_2 / G) &= Q_0, \text{ if } \mathcal{L}(A) \setminus \{b_1, b_2\} \neq \emptyset; \\ \mathcal{R}(\exists b_1 / G \cap \nexists b_2) &= Q_1, \text{ if } b_1 \in \mathcal{L}(A) \text{ and } \mathcal{L}(G) \setminus \{b_2\} \neq \emptyset; \\ \mathcal{R}(\exists b_2 / G \cap \nexists b_1) &= Q_2, \text{ if } b_2 \in \mathcal{L}(A) \text{ and } \mathcal{L}(G) \setminus \{b_1\} \neq \emptyset. \end{aligned}$$

Corollary 2. *The polymorphic operation “ $\circ : \text{PSP}[\Gamma] \times \text{PSP}[\Gamma] \Rightarrow \text{PSP}_+[\Gamma]$ ” is well-defined by the partial operation \circ in Def. 11 and the partial operation \circ in Ex. 10.*

Lemma 23. *For any two pairing specs $\mathbf{p}_1, \mathbf{p}_2$, we have that $\mathbf{p}_1 \circ \mathbf{p}_2$ can be computed in time $O(|\mathbf{p}_1| + |\mathbf{p}_2|)$.*

Proof. Follows from Lemma 3 and the fact that $|\mathbf{p}_1 \circ \mathbf{p}_2| = O(|\mathbf{p}_1| + |\mathbf{p}_2|)$ as seen in Def. 11.

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

Lemma 24. *We have that $\mathcal{R}(\hat{t} \circ \hat{s}) = \mathcal{R}(\hat{t} \circ \hat{s})$.*

Proof. We show the direction $\mathcal{R}(\hat{t} \circ \hat{s}) \subseteq \mathcal{R}(\hat{t} \circ \hat{s})$; the other direction is similar. Let $(u, v) \in \mathcal{R}(\hat{t} \circ \hat{s})$. Then there is an accepting path $P = \langle q_{i-1}, \mathbf{p}_i, q_i \rangle_{i=1}^\ell$ in $\hat{t} \circ \hat{s}$ such that

$$(u, v) \in \mathcal{R}(\mathbf{p}_1) \cdots \mathcal{R}(\mathbf{p}_\ell).$$

For each transition $e = (q_{i-1}, \mathbf{p}_i, q_i)$, define the triple $(q_{i-1}, \mathbf{p}'_i, q_i)$ as follows: $\mathbf{p}'_i = \mathbf{p}_i$, if e is in $\hat{t} \circ \hat{s}$; else, by Def. 12, there is a transition $(q_{i-1}, \mathbf{p}'_i, q_i)$ in $\hat{t} \circ \hat{s}$ such that \mathbf{p}'_i is a \oplus -sum of terms that include \mathbf{p}_i . Then, the sequence $P' = \langle q_{i-1}, \mathbf{p}'_i, q_i \rangle_{i=1}^{\ell}$ is an accepting path of $\hat{t} \circ \hat{s}$ such that

$$(u, v) \in \mathcal{R}(\mathbf{p}'_1) \cdots \mathcal{R}(\mathbf{p}'_{\ell}).$$

Thus, $(u, v) \in \mathcal{R}(\hat{t} \circ \hat{s})$.

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

Proof. The algorithm computes the transducer $\hat{t} \circ \hat{s}$ using the product construction in Def. 10. As the composition $\mathbf{p} \circ \mathbf{p}'$ of any two labels of \hat{t}, \hat{s} can be computed in linear time, we have that $\hat{t} \circ \hat{s}$ can be computed in time $O(|\delta| \|\delta'\| + |\delta'\| \|\delta|)$. Then, in linear time, the algorithm replaces each transition $(p, \mathbf{p}_1 \oplus \cdots \oplus \mathbf{p}_{\ell}, q)$ of $\hat{t} \circ \hat{s}$, with $\ell > 1$, with the ℓ transitions (p, \mathbf{p}_i, q) . Now we have

$$\mathcal{R}(\hat{t} \odot \hat{s}) = \mathcal{R}(\hat{t} \circ \hat{s}) \tag{16}$$

$$= \mathcal{R}(\exp \hat{t} \circ \exp \hat{s}) \tag{17}$$

$$= \mathcal{R}(\exp \hat{t}) \circ \mathcal{R}(\exp \hat{s}) \tag{18}$$

$$= \mathcal{R}(\hat{t}) \circ \mathcal{R}(\hat{s}). \tag{19}$$

Statement (17) follows from Theorem 4 and Ex. 13, and statement (18) follows from Lemma 8.

13 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 [2], the authors present an algorithm `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. In view of Lemma 6, we have that

$$\text{for trim } \hat{t}, \text{ identityP}(\hat{t}) \text{ works in time } O(|\delta| |\Delta|). \tag{20}$$

The algorithm `functionalityP`(\hat{s}) deciding functionality of a transducer $\hat{t} = (Q, I, \delta, I, F)$ first constructs the *square transducer* \hat{u} , [5], 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}^{ε} . 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

$$\text{for trim } \hat{t}, \text{ functionalityP}(\hat{t}) \text{ works in time } O(|\delta|^2 |\Delta|). \tag{21}$$

Lemma 25. *Let $\hat{s} = (Q, \text{PSP}[I], \delta, I, F)$ be a trim transducer with set specs. If any label \mathbf{p} of \hat{s} satisfies one of the following conditions then \hat{s} does not realize an identity. (Below, F, G are set specs other than e .)*

(C1) \mathbf{p} is of the form F/G or F/e or e/G , and $|\mathcal{L}(F)| > 1$ or $|\mathcal{L}(G)| > 1$.

In the following conditions, \mathbf{p} is of the form $F/G \neq$.

(C2) $|\mathcal{L}(F)| > 2$ or $|\mathcal{L}(G)| > 2$.

(C3) $|\mathcal{L}(F)| = 2$ and $|\mathcal{L}(G)| = 2$.

(C4) $|\mathcal{L}(F)| = 1$ and $|\mathcal{L}(G)| = 2$ and $\mathcal{L}(F) \cap \mathcal{L}(G) = \emptyset$.

(C5) $|\mathcal{L}(F)| = 2$ and $|\mathcal{L}(G)| = 1$ and $\mathcal{L}(F) \cap \mathcal{L}(G) = \emptyset$.

Testing whether there is a label of \hat{s} satisfying one of the above conditions can be done in time $O(\|\delta\|)$.

Proof. Suppose (C1) is true. We only present the subcase where $\mathbf{p} = F/G$ and $|\mathcal{L}(F)| > 1$ (the other subcases can be dealt with similarly). Then, there are $f_1, f_2 \in \mathcal{L}(F)$, with $f_1 \neq f_2$, and $y \in \mathcal{L}(G)$. Also, $\text{exp } \hat{s}$ has two transitions of the form $(p, f_1/y, q)$ and $(p, f_2/y, q)$. As \hat{s} is trim, there is a path from I to p with some label u/v and a path from p to F with some label u'/v' . As $(uf_1u', vyv'), (uf_2u', vyv') \in \mathcal{R}(\text{exp } \hat{s})$ and $f_1 \neq f_2$, $\text{exp } \hat{s}$ cannot realize an identity. Now suppose one of (C2)–(C5) is true. One works as above and shows that again $\text{exp } \hat{s}$ cannot realize an identity. For the time complexity, Lemma 3 implies that each condition can be tested in time $O(\mathbf{p})$. For all transitions $(p, \mathbf{p}, q) \in \delta$ this can be done in time $O(\|\delta\|)$.

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

Proof. As \hat{s} is trim, we have that $|Q| \leq 2|\delta| + 1$. First, the algorithm goes through the labels of \hat{s} and returns **False** the first time a label \mathbf{p} satisfies one of the conditions (C1)–(C5) in Lemma 25. Now suppose that no label \mathbf{p} of \hat{s} satisfies any of those conditions. Then, the algorithm computes $\text{exp } \hat{s}$ and returns what **identityP**($\text{exp } \hat{s}$) returns. For each transition $(p, \mathbf{p}, q) \in \delta$ the corresponding transition(s) $(p, x/y, q) \in \delta_{\text{exp}}$ are computed depending on the following five cases about the form of \mathbf{p} .

1. (e/e): Then, $x/y = e/e$.
2. (F/G) or (F/e) or (e/G): As (C1) is false, $\mathcal{L}(F) = \{f\}$ and/or $\mathcal{L}(G) = \{g\}$. Then $x/y = f/g$ or $x/y = f/e$ or $x/y = e/g$, depending on whether $\mathbf{p} = F/G$ or $\mathbf{p} = F/e$ or $\mathbf{p} = e/G$, respectively.
3. ($F/=$): $x/y \in \{(f, f) \mid f \in \mathcal{L}(F)\}$.
4. ($F/G \neq$): with $\mathcal{L}(F) = \{f\}$ and $\mathcal{L}(G) = \{g\}$. If $f = g$ then $\mathcal{R}(\mathbf{p}) = \emptyset$, so no label x/y is defined. If $f \neq g$ then $x/y = f/g$.
5. ($F/G \neq$): with $\mathcal{L}(F) = \{f\}$ and $\mathcal{L}(G) = \{f, g\}$, or $\mathcal{L}(F) = \{f, g\}$ and $\mathcal{L}(G) = \{g\}$. Then $x/y = f/g$.

All cases other than the third one result in at most one transition for each $(p, \mathbf{p}, q) \in \delta$. The third case results into $O(|I|)$ transitions. Thus, $|\delta_{\text{exp}}| =$

$O(|\delta||\Gamma|)$. Then, as $|\exp \hat{s}| = |\delta_{\text{exp}}| + |Q|$ and $|Q| \leq 2|\delta| + 1$, we have that

$$|\delta_{\text{exp}}| = O(|\delta||\Gamma|) \quad \text{and} \quad |\exp \hat{s}| = O(|\Gamma||\delta|). \quad (22)$$

The correctness of the algorithm follows from Lemma 25 and the fact that $\mathcal{R}(\hat{s}) = \mathcal{R}(\exp \hat{s})$.

Now we establish the claim about the time complexity. The total time consists of three parts: $T_1 =$ time to test conditions (C1)–(C5); $T_2 =$ time to construct $\exp \hat{s}$; and $T_3 =$ time to execute $\text{identityP}(\exp \hat{s})$. Lemma 25 implies that $T_1 = O(|\delta|)$. For T_2 , we have that

$$T_2 = \sum_{e=(p,p,q) \in \delta} C_p,$$

where C_p is the cost of computing the set of x/y for which $(p, x/y, q) \in \delta_{\text{exp}}$. We show that $C_p = O(|\Gamma|)$, which implies that $T_2 = O(|\delta||\Gamma|)$. Using Lemma 3, testing for things like $|\mathcal{L}(F)| \geq 2$ can be done in time $O(|F|)$ and also the same time for computing the single element of $\mathcal{L}(F)$ when $|\mathcal{L}(F)| = 1$. The most time intensive task can be in the third case above: compute $\mathcal{L}(F)$ when $F = \exists w$ and $|w| = |\Gamma| - 1$, or $F = \exists w$ and $|w| = 1$. In the former case, $\mathcal{L}(F)$ is computed in time $O(|w|)$ by simply reading off w . In the latter case, we can read Γ and make the word $u = \text{wo}(\Gamma)$, and then use Lemma 1 to compute $\exists u \cap \exists w$ in time $O(|\Gamma|)$, which is of the form $\exists v$ and equal to $\mathcal{L}(F)$. For T_3 , statement (20) implies that $\text{identityP}(\exp \hat{s})$ works in time $O(|\delta_{\text{exp}}| + |Q||\Gamma|)$, which is $O(|\delta||\Gamma|)$ using (22) and $|Q| \leq 2|\delta| + 1$. Hence, $T_3 = O(|\delta||\Gamma|)$. Thus, $T_1 + T_2 + T_3 = O(|\delta||\Gamma|)$ using Remark 8. \square

Remark 12. Consider the trim transducer \hat{s} with set specs in the above theorem. Of course one can test whether it realizes an identity by simply using $\text{identityP}(\exp \hat{s})$, which would work in time $O(|\delta_{\text{exp}}||\Gamma|)$ according to (20). This time complexity is clearly higher than the time $O(|\delta||\Gamma|)$ in the above theorem when $|\delta_{\text{exp}}|$ 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 $\hat{s} = (Q, \text{PSP}[\Gamma], \delta, I, F)$ with set specs is functional can be answered in time $O(|\delta|^2|\Gamma|)$.*

Proof. Consider any trim transducer \hat{s} with set specs. The algorithm consists of two main parts. First, the algorithm computes \hat{s}^{-1} and then the transducer with set specs $\hat{u} = \hat{s} \circ \hat{s}^{-1}$ using the product construction in Def. 10. The second part is to test whether \hat{u} realizes an identity using Theorem 5. As the composition of any two labels β, β' of \hat{s}, \hat{s}^{-1} results in at most three labels, we have that \hat{u} has $O(|\delta|^2)$ transitions and is of size $O(|\delta||\delta|)$, and can be computed in time $O(|\delta||\delta|)$. Thus, testing \hat{u} for identity can be done in time $O(|\delta|^2|\Gamma|)$. So the total time of the algorithm is of order $|\delta||\delta| + |\delta|^2|\Gamma|$, which is $O(|\delta|^2|\Gamma|)$ by

Remark 8. For the correctness of the algorithm we have that

$$\mathcal{R}(\hat{s}) \text{ is functional iff } \mathcal{R}(\exp \hat{s}) \text{ is functional} \quad (23)$$

$$\text{iff } \mathcal{R}(\exp \hat{s} \circ (\exp \hat{s})^{-1}) \text{ is an identity} \quad (24)$$

$$\text{iff } \mathcal{R}(\exp \hat{s} \circ (\exp \hat{s}^{-1})) \text{ is an identity} \quad (25)$$

$$\text{iff } \mathcal{R}(\exp \hat{s}) \circ \mathcal{R}(\exp \hat{s}^{-1}) \text{ is an identity} \quad (26)$$

$$\text{iff } \mathcal{R}(\hat{s}) \circ \mathcal{R}(\hat{s}^{-1}) \text{ is an identity.} \quad (27)$$

Statement (24) follows from the fact that a relation R is functional iff $R \circ R^{-1}$ is an identity—see also Lemma 5 of [2]. Statement (25) follows from Lemma 9. \square

Remark 13. Consider the trim transducer \hat{s} with set specs in the above theorem. Of course one can test whether \hat{s} is functional by simply using `functionalityP(exp \hat{s})`, which would work in time $O(|\delta_{\text{exp}}|^2|T|)$ according to (21). This time complexity is clearly higher than the time $O(|\delta|^2|T|)$ in the above theorem when $|\delta_{\text{exp}}|$ is of order $|\delta||T|$ or $|\delta||T|^2$ (for example if \hat{s} involves labels $\forall/=$ or \forall/\forall).

14 Transducers and Independent Languages

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

$$u, v \in L \text{ and } v \in \hat{t}(u) \text{ implies } u = v. \quad (28)$$

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

$$\hat{t}(L) \cap L = \emptyset. \quad (29)$$

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 $\hat{t}_{\text{sub}2}$ describes all the 1-substitution error-detecting languages and \hat{t}_{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 (28).

Remark 14. The witness version of the property satisfaction question for input-altering transducers \hat{s} (see Eq. (29)) 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

$$\text{nonEmptyW}(\hat{s} \downarrow \hat{a} \uparrow \hat{a}).$$

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 3. *Let $\hat{s} = (Q, \text{PSP}[\Gamma], \delta, I, F)$ 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*

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

Proof. The statement about the complexity follows from Lemma 19. Then, we have

$$\mathcal{R}(\hat{s} \downarrow \hat{b}) = \mathcal{R}(\text{exp } \hat{s} \downarrow \text{exp } \hat{b}) \tag{30}$$

$$= \mathcal{R}(\text{exp } \hat{s}) \downarrow \mathcal{L}(\text{exp } \hat{b}) \tag{31}$$

$$= \mathcal{R}(\hat{s}) \downarrow \mathcal{L}(\hat{b}). \tag{32}$$

Statement (30) follows from Theorem 4 and Ex. 13, and statement (31) follows from Lemma 8.

Corollary 4. *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 14. 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 $\text{exp } \hat{t}_{\text{sub2}}[\Gamma]$ to decide the question, the required time would be $O(|\Gamma|^2 \cdot |\hat{b}|^2)$.

15 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 $O(|w|)$ —see for instance the DFA with set specs in Fig. 3. 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 obtain a few results like the product construction in Theorem 4. 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 partial derivatives can be defined for regular expressions involving labels other than set and pairing specs⁴.

References

1. Parosh Aziz Abdulla, Johann Deneux, and Lisa KaatiMarcus Nilsson. Minimization of non-deterministic automata with large alphabets. In J. Farré, I. Litovsky, and S. Schmitz, editors, *Proceedings of CIAA 2005, Sydney, Australia*, volume 3845 of *Lecture Notes in Computer Science*, pages 31–42, 2006.
2. Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.
3. V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comput. Sci.*, 155(2):291–319, 1996.
4. Rafaela Bastos, Sabine Broda, António Machiavelo, Nelma Moreira, and Rogério Reis. On the average complexity of partial derivative automata for semi-extended expressions. *Journal of Automata, Languages and Combinatorics*, 22(1–3):5–28, 2017.
5. Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: An efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.
6. Sabine Broda, António Machiavelo, Nelma Moreira, and Rogério Reis. On the average state complexity of partial derivative automata: an analytic combinatorics approach. *International Journal of Foundations of Computer Science*, 22(7):1593–1606, 2011. MR2865339.
7. Janusz A. Brzozowski and Edward J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. Electronic Computers*, 12:67–76, 1963.
8. John Brzozowski. Derivatives of regular expressions. *J. Association for Computer Machinery*, (11):481–494, 1964.
9. Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Partial derivatives of an extended regular expression. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *Proc. 5th LATA 2011*, volume 6638, pages 179–191. Springer, 2011.
10. J. M. Champarnaud and D. Ziadi. From Mirkin’s prebases to Antimirov’s word partial derivatives. *Fundam. Inform.*, 45(3):195–205, 2001.
11. J. M. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theoret. Comput. Sci.*, 289:137–163, 2002.
12. Akim Demaille. Derived-term automata of multitape rational expressions. In Yo-Sub Han and Kai Salomaa, editors, *Proc. 21st CIAA 2016*, volume 9705, pages 51–63. Springer, 2016.
13. FAdo. Tools for formal languages manipulation. URL address: <http://fado.dcc.fc.up.pt/> Accessed in April, 2018.

⁴ While we have not obtained in this work the partial derivative transducer corresponding to a regular expression involving pairing specs, it is our immediate plan to do so.

14. Stavros Konstantinidis. Transducers and the properties of error-detection, error-correction and finite-delay decodability. *Journal Of Universal Computer Science*, 8:278–291, 2002.
15. Stavros Konstantinidis. Applications of transducers in independent languages, word distances, codes. In Giovanni Pighizzini and Cezar Câmpeanu, editors, *Proceedings of DCFs 2017*, number 10316 in Lecture Notes in Computer Science, pages 45–62, 2017.
16. Sylvain Lombardy and Jacques Sakarovitch. Derivatives of rational expressions with multiplicity. *Theor. Comput. Sci.*, 332(1-3):141–177, 2005.
17. Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989.
18. B. G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5:51–57, 1966.
19. Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, Berlin, 2009.
20. Jacques Sakarovitch. Automata and rational expressions. *arXiv.org*, arXiv:1502.03573, 2015.
21. H. J. Szyr and Gabriel Thierrin. Codes and binary relations. In Marie Paule Malliavin, editor, *Séminaire d'Algèbre Paul Dubreil, Paris 1975–1976 (29ème Année)*, volume 586 of *Lecture Notes in Mathematics*, pages 180–188, 1977.
22. Ken Thompson. Regular expression search algorithm. *Communications of the ACM (CACM)*, 11:419–422, 1968.
23. Margus Veanes. Applications of symbolic finite automata. In S. Konstantinidis, editor, *Proceedings of CIAA 2013*, volume 7982 of *Lecture Notes in Computer Science*, pages 16–23, 2013.
24. Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaï Björner. Symbolic finite state transducers: Algorithms and applications. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, pages 137–150, 2012.
25. Derick Wood. *Theory of Computation*. Harper & Row, New York, 1987.
26. Sheng Yu. Regular languages. In [?], pages 41–110.