International Journal of Foundations of Computer Science

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

Stavros Konstantinidis, Nelma Moreira, Rogério Reis, Joshua Young

Mathematics and Computing Science, Saint Mary's University,
923 Robie Str., Halifax, Nova Scotia BH 3C3, Canada
s.konstantinidis@smu.ca, jyo04@hotmail.com

CMUP & DCC, Faculdade de Ciências da Universidade do Porto,
Rua do Campo Alegre 4169-007 Porto, Portugal
nam@dcc.fc.up.pt, rvr@dcc.fc.up.pt

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

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 transducers with set specs we obtain further algorithms that can be applied to questions about independent regular languages as well as a decision question about synchronous transducers.

Keywords: Alphabet-invariant transducers; regular expressions; automata; algorithms; monoids.

1. Introduction

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

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

*Research supported by NSERC (Canada) and by FCT project UID/MAT/00144/2013 (Portugal).
Due to page restrictions, results on *partial derivatives* will be treated in a companion publication.

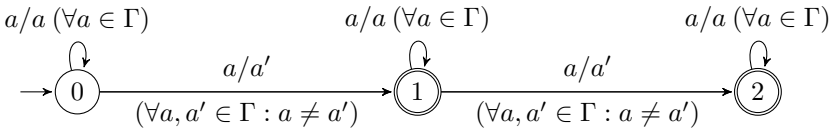

Fig. 1: The transducer realizes the relation of all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2.

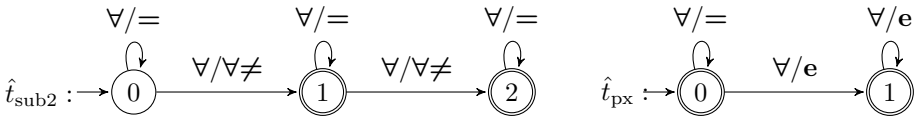

Fig. 2: Transducers over pairing specs. The left one realizes all (u, v) such that $u \neq v$ and the Hamming distance of u, v is at most 2. Transducer $\hat{t}_{\text{px}}$ realizes the relation of all (u, v) such that v is a prefix of u.

where n is variable, and the 2D regular expression

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

where $\mathbf{e}$ 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 $\hat{t}_{\text{sub2}}$ in Fig. 2. In that figure, the label $(\forall/=)$ represents the set $\{(a, a) \mid a \in \Gamma\}$ and the label $(\forall/\forall\neq)$ represents the set $\{(a, a') \mid a, a' \in \Gamma, a \neq a'\}$ (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

$$(\forall/=)^*(\forall/\mathbf{e})^*.$$

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.

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 a *label set*, where each special label $\beta \in B$ is simply a nonempty 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 or the string representation of M's neutral element. 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 associated monoid M. Then, we consider further results specific to the cases of (i) 1D regular expressions and automata (monoid $M = \Gamma^*$) with labels called *set specs*, (ii) 2D regular expressions and transducers (monoid $M = \Gamma^* \times \Gamma^*$) with special labels (called *pairing specs*).

A labelled graph in this work can be considered to be a syntactic version of an automaton over the monoid M in the sense of [12]. Thus, we make no attempt to define regular expressions and automata outside of monoids; rather we use monoid-based regular expressions and automata as a foundation such that (i) one can define such objects with alphabet invariant labels or with a priori unknown label sets B, as long as each of the labels represents a subset of a known monoid; (ii) many known algorithms and constructions on monoid-based regular expressions and automata are extended to work directly and as efficiently on certain type B objects.

We also mention the framework of symbolic automata and transducers of [16, 15]. 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 [16, 15] 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 set specs work well with nondeterministic automata and transducers, this might not be true when dealing with deterministic ones.

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 whose elements represent subsets of Γ, 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 $\operatorname{mon} B$; that is, $\mathcal{I}(\beta)$ is a subset of $\operatorname{mon} B$ 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. Section 8 defines regular expressions $\boldsymbol{r}$ over any label set B and their behaviour $\mathcal{I}(\boldsymbol{r})$, and establishes the equivalence of type B graphs and type B regular expressions (see Theorem 34 and Corollary 36). Section 9 considers the possibility of defining 'higher level' versions of existing product constructions 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 label sets B, B', returning an element of some label set C, and also partially defined on the elements of the monoids $\operatorname{mon} B$ and $\operatorname{mon} B'$, returning an element of the monoid $\operatorname{mon} C$. In this case, if $\odot$ is known to work on automata/transducers over $\operatorname{mon} B, \operatorname{mon} B'$ then it would also work on type B, B' graphs (see Theorem 44). Section 10 presents some basic algorithms on automata with set specs and transducers with set specs. Section 11 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 56). Section 12 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 58 and Theorem 60). Section 13 shows that transducers with set specs (i) can be processed directly (without expanding them) and efficiently to answer the witness version of the property satisfaction question for regular languages (see Corollary 64 and Example 65) and (ii) can be used to decide efficiently whether a given synchronous transducer represents a left synchronous relation (Corollary 66). 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 f is a *partial mapping* of a set S into a set T we shall use the notation

$$f : S \dashrightarrow T$$

We shall write $f(s) = \perp$ to *indicate that f is not defined on $s \in S$*. We shall occassionally use the term *operator* for (partial) mappings.

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

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

We shall denote by '$<$' the total order of Ω. As usual we use the term *string* or *word* to refer to any finite sequence of symbols. The *empty string* is denoted by ε. For any string w we say that w is *sorted* if the symbols contained in w occur in the left to right direction according to the total order of Ω. For example, the word 012 is sorted, but 021 is not sorted. For any set of symbols S, we use the notation

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

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

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

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

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

Algorithmic convention about alphabet symbols. We shall consider algorithms on automata and transducers where the alphabets Γ involved are not of fixed size and, therefore, $|\Gamma| \to \infty$; thus, the alphabet size $|\Gamma|$ is accounted for in time complexity estimates. Moreover, we assume that each Γ-symbol is of size $O(1)$. This approach is also used in related literature (e.g., [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 $|V| + |E| \to \infty$, but the size of each vertex is implicitly considered to be $O(1)$, [10]. 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 access only a part of Γ or some information about Γ such as $|\Gamma|$. We assume that Γ has been preprocessed such that the value of $|\Gamma|$ is available and is $O(\log |\Gamma|)$ bits long and the *minimum symbol* $\min \Gamma$ of Γ is also available. In particular, we assume that we have available a *sorted array* ARR_Γ consisting of all Γ-symbols. While this is a convenient assumption, if in fact it is not applicable then one can make the array from Γ in time $O\big(|\Gamma| \log |\Gamma|\big)$. Then, the minimum symbol of Γ is simply $\mathrm{ARR}_\Gamma[0]$.

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

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

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

Relations. Let Σ, Δ be alphabets. A (binary word) *relation* of type $[\Sigma, \Delta]$ is a subset R of $\Sigma^* \times \Delta^*$. Then, $R^{-1} = \{(v, u) : (u, v) \in R\}$.

3. Set Specifications

Here we define expressions, called set specs, that are intended to represent nonempty subsets of the alphabet Γ. 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 three forms*

$$\forall \qquad \exists w \qquad \nexists w$$

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

We shall need the symbol $\mathbf{e}$ to represent the empty string ε. We shall use the abbreviation

$$\mathrm{SSP}_\varepsilon \;=\; \mathrm{SSP} \cup \{\mathbf{e}\}.$$

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

$\mathbf{e} \cap \mathbf{e} = \mathbf{e}, \quad \mathbf{e} \cap F = F \cap \mathbf{e} = \bot$

$\forall \cap F = F \cap \forall = F$

$\exists u \cap \exists v = \exists \,\mathrm{wo}\,\big(\mathrm{alph}\,u \cap \mathrm{alph}\,v\big), \quad \text{if } \big(\mathrm{alph}\,u \cap \mathrm{alph}\,v\big) \neq \emptyset$

$\exists u \cap \exists v = \bot, \quad \text{if } \big(\mathrm{alph}\,u \cap \mathrm{alph}\,v\big) = \emptyset$

$\nexists u \cap \nexists v = \nexists \,\mathrm{wo}\,\big(\mathrm{alph}\,u \cup \mathrm{alph}\,v\big)$

$\exists u \cap \nexists v = \exists \,\mathrm{wo}\,\big(\mathrm{alph}\,u \setminus \mathrm{alph}\,v\big), \quad \text{if } \big(\mathrm{alph}\,u \setminus \mathrm{alph}\,v\big) \neq \emptyset$

$\exists u \cap \nexists v = \bot, \quad \text{if } \big(\mathrm{alph}\,u \setminus \mathrm{alph}\,v\big) = \emptyset$

$\nexists u \cap \exists v = \exists v \cap \nexists u$

Example 2. *As any set spec F is a string, it has a length $|F|$. We have that $|\forall| = 1$ and $|\exists w| = 1 + |w|$. Also,*

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

Lemma 3. *Given any $G, F \in \mathrm{SSP}_\varepsilon$, $G \cap F$ can be computed in time $O(|G| + |F|)$.*

Proof. The required algorithm works as follows. If either of G, F is $\mathbf{e}$ 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 through 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 $G = \nexists u$ and $F = \nexists v$. 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 $\nexists w$. Now suppose that $G = \exists u$ and $F = \nexists v$. 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 4. *Let Γ be an alphabet of reference and let F be a set spec. We say that F* respects Γ*, if either $F = \forall$, or F is of the form $\exists w$ or $\nexists w$ and the following restrictions hold:*

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

The language $\mathcal{L}(F)$ *of F (*with respect *to Γ) is the subset of Γ defined as follows:*

$$\mathcal{L}(\forall) = \Gamma, \qquad \mathcal{L}(\exists w) = \operatorname{alph} w, \qquad \mathcal{L}(\nexists w) = \Gamma \setminus \operatorname{alph} w.$$

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

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

We also define $\mathcal{L}(\mathbf{e}) = \{\varepsilon\}$ *and* $\mathrm{SSP}[\Gamma]_\varepsilon = \mathrm{SSP}[\Gamma] \cup \{\mathbf{e}\}$.

Remark 5. *In the above definition, the requirement $|w| < |\Gamma|$ implies that there is at least one Γ-symbol that does not occur in w. Thus, to represent Γ we must use $\forall$ as opposed to the longer set spec $\exists\,\mathrm{wo}(\Gamma)$.*

Lemma 6. *Let Γ be an alphabet of reference and let $G, F \in \mathrm{SSP}[\Gamma]_\varepsilon$. 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 $(\operatorname{alph} w) \notin \{\emptyset, \Gamma\}$, as $0 < |w| < |\Gamma|$; and if $F = \nexists w$ then again $(\Gamma \setminus \operatorname{alph} w) \notin \{\emptyset, \Gamma\}$. For the second statement, we consider the definition of the operation '$\cap$' as well as the above definition. Clearly the statement holds, if $G = F = \mathbf{e}$, or if one of G, F is $\forall$ and the other one is not $\mathbf{e}$. 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 $\mathcal{L}(F) = \Gamma \setminus \operatorname{alph} v$, so $\mathcal{L}(G) \cap \mathcal{L}(F) = \operatorname{alph} u \setminus \operatorname{alph} v$, 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 4. □

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

Lemma 7. *Let Γ be an alphabet of reference and let F 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 $|w|_g > 0$", "$F = \nexists w$ 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 "$F = \exists g$", "$F = \nexists w$ and $|w| = |\Gamma| - 1$ and $|w|_g = 0$". For the third statement, we note that the condition to test is equivalent to one of $|\Gamma| \geq k$, $|w| \geq k$, $|\Gamma| - |w| \geq k$, depending on whether F is one of $\forall, \exists w, \nexists w$. The last case requires time $O(|F|)$ to compute $|w|$ and then $O(\log |\Gamma| + \log |w|)$ time for arithmetic operations, which is $O(\log |\Gamma|)$ as $|w| < |\Gamma|$. For the fourth statement, we note that $|\mathcal{L}(F)| = 1$ is equivalent to whether "$F = \exists g$ and $|g| = 1$" or "$F = \nexists w$ and $|w| = |\Gamma| - 1$". In the former case, the algorithm returns g. In the latter case, we use the algorithm $\mathrm{notIn}(w)$ to get the desired symbol in $\Gamma \setminus \mathrm{alph}\, w$. The latter case is the worse of the two, and works in time $O(|F| + \log |\Gamma|)$ to compute $|w|$ and test whether $|w| = |\Gamma| - 1$, plus time $O(|F|)$ to execute $\mathrm{notIn}(w)$ (see the bound in (1)). The total time is $O(|F|)$, as $|F| = |\Gamma|$. For the fifth statement, if $F = \forall$ or $F = \exists w$ the algorithm simply returns $\mathrm{ARR}_\Gamma[0]$ or $w[0]$, respectively. The worst case is when $F = \nexists w$, where, as before, the algorithm uses $\mathrm{notIn}(w)$ requiring time $O(|F|)$. For the sixth statement, the algorithm first finds any $g_1 \in \mathcal{L}(F)$, then computes the set spec $B = F \cap \nexists g_1$ 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 $\big((\Gamma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})\big) \setminus \{(\varepsilon, \varepsilon)\}$. First, we define their syntax and then their semantics.

Definition 8. *A* pairing specification*, or* pairing spec *for short, is a string of the form*

$$\mathbf{e}/G \qquad F/\mathbf{e} \qquad F/G \qquad F/= \qquad F/G{\neq} \tag{2}$$

where F, G are set specs. The set of pairing specs is denoted by PSP.

The *inverse* p^{-1} of a pairing spec p is defined as follows depending on the possible forms of p displayed in (2):

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

We also define $(\mathbf{e}/\mathbf{e})^{-1} = (\mathbf{e}/\mathbf{e})$.

Example 9. *As a pairing spec* p *is a string, it has a length* $|\mathsf{p}|$*. We have that* $|\forall/\mathbf{e}| = 3$ *and* $|\exists u/\nexists v| = 3 + |u| + |v|$*. Also,* $(\forall/\mathbf{e})^{-1} = (\mathbf{e}/\forall)$ *and* $(\exists u/\forall{\neq})^{-1} = (\forall/\exists u{\neq})$*.*

Definition 10. *A pairing spec is called* alphabet invariant *if it contains no set spec of the form* $\exists w, \nexists w$*. The set of alphabet invariant pairing specs is denoted by* $\mathrm{PSP}^{\mathrm{invar}}$*.*

Definition 11. *Let* Γ *be an alphabet of reference and let* p *be a pairing spec. We say that* p respects Γ*, if any set spec occurring in* p *respects* Γ*. The set of pairing specs that respect* Γ *is denoted as follows*

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

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

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

We also define $\mathcal{R}(\mathbf{e}/\mathbf{e}) = \{(\varepsilon, \varepsilon)\}$ *and* $\mathrm{PSP}[\Gamma]_\varepsilon = \mathrm{PSP}[\Gamma] \cup \{\mathbf{e}/\mathbf{e}\}$*.*

Remark 12. *All the alphabet invariant pairing specs are*

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

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

Lemma 13. *Let* $\mathsf{p} \in \mathrm{PSP}[\Gamma]$*. The following statements hold true.*

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

Proof. The first statement follows from Definition 11 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 8 and 11. □

Some notation on pairing specs. Let $\mathsf{p} \in \mathrm{PSP}[\Gamma]_\varepsilon$. The *left* part, $\mathrm{left}\,\mathsf{p}$, of p is the string on the left of the symbol '/', and the *right* part, $\mathrm{right}\,\mathsf{p}$, of p is the string on the right of '/'. We have the following examples:

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

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

$$\mathrm{rset}(\mathbf{e}/G) = G, \quad \mathrm{rset}(F/\mathbf{e}) = \mathbf{e}, \qquad \mathrm{rset}(F/G) = G,$$
$$\mathrm{rset}(F/=) = F, \quad \mathrm{rset}(F/G\neq) = G, \quad \mathrm{rset}(\mathbf{e}/\mathbf{e}) = \mathbf{e}.$$

The above notation implies

$$\mathcal{R}(\mathsf{p}) \subseteq \mathcal{L}(\mathrm{left}\,\mathsf{p}) \times \mathcal{L}(\mathrm{rset}\,\mathsf{p}). \tag{3}$$

Lemma 14. *If* $\mathsf{p} \in \mathrm{PSP}[\Gamma]$ *then* $|\mathsf{p}| \leq 2|\Gamma| + 2$.

Proof. Follows from Lemma 6. □

5. Label Sets and their Monoid 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 15. *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* $\mathbf{e}$*:* $\underline{\varepsilon} = \mathbf{e}$*.*

(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,* $\underline{(\varepsilon, \varepsilon)} = \mathbf{e}/\mathbf{e}$*.*

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

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

where M is a monoid. Thus, the behaviour $\mathcal{I}(\beta)$ of a label $\beta \in B$ is a subset of M. We shall consider label sets B with *fixed behaviours*, so we shall denote by $\operatorname{mon} B$ 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 $\operatorname{mon} B_1 = \operatorname{mon} B_2$ 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 $\operatorname{mon} B_1 = \operatorname{mon} B_2$. This convention is applied in the example below: we use $\mathcal{L}$ for the behaviour of both the label sets Σ and $\mathrm{SSP}[\Gamma]$.

Example 16. *We shall use some of the following label sets and their fixed label behaviours.*

(1) Σ *with behaviour* $\mathcal{L} : \Sigma \to 2^{\Sigma^*}$ *such that* $\mathcal{L}(g) = \{g\}$*, for* $g \in \Sigma$*. Thus,* $\operatorname{mon} \Sigma = \Sigma^*$*.*

(2) $\mathrm{SSP}[\Gamma]$ *with behaviour* $\mathcal{L} : \mathrm{SSP}[\Gamma] \to 2^{\Gamma^*}$*, as specified in Def. 4. Thus,* $\operatorname{mon} \mathrm{SSP}[\Gamma] = \Gamma^*$*.*

(3) $\mathrm{REG}\,\Sigma$ *= all regular expressions over* Σ *with behaviour* $\mathcal{L} : \mathrm{REG}\,\Sigma \to 2^{\Sigma^*}$ *such that* $\mathcal{L}(\boldsymbol{r})$ *is the language of the regular expression* $\boldsymbol{r}$*. Thus,* $\operatorname{mon} \mathrm{REG}\,\Sigma = \Sigma^*$*.*

(4) $[\Sigma, \Delta] = \{x/y \mid x \in \Sigma \cup \{\mathbf{e}\}, y \in \Delta \cup \{\mathbf{e}\}\} \setminus \{\mathbf{e}/\mathbf{e}\}$ *with behaviour*

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

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

(5) $\mathrm{PSP}[\Gamma]$ *with behaviour* $\mathcal{R} : \mathrm{PSP}[\Gamma] \to 2^{\Gamma^* \times \Gamma^*}$ *as specified in Def. 11. Thus,* $\operatorname{mon} \mathrm{PSP}[\Gamma] = \Gamma^* \times \Gamma^*$*.*

(6) $\mathrm{PSP}^{\mathrm{invar}}$ *with behaviour* $\mathcal{R}_\perp : \mathrm{PSP}^{\mathrm{invar}} \to \{\emptyset\}$*. Thus,* $\mathcal{I}(\beta) = \emptyset$*, for any* $\beta \in \mathrm{PSP}^{\mathrm{invar}}$*.*

(7) 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*

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

(8) $[\mathrm{REG}\,\Sigma, \mathrm{REG}\,\Delta]$ *with behaviour* $\mathcal{R}$ *in the monoid* $\Sigma^* \times \Delta^*$ *such that* $\mathcal{R}(\boldsymbol{r}/\boldsymbol{s}) = \mathcal{L}(\boldsymbol{r}) \times \mathcal{L}(\boldsymbol{s})$*, for any* $\boldsymbol{r} \in \mathrm{REG}\,\Sigma$ *and* $\boldsymbol{s} \in \mathrm{REG}\,\Delta$*.*

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

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

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

Abbreviation. Let B be any label set. We shall see further below that type B regular expressions and graphs are constructed using labels in $B \cup \{\underline{\varepsilon_{\mathrm{mon}\,B}}\}$. Thus, we define

$$B_\varepsilon \;=\; B \cup \{\underline{\varepsilon_{\mathrm{mon}\,B}}\}.$$

The term *label shall mean an element of* B_ε (unless we specify a label *in* B).

Thus, $\Sigma_\varepsilon = \Sigma \cup \{\mathbf{e}\}$ and $\mathrm{SSP}[\Gamma]_\varepsilon = \mathrm{SSP}[\Gamma] \cup \{\mathbf{e}\}$ and $[\Sigma, \Delta]_\varepsilon = [\Sigma, \Delta] \cup \{\mathbf{e}/\mathbf{e}\}$.

Remark 17. *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 monoid* $\mathrm{mon}\,X$*.*

6. Labelled Graphs, Automata, Transducers

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

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

such that

- Q is a nonempty set whose elements are called *states*;
- $I \subseteq Q$ is the nonempty set of initial, or start states;
- $F \subseteq Q$ is the set of final states;
- δ is a set, called the set of *edges* or *transitions*, consisting of triples (p, β, q) such that $p, q \in Q$ and $\beta \in B \cup \{\underline{\varepsilon_{\mathrm{mon}\,B}}\}$. The set of *labels of* $\hat{g}$ is the set $\mathrm{Labels}(\hat{g}) = \{\beta \mid (p, \beta, q) \in \delta\}$.

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

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

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

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

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

Remark 20. *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* $\operatorname{REG}\Sigma$.

Lemma 21. *For each type B graph $\hat{g} = (Q, B, \delta, I, F)$, we have that*

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

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 $\delta_{\exp}$, for each $i = 1, \ldots, \ell$, 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 $\hat{g}^\varepsilon$. 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 $\hat{g}^\varepsilon$ contains transitions $(q_{i-1}, \underline{\varepsilon_{\mathrm{mon}\, B}}, q_i)$ with $q_{i-1} = q_i$, then these transitions can be omitted resulting into an accepting path of $\hat{g}$. □

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 $\mathbf{e}$ and $\mathbf{e}/\mathbf{e}$, which are of fixed length. Thus, we shall assume that $\underline{\varepsilon_{\mathrm{mon}\, B}} = O(1)$, for any label set B. This implies that

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

Definition 22. *Let Σ, Δ, Γ be alphabets.*

(1) An ordinary automaton *(over Σ) is a labelled graph $\hat{a} = (Q, \Sigma, \delta, I, F)$. The* language $\mathcal{L}(\hat{a})$ *accepted by $\hat{a}$ is the behaviour of $\hat{a}$ with respect to the label set Σ. An* automaton*, or ε-*NFA*, (over Σ) is a labelled graph $\hat{a} = (Q, B, \delta, I, F)$ such that $\Sigma \subseteq B$, $\mathrm{mon}\, B = \mathrm{mon}\, \Sigma = \Sigma^*$, and $\mathrm{Labels}(\hat{a}) \subseteq \Sigma \cup \{\mathbf{e}\}$. Thus, $\mathcal{L}(\sigma) = \{\sigma\}$ for $\sigma \in \Sigma$, and $\mathcal{L}(\beta) \subseteq \Sigma^*$ for $\beta \in B$. If $\mathrm{Labels}(\hat{a}) \subseteq \Sigma$ then $\hat{a}$ is called an* NFA*. If in $\hat{a}$ we replace B with Σ we get an ordinary automaton that is otherwise identical to $\hat{a}$.*

(2) An automaton with set specs *is a labelled graph $\hat{b} = (Q, \mathrm{SSP}[\Gamma], \delta, I, F)$. The* language $\mathcal{L}(\hat{b})$ *accepted by $\hat{b}$ is the behaviour of $\hat{b}$ with respect to the label set $\mathrm{SSP}[\Gamma]$.*

(3) An ordinary transducer *(over Σ, Δ) is a labelled graph $\hat{t} = (Q, [\Sigma, \Delta], \delta, I, F)$. The* relation $\mathcal{R}(\hat{t})$ *realized by $\hat{t}$ is the behaviour of $\hat{t}$ with respect to the label set $[\Sigma, \Delta]$. A* transducer *(over Σ, Δ) is a labelled graph $\hat{t} = (Q, C, \delta, I, F)$ such that $[\Sigma, \Delta] \subseteq C$, $\mathrm{mon}\, C = \mathrm{mon}[\Sigma, \Delta] = \Sigma^* \times \Delta^*$, and $\mathrm{Labels}(\hat{t}) \subseteq [\Sigma, \Delta] \cup \{\mathbf{e}/\mathbf{e}\}$. Thus, $\mathcal{R}(x/y) = \{(x, y)\}$ for $x/y \in [\Sigma, \Delta]$, and $\mathcal{R}(\beta) \subseteq \Sigma^* \times \Delta^*$ for $\beta \in C$. If in $\hat{t}$ we replace C with $[\Sigma, \Delta]$ we get an ordinary transducer that is otherwise identical to $\hat{t}$.*

(4) A transducer with set specs *is a labelled graph $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$; that is, $\hat{s}$ is a type $\mathrm{PSP}[\Gamma]$ graph. The* relation $\mathcal{R}(\hat{s})$ realized *by $\hat{s}$ is the behaviour of $\hat{s}$ with respect to the label set $\mathrm{PSP}[\Gamma]$.*

(5) An alphabet invariant transducer *is a labelled graph $\hat{i} = (Q, \mathrm{PSP}^{\mathrm{invar}}, \delta, I, F)$. If Γ is an alphabet then the* Γ-version *of $\hat{i}$ is the transducer with set specs $\hat{i}[\Gamma] = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$.*

The above definitions of ordinary 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 $\mathbf{e}$. 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.

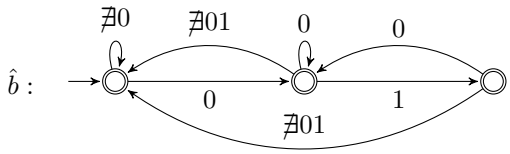

Fig. 3: Automaton with set specs accepting all strings over $\Gamma = \{0, \ldots, n-1\}$ that do not contain 011.

The terms automaton and transducer (as opposed to the ordinary ones) allow for more flexibility in the use of label sets as components of these objects.

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 23. *The size of an alphabet invariant transducer $\hat{\imath}$ is of the same order of magnitude as $|Q| + |\delta|$.*

Lemma 24. *If $\hat{b}$ is an automaton with set specs then $\exp \hat{b}$ is an automaton. If $\hat{s}$ is a transducer with set specs then $\exp \hat{s}$ is a transducer (in standard form).*

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

Example 25. *The transducers shown in Fig. 2 are alphabet invariant. Both transducers are much more succinct compared to their expanded Γ-versions, as $|\Gamma| \to \infty$:*

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

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

Following [18], if $\hat{t} = (Q, [\Sigma, \Delta], \delta, I, F)$ is an ordinary transducer then $\hat{t}^{-1}$ is the ordinary transducer $(Q, [\Delta, \Sigma], \delta', I, F)$, where $\delta' = \{(p, y/x, q) \mid (p, x/y, q) \in \delta\}$, such that

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

Lemma 26. *For each transducer $\hat{s}$ with set specs we have that*

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

Proof. The first identity follows from two facts: (i) $\exp(\hat{s}^{-1})$ has transitions of the form $(p, y/x, q)$, where $y/x \in \mathcal{R}(\mathsf{p}^{-1})$ and (p, 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 $x/y \in \mathcal{R}(\mathsf{p})$ and (p, p, q) is a transition in $\hat{s}$. The second identity follows from (4) and Definition 19. □

Remark 27. *Let $\hat{t} = (Q, \Gamma, \delta, I, F)$ be a transducer with set specs. By Lemma 14, we have that*

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

7. Rational Operations

The three standard *rational operations* (union, catenation, star) on ordinary 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

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

The graph $\hat{g} \cup \hat{g}'$ of type $C = B \cup B'$ 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 $(s, \underline{\varepsilon_{\operatorname{mon} B}}, p)$, for all $p \in I \cup I'$.

The graph $\hat{g} \cdot \hat{g}'$ of type $C = B \cup B'$ 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 $(f, \underline{\varepsilon_{\operatorname{mon} B}}, q)$, for all $f \in F$, and E' is the set of transitions $(q, \underline{\varepsilon_{\operatorname{mon} B}}, i')$, for all $i' \in I'$.

The graph $\hat{g}^*$ of type B is defined as follows

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

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

Lemma 28. *Let $\hat{g} = (Q, B, \delta, I, F)$ and $\hat{g}' = (Q', B', \delta', I', F')$ be trim labelled graphs such that* $\operatorname{mon} B = \operatorname{mon} B'$.

(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

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

But then, one works at the level of the monoid $\operatorname{mon} B$ and the proofs are essentially the same as the ones for the case of ordinary automata (see e.g. [17]).

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 [12]).

Definition 29. *Let B be a label set with behaviour $\mathcal{I}$ such that no $\beta \in B$ contains the special symbol $\oslash$. The set* $\operatorname{REG} B$ *of* type B regular expressions *is the set of strings consisting of the 1-symbol string $\oslash$ and the strings in the set Z that is defined inductively as follows.*

- $\underline{\varepsilon_{\operatorname{mon} B}}$ *is in Z.*
- *Every $\beta \in B$ is in Z.*
- *If $\boldsymbol{r}, \boldsymbol{s} \in Z$ then $(\boldsymbol{r} + \boldsymbol{s}), (\boldsymbol{r}\boldsymbol{s}), (\boldsymbol{r}^*)$ are in Z.*

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

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

Remark 30. *The above definition implies that $\oslash$ occurs in any $\boldsymbol{r} \in \operatorname{REG} B$ if and only if $\boldsymbol{r} = \oslash$.*

Example 31. *Let Σ, Δ be alphabets. Using Σ as a label set, we have that* $\operatorname{REG} \Sigma$ *is the set of ordinary regular expressions over Σ. For the label set $[\Sigma, \Delta]$, we have that* $\operatorname{REG}[\Sigma, \Delta]$ *is the set of rational expressions over $\Sigma^* \times \Delta^*$ in the sense of* [12].

Example 32. *Let $\Gamma = \{0, 1, \ldots, n-1\}$. In type* $\operatorname{SSP}[\Gamma]$ *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 is $0 + \cdots + (n-1)$, which is of size $O(n)$. Similarly, the following regular expression over* $\operatorname{PSP}[\Gamma]$

$$(\forall/=)^* \; (\forall/\forall\neq) \; (\forall/=)^* \tag{5}$$

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 $[\Gamma, \Gamma]$ is

$$\big(0/0 + \cdots + (n-1)/(n-1)\big)^* \; \big(\boldsymbol{r}_0 + \cdots + \boldsymbol{r}_{n-1}\big) \; \big(0/0 + \cdots + (n-1)/(n-1)\big)^*$$

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

Example 33. *Consider the UNIX utility* `tr`*. For any strings* u, v *of length* $\ell > 0$*, the command*[a]

`tr` u v

can be 'simulated' by the following regular expression of type PSP[ASCII]

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

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

`tr` $-$`d` u

can be 'simulated' by the following regular expression of type PSP[ASCII]

$$\big(\exists u/\mathbf{e} + \nexists u/=\big)^*$$

For the command

`tr` $-$`s` u

it seems that any regular expression over PSP[ASCII] *cannot be of size* $O(\ell^2)$.

The Thompson method, [14], of converting an ordinary regular expression over Σ—a type Σ regular expression in the present terminology—to an ordinary automaton can be extended without complications to work with type B regular expressions, for any label set B, using Lemma 28.

Theorem 34. *Let* B *be a label set with behaviour* $\mathcal{I}$*. For each type* B *regular expression* $\boldsymbol{r}$*, there is a type* B *graph* $\hat{g}(\boldsymbol{r})$ *such that*

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

For the converse of the above theorem, we shall extend the state elimination method of automata, [4], to labelled graphs.

Let $\hat{g} = \big(Q, B, \delta, \{s\}, \{f\}\big)$ 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 $\boldsymbol{r}_{p,r} = \beta_1 + \cdots + \beta_m$, 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

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

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

[a]The command "`tr` u v" takes as input a file F and outputs its characters except that each character $u[i]$ occurring in F is outputted as $v[i]$. The command "`tr` $-$`d` u" outputs the characters in the input file omitting those occurring in u. The command "`tr` $-$`s` u" outputs the characters in the input file suppressing any (consecutive) repetitions of each character $u[i]$.

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

Lemma 35. *Let $\hat{g} = \big(Q, B, \delta, \{s\}, \{f\}\big)$ 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 [17]. 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 $\ell \in \mathbb{N}$, $m = m_1 \cdots m_\ell$ 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

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

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

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

is a transition in δ'—see (6). Moreover, $m_b \cdots m_{b+r} \in \mathcal{I}(\beta_b)\mathcal{I}(\boldsymbol{r}_{q,q}^*)\mathcal{I}(\beta_{b+r})$. 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 $\ell \geq 1$, $m = m_1 \cdots m_\ell$ and each $m_i \in \mathcal{I}(\beta_i)$. For each $i \in \{1, \ldots, \ell\}$, 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

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

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

$$m_i = k_i'k_{i,1}\cdots k_{i,r}k_i'', \;\; k_i' \in \mathcal{I}(\alpha_1), \;\; k_i'' \in \mathcal{I}(\alpha_2), \;\; k_{i,1}, \ldots, k_{i,r} \in \mathcal{I}(\boldsymbol{r}_{q,q}).$$

Then, the path P_i is

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

which has r repetitions of $(q, \boldsymbol{r}_{q,q}, q)$. 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 $\mathcal{I}(\hat{h}) = \mathcal{I}(\boldsymbol{r}_{s,f})$. Thus, we have the following consequence of Lemma 35.

Corollary 36. *Let B be a label set with behaviour $\mathcal{I}$. For each type B graph $\hat{g}$ there is a type B regular expression $\boldsymbol{r}$ such that $\mathcal{I}(\hat{g}) = \mathcal{I}(\boldsymbol{r})$.*

9. 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 $\operatorname{mon} B, \operatorname{mon} B'$ such that, when defined, the product $m \odot m'$ of two monoid elements belongs to $\operatorname{mon} C$. We shall call $\odot$ a *polymorphic* operation (in analogy to polymorphic operations in programming languages) when $\mathcal{I}(\beta \odot \beta') = \mathcal{I}_1(\beta) \odot \mathcal{I}_2(\beta')$ 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.

Below, the outcome of a label operation $\beta \odot \beta'$ could be $\perp$ (undefined). For convenience we shall write $\mathcal{I}(\perp) = \emptyset$. For any $S \subseteq \operatorname{mon} B$ and $S' \subseteq \operatorname{mon} B'$, we shall use the notation

$$S \odot S' = \{m \odot m' \mid m \in S, m' \in S', m \odot m' \neq \perp\}.$$

Example 37. *We shall consider the following partial monoid operations, which are better known when applied to subsets of the monoid.*

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

$$R \downarrow L = R \cap (L \times \Delta^*). \tag{7}$$

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

$$R \uparrow L = R \cap (\Sigma^* \times L). \tag{8}$$

Definition 38. *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:* $\odot : B_\varepsilon \times B'_\varepsilon \dashrightarrow C_\varepsilon$
- *It is a partial mapping:* $\odot : \operatorname{mon} B \times \operatorname{mon} B' \dashrightarrow \operatorname{mon} C$*.*
- *For all* $\beta \in B_\varepsilon$ *and* $\beta' \in B'_\varepsilon$ *we have*

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

Example 39. *The following polymorphic operations are based on label sets of ordinary automata and transducers using the monoid operations in Ex. 37.*

- *"*$\cap : \Sigma \times \Sigma \Rightarrow \Sigma$*" is defined by*

  - *the label operation* $\cap : \Sigma_\varepsilon \times \Sigma_\varepsilon \dashrightarrow \Sigma_\varepsilon$ *such that* $x \cap y = x$, *if* $x = y$, *else* $x \cap y = \bot$; *and*
  - *the monoid operation* $\cap : \Sigma^* \times \Sigma^* \dashrightarrow \Sigma^*$.

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

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

  Obviously, $\mathcal{R}((x/y) \downarrow z) = \mathcal{R}(x/y) \downarrow \mathcal{L}(z)$.
- "$\uparrow : [\Sigma, \Delta] \times \Delta \Rightarrow [\Sigma, \Delta]$" *is defined by*
  - *the label operation* $\uparrow : [\Sigma, \Delta]_\varepsilon \times \Delta_\varepsilon \dashrightarrow [\Sigma, \Delta]_\varepsilon$ *such that* $(x/y) \uparrow z = (x/y)$ *if* $y = z$, *else* $(x/y) \uparrow z = \bot$; *and*
  - *the monoid 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 40. *The following polymorphic operations are based on label sets of automata and transducers with set specs.*

- "$\cap : \mathrm{SSP}[\Gamma] \times \mathrm{SSP}[\Gamma] \Rightarrow \mathrm{SSP}[\Gamma]$" *is defined by the monoid operation* $\cap : \Gamma^* \times \Gamma^* \dashrightarrow \Gamma^*$, *and the label operation* $\cap : \mathrm{SSP}[\Gamma]_\varepsilon \times \mathrm{SSP}[\Gamma]_\varepsilon \dashrightarrow \mathrm{SSP}[\Gamma]_\varepsilon$ *defined in Section 3. By Lemma 6 and having* $\mathcal{L}(\bot) = \emptyset$, *for any* $\alpha, \beta \in \mathrm{SSP}[\Gamma]_\varepsilon$, *we have that*

$$\mathcal{L}(\alpha \cap \beta) = \mathcal{L}(\alpha) \cap \mathcal{L}(\beta).$$

- "$\downarrow : \mathrm{PSP}[\Gamma] \times \Gamma \Rightarrow \mathrm{PSP}[\Gamma]$" *is defined by the monoid operation* $\downarrow : (\Sigma^* \times \Delta^*) \times \Sigma^* \dashrightarrow (\Sigma^* \times \Delta^*)$ *and by the label operation* $\downarrow : \mathrm{PSP}[\Gamma]_\varepsilon \times \Gamma_\varepsilon \dashrightarrow \mathrm{PSP}[\Gamma]_\varepsilon$ *such that* $\mathbf{e}/\mathbf{e} \downarrow \mathbf{e} = \mathbf{e}/\mathbf{e}$ *and* $\mathbf{e}/\mathbf{e} \downarrow g = \bot$ *for* $g \in \Gamma$, *and for* $\mathsf{p} \in \mathrm{PSP}[\Gamma]$ *and* $x \in \Gamma_\varepsilon$

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

  Assuming $\beta \in \mathrm{PSP}[\Gamma]_\varepsilon$ *and having* $\mathcal{R}(\bot) = \emptyset$, *we have that*

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

  Moreover we have that $\beta \downarrow x$ *can be computed from* β *and* x *in time* $O(|\beta|)$.

- *"*$\uparrow\colon \mathrm{PSP}[\Gamma]\times\Delta \Rightarrow \mathrm{PSP}[\Gamma]$*" is defined by the monoid operation* $\uparrow\colon (\Sigma^*\times\Delta^*)\times\Delta^* \dashrightarrow (\Sigma^*\times\Delta^*)$*, and by the label operation* $\uparrow\colon \mathrm{PSP}[\Gamma]_\varepsilon \times \Delta_\varepsilon \dashrightarrow \mathrm{PSP}[\Gamma]_\varepsilon$ *such that* $\mathbf{e}/\mathbf{e}\uparrow\mathbf{e} = \mathbf{e}/\mathbf{e}$ *and* $\mathbf{e}/\mathbf{e}\uparrow g = \bot$ *for* $g\in\Gamma$*, and* $\mathsf{p}\uparrow x = (\mathsf{p}^{-1}\downarrow x)^{-1}$ *for* $\mathsf{p}\in\mathrm{PSP}[\Gamma]$ *and* $x\in\Gamma_\varepsilon$*. Assuming* $\beta\in\mathrm{PSP}[\Gamma]_\varepsilon$*, we have that*

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

Moreover we have that $\beta\uparrow x$ *can be computed from* β *and* x *in time* $O(|\beta|)$*.*

In Sect. 11, we define the polymorphic operation '$\circ$*' between pairing specs.*

Definition 41. *Let* $\hat{g} = (Q, B, \delta, I, F)$ *and* $\hat{g}' = (Q', B', \delta', I', F')$ *be type* B *and* B'*, respectively, graphs and let "*$\odot : B\times B' \Rightarrow C$*" be a polymorphic operation. The* product $\hat{g}\odot\hat{g}'$ *is the type* C *graph*

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

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

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

Example 42. *Here we recall a few known examples of product constructions involving automata and transducers.*

(1) For two automata $\hat{a},\hat{a}'$*, the automaton* $\hat{a}\cap\hat{a}'$ *is 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 $\hat{t},\hat{t}'$*, the transducer* $\hat{t}\circ\hat{t}'$ *is 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* $\hat{a}$*, the transducer* $\hat{t}\downarrow\hat{a}$ *is such that*

$$\mathcal{R}(\hat{t}\downarrow\hat{a}) = \mathcal{R}(\hat{t})\downarrow\mathcal{L}(\hat{a}).$$

Similarly, the transducer $\hat{t}\uparrow\hat{a}$ *is such that*

$$\mathcal{R}(\hat{t}\uparrow\hat{a}) = \mathcal{R}(\hat{t})\uparrow\mathcal{L}(\hat{a}).$$

These product constructions were used in [8] *to answer algorithmic questions about independent languages—see Sect. 13.*

Lemma 43. *The following statements hold true about the product graph* $\hat{g}\odot\hat{g}' = (P, C, \delta\odot\delta', I\times I', F\times F')$ *of two trim labelled graphs* $\hat{g},\hat{g}'$ *as defined in Def. 41.*

(1) $|P| = O(|\delta||\delta'|)$ *and* $|\delta \odot \delta'| \leq |\delta||\delta'|$.

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

Proof. As $P \subseteq Q \times Q'$, Lemma 18 implies that $|P| \leq (2|\delta| + 1)(2|\delta'| + 1)$, so $|P| = O(|\delta||\delta'|)$. As we get at most one transition in $\delta \odot \delta'$ for each pair of transitions in δ and δ', we have that $|\delta \odot \delta'| \leq |\delta||\delta'|$. For the second statement, we have that $\delta \odot \delta'$ 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 $\beta \odot \beta'$ 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 44. *If "*$\odot : B \times B' \Rightarrow C$*" is a polymorphic operation and* $\hat{g}, \hat{g}'$ *are type* B, B'*, respectively, graphs then* $\hat{g} \odot \hat{g}'$ *is a type* C *graph such that*

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

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 $\beta \in B_\varepsilon$ and $m \in \mathcal{I}_1(\beta)$; 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 $\beta' \in B'_\varepsilon$ and $m' \in \mathcal{I}_2(\beta')$; where we used $\mathcal{I}_1, \mathcal{I}_2$ for the behaviours of B, B'. Also, if $\beta \odot \beta' \neq \bot$ and $m \odot m' \neq \bot$ then

$\big((p, p'), \beta \odot \beta', (q, q')\big)$ is a transition of $\hat{g} \odot \hat{g}'$ and
$\big((p, p'), \underline{m \odot m'}, (q, q')\big)$ 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 $i = 1, \ldots, \ell$, 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 $i = 1, \ldots, \ell$, 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 $\hat{g}'$. 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, $m \in \mathcal{I}(\exp \hat{g} \odot \exp \hat{g}')$. □

How to apply the above theorem. Suppose that we have a known product construction $\odot$ on labelled graphs $\hat{u}, \hat{u}'$ over monoids M, M' (see Ex. 42), where $\mathcal{I}(\hat{u} \odot \hat{u}') = \mathcal{I}(\hat{u}) \odot \mathcal{I}(\hat{u}')$. We can apply a 'higher level' version of $\odot$ on labelled graphs $\hat{g}, \hat{g}'$ of some types B, B' with behaviours in the monoids M, M'. This would avoid expanding $\hat{g}$ and $\hat{g}'$ to $\hat{u}$ and $\hat{u}'$, and the theorem establishes that the behaviour of $\hat{g} \odot \hat{g}'$ is correct, that is, $\mathcal{I}(\hat{g} \odot \hat{g}') = \mathcal{I}(\hat{u} \odot \hat{u}')$. We apply the theorem in Lemma 46.2, in Theorem 56, in Corollary 63, and in Corollary 66.

10. 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 13).

Remark 45. *For every ordinary automaton $\hat{a} = (Q, \Gamma, \delta, I, F)$, one can make in linear time an automaton with set specs $\hat{a}' = (Q, \mathrm{SSP}[\Gamma], \delta', I, F)$ such that, δ' consists of all transitions $(p, \mathbf{e}, q) \in \delta$ (if any) union all transitions $(p, \exists g, q)$ where $(p, g, q) \in \delta$ and $g \in \Gamma$.*

Lemma 46. *Consider a string w and two automata with set specs*

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

(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 7. The algorithm also constructs a labelled 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 43 implies that $\hat{b} \cap \hat{b}'$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Now we have

$$\begin{aligned} \mathcal{L}(\hat{b} \cap \hat{b}') &= \mathcal{L}(\exp \hat{b} \cap \exp \hat{b}') && (9)\\ &= \mathcal{L}(\exp \hat{b}) \cap \mathcal{L}(\exp \hat{b}') && (10)\\ &= \mathcal{L}(\hat{b}) \cap \mathcal{L}(\hat{b}') && (11) \end{aligned}$$

Statement (9) follows from the fact that "$\cap$: $\mathrm{SSP}[\Gamma] \times \mathrm{SSP}[\Gamma] \Rightarrow \mathrm{SSP}[\Gamma]$" is a polymorphic operation—see Theorem 44 and Ex. 40. Statement (10) follows from the fact that each $\exp \hat{b}, \exp \hat{b}'$ is an automaton and the operation $\cap$ is well-defined on these objects—see Lemma 24 and Ex. 42.

For the third statement, one makes an automaton with set specs $\hat{b}_w$ accepting $\{w\}$, then computes $\hat{a} = \hat{b}_w \cap \hat{b}$, and then uses $\texttt{emptyP}(\hat{a})$ to get the desired answer. □

Lemma 47. *Let $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ be a trim transducer with set specs, $\hat{a} = (Q', \Gamma, \delta', I', F')$ be a trim automaton and (u, v) be a pair of words.*

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

Proof. The first statement is completely analogous to the first statement of Lemma 46. For the second statement, we compute the product $\hat{s} \downarrow \hat{a}$. As the product $\mathtt{p} \downarrow x$ of two labels can be computed in linear time, Lemma 43 implies that $\hat{s} \downarrow \hat{a}$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Now we have

$$\begin{aligned} \mathcal{R}(\hat{s} \downarrow \hat{a}) &= \mathcal{R}(\exp \hat{s} \downarrow \exp \hat{a}) && (12)\\ &= \mathcal{R}(\exp \hat{s}) \downarrow \mathcal{L}(\exp \hat{a}) && (13)\\ &= \mathcal{R}(\hat{s}) \downarrow \mathcal{L}(\hat{a}) && (14) \end{aligned}$$

Statement (12) follows from the fact that '$\downarrow$: $\mathrm{PSP}[\Gamma]\times\Gamma \Rightarrow \mathrm{PSP}[\Gamma]$" is a polymorphic operation—see Theorem 44 and Ex. 40. Statement (13) follows from the fact that $\exp \hat{s}$ is a transducer and $\exp \hat{a}$ is an automaton and the operation $\downarrow$ is well-defined on these objects—see Lemma 24 and Ex. 42.

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 $\texttt{emptyP}(\hat{t})$ to get the desired answer. □

11. Composition of Transducers with Set Specifications

Here we are interested in defining the composition $\mathsf{p}_1 \circ \mathsf{p}_2$ of two pairing specs in a way that $\mathcal{R}(\mathsf{p}_1) \circ \mathcal{R}(\mathsf{p}_2)$ is equal to $\mathcal{R}(\mathsf{p}_1 \circ \mathsf{p}_2)$. By Definition 11, the operator $\mathcal{R}()$ is defined with respect to an alphabet of reference Γ, so the value of $\mathsf{p}_1 \circ \mathsf{p}_2$ should depend on Γ. It turns out that, for a particular subcase about the structure of $\mathsf{p}_1, \mathsf{p}_2$, the operation $\mathsf{p}_1 \circ \mathsf{p}_2$ can produce two or three pairing specs. To account for this, we define a new label set:

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

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

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

Definition 48. *Let Γ be an alphabet of reference. The label operation*

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

is defined between any $\mathsf{p}_1, \mathsf{p}_2 \in \mathrm{PSP}[\Gamma]_\varepsilon$ as follows—again $\perp$ means undefined.

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

Now we assume that the above condition is not true and we consider the possible structure of $\mathsf{p}_1, \mathsf{p}_2$ using $A, B, F, G \in \mathrm{SSP}[\Gamma]$ and $W, X, Y, Z \in \mathrm{SSP}[\Gamma]_\varepsilon$ as components—thus, we assume below that $\mathcal{L}(B) \cap \mathcal{L}(F) \neq \emptyset$ and $\mathcal{L}(X) \cap \mathcal{L}(Y) \neq \emptyset$.

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

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

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

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

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

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

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

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

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

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

$\boldsymbol{D}$ *includes* $A \cap \nexists b_1 b_2/G$*, if* $\mathcal{L}(A) \setminus \{b_1, b_2\} \neq \emptyset$*;*
$\boldsymbol{D}$ *includes* $\exists b_1/G \cap \nexists b_2$*, if* $b_1 \in \mathcal{L}(A)$ *and* $\mathcal{L}(G) \setminus \{b_2\} \neq \emptyset$*;*
$\boldsymbol{D}$ *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 $\boldsymbol{D} = \bot$ *if none of the above three conditions is true.*

Remark 49. *In the above definition, we have omitted cases where* $\mathsf{p}_1 \circ \mathsf{p}_2$ *is obviously undefined. For example, as* $F/{=}$ *and* $F/G{\neq}$ *are only defined when* $F, G \neq \mathbf{e}$*, we omit the case* $(W/\mathbf{e}) \circ (F/{=})$*.*

Remark 50. *If we allowed* $\bot$ *to be a pairing spec, then the set* $\mathrm{PSP}[\Gamma]$ *with the composition operation '*$\circ$*' would be 'nearly' a semigroup: the subcase "*$(A/B{\neq}) \circ (F/G{\neq})$ *with* $\mathcal{L}(B \cap F) = \{b_1, b_2\}$*" in the above definition is the only one where the result of the composition is not necessarily a single pairing spec. For example, let the alphabet* Γ *be* $\{0, 1, 2\}$ *and* $A = \exists 01$*,* $B = F = \exists 12$*, and* $G = \exists 012$*. Then,*

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

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

Lemma 51. *The relation* $\mathcal{R}(\mathsf{p}_1 \circ \mathsf{p}_2)$ *is equal to* $\mathcal{R}(\mathsf{p}_1) \circ \mathcal{R}(\mathsf{p}_2)$*, for any* $\mathsf{p}_1, \mathsf{p}_2 \in \mathrm{PSP}[\Gamma]_\varepsilon$*.*

Proof. We shall use the following shorthand notation:

$Q = \mathcal{R}(\mathsf{p}_1) \circ \mathcal{R}(\mathsf{p}_2)$ and $R = \mathcal{R}(\mathsf{p}_1 \circ \mathsf{p}_2)$ and $\mathcal{R}(\bot) = \emptyset$.

We shall distinguish several cases about the form of $\mathsf{p}_1 \circ \mathsf{p}_2$ according to Def. 48. By looking at that definition and using (3), we have that

$$Q, R \ \subseteq \ \mathcal{L}(\mathrm{left}\, \mathsf{p}_1) \times \mathcal{L}(\mathrm{rset}\, \mathsf{p}_2) \tag{15}$$

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 $(W/B) \circ \big(F/G{\neq}\big)$. 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 <u>second</u> subcase, $\mathcal{L}(B \cap F) = \{b\}$, for some $b \in \Gamma$, and $\mathcal{L}(G) \setminus \{b\} \neq \emptyset$. Then, $R = \mathcal{L}(W) \times \mathcal{L}(G \cap \not\ni b)$. 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 <u>third</u> 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 $(W/B) \circ \big(F/G{\neq}\big)$.

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 $\mathcal{R}(A/B \cap F{\neq}) = R$; 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 <u>second</u> 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 <u>third</u> 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:

$$Q_0 = \{(a, g) \mid a \in \mathcal{L}(A) \setminus \{b_1, b_2\}, g \in \mathcal{L}(G)\}$$

$Q_1 = \mathcal{L}(A) \times \mathcal{L}(G) \cap \{(b_1, g) \mid g \notin \mathcal{L}(G) \setminus \{b_2\}\}$
$Q_2 = \mathcal{L}(A) \times \mathcal{L}(G) \cap \{(b_2, g) \mid g \notin \mathcal{L}(G) \setminus \{b_1\}\}$

Then we have that

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

Corollary 52. *The polymorphic operation "$\circ : \mathrm{PSP}[\Gamma] \times \mathrm{PSP}[\Gamma] \Rightarrow \mathrm{PSP}_+[\Gamma]$" is well-defined by the partial operation $\circ$ in Def. 48 and the monoid operation $\circ$ in Ex. 37.*

Lemma 53. *For any $\mathsf{p}_1, \mathsf{p}_2 \in \mathrm{PSP}[\Gamma]_\varepsilon$, we have that $\mathsf{p}_1 \circ \mathsf{p}_2$ can be computed in time $O\big(|\mathsf{p}_1| + |\mathsf{p}_2|\big)$.*

Proof. Follows from Lemma 7 and the fact that $|\mathsf{p}_1 \circ \mathsf{p}_2| = O(|\mathsf{p}_1| + |\mathsf{p}_2|)$ as seen in Def. 48. □

Definition 54. *Let $\hat{t} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ and $\hat{s} = (Q', \mathrm{PSP}[\Gamma], \delta', I', F')$ be transducers with set specs. The transducer $\hat{t} \circledcirc \hat{s}$ with set specs is defined as follows. First compute the transducer $\hat{t} \circ \hat{s}$ with labels in $\mathrm{PSP}_+[\Gamma]$. Then, $\hat{t} \circledcirc \hat{s}$ results when each transition $(p, \mathsf{p}_1 \oplus \cdots \oplus \mathsf{p}_\ell, q)$ of $\hat{t} \circ \hat{s}$, with $\ell > 1$, is replaced with the ℓ transitions (p, p_i, q).*

Lemma 55. *We have that $\mathcal{R}(\hat{t} \circledcirc \hat{s}) = \mathcal{R}(\hat{t} \circ \hat{s})$.*

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

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

For each transition $e = (q_{i-1}, \mathsf{p}_i, q_i)$, define the triple $(q_{i-1}, \mathsf{p}'_i, q_i)$ as follows: $\mathsf{p}'_i = \mathsf{p}_i$, if e is in $\hat{t} \circ \hat{s}$; else, by Def. 54, there is a transition $(q_{i-1}, \mathsf{p}'_i, q_i)$ in $\hat{t} \circ \hat{s}$ such that p'_i is a $\oplus$-sum of terms that include p_i. Then, the sequence $P' = \langle q_{i-1}, \mathsf{p}'_i, q_i \rangle_{i=1}^{\ell}$ is an accepting path of $\hat{t} \circ \hat{s}$ such that

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

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

Theorem 56. *For any two trim transducers $\hat{t} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ and $\hat{s} = (Q', \mathrm{PSP}[\Gamma], \delta', I', F')$ with set specs, $\hat{t} \circledcirc \hat{s}$ can be computed in time $O(|\Gamma| + |\delta|\|\delta'\| + |\delta'|\|\delta\|)$. Moreover, $\mathcal{R}(\hat{t} \circledcirc \hat{s}) = \mathcal{R}(\hat{t}) \circ \mathcal{R}(\hat{s})$.*

Proof. The algorithm computes the transducer $\hat{t} \circ \hat{s}$ using the product construction in Def. 41. As the composition $\mathsf{p} \circ \mathsf{p}'$ 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 $(p, \mathsf{p}_1 \oplus \cdots \oplus \mathsf{p}_\ell, q)$ of $\hat{t} \circ \hat{s}$, with $\ell > 1$, with the ℓ transitions (p, p_i, q). Now we have

$$\begin{aligned} \mathcal{R}(\hat{t} \circledcirc \hat{s}) &= \mathcal{R}(\hat{t} \circ \hat{s}) && (16)\\ &= \mathcal{R}(\exp \hat{t} \circ \exp \hat{s}) && (17)\\ &= \mathcal{R}(\exp \hat{t}) \circ \mathcal{R}(\exp \hat{s}) && (18)\\ &= \mathcal{R}(\hat{t}) \circ \mathcal{R}(\hat{s}). && (19) \end{aligned}$$

Statement (17) follows from Theorem 44 and Corollary 52, and statement (18) follows from Lemma 24. □

12. Transducer Identity and Functionality

The question of whether a given transducer is functional is of central importance in the theory of rational relations [11]. 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 $\texttt{identityP}(\hat{t})$ that works in time $O(|\delta| + |Q||\Delta|)$ and tells whether $\hat{t} = (Q, [\Sigma, \Delta], \delta, I, F)$ realizes an identity. In view of Lemma 18, we have that

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

The algorithm $\texttt{functionalityP}(\hat{s})$ deciding functionality of a transducer $\hat{t} = (Q, [\Sigma, \Delta], \delta, I, F)$ first constructs the *square transducer* $\hat{u}$, [3], in which the set of transitions $\delta_{\hat{u}}$ consists of tuples $((p, p'), y/y', (q, q'))$ such that $(p, x/y, q)$ and $(p', x/y', q')$ are any transitions in $\hat{t}^{\varepsilon}$. Then, it follows that $\hat{t}$ is functional if and only if $\hat{u}$ realizes an identity. Note that $\hat{u}$ has $O(|\delta|^2)$ transitions and its graph size is $O(|\hat{t}|^2)$. Thus, we have that

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

Lemma 57. *Let $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, H)$ be a trim transducer with set specs. If any label p of $\hat{s}$ satisfies one of the following conditions then $\hat{s}$ does not realize an identity. (Below, F, G are set specs.)*

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

In the following conditions, 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 $\mathsf{p} = F/G$ 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, $\exp \hat{s}$ 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 $(uf_1u', vyv'), (uf_2u', vyv') \in \mathcal{R}(\exp \hat{s})$ and $f_1 \neq f_2$, $\exp \hat{s}$ cannot realize an identity. Now suppose one of (C2)–(C5) is true. One works as above and shows that again $\exp \hat{s}$ cannot realize an identity. For the time complexity, Lemma 7 implies that each condition can be tested in time $O(\mathsf{p})$. For all transitions $(p, \mathsf{p}, q) \in \delta$ this can be done in time $O(\|\delta\|)$. □

Theorem 58. *The question of whether a trim transducer $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, H)$ with set specs realizes an identity can be answered in time $O\big(|\delta||\Gamma|\big)$.*

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 p satisfies one of the conditions (C1)–(C5) in Lemma 57. Now suppose that no label p of $\hat{s}$ satisfies any of those conditions. Then, the algorithm computes $\exp \hat{s}$ and returns what $\texttt{identityP}(\exp \hat{s})$ returns. For each transition $(p, \mathsf{p}, q) \in \delta$ the corresponding transition(s) $(p, x/y, q) \in \delta_{\exp}$ are computed depending on the following five cases about the form of p.

(1) $(\mathbf{e}/\mathbf{e})$: Then, $x/y = \mathbf{e}/\mathbf{e}$.
(2) (F/G) or $(F/\mathbf{e})$ or $(\mathbf{e}/G)$: As (C1) is false, $\mathcal{L}(F) = \{f\}$ and/or $\mathcal{L}(G) = \{g\}$. Then $x/y = f/g$ or $x/y = f/\mathbf{e}$ or $x/y = \mathbf{e}/g$, depending on whether $\mathsf{p} = F/G$ or $\mathsf{p} = F/\mathbf{e}$ or $\mathsf{p} = \mathbf{e}/G$, 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 $\mathcal{R}(\mathsf{p}) = \emptyset$, 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 $(p, \mathsf{p}, q) \in \delta$. The third case results into $O(|\Gamma|)$ transitions. Thus, $|\delta_{\exp}| = O(|\delta||\Gamma|)$. Then, as $|\exp \hat{s}| = |\delta_{\exp}| + |Q|$ and $|Q| \leq 2|\delta| + 1$, we have that

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

The correctness of the algorithm follows from Lemma 57 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 $\texttt{identityP}(\exp \hat{s})$. Lemma 57 implies that $T_1 = O(\|\delta\|)$. For T_2, we have that

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

where C_{p} is the cost of computing the set of x/y for which $(p, x/y, q) \in \delta_{\exp}$. We show that $C_{\mathsf{p}} = O(|\Gamma|)$, which implies that $T_2 = O\big(|\delta||\Gamma|\big)$. Using Lemma 7, 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 $F = \nexists w$ 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 $u = \mathrm{wo}(\Gamma)$, and then use Lemma 3 to compute $\exists u \cap \nexists w$ in time $O(|\Gamma|)$, which is of the form $\exists v$ and equal to $\mathcal{L}(F)$. For T_3, statement (20) implies that $\texttt{identityP}(\exp \hat{s})$ works in time $O\big(|\delta_{\exp}| + |Q||\Gamma|\big)$, 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 27. □

Remark 59. *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* $\texttt{identityP}(\exp \hat{s})$*, which would work in time $O(|\delta_{\exp}||\Gamma|)$ according to* (20)*. This time complexity is clearly higher than the time $O(|\delta||\Gamma|)$ in the above theorem when $|\delta_{\exp}|$ is of order $|\delta||\Gamma|$ or $|\delta||\Gamma|^2$ (for example if $\hat{s}$ involves labels $\forall/=$ or $\forall/\forall$).*

Theorem 60. *The question of whether a trim transducer $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, H)$ 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. 41. The second part is to test whether $\hat{u}$ realizes an identity using Theorem 56. 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 27. For the correctness of the algorithm we have that

$$\begin{aligned}
\mathcal{R}(\hat{s}) \text{ is functional iff } & \mathcal{R}(\exp \hat{s}) \text{ is functional} && (23)\\
\text{iff } & \mathcal{R}(\exp \hat{s} \circ (\exp \hat{s})^{-1}) \text{ is an identity} && (24)\\
\text{iff } & \mathcal{R}(\exp \hat{s} \circ (\exp \hat{s}^{-1})) \text{ is an identity} && (25)\\
\text{iff } & \mathcal{R}(\exp \hat{s}) \circ \mathcal{R}(\exp \hat{s}^{-1}) \text{ is an identity} && (26)\\
\text{iff } & \mathcal{R}(\hat{s}) \circ \mathcal{R}(\hat{s}^{-1}) \text{ is an identity.} && (27)
\end{aligned}$$

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 26. □

Remark 61. *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* $\texttt{functionalityP}(\exp \hat{s})$*, which would work in time $O(|\delta_{\exp}|^2|\Gamma|)$ according to* (21)*. This time complexity is*

clearly higher than the time $O(|\delta|^2|\Gamma|)$ *in the above theorem when* $|\delta_{\exp}|$ *is of order* $|\delta||\Gamma|$ *or* $|\delta||\Gamma|^2$ *(for example if* $\hat{s}$ *involves labels* $\forall/=$ *or* $\forall/\forall$*).*

13. Some Applications in Independent Languages and Synchronous Transducers

Here we show that some algorithms about independent regular languages and synchronous transducers can be improved in terms of time complexity by employing transducers with set specs (see Example 65 and Corollary 66).

Let $\hat{t}$ be a transducer. A language L is called $\hat{t}$-*independent*, [13], if

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

If the transducer $\hat{t}$ is input-altering then, [9], the above condition is equivalent to

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

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 $\hat{t}_{\text{sub2}}$ describes all the 1-substitution error-detecting languages and $\hat{t}_{\text{px}}$ 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 62. *The witness version of the property satisfaction question for input-altering ordinary transducers* $\hat{s}$ *(see Eq.* (29)*) can be answered in time* $O(|\hat{s}| \cdot |\hat{a}|^2)$*, where* $\hat{a}$ *is the given ordinary automaton accepting* L *(see* [9]*). This can be done using the function call*

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

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 63. *Let* $\hat{s} = (Q, \mathrm{PSP}[\Gamma], \delta, I, F)$ *be a transducer with set specs and let* $\hat{b} = (Q', \Gamma, \delta', I', F')$ *be an ordinary automaton. The type* $[\Gamma, \Gamma]$ *transducers* $\hat{s} \downarrow \hat{b}$ *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 43. Then, we have

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

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

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

Statement (30) follows from Theorem 44 and Ex. 40, and statement (31) follows from Lemma 24. The proof for $\mathcal{R}(\hat{s} \uparrow \hat{b})$ is similar. □

Corollary 64. *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 65. *We can apply the above corollary to the transducer with set specs $\hat{t}_{\mathrm{sub2}}[\Gamma]$ of Example 25, 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 transducer $\exp \hat{t}_{\mathrm{sub2}}[\Gamma]$ to decide the question, the required time would be $O(|\Gamma|^2 \cdot |\hat{b}|^2)$.*

There are cases when we want to view the labels of a type B graph as single symbols. In [5] for example, a synchronous transducer $\hat{s}$ of type $[\Gamma, \Gamma]$ is viewed as an ordinary automaton $\hat{s}^\dagger$ over the alphabet $\Gamma \times \Gamma$, and this is helpful in the study of synchronous relations. A transducer is *synchronous* if each transition label x/y is such that $x, y \neq \mathbf{e}$, that is, $x, y \in \Gamma$. Suppose that $\Gamma = \Sigma \cup \{\#\}$ with $\# \notin \Sigma$. In [5], a relation $R \subseteq \Sigma^* \times \Sigma^*$ is *left synchronous*, if there is a synchronous transducer $\hat{s}$ of type $[\Gamma, \Gamma]$ such that

$$\mathcal{R}(\hat{s}) = \{(u, v\#^{|u|-|v|}) \mid (u, v) \in R \wedge |u| \geq |v|\} \cup \{(u\#^{|v|-|u|}, v) \mid (u, v) \in R \wedge |u| < |v|\}.$$

In this case, we say that $\hat{s}$ *represents* the left synchronous relation R. It can be shown that a synchronous transducer $\hat{s}$ of type $[\Gamma, \Gamma]$ represents a left synchronous relation if and only if

$$\mathcal{L}(\hat{s}^\dagger) \subseteq \Big(\uplus_{x,y \in \Sigma} (x/y)^\dagger \Big)^* \Big(\big(\uplus_{x \in \Sigma} (x/\#)^\dagger \big)^* \; + \; \big(\uplus_{y \in \Sigma} (\#/y)^\dagger \big)^* \Big), \tag{33}$$

where (i) $(x/y)^\dagger$ denotes the single symbol (in Ω) for the label x/y; (ii) $\hat{s}^\dagger$ is the automaton of type $\Delta \triangleq \{(x/y)^\dagger \mid x/y \in [\Gamma, \Gamma]\}$ resulting if each transition label x/y of $\hat{s}$ is replaced with $(x/y)^\dagger$; (iii) the right hand side of (33) is a language written as regular expression such that the notation $\uplus_{i=1,\ldots,k} \sigma_i$ is shorthand for $\sigma_1 + \cdots + \sigma_k$.

We want an efficient test for (33). Let L be the language on the right hand side of (33). Then, (33) is equivalent to $\mathcal{L}(\hat{s}^\dagger) \cap (\Delta^* \setminus L) = \emptyset$, and we can define an ordinary automaton $\hat{b}^\dagger$ over Δ such that $\mathcal{L}(\hat{b}^\dagger) = \Delta^* \setminus L$—see Fig 4. Then, we need to test the condition

$$\mathcal{L}(\hat{s}^\dagger) \; \cap \; \mathcal{L}(\hat{b}^\dagger) \; = \; \emptyset \tag{34}$$

As $\hat{s}^\dagger$ and $\hat{b}^\dagger$ are ordinary automata over Δ, we can test (34) in time $O(|\hat{s}||\Gamma|^2)$ by computing the automaton $\hat{s}^\dagger \cap \hat{b}^\dagger$ and testing whether it has a path from an initial to a final state.

Next we show that testing (34) can be done in time $O(|\hat{s}|)$. First note that by dropping † from the labels of $\hat{b}^\dagger$ we get the transducer $\hat{b}$ of type $[\Gamma, \Gamma]$. But then we can define the type PSP[Γ] graph $\hat{u}$ such that $\mathcal{R}(\exp \hat{u}) = \mathcal{R}(\hat{b})$ and $|\hat{u}| = O(1)$—see Fig 5. Our goal now is to define a new condition equivalent to (34) that can be tested in time $O(|\hat{s}|)$. We consider Δ to be the label set with $\operatorname{mon} \Delta = \Delta^*$ and behaviour $\mathcal{L}((x/y)^\dagger) = \{(x/y)^\dagger\}$, for all $(x/y)^\dagger \in \Delta$. We also define

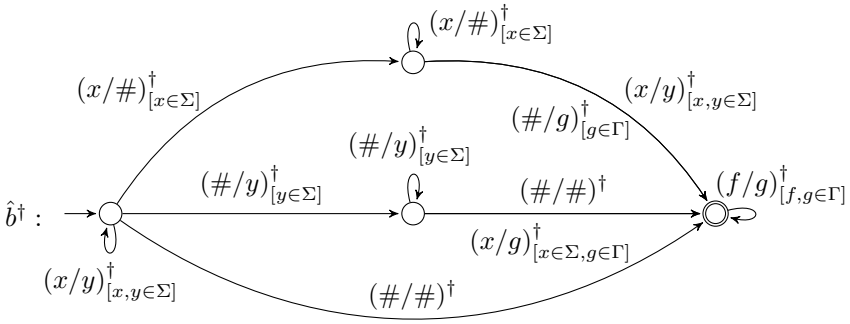

Fig. 4: The automaton $\hat{b}^\dagger$ accepts $\Delta^* \setminus L$, where L is the language in the right hand side of (33) and $\Delta = \{(x/y)^\dagger \mid x, y \in \Gamma\}$.

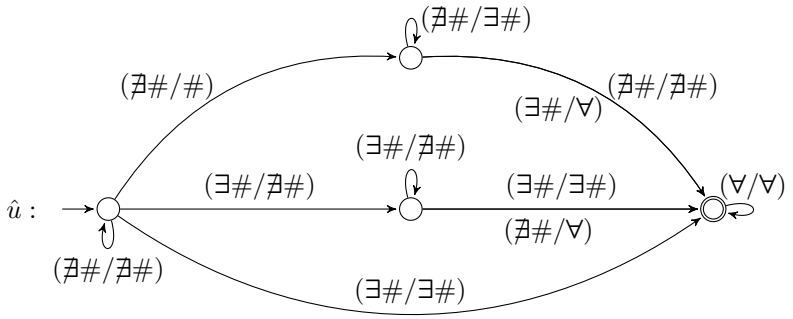

Fig. 5: The type PSP[Γ] graph $\hat{u}$ is such that $\mathcal{R}(\hat{u}) = \mathcal{R}(\hat{b})$, where $\hat{b}$ is the type $[\Gamma, \Gamma]$ transducer resulting by dropping from $\hat{b}^\dagger$ all †'s.

- the label set $\mathrm{PSP}[\Gamma]^\dagger = \{\mathsf{p}^\dagger \mid \mathsf{p} \in \mathrm{PSP}[\Gamma]\}$ with $\operatorname{mon}\mathrm{PSP}[\Gamma]^\dagger = \Delta^*$ and behaviour
$$\mathcal{L}(\mathsf{p}^\dagger) = \{(x/y)^\dagger \mid (x, y) \in \mathcal{R}(\mathsf{p})\};$$
- the polymorphic operation "$[\Gamma, \Gamma]^\dagger \cap \mathrm{PSP}[\Gamma]^\dagger \Rightarrow [\Gamma, \Gamma]^\dagger$" based on the standard monoid operation $\cap : \Delta^* \times \Delta^* \dashrightarrow \Delta^*$ (recall Ex. 37) and on the label operation $\cap : [\Gamma, \Gamma]^\dagger_\varepsilon \times \mathrm{PSP}[\Gamma]^\dagger_\varepsilon \dashrightarrow [\Gamma, \Gamma]^\dagger_\varepsilon$ such that $(x/y)^\dagger \cap \mathsf{p}^\dagger = (x/y)^\dagger$ if $(x, y) \in \mathcal{R}(\mathsf{p})$, that is, $(x/y)^\dagger \in \mathcal{L}(\mathsf{p}^\dagger)$; and $\perp$ otherwise. One confirms that $\mathcal{L}\big((x/y)^\dagger \cap \mathsf{p}^\dagger\big) = \mathcal{L}\big((x/y)^\dagger\big) \cap \mathcal{L}(\mathsf{p}^\dagger)$.

By Theorem 44 and the fact that $\exp(\hat{s}^\dagger), \exp(\hat{u}^\dagger)$ are automata over Δ, we have that $\mathcal{L}(\hat{s}^\dagger \cap \hat{u}^\dagger) = \mathcal{L}(\hat{s}^\dagger) \cap \mathcal{L}(\hat{u}^\dagger)$. If we show that $\mathcal{L}(\hat{u}^\dagger) = \mathcal{L}(\hat{b}^\dagger)$, then condition (34)

would be equivalent to

$$\mathcal{L}(\hat{s}^\dagger \cap \hat{u}^\dagger) = \emptyset. \tag{35}$$

That $\mathcal{L}(\hat{u}^\dagger) = \mathcal{L}(\hat{b}^\dagger)$ is indeed true follows when we note that (i) the graphs $\hat{u}^\dagger$ and $\hat{b}^\dagger$ are isomorphic; (ii) a transition $(p, \mathsf{p}^\dagger, q)$ of $\hat{u}^\dagger$ expands exactly to the transitions $(p, (x/y)^\dagger, q)$ of $\hat{b}^\dagger$ for all $(x/y)^\dagger \in \mathcal{L}(\mathsf{p}^\dagger)$. Finally, we note that, as $|\hat{u}| = O(1)$, we have $|\hat{s}^\dagger \cap \hat{u}^\dagger| = O(|\hat{s}|)$; hence, (35) can be tested in time $O(|\hat{s}|)$.

Corollary 66. *Whether a given synchronous transducer of type* $[\Sigma \cup \{\#\}, \Sigma \cup \{\#\}]$ *represents a left synchronous relation can be decided in time* $O(|\hat{s}|)$.

14. 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 other algorithms (not considered here) can be extended to regular expressions and transducers over pairing specs.

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, [7], 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 44. A research direction is to investigate whether more results can be obtained at this level, or what results obtain for different label sets. For example, for set specs F, G, one can add to PSP[Γ] the new labels $F/G<$ and $F/G>$ with their obvious behaviours. In this case, Lemma 47 would still hold, but composition of labels becomes problematic.

We close by noting that a concept of label set similar to the one defined here is considered in [6]. In particular, [6] considers label sets with weights, and the objectives of that work are different from the ones here.

References

[1] P. A. Abdulla, J. Deneux and L. K. Nilsson, Minimization of non-deterministic automata with large alphabets, *Proceedings of CIAA 2005, Sydney, Australia*, eds. J. Farré, I. Litovsky and S. Schmitz *Lecture Notes in Computer Science* **3845** (2006), pp. 31–42.

[2] C. Allauzen and M. Mohri, Efficient algorithms for testing the twins property, *Journal of Automata, Languages and Combinatorics* **8**(2) (2003) 117–144.

[3] M.-P. Béal, O. Carton, C. Prieur and J. Sakarovitch, Squaring transducers: An efficient procedure for deciding functionality and sequentiality, *Theoretical Computer Science* **292**(1) (2003) 45–63.
[4] J. A. Brzozowski and E. J. McCluskey, Signal flow graph techniques for sequential circuit state diagrams, *IEEE Trans. Electronic Computers* **12** (1963) 67–76.
[5] O. Carton, Left and right synchronous relations, *Proceedings of DLT 2009*, ed. V. Diekert *Lecture Notes in Computer Science* **5583** (2009), pp. 170–182.
[6] A. Demaille, A. Duret-Lutz, S. Lombardy, L. Saiu and J. Sakarovitch, A type system for weighted automata and rational expressions, *Proceedings of CIAA 2014*, eds. M. Holzer and M. Kutrib *Lecture Notes in Computer Science* **8587** (2014), pp. 162–175.
[7] FAdo, Tools for formal languages manipulation URL address: `http://fado.dcc.fc.up.pt/` Accessed in April, 2018.
[8] S. Konstantinidis, Transducers and the properties of error-detection, error-correction and finite-delay decodability, *Journal of Universal Computer Science* **8** (2002) 278–291.
[9] S. Konstantinidis, Applications of transducers in independent languages, word distances, codes, *Proceedings of DCFS 2017*, eds. G. Pighizzini and C. Câmpeanu *Lecture Notes in Computer Science* **10316** (2017), pp. 45–62.
[10] U. Manber, *Introduction to Algorithms: A Creative Approach* (Addison-Wesley, 1989).
[11] J. Sakarovitch, *Elements of Automata Theory* (Cambridge University Press, Berlin, 2009).
[12] J. Sakarovitch, Automata and rational expressions, *arXiv.org* **arXiv:1502.03573** (2015).
[13] H. J. Shyr and G. Thierrin, Codes and binary relations, *Séminaire d'Algèbre Paul Dubreil, Paris 1975–1976 (29ème Année)*, ed. M. P. Malliavin *Lecture Notes in Mathematics* **586** (1977), pp. 180–188.
[14] K. Thompson, Regular expression search algorithm, *Communications of the ACM (CACM)* **11** (1968) 419–422.
[15] M. Veanes, Applications of symbolic finite automata, *Proceedings of CIAA 2013*, ed. S. Konstantinidis *Lecture Notes in Computer Science* **7982** (2013), pp. 16–23.
[16] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar and N. Bjorner, Symbolic finite state transducers: Algorithms and applications, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012*, eds. J. Field and M. Hicks (2012), pp. 137–150.
[17] D. Wood, *Theory of Computation* (Harper & Row, New York, 1987).
[18] S. Yu, Regular languages, *Handbook of Formal Languages, Vol. I*, eds. G. Rozenberg and A. Salomaa 1997, pp. 41–110.