Journal of Automata, Languages and Combinatorics **u** (v) w, x–y
© Otto-von-Guericke-Universität Magdeburg

Valid Identity Problem for Shuffle Regular Expressions

Albert R. Meyer
MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
e-mail: meyer@lcs.mit.edu

and

Alexander Rabinovich
Department of Computer Science,
The Sackler School of Exact Sciences,
Tel Aviv University, Israel 69978
e-mail: rabino@math.tau.ac.il

ABSTRACT

We show that the following problem is decidable: given are expressions E_1 and E_2 constructed from variables by the regular operations and shuffle. Is the identity $E_1 = E_2$ true for all instantiations of its variables by formal languages?

Keywords: Decidability, valid equations, shuffle regular expressions

1. Introduction

We consider valid identity decision problem for shuffle regular expressions. Namely, given are expressions E_1 and E_2 constructed from variables by the regular operations and shuffle. Is the identity $E_1 = E_2$ true for all instantiations of its variables by formal languages?

For example, the identity $(X^*Y^*)^* = (X + Y)^*$ is true because for all languages L_1 and L_2, the languages $(L_1^*L_2^*)^*$ and $(L_1 + L_2)^*$ are the same.

The above identity contains only regular operations: concatenation, union and iteration. An easy 'folk' theorem (see e.g., [10] Exercise 14, Chapter 3) shows that the validity of an identity over regular operations can be verified by instantiating the language variables as single letters. For example, in order to check the validity of $(X^*Y^*)^* = (X + Y)^*$ we instantiate the variables X and Y by single letters a and b and verify that $(a^*b^*)^* = (a + b)^*$. Checking this variable-free identity is a routine matter of checking equivalence of finite state automata.

In concurrency a very important role is played by parallel composition operators. The simplest of these operators is the non-communicating parallel connective $||$, corresponding to shuffle of languages. The above folk theorem fails for the expressions containing shuffle. For example, for single letters a and b, the languages $a||b$ and $ab + ba$ are the same. However, the identity $X||Y = XY + YX$ is not true (indeed, instantiate X by a and Y by bc).

An algorithm for the valid identity problem is provided in this paper. In order to check the validity of an identity $E_1(X_1, \ldots, X_k) = E_2(X_1, \ldots, X_k)$ we will specify (see Theorem 3) finite languages $L_1, \ldots, L_k$ (the languages depend on E_1 and E_2) such that the identity is valid iff the variable-free identity obtained through instantiation of $X_1, \ldots, X_k$ by $L_1, \ldots, L_k$ is true. Checking this last variable-free identity is reduced to the checking of language equivalence of finite state automata.

The valid identity problem is formulated in terms of interleaving semantics. The decision algorithm is also given in interleaving terms. However, we developed and proved the correctness of the algorithm by appealing to a partial order semantics.

The paper is organized as follows. In Section 2 basic definitions are provided. In Section 3 the main theorems are stated and in Section 4 we analyze the complexity of the decision problem. Section 5 and Section 6 provide background on pomsets - partially ordered multisets. In Sections 7 and 8 the proof of the main theorem is given. Section 9 - conclusion and further results.

The results of this paper were announced in [5].

2. Shuffle Regular Expressions

We presuppose two fixed infinite sets

$$\begin{aligned} Act &= \{a,\ a_1, \ldots b,\ b_1, \ldots\} \quad \text{the actions} \\ Var &= \{X,\ X_1, \ldots Y,\ Y_1, \ldots\} \quad \text{the variable symbols.} \end{aligned}$$

Shuffle regular expressions are defined by the following grammar:

$E ::= \ x \mid c \mid E + E \mid E; E \mid E||E \mid E^*$, where x ranges over alphabet Var of variable symbols and, c ranges over alphabet Act of constant symbols.

We denote by $FVar(E)$ the set of variables which occur in E.

We say that E is a *variable free* expression if $FVar(E) = \emptyset$.

We say that E is a *constant free* expression if E does not contain actions.

We use notation $E\{E_1/X_1 \ldots E_n/X_n\}$ for the expression obtained from E by simultaneous substitution of E_i for X_i. We use $\sum_{i=1}^{n} E_i$ as an abbreviation for $E_1 + E_2 + \ldots + E_n$.

A string is a finite sequence of actions; we use w, u to range over strings. A string language is a set of strings; we use L to range over string languages.

The operations sum, concatenation, iteration and shuffle are defined in a standard way on the string languages.

We recall that a string w belongs to the shuffle of languages L_1 and L_2 if $w = w_1u_1w_2u_2 \ldots w_ku_k$ where $w_1w_2 \ldots w_k \in L_1$ and $u_1u_2 \ldots u_k \in L_2$.

$[\![a]\!]\sigma = \{a\}$
$[\![x]\!]\sigma = \sigma(x)$
$[\![E_1 + E_2]\!]\sigma =$ Union of $[\![E_1]\!]\sigma$ and $[\![E_2]\!]\sigma$
$[\![E_1; E_2]\!]\sigma =$ Concatenation of $[\![E_1]\!]\sigma$ and $[\![E_2]\!]\sigma$
$[\![E_1||E_2]\!]\sigma =$ Shuffle of $[\![E_1]\!]\sigma$ and $[\![E_2]\!]\sigma$
$[\![E_1^*]\!]\sigma = ([\![E]\!]\sigma)^*$

Figure 1: Definition of $[\![E]\!]\sigma$

A string language environment for $\{X_1 \dots X_n\}$ is a function which assigns to X_i a string language. For an expression E and a string language environment σ for a set that contains the free variables of E, the string language $[\![E]\!]\sigma$ is assigned in a standard way by structural induction on the expressions (see Fig. 1).

It is clear that if $\sigma(x) = \sigma'(x)$ for every $x \in FVar(E)$ then $[\![E]\!]\sigma = [\![E]\!]\sigma'$

3. The Valid Identity Problem

We will consider the following decision problem
Valid Identity Problem:
Input: A pair of shuffle regular expressions E_1 and E_2.
Question: Is the identity $E_1 = E_2$ valid, i.e., are the languages $[\![E_1]\!]\sigma$ and $[\![E_2]\!]\sigma$ equal for every string language environment σ for $FVar(E_1) \cup FVar(E_2)$.

The main technical result of our paper is

Theorem 1 *The valid identity problem for shuffle regular expressions is decidable.*

Theorem 1 follows from the next two theorems.

Theorem 2 *The valid identity problem for variable free shuffle regular expressions is decidable.*

Proof. The problem is easily reduced to the problem of equivalence of finite state automata. □

Notation: The shuffle nesting of an expression E is denoted by $sn(E)$ and is defined in Fig. 2.

Theorem 3 *Let E_1 and E_2 be shuffle regular expressions over variables $X_1, \dots, X_n$ such that the shuffle nesting of E_1 and E_2 is less than k.*
Let $\{\langle m_{i,j},\ S\rangle,\ \langle m_{i,j},\ F\rangle :\ i = 1, \dots, n;\ j = 1, \dots, k\}$ be distinct actions which do not occur in the expressions E_1, E_2. Let $SC_i^{(k)}$ be the expression $\sum_{j=1}^{k} \langle m_{i,j},\ S\rangle; \langle m_{i,j},\ F\rangle$. Then identity $E_1 = E_2$ is valid if and only if the variable free identity $E_1\{SC_1^{(k)}/X_1 \dots SC_n^{(k)}/X_n\} = E_2\{SC_1^{(k)}/X_1 \dots SC_n^{(k)}/X_n\}$ is valid.

$$sn(X) = sn(a) = 0$$
$$sn(E_1 + E_2) = sn(E_1; E_2) = max(sn(E_1), sn(E_2))$$
$$sn(E_1||E_2) = sn(E_1) + sn(E_2) + 1$$
$$sn(E^*) = sn(E)$$

Figure 2: Shuffle nesting of Expressions

Proof. In order to prove this theorem we appeal to the notions which were developed in the casual approach to concurrency. Theorem 3 follows from Theorem 8 and Theorem 10(2), below. □

4. Complexity of the valid identity problem

Given is a variable free shuffle-regular expression E of length n and shuffle nesting k. It is easy to construct a nondeterministic finite state automaton of the size bounded by $(n/k)^k$ that accepts the language definable by E. The equivalence of NFA's can be decided by a nondeterministic Turing machine in space proportional to the size of the automata (see e.g. [3] Theorem 13.14). Therefore, the validity of variable free shuffle-regular identity can be decided in space $O((n/k)^k)$.

Note that if an expression E has shuffle nesting less than k then $E\{SC_1^{(k)}/X_1, \ldots SC_n^{(k)}/X_n\}$ has the same shuffle nesting as E and that $size(E\{SC_1^{(k)}/X_1, \ldots SC_n^{(k)}/X_n\}) \leq 2 \times k \times size(E)$.

From Theorem 3 and the above remarks it follows that the validity of shuffle-regular identity for expressions of shuffle nesting k can be decided in nondeterministic space $O((2n)^k)$. Observing that $k < n/2$, we obtain $NSPACE(d^{nlog(n)})$ upper bound for valid identity problem.

Mayer and Stockmeyer [4] provided $O(c^{\sqrt{n}})$ nondeterministic space lower bound for the valid identity problem of the variable free shuffle regular expressions. They also have shown that the problem is EXSPACE complete (with respect to the polynomial time reductions).

Therefore,

Theorem 4 *The valid identity problem is in $NSPACE(d^{nlog(n)}) \setminus NSPACE(c^{\sqrt{n}})$ for some $c,\ d > 1$. The problem is also EXSPACE complete.*

We conclude with
Open Problem: Find a lower bound for the problem of validity of constant free identities.

5. Pomsets

Definition 1 *(Pratt [8]) A concrete pomset P over set Σ of labels consists of a set of events $Events_P$ which are partially ordered by a relation $\leq_P$ and a function lab_P*

from $Events_P$ into Σ. A function f is an isomorphism between concrete pomsets P_1 and P_2 if it is a label preserving isomorphism between the partial orders of P_1 and P_2. An (abstract) pomset is an isomorphism class of concrete pomsets.

Throughout the paper we provide some definitions and constructions for concrete pomsets. All these definitions/constructions are extended in a natural way to the abstract pomsets.

Definition 2 *Events e_1 and e_2 of a pomset P are* **concurrent** *(notation $e_1\ co_P\ e_2$) if neither $e_1 \leq_P e_2$ nor $e_2 \leq_P e_1$.*

Definition 3 *The* **width** *of a pomset P is the maximal number of mutually concurrent events in P.*

Definition 4 *A* **pomset language** *over Σ is a set of pomsets over Σ. We say that a pomset language PL has width at most n if all pomsets in PL have width less or equal than n.*

Definition 5 *A concrete pomset P is an* **augmentation** *of a concrete pomset Q if $Events_P = Events_Q$, $lab_P = lab_Q$ and $e_1 \leq_Q e_2$ implies $e_1 \leq_P e_2$ for all $e_1, e_2 \in Events_P$.*

Definition 6 *A concrete pomset P is a* **linearly ordered pomset** *if $\leq_P$ is a linear order over $Events_P$.*

We will identify a linearly ordered pomset over a label set Σ with the corresponding string over alphabet Σ. Also every string language is considered as a pomset language.

Definition 7 *The* **linearization** *of a pomset language PL (notation $Lin(PL)$) is the string language L such that $w \in L$ iff w is a linearly ordered augmentation of a pomset $P \in PL$.*

Notations: A pomset containing only one event labeled by l will be denoted by l. The pomset language containing only one pomset P will be denoted by $\{P\}$; in particular, the language containing only the one element pomset labeled by l will be denoted by $\{l\}$.

6. Refinement

Let P be a pomset and f be a function which assigns a pomset to every event of P.

The **f-expansion** of P is a pomset Q obtained by replacing every event of P by its image. Formally,

1. $Events_Q = \{(e, e') \ : e \in P,\ e' \in f(e)\}$;
2. $(e_1, e_2) \leq_Q (e_3, e_4)$ if either $e_1 \leq_P e_3$ or $e_1 = e_3$ and $e_2 \leq_{f(e_1)} e_4$;
3. $lab_Q((e, e')) = lab_{f(e)}(e')$.

We use the notation $Expan(P, f)$ for the f expansion of pomset P.

Definition 8 *A pomset language environment for a set of labels Σ is a function which assigns a pomset language to every label in Σ.*

Notations: We use the notation $[l_1 \to PL_1,\ l_2 \to PL_2, \ldots, l_n \to PL_n]$ for the pomset environment which maps l_i to PL_i, $i = 1, \ldots, n$. We denote by $PLE(\Sigma)$ the set of pomset language environments for Σ. We use α, β to range over pomset language environments. We denote by $SLE(\Sigma)$ the set of string language environments for Σ. We use σ, τ to range over string language environments.

Definition 9 *Let P be a pomset. Let f be a function which assigns a pomset to every event of P and let α be a pomset language environment for Σ. The function f is* **consistent** *with α if for every event e*

1. *$f(e) \in \alpha(lab_P(e))$ if $lab_p(e) \in \Sigma$,*
2. *$f(e) = \{lab_P(e)\}$ otherwise.*

Definition 10 *Let α be a pomset language environment for Σ. The* **α-refinement** *$REF(PL, \alpha)$ of a pomset language PL is $\{Expan(P, f) : P \in PL$ and f is consistent with $\alpha\}$.*

The refinement operation has properties similar to the substitution operation:

Lemma 5 *Let PL be a pomset language over an alphabet Σ and α be a pomset language environment for an alphabet Σ'. If $\Sigma \cap \Sigma' \subseteq \{l_1, \ldots, l_k\}$ then $REF((REF(PL, [l_1 \to PL_1, \ldots, l_k \to PL_k]), \alpha) = REF(PL, [l_1 \to REF(PL_1, \alpha), \ldots, l_k \to REF(PL_k, \alpha)])$.*

The next lemma states how linearization operator interacts with refinement.

Lemma 6 *$Lin(REF(PL, [l_1 \to PL_1, \ldots, l_k \to PL_k])) = Lin(REF(PL, [l_1 \to Lin(PL_1), \ldots, l_k \to Lin(PL_k)]))$.*

7. Operations definable by pomset languages

Definition 11 *The* **application of a pomset language** *PL to a string language environment τ (notation $PL \bullet \tau$) is the string language defined as $Lin(REF(PL, \tau))$.*

Definition 12 *(Pratt [9]) Let F be a function from string language environments for Σ into string languages. We say that F is* **definable by a pomset language** *PL if $F\tau = PL \bullet \tau$ for all $\tau \in SLE(\Sigma)$.*

Observation 7 *The regular operations and shuffle are pomset language definable, namely*

$$[\![a]\!]^{pom} = \{a\}$$
$$[\![x]\!]^{pom} = \{x\}$$
$$[\![E_1 + E_2]\!]^{pom} = REF(SUM(X,Y),\ [X \to [\![E_1]\!]^{pom},\ Y \to [\![E_2]\!]^{pom}])$$
$$[\![E_1; E_2]\!]^{pom} = REF(SEQ(X,Y),\ [X \to [\![E_1]\!]^{pom},\ Y \to [\![E_2]\!]^{pom}])$$
$$[\![E_1 || E_2]\!]^{pom} = REF(PAR(X,Y),\ [X \to [\![E_1]\!]^{pom},\ Y \to [\![E_2]\!]^{pom}])$$
$$[\![E_1^*]\!]^{pom} = REF(ITER(X),\ [X \to [\![E]\!]^{pom}])$$

Figure 3: Pomset Semantics

1. *Let $PAR(X,Y)$ be the pomset consisting of two unordered events labeled by X and Y. It is easy to see that $[\![X||Y]\!]$ defines the same operation as $\{PAR(X,Y)\}$ over the string environments for $\{X,Y\}$.*
2. *Let $SEQ(X,Y)$ be a pomset with two events labeled by X and Y such that the event X precedes the event Y. It is easy to see that $[\![X;Y]\!]$ defines the same operation as $\{SEQ(X,Y)\}$ over the string environments for $\{X,Y\}$.*
3. *Let $SUM(X,Y)$ be the pomset language consisting of two one element pomsets X and Y. It is easy to see that $[\![X+Y]\!]$ defines the same operation as $SUM(X,Y)$ over the string environments for $\{X,Y\}$.*
4. *Let $ITER(X)$ be the pomset language which consists of all finite strings over symbol X. It is easy to see that $[\![X^*]\!]$ defines the same operation as $ITER(X)$ over the string environments for $\{X\}$.*

Theorem 8 *For every shuffle regular expression E, the operation $\lambda\sigma.\ [\![E]\!]\sigma$ is definable by a pomset language with width equal to $1 + sn(E)$.*

Proof. By structural induction on the expressions the pomset language $[\![E]\!]^{pom}$ is assigned to every shuffle regular expression (see Fig. 3). Relying on Lemma 5, Lemma 6 and Observation 7, it is easy to show that $[\![E]\!]\sigma = [\![E]\!]^{pom} \bullet \sigma$, where σ is any language environment σ for the $FVar(E)$. □

Definition 13 *A string language environment $[l_1 \to L_1, \ldots l_n \to L_n]$, is called a split-choice environment for $\{l_1 \ldots l_n\}$ if every L_i contains only strings of length two.*

Lemma 9 *$PL \bullet \tau = PL' \bullet \tau$ for every string language environment τ for Σ iff $PL \bullet \tau = PL' \bullet \tau$ for every split-choice environment τ for Σ.*

This lemma can be strengthened as follows:

Theorem 10 *Let PL and PL' be pomset languages over an alphabet Σ and let $\{\langle m_{i,j},\ S\rangle,\ \langle m_{i,j}\ F\rangle\ :\ i = 1, \ldots, n\ ,$ and $1 \le j\}$ be distinct labels not in Σ. Let $L_i^{(\infty)}$ be string language $\{\langle m_{i,j},\ S\rangle\langle m_{i,j}\ F\rangle\ : 1 \le j\}$ and let $L_i^{(k)}$ be string language $\{\ \langle m_{i,j},\ S\rangle\langle m_{i,j}\ F\rangle\ : j = 1, 2, \ldots, k\}$.*

1. *$PL \bullet \tau = PL' \bullet \tau$ for every string language environment τ for $\{m_1, \ldots, m_n\}$ iff $PL \bullet [m_1 \to L_1^{(\infty)}, \ldots, m_n \to L_n^{(\infty)}] = PL' \bullet [m_1 \to L_1^{(\infty)}, \ldots, m_n \to L_n^{(\infty)}]$.*
2. *Let PL and PL' be pomset languages of width at most k. Then $PL \bullet \tau = PL' \bullet \tau$ for every string language environment τ for $\{m_1, \ldots, m_n\}$ if and only if $PL \bullet [m_1 \to L_1^{(k)}, \ldots, m_n \to L_n^{(k)}] = PL' \bullet [m_1 \to L_1^{(k)}, \ldots, m_n \to L_n^{(k)}]$.*

Proof. Follows from Corollary 15 and Corollary 17 in Section 8. □

Remarks (1) The above theorem can be strengthened as follows: Let k_i be the bound on the number of mutual concurrent events labeled by m_i in the pomset languages PL and PL'. Then $PL \bullet \tau = PL' \bullet \tau$ for every string language environment τ for $\{m_1, \ldots, m_n\}$ if and only if
$PL \bullet [m_1 \to L_1^{(k_1)}, \ldots, m_n \to L_n^{(k_n)}] = PL' \bullet [m_1 \to L_1^{(k_1)}, \ldots, m_n \to L_n^{(k_n)}]$.
(2) Weaker versions of the above mentioned theorem have appeared in the literature.

Gischer [2] considered the operations definable by pomsets. One of his results can be stated as follows: for pomsets P and P' with less than k events, $\{P\} \bullet \tau = \{P'\} \bullet \tau$ for every string language environment τ for $\{m_1, \ldots, m_n\}$ if and only if $\{P\} \bullet [m_1 \to L_1^{(k)}, \ldots, m_n \to L_n^{(k)}] = \{P'\} \bullet [m_1 \to L_1^{(k)}, \ldots, m_n \to L_n^{(k)}]$.

In [7] the special pomsets which are called semi-words are considered. A pomset is a semi-word if no events with the same label are concurrent. It was proved in [7] that for semi-word languages SWL and SWL' the following theorem holds: $SWL \bullet \tau = SWL' \bullet \tau$ for every string language environment τ for $\{m_1, \ldots, m_n\}$ if and only if
$SWL \bullet [m_1 \to L_1^{(1)}, \ldots, m_n \to L_n^{(1)}] = SWL' \bullet [m_1 \to L_1^{(1)}, \ldots, m_n \to L_n^{(1)}]$.

(3) The proof of Theorem 10 given in the next section is based on the proofs of these two weaker versions.

8. Proof of Theorem 10

8.1. Definitions and Some Preliminary Lemmas

In this section we collect some definitions and lemmas that will be referred later. Some of the lemmas follow directly from the definitions and their proofs are omitted.

Notation: Let s be a string and pos be a number between 1 and $length(s)$. We use notation $s(pos)$ for the letter of s at position pos. If $pos_0 < pos_1 < \ldots pos_k$ and $T = \{pos_i \ : i \leq k\}$ then string $s(T)$ is defined as $s(pos_0)s(pos_1)\ldots s(pos_k)$.

Definition 14 *Let τ be a language environment for Σ, let s be a string and P be a pomset. A function $\pi : \{1, 2, \ldots, length(s)\} \to Events_P$ is called a parsing of s wrt P and τ if*

1. *$v_1 <_P v_2$ implies that $\pi^{-1}(v_1)$ precedes $\pi^{-1}(v_2)$, i.e., $\pi(pos_1) = v_1$ and $\pi(pos_2) = v_2$ implies $pos_1 < pos_2$.*

2. $lab_P(v) \notin \Sigma$ *implies* $\pi^{-1}(v)$ *is a singleton position and it is labeled by* $lab_P(v)$.
3. $lab_P(v) \in \Sigma$ *implies that* $s(\pi^{-1}(v))$ *belongs to* $\tau(lab_P(v))$.

Lemma 11 *$s \in P \bullet \tau$ if and only if there exists a parsing of s wrt P and τ.*

Definition 15 *(Projection) The projection of a string $s = a_1 a_2 \dots a_k$ on an alphabet Σ is the string obtained from s by replacing all a_i that are not in Σ by the empty string ϵ.*

Definition 16 *(Balanced strings) Let Δ_0 and $\Delta \times \{S,\ F\}$ be disjoint sets of labels. A string s over $\Delta_0 \cup \Delta \times \{S,\ F\}$ is Δ balanced if for every $c \in \Delta$, the projection of s on $\{\langle c,\ S\rangle, \langle c,\ F\rangle\}$ is a string in the language $(\langle c,\ S\rangle\langle c,\ F\rangle)^*$.*

Definition 17 *(Matching positions) A position pos_2 is the* right match *for a position pos_1 in a string s if*

1. $pos_1 < pos_2$.
2. $s(pos_1) = \langle c,\ S\rangle$ *and* $s(pos_2) = \langle c,\ F\rangle$
3. $s(pos) \notin \{\langle c,\ S\rangle,\ \langle c,\ F\rangle\}$ *for any position pos between pos_1 and pos_2.*

We say that pos_1 is the left match of pos_2 if pos_2 is the right match of pos_1.

Lemma 12 *If s is Δ balanced then there is a function $\mathbf{rmatch}_s$ from the positions labeled by $\Delta \times \{S\}$ to the positions labeled by $\Delta \times \{F\}$ such that $\mathbf{rmatch}_s(pos)$ is the right match of pos.*

The inverse of $\mathbf{rmatch}_s$ will be denoted by $\mathbf{lmatch}_s$. Often, when s is clear from the context, we will write **rmatch** for $\mathbf{rmatch}_s$.

Lemma 13 *Let s be a Δ balanced string and τ be a language environment such that for every label l in the domain of τ the language $\tau(l)$ is a subset of $\{\langle a,\ S\rangle\langle a,\ F\rangle : a \in \Delta\}$. If $s \in P \bullet \tau$ then the following conditions are equivalent:*

1. $pos_2 = \mathbf{rmatch}(pos_1)$
2. $pos_1 < pos_2$ *and* $\pi(pos_1) = \pi(pos_2)$ *for every parsing π of s wrt P and τ.*
3. $pos_1 < pos_2$ *and* $\pi(pos_1) = \pi(pos_2)$ *for a parsing π of s wrt P and τ.*

Proof. $1 \Rightarrow 2$. Let π be a parsing of s wrt P and τ. Let Bad be $\{p :\ \pi(p) \neq \pi(\mathbf{rmatch}(p))$. It is enough to show that Bad is the empty set.

Suppose for the sake of contradiction that Bad is not empty. Let p_1 be the minimal element of Bad and let $p_2 = \mathbf{rmatch}(p_1)$. From Definition 17 it follows that $p_1 < p_2$ and there is some $c \in \Delta$ such that $s(p_1) = \langle c,\ S\rangle$ and $s(p_2) = \langle c,\ F\rangle$.

Let $v_1 = \pi(p_1)$ and let $v_2 = \pi(p_2)$. Note that $v_1 \neq v_2$, because $p_1 \in Bad$.

Since π is a parsing of s wrt P and τ there is position p_3 of s such that p_3 is labeled by $\langle c,\ S\rangle$ and $p_3 < p_2$ and $\pi^{-1}(v_2) = \{p_3,\ p_2\}$. Note that p_3 cannot be between p_1 and p_2 because $p_2 = \mathbf{rmatch}(p_1)$ and condition 3 of Definition 17. Therefore $p_3 < p_1$.

Note that $s(p_3) = s(p_1) = \langle c,\ S\rangle$. Since s is balanced there is p_4 such that $p_4 = \mathbf{rmatch}(p_3)$ and $p_4 < p_1$.

By the choice of p_1 as the minimal element of Bad it follows that $\pi(p_4) = \pi(p_3) = v_2 = \pi(p_2)$. Therefore $\pi^{-1}(v_2)$ contains at least three different elements and therefore $s(\pi^{-1}(v_2))$ cannot belong to $\tau(lab_P(v_2)) \subseteq \{\langle a,\ S\rangle\langle a,\ F\rangle :\ a \in \Delta\}$. This contradicts the assumption that π is a parsing of s wrt P and τ.

$2 \Rightarrow 3$. The implication is immediate, because from $s \in P \bullet \tau$, by Lemma 11, it follows that there exists a parsing of s wrt P and τ.

$3 \Rightarrow 1$: Suppose for the purpose of contradiction that $pos_1 < pos_2$ and $\pi(pos_1) = \pi(pos_2) = v$ and $pos_2 \neq \mathbf{rmatch}(pos_1)$. Let $p = \mathbf{rmatch}(pos_1)$,

Since we have already proved the implication $1 \Rightarrow 2$, it follows that $\pi(p) = \pi(pos_1) = v$. Therefore $\pi^{-1}(v)$ contains at least three different elements p, pos_1, pos_2 and π cannot be a parsing of s wrt τ. □

8.2. Proof of Theorem 10

In this and the next subsections we will use the following
Notations: Let $m_{i,j}$ and m_i be distinct labels $i = 1, \ldots, n$ and $j \geq 1$. We will use the following notations:

- $\Delta_i^{(r)}$ is the notation for the set of labels $\{m_{i,j},:\ i = 1, \ldots, n\ \ and\ 1 \leq\ j \leq r\}$
- $\Delta_i^{(\omega)}$ is the notation for the set of labels $\{m_{i,j},:\ i = 1, \ldots, n\ ,\ and\ 1 \leq j\}$
- $\Delta^{(\omega)}$ is the union of $\Delta_i^{(\omega)}$.
- $\Delta^{(r)}$ is the union of $\Delta_i^{(r)}$.
- $splitchoice^{(r)}$ is the language environment that assigns to m_i the language $\{\langle m_{i,j},\ S\rangle\langle m_{i,j},\ F\rangle\ :\ 1 \leq j \leq r\}$. The environment $splitchoise^{(\omega)}$ is defined similarly.

Proviso: (1) We assume that the sets $\Delta^{(\omega)}$, $\Delta^{(\omega)} \times \{S,\ F\}$ and $\{m_1, \ldots m_n\}$ are disjoint. (2) All language environments in the rest of the paper are for the alphabet $\{m_1, \ldots m_n\}$.

The first two paragraphs of Gischer's proof of Theorem 2.4 in [2] can be summarized in our terminology as follows:

Theorem 14 *(Gischer [2]) Let P be a pomset and let τ be a language environment for $\{m_1, \ldots m_n\}$. If $s \in P \bullet \tau$ then there exists a string t such that*

1. *$t \in P \bullet splitchoise^{(\omega)}$*
2. *t is $\Delta^{(\omega)}$ balanced*
3. *$\forall Q.t \in Q \bullet splitchoise^{(\omega)}$ implies $s \in Q \bullet \tau$*

Corollary 15 *Let PL_1 and PL_2 be pomset languages. Then $PL_1 \bullet splitchoise^{(\omega)} \subseteq PL_2 \bullet splitchoise^{(\omega)}$ if and only if $\forall \tau.\ PL_1 \bullet \tau \subseteq PL_2 \bullet \tau$*

Proof. The direction from the right to the left is trivial. Below a proof for the other direction is presented. Namely, from

$$PL_1 \bullet splitchoise^{(\omega)} \subseteq PL_2 \bullet splitchoise^{(\omega)} \tag{1}$$

and

$$s \in PL_1 \bullet \tau \tag{2}$$

we will derive that $s \in PL_2 \bullet \tau$.

From (2) it follows that

$$\text{there exists } P_1 \in PL_1 \text{ such that } s \in P_1 \bullet \tau. \tag{3}$$

From (3), by Theorem 14, we obtain that

$$\text{there exists } t \in P_1 \bullet splitchoise^{(\omega)} \text{ such that} \tag{4}$$

$$\forall Q.t \in Q \bullet splitchoise^{(\omega)} \text{ implies } s \in Q \bullet \tau \tag{5}$$

From (1), (3) and (4) it follows that there exists $P_2 \in PL_2$ such that $t \in P_2 \bullet splitchoise^{(\omega)}$, therefore, by (5), we obtain that $s \in P_2 \bullet \tau \subseteq PL_2 \bullet \tau$ and this completes the proof.

□

Lemma 16 *If P is a pomset of width not greater than k and $s \in P \bullet splitchoise^{(\omega)}$ then there exists string t such that*

1. *$t \in P \bullet splitchoise^{(k)}$*
2. *t is $\Delta^{(k)}$ balanced*
3. *$\forall Q.t \in Q \bullet splitchoise^{(k)}$ implies $s \in Q \bullet splitchoise^{(\omega)}$*

Proof. The proof will be given at the end of the next subsection. □

From Theorem 14 and Lemma 16, it follows

Corollary 17 *Let PL_1 and PL_2 be pomset languages of parallel width not greater than k. Then $PL_1 \bullet splitchoise^{(k)} \subseteq PL_2 \bullet splitchoise^{(k)}$ if and only if $\forall \tau. PL_1 \bullet \tau \subseteq PL_2 \bullet \tau$*

Proof. The proof is similar to the proof of corollary 15.

The direction from the right to the left is trivial. Below a proof for the other direction is presented. Namely, from

$$PL_1 \bullet splitchoise^{(k)} \subseteq PL_2 \bullet splitchoise^{(k)}$$
$$\text{and parallel width of } PL_1,\ PL_2 \text{ is not greater than } k \tag{6}$$

and

$$s' \in PL_1 \bullet \tau \tag{7}$$

we will derive that $s' \in PL_2 \bullet \tau$.

From (7) it follows that

there exists $P_1 \in PL_1$ such that $s' \in P_1 \bullet \tau$. (8)

From (8), by Theorem 14, we obtain that

there exists $s \in P_1 \bullet splitchoise^{(\omega)}$ such that (9)

$\forall Q.s \in Q \bullet splitchoise^{(\omega)}$ implies $s' \in Q \bullet \tau$ (10)

From (9), by Lemma 16, we derive that

there exists $t \in P_1 \bullet splitchoise^{(k)}$ such that (11)

$\forall Q.t \in Q \bullet splitchoise^{(\omega)}$ implies $s \in Q \bullet splitchoise^{(\omega)}$ (12)

Since $PL_1 \bullet splitchoise^{(k)} \subseteq PL_2 \bullet splitchoise^{(k)}$, it follows that there exists $P_2 \in PL_2$ such that $t \in P_2 \bullet splitchoise^{(k)}$, therefore, by (12), we obtain that $s \in P_2 \bullet splitchoise^{(\omega)}$ and by (10) we obtain that $s' \in P_2 \bullet \tau \subseteq PL_2 \bullet \tau$.

This completes our proof. □

8.3. Intervals and Proof of Lemma 16

Given is a linearly ordered set. Its interval $[a,\ b]$ is the set $\{x :\ a \leq x \leq b\}$. Two intervals $[a, b]$ and $[c, d]$ overlap if their intersection is not empty.

Lemma 18 *Let $\mathcal{F}$ be a set of intervals such that the intersection of any $k+1$ intervals from $\mathcal{F}$ is empty. Then there exists a partition $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k$ of $\mathcal{F}$ such that for any i the intersection of any two intervals from $\mathcal{F}_i$ is empty.*

Proof. By induction on the number of intervals in $\mathcal{F}$.

The basis is trivial.

Inductive step: Let $\mathcal{F} = \{[a_i,\ b_i]\ : i < m+1\}$ be a set of $m+1$ intervals. Let $a_{max} = max\{a_i\ : i < m+1\}$ and let $\mathcal{F}' = \mathcal{F} \setminus [a_{max},\ b_{max}]$. By the inductive hypothesis there exists a partition $\mathcal{F}'_1, \mathcal{F}'_2, \ldots, \mathcal{F}'_k$ of $\mathcal{F}'$ such that the intersection of any two intervals from $\mathcal{F}'_i$ is empty.

Let $\mathcal{G}$ be the set of intervals from $\mathcal{F}'$ that overlap with $[a_{max}, b_{max}]$. By the choice of a_{max} it follows that a_{max} belongs to every interval in $\mathcal{G}$ and therefore a_{max} belongs to $|\mathcal{G}|+1$ intervals from $\mathcal{F}$. Therefore $|\mathcal{G}| < k$ and therefore, there exists j such that $\mathcal{F}'_j$ does not contain any interval from $\mathcal{G}$. Hence, the intersection of $[a_{max},\ b_{max}]$ with every interval in $\mathcal{F}'_j$ is empty.

The required partition of $\mathcal{F}$ is obtained from the partition of $\mathcal{F}'$ by adding $[a_{max},\ b_{max}]$ to $\mathcal{F}'_j$. □

Remark: Our proof shows that Lemma 18 holds for the finite sets of intervals. Only this finite version will be used later. However, by standard compactness arguments, from the finite version of the Lemma it follows the validity of the Lemma for the infinite sets of intervals.

Definition 18 *Let s be $\Delta^{(\omega)}$ balanced string. For every i and j we define the set of intervals $G_s^{i,j}$ as follows: $[pos_1,\ pos_2] \in G_s^{i,j}$ if $s(pos_1) = \langle m_{i,j}, S\rangle$ and $pos_2 = \mathbf{rmatch}(pos_1)$. The set of intervals G_s^i is defined as the union of the sets $G_s^{i,j}$.*

Observe that because s is $\Delta^{(\omega)}$ balanced string any two intervals from $G_s^{i,j}$ have empty intersection.

Lemma 19 *Let s and t be $\Delta^{(\omega)}$ balanced strings. If $G_s^i = G_t^i$ for $i = 1,\dots n$ and $\forall p.\ s(p) \neq t(p) \Rightarrow (s(p) \in \Delta^{(\omega)} \times \{S,\ F\} \wedge t(p) \in \Delta^{(\omega)} \times \{S,\ F\})$ then $t \in P \bullet splitchoise^{(\omega)}$ if and only if $s \in P \bullet splitchoise^{(\omega)}$.*

Proof. We will show that if $t \in P \bullet splitchoise^{(\omega)}$ then $s \in P \bullet splitchoise^{(\omega)}$. The other direction follows by symmetry.

Since $t \in P \bullet splitchoise^{(\omega)}$ there exists a parsing π of t wrt P and $splitchoise^{(\omega)}$. Therefore, by Lemma 11, it is sufficient to show that π is a parsing of s wrt P and $splitchoise^{(\omega)}$.

Below we will check that the requirements (1), (2) and (3) of Definition 14 are satisfied.

(1) follows immediately, because π is a parsing of t.

(2) If $lab_p(v) \notin \{m_1,\dots m_n\}$ then $\pi^{-1}(v)$ is singleton and $t((\pi^{-1}(v)) = lab_P(v)$, because π is a parsing of t. Therefore, from the assumption $\forall p.\ s(p) \neq t(p) \Rightarrow (s(p) \in \Delta^{(\omega)} \times \{S,\ F\} \wedge t(p) \in \Delta^{(\omega)} \times \{S,\ F\})$ it follows that $s((\pi^{-1}(v)) = lab_P(v)$ and this proves condition (2) of Definition 14.

(3) $lab_p(v) = m_i$. In this case $\pi^{-1}(v)$ contains two elements pos_1 and pos_2 because π is parsing of t wrt P and $splitchoise^{(\omega)}$. Moreover, there is j such that $t(pos_1) = \langle m_{i,j},\ S\rangle$ and $t(pos_2) = \langle m_{i,j},\ F\rangle$.

By Lemma 13, we have $pos_2 = \mathbf{rmatch}_t(pos_1)$. Therefore, $[pos_1,\ pos_2] \in G_t^{i,j}$

Note that $G_t^{i,j} \subseteq G_t^i = G_s^i$. Therefore. $[pos_1,\ pos_2] \in G_s^{i,r}$ for some r. Consequently, $s(pos_1) = \langle m_{i,r},\ S\rangle$ and $s(pos_2) = \langle m_{i,r},\ F\rangle$.

Thus $s(\pi^{-1}(v)) = s(pos_1)s(pos_2) = \langle m_{i,r},\ S\rangle\langle m_{i,r},\ F\rangle \in splitchoise^{(\omega)}(m_i)$. and $lab_P(v) = m_i$. This proves condition (3) of Definition 14. □

Lemma 20 *Let P be a pomset with at most k_i mutual concurrent events labeled by m_i. If $s \in P \bullet splitchoise^{(\omega)}$ is $\Delta^{(\omega)}$ balanced string then the intersection of any k_i+1 interval from G_s^i is empty.*

Proof. Let $H = \{[a_j,\ b_j]\ : 1 \leq j \leq k_i + 1\}$ be a set of $k_i + 1$ intervals in G_s^i. We will show that there are two intervals in H with the empty intersection. This will imply the Lemma.

Observe that $b_j = \mathbf{rmatch}(a_j)$ and $s(a_j) \in \Delta_i^{(\omega)} \times \{S\}$, by the definition of G_s^i.

Let π be a parsing of s wrt P and $splitchoise^{(\omega)}$. From Lemma 13, it follows that for $j = 1,\dots k_i + 1$

- $\pi(a_j) = \pi(b_j) = v_j$, and
- $lab(v_j) = m_i$ and
- $j \neq j'$ implies $v_j \neq v_{j'}$

Since in P there are at most k_i mutual concurrent events labeled by m_i, it follows that there exist j_1, $j_2 < k_i + 1$ such that $v_{j_1} <_P v_{j_2}$.

Therefore, by Definition 14(1), it follows that $\pi^{-1}(v_{j_i}) \ni b_{j_1} < a_{j_2} \in \pi^{-1}(v_{j_2})$. Thus the intersection of $[a_{j_1},\ b_{j_i}]$ and $[a_{j_2},\ b_{j_2}]$ is empty. □

Now we are ready to prove a lemma that is slightly weaker than Lemma 16.

Lemma 21 *If P is a pomset of width not greater than k and s is $\Delta^{(\omega)}$ balanced string in $P \bullet splitchoise^{(\omega)}$ then there exists a string t such that*

1. *$t \in P \bullet splitchoise^{(k)}$*
2. *t is $\Delta^{(k)}$ balanced*
3. *$\forall Q. t \in Q \bullet splitchoise^{(k)}$ iff $s \in Q \bullet splitchoise^{(\omega)}$*

Proof. Let P be a pomset of width k and let s be $\Delta^{(\omega)}$ balanced string in $P \bullet splitchoise^{(\omega)}$. By Lemma 20, for any i the intersection of $k+1$ intervals from G^i_s is empty. Therefore by Lemma 18, there exists a partition of G^i_s into k sets of intervals $\mathcal{F}^i_1, \ldots \mathcal{F}^i_k$ such that no two intervals from $\mathcal{F}^i_j$ overlap.

Define a string t by renaming the labels of some matching pairs of s as follows:

$$t(p) = \begin{cases} s(p) & \text{if } s(p) \notin \Delta^{(\omega)} \times \{S,\ F\} \\ \langle m_{i,j}, S\rangle & \text{if } p \text{ is the left end of an interval in } \mathcal{F}^i_j \\ \langle m_{i,j}, F\rangle & \text{if } p \text{ it the right end of an interval in } \mathcal{F}^i_j \end{cases}$$

Note that every position of s whose label is in $\Delta^{(\omega)} \times \{S,\ F\}$ is an endpoint of exactly one interval from $\cup_{i=1}^{i=n} G^i_s = \cup_{i=1}^{i=n} \cup_{j=1}^{j=k} \mathcal{F}^i_j$. Therefore, t is well defined.

We are going to show that t satisfies the conclusions of Lemma 21.

Observe that intervals in $\mathcal{F}^i_j$ do not overlap, hence t is $\Delta^{(\omega)}$ balanced. This proves Lemma 21(2).

From the definition of t it follows that $G^{i,j}_t = \mathcal{F}^i_j$ and therefore

$$G^i_t = \cup_{j=1}^{j=k} \mathcal{F}^i_j = G^i_s. \tag{13}$$

Also observe that

$$\forall p.\ s(p) \neq t(p) \Rightarrow (s(p) \in \Delta^{(\omega)} \times \{S,\ F\} \wedge t(p) \in \Delta^{(\omega)} \times \{S,\ F\}). \tag{14}$$

Hence by Lemma 19, we obtain that

$$\forall Q.\ (s \in Q \bullet splitchoise^{(\omega)} \text{ iff } t \in Q \bullet splitchoise^{(\omega)}). \tag{15}$$

Note that by the construction of t, if $t(p) \in \Delta^{(\omega)} \times \{S,\ F\}$ then $t(p) \in \Delta^{(k)} \times \{S,\ F\}$. This fact together with (15) imply

$$\forall Q.(s \in Q \bullet splitchoise^{(\omega)} \text{ if and only if } t \in Q \bullet splitchoise^{(k)}). \tag{16}$$

From (16), we obtain Lemma 21(3) and from (16) and the assumption that $s \in P \bullet splitchoise^{(\omega)}$ we obtain Lemma 21(1). □

Now we are ready to prove Lemma 16.

Proof. (of Lemma 16). Assume that $s \in P \bullet splitchoise^{(\omega)}$. Then, by Theorem 14, there exists s' such that s' is $\Delta^{(\omega)}$ balanced $s' \in P \bullet splitchoise^{(\omega)}$ and

$$\forall Q.s' \in Q \bullet splitchoise^{(\omega)} \text{ implies } s \in Q \bullet splitchoise^{(\omega)}. \tag{17}$$

Hence s' satisfies the assumption of Lemma 21 and therefore there exists t such that $t \in P \bullet splitchoise^{(k)}$, t is $\Delta^{(k)}$ balanced (this establishes (1) and (2) of Lemma 16) and

$$\forall Q.t \in Q \bullet splitchoise^{(\omega)} \text{ implies } s' \in Q \bullet splitchoise^{(\omega)}. \tag{18}$$

From (17) and (18), we obtain that t also satisfies (3) of Lemma 16. □

9. Related Works and Further Results

9.1. The Role of Intervals and of Balanced Strings

Let $\langle V, \leq\rangle$ be a partially ordered set (poset). Assume that we want to assign durations to the events in V and to schedule the executions of the events in such a way that $e_1 < e_2$ if and only if the execution of e_2 starts after the execution of e_1 has been completed. Not every partially ordered set allows such a scheduling.

A poset $\langle V, \leq\rangle$ has an interval representation if there exists a linearly ordered set $\langle L, \leq_L\rangle$ and functions $Start$, $Finish$ from V into L such that $Start(e) \leq_L Finish(e)$ and $e_1 \leq e_2$ iff $Finish(e_1) \leq_L Start(e_2)$. It was shown by Russell and Wiener [11] that a poset has an interval representation if it has the following property:

$$\forall e_1e_2e_3e_4.\ (e_1 \leq e_2 \wedge e_3 \leq e_4) \Rightarrow (e_1 \leq e_4 \vee e_3 \leq e_2) \tag{19}$$

Let $P = \langle V_P, \leq_P, lab_P\rangle$ be a pomset. P is an interval pomset if $\langle V_P, \leq_P\rangle$ has an interval representation.

The role of balanced strings for interval pomsets is similar to the role of interval representations for posets.

Let s be $\Delta^{(\omega)}$ balanced string. For the sake of simplicity we assume that s is over alphabet $\Delta^{(\omega)} \times \{S, F\}$.

Consider the interval pomset P_s over the alphabet $\{m_1, \ldots m_n\}$ defined as follows: (1) The events of P_s are the matching pairs $[pos, pos']$ in s. (2) lab_{P_s} is defined as $lab_{P_s}([pos, pos']) = m_i$ if $\exists j.\ s(pos) = \langle m_{i,j}, S\rangle$. (3) $[pos_1, pos_1'] \leq [pos_2, pos_2']$ iff $pos_1' \leq pos_2$.

It is clear that $s \in P_s \bullet splitchoise^{(\omega)}$. Moreover, from our proofs it follows that $s \in P \bullet splitchoise^{(\omega)}$ iff P_s is an augmentation of P. From Lemma 21 we can obtain that if P_s has width $\leq k$ then there exists a balanced string t over alphabet $\Delta^{(k)}$ such that P_t is isomorphic to P_s. From the Gischer's theorem (Theorem 14) and Lemma 16 it follows that for any (not necessary balanced) string s there are balanced strings $t_1 \ldots t_m$ such that $s \in P \bullet splitchoise^{(\omega)}$ iff there is i such that P_{t_i} is an augmentation of P.

Recall [7] that pomset languages PL_1 and PL_2 over Σ are interval equivalent if

$$\forall P.\ P \text{ is interval pomset } \Rightarrow (P \in Aug(PL_1) \text{ if and only if } P \in Aug(PL_2)a) \tag{20}$$

It was shown in [7] that PL_1 and PL_2 are interval equivalent iff $\forall \tau.\ PL_1 \bullet \tau = PL_2 \bullet \tau$.

Our results generalize the results of [7]. In the above terminology we have shown that pomset languages PL_1 and PL_2 of width $\leq k$ are interval equivalent iff $PL_1 \bullet splitchoise^{(k)} = PL_2 \bullet splitchoise^{(k)}$.

Another technical contribution of this paper is that we consider the partition of labels into atomic At and not atomic Σ. A pomset P over $At \cup \Sigma$ has $At - \Sigma$ interval representation if there exists a linear order $\langle L, \ \leq_L \rangle$ and functions $Start$, $Finish$ from V into L such that (1) $Start(e) \leq_L Finish(e)$, (2) $Start(e) = Finish(e)$ for every event e with an atomic label and (3) $e_1 \leq e_2$ iff $Finish(e_1) \leq_L Start(e_2)$.

It is easy to see that P is an $At - \Sigma$ interval pomset if it satisfies (19) and all its atomic events are linearly ordered. The definition of $At - \Sigma$ interval equivalence is similar to the definition of interval equivalence and from our proofs it follows that PL_1 and PL_2 are $At - \Sigma$ interval equivalent iff $PL_1 \bullet \tau = PL_2 \bullet \tau$ for every language environment τ for Σ.

Also it is easy to show that $At - \Sigma$ interval equivalence is a congruence with respect to refinement, i.e., if PL is $At - \Sigma$ interval equivalent to PL' and PL_1 is $At - \Sigma$ interval equivalent to PL'_1 and $l \in \Sigma$ then $REF(PL, [l \to PL_1])$ is $At - \Sigma$ interval equivalent to $REF(PL', [l \to PL'_1])$.

9.2. Extension by Pomset Language Definable Operations

Let OP be an n-ary operation on string languages. We say that OP is effective on regular languages if there exists an algorithm which constructs a finite automaton for the language $OP(L_1, \ldots, L_n)$ from finite automata for $L_1, \ldots, L_n$.

We say that OP is definable by (finite width) pomset language if there exists a (finite width) pomset language PL such that $PL \bullet [1 \to L_1, \ldots, n \to L_n] = OP(L_1, \ldots, L_n)$ for any languages $L_1, \ldots, L_n$.

Note that the operations definable by finite pomset languages are effective on regular languages. Among such operations are operations which are not definable by any shuffle regular expressions.

The valid identity problem is decidable for the expressions constructed over any set of operations which are effective on regular languages and are definable by finite width pomset languages.

9.3. Expressions Without Constants

Bloom and Ésik (see Corollary 6.35 and Section 8 in [1]) proved independently the decidability of the valid identity problem for constant free shuffle regular expressions.

Micciancio [6] proved the decidability of the valid identity problem for constant free shuffle-intersection regular expressions. These expressions are defined by the following grammar: $E ::= \ x \mid E \cap E \mid E + E \mid E; E \mid E || E \mid E^*$, where x ranges over variable symbols.

Note that the intersection is not a pomset language definable operation. Micciancio's very interesting proof is given in terms of interleaving semantics and does not use explicitly pomsets. For this proof it is essential that no constant symbols are presented in the expressions.

Yotam Abramson observed that an identity between constant free shuffle-intersection regular expressions is valid if it is valid for the interpretations of the variables as languages over the binary alphabet $\{0, 1\}$.

Acknowledgments

We would like to thank Yoram Hirshfeld and Boris Trakhtenbrot for stimulating discussions and many constructive suggestions.

References

[1] S. L. Bloom and Z. Ésik. Free shuffle algebras in language varieties. Theoretical Computer Science, 163:55-98, 1996.

[2] J. Gischer. Partial Orders and Theory of Shuffle. PhD thesis, Stanford University, 1984.

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Lamguages and Computations*, Addison-Wesley, Reading, MA, 1979.

[4] A. Mayer and L. Stockmeyer. The complexity of word problem - this time with interleaving. Information and Computation, pages 293-311, Vol 115, 1994.

[5] A. R. Meyer and A. Rabinovich. A solution of an interleaving decision problem by a partial order techniques. In *Proceedings of Workshop on Partial-Order Methods in Verification*, pp 203-212, Editors G. Holzmann, D. Peled, V. Pratt, AMS-DIMACS Series in Discrete Mathematics, 1996.

[6] D. Micciancio. The Validity problem for Extended Regular Expressions. M. Sc. thesis, MIT, 1996.

[7] M.. Nielsen, U. Engberg, and K. Larsen. A Simple Process Language with Refinement. In *REX Workshop on Linear Time, Branching time and Partial Order in Logics and Models for Concurrency* , volume 354 of *Lect. Notes in Computer Science*, pages 523-548, Springer Verlag, 1990.

[8] V. R. Pratt, On the composition of Processes. In *Proc. of the Ninth Annual ACM Symposium on Principles of Programming Languages*, 1982.

[9] V. R. Pratt, Modeling concurrency with partial orders. *J. of Parallel Processing,* 15(1986), 33-71.

[10] B. A. Trakhtenbrot and Y. M. Barzdin. Finite Automata, North Holland Amsterdam, 1973.

[11] N. Wiener. A contribution to the theory of relative position. In *Proc. Camb. Philos. Soc.* 17, pp 441-449, 1914.