Theoretical Computer Science 38 (1985) 223-247
North-Holland

EXTENDING REGULAR EXPRESSIONS WITH ITERATED SHUFFLE

Matthias JANTZEN

Fachbereich Informatik (TGI), Universität Hamburg, Rothenbaumschaussee 67, D-2000 Hamburg 13, Fed. Rep. Germany

Communicated by T. Ito
Received May 1984
Revised November 1984

Abstract. It is shown that every finite expression which uses the operations union, product, Kleene star, and iterated shuffle in any order, starting with finite sets, defines a language which can be recognized non-deterministically by some multicounter machine in quasirealtime. It is known that this family is in general not closed with respect to iterated shuffle. As a consequence of the characterization each such language is in NSPACE(log n) and thus in P. However, if P $\neq$ NP, then also neither P nor NSPACE(log n) are closed under iterated shuffle. The proof uses the new concept of so-called shuffle schemes and a number of results on algebraic language theory.

Introduction

Recently, the operations shuffle and iterated shuffle have got more and more attention. Together with other operations on languages they have been used to define various classes of finite expressions that aim to describe sequentialized execution histories of concurrent processes or sometimes the processes themselves [4, 17-24].

The iterated shuffle operation sometimes allows a very short description of an otherwise lengthy process specification. For instance, the unbounded readers-writers problem, which cannot be modelled by any ordinary Petri net, has the following short 'solution':

$$(((startread \cdot read \cdot endread)^{\circledast} \sqcup\!\sqcup requestwrite) \cdot write)^*,$$

where $^{\circledast}$ denotes the *iterated shuffle*, and $\sqcup\!\sqcup$ denotes the *shuffle operation*.

Most application oriented description languages, such as flow expressions [23] or event expressions [19, 21] use the regular operations (union, product, Kleene star), the shuffle operations (shuffle, iterated shuffle), and in addition to that further operations to control synchronization, such as lock symbols, inverse shuffle, or cancellation. These additional operations are able to mimic intersection and erasing homomorphisms which causes the corresponding language classes to equal the family of all recursively enumerable sets (see [2, 14, 19, 25]).

In [2] and in [14, 25] it has been show that flow expressions describe all the recursively enumerable sets. In fact, any rec. enum. set can be specified by some finite expression that uses only three different types of operations x, y, and z, where $x \in$ {inverse shuffle, cancellation}, $y \in$ {shuffle, product}, and $z =$ iterated shuffle [14]. Thus these combinations of operations are very powerful and the question arises, which of the many other nontrivial combinations of operations, one of them being iterated shuffle, are as powerful as the above or hopefully less complex.

More combinations of operations that can be used to define all rec. enum. sets have been studied in [14], but all of them include some type of erasing.

In [1], flow expressions have been studied that do not allow any nesting of Kleene star and iterated shuffle. It is there shown that these classes of restricted flow expressions define all, and only, homomorphic images of Petri net languages. However, these flow expressions still use the so-called lock symbols and thus again provide a mechanism for erasing.

Extending the regular operations only with shuffle and iterated shuffle avoids erasing and yields the so-called shuffle expressions [23] or equivalently C-expressions [17].

Obviously, every shuffle expression language is context sensitive and this is the best result known so far (see [16]). The claim [14, Corollary 3.14] that there exists a shuffle expression language which cannot be accepted by any multicounter machine in quasirealtime, unfortunately is based on an incorrect example. The new results presented here will show why.

Now, to gain more insight in the complexity and power of the iterated shuffle operation it is useful to first study subclasses of the shuffle expression languages. One of those is the family $\mathcal{Shuf}$, which is the smallest class of languages containing the finite sets and closed under the operations union, shuffle and iterated shuffle. This class contains nonregular sets and has been studied in [12, 14]. It was shown there that each language $L \in \mathcal{Shuf}$ is an element of the least intersection closed trio generated by the semi-Dyck set $D_1 := (ab)^{\circledast}$. This trio can be characterized using Petri nets ([9]) as the class $\mathcal{CSS}$ or else as certain restricted classes of multicounter languages as defined in [8, 12, 13]. From these characterizations it follows easily that $\mathcal{Shuf}$ is a subclass of P, the class of languages acceptable in deterministic polynomial time. Another proof for this result was later given in [26]. Both proofs relied on an important property of the family $\mathcal{Shuf}$, namely, that any language of that class can be written by some finite expression in normal form, where the iterated shuffle is not nested. Whether nesting of iterated shuffle is crucial for defining complicated languages or not, depends of course not only on the whole set of operations allowed, but also on the basic sets one starts with. For instance, it was shown in [14] that the language $L = \{ab^n cde^n f \mid n \geq 0\}^{\circledast}$ is not acceptable by any multicounter machine in quasirealtime, which is also a consequence of a result in [26], stating that this language L is NP-complete, once we assume P $\neq$ NP.

Knowing all this, it is now the question whether arbitrary nesting of iterated shuffle and Kleene star is powerful enough to also define NP-complete sets by starting with regular sets.

We show here that any language which can be defined from the regular sets by applying finitely many of the four operations union, product, Kleene star, and iterated shuffle in any order, is in fact acceptable by some nondeterministic multi-counter machine that operates in quasirealtime, and thus in P. This significantly generalizes [26, Theorem 5.1], corrects [14, Corollary 3.11] and adds to the research in [1, 2, 4, 14–26], and last but not least solves the open problem in [19].

For proving complicated equations involving shuffle operations we introduce the notion of *shuffle schemes*. These shuffle schemes combine informal graphical representations of shuffled strings with the necessary preciseness to give rigorous proofs. The advantage of this concept lies in its ability to replace clumsy string-oriented proofs by more natural graphical transformations.

Unfortunately, the question as to whether every shuffle expression language is a multicounter language, or at least an element of the class P, still remains unanswered and appears to be difficult to solve.

1. Notation and basic definitions

To have a concise description for classes of languages defined by finite expressions we use the following notation which is in accordance with standard formal language theory. The notation used in [14] is partly repeated here.

A *language* L is a subset of the free monoid X^* generated by the finite alphabet X. The *length* of a string $w \in X^*$ is denoted by $\lg(w)$, and $\mathrm{card}(X)$ denotes the *cardinality* of X.

A *family of languages* $\mathscr{L}$ is a nonempty set of languages which is closed under isomorphisms, i.e. change of alphabets, such that not all elements $L \in \mathscr{L}$ are the empty set.

In accordance with [3] we define, slightly simplifying the notation: A mapping $\mathcal{O}$ from families of languages to families of languages is called an *operator*, if for all families of languages $\mathscr{L}$ and $\mathscr{L}'$, $\mathscr{L} \subseteq \mathscr{L}'$ implies $\mathcal{O}(\mathscr{L}) \subseteq \mathcal{O}(\mathscr{L}')$, and for each $L \in \mathcal{O}(\mathscr{L})$ there exists a finite family $\mathscr{L}''$ such that $L \in \mathcal{O}(\mathscr{L}'')$.

If $\mathcal{O}$ and $\mathcal{O}'$ are operators satisfying $\mathcal{O}(\mathscr{L}) \subseteq \mathcal{O}'(\mathscr{L})$ for all families of languages $\mathscr{L}$, then we write $\mathcal{O} \leqslant \mathcal{O}'$. An operator $\mathcal{O}$ is called a *closure operator*, if for all families of languages $\mathscr{L}$ one has $\mathscr{L} \subseteq \mathcal{O}(\mathscr{L})$ and $\mathcal{O}(\mathcal{O}(\mathscr{L})) = \mathcal{O}(\mathscr{L})$.

If $\mathcal{O}_1, \ldots, \mathcal{O}_n$ are operators then $(\mathcal{O}_1, \ldots, \mathcal{O}_n)(\mathscr{L})$ is the least family of languages containing $\mathscr{L}$ and closed under finitely many, including none, applications of the operators $\mathcal{O}_i$, $1 \leqslant i \leqslant n$. Thus by definition $(\mathcal{O}_1, \ldots, \mathcal{O}_n)$ is a closure operator. If $\mathcal{O}$ is already a closure operator, then we write $\mathcal{O}(\mathscr{L})$ instead of $(\mathcal{O})(\mathscr{L})$.

Usually operators are defined on the basis of operations on languages. For example, if $\cdot$ is the concatenation between languages, then $\mathscr{L} \cdot \mathscr{L}' := \{L \cdot L' \mid L \in \mathscr{L}, L' \in \mathscr{L}'\}$ is the natural generalization to families of languages $\mathscr{L}$ and $\mathscr{L}'$. Hence by our notation $(\cdot)(\mathscr{L})$ denotes the closure of the family $\mathscr{L}$ under product, more precisely: the smallest family containing $\mathscr{L}$ and which is closed under product.

The expression $(\mathcal{O}_1, \ldots, \mathcal{O}_n)(\mathcal{O}'_1, \ldots, \mathcal{O}'_m)(\mathcal{L})$ is a shorthand for $(\mathcal{O}_1, \ldots, \mathcal{O}_n)((\mathcal{O}'_1, \ldots, \mathcal{O}'_m)(\mathcal{L}))$.

The following unary operators on families of languages all are closure operators. The definitions of the underlying operations on languages may be found in [3, 6].

$\hat{\mathcal{H}}$ is the class of all homomorphisms between languages.

$\mathcal{H}$ is the class of all non-erasing homomorphisms:

$$\mathcal{H} := \{h \in \hat{\mathcal{H}} \mid \lg(h(w)) \neq 0 \text{ for all } w \neq \lambda\}.$$

$\mathcal{H}^{-1}$ is the class of all inverse homomorphisms.

$\mathcal{H}^{cod}$ is the class of all codings or length-preserving homomorphisms:

$$\mathcal{H}^{cod} := \{h \in \hat{\mathcal{H}} \mid \lg(h(w)) = \lg(w) \text{ for all words } w\}.$$

$\mathcal{H}^{dec}$ is the class of all length-decreasing homorphisms, also called weak codings or alphabetical homomorphisms:

$$H^{dec} := \{h \in \hat{\mathcal{H}} \mid \lg(h(w)) \leqslant \lg(w) \text{ for all words } w\}.$$

Furthermore we shall use, as is usually done in formal language theory, the following abbreviations: $\mathcal{M} := (\mathcal{H}, \mathcal{H}^{-1}, \wedge \mathcal{R})$ is the class of all trio operations, where $\wedge \mathcal{R}$ denotes intersection with regular sets. It is well known that $\mathcal{M}(\mathcal{L}) = (\mathcal{H}^{cod}, \mathcal{H}^{-1}, \wedge \mathcal{R})(\mathcal{L})$.

Similarly, we use the abbreviations

$$\hat{\mathcal{M}} := (\hat{\mathcal{H}}, \mathcal{H}^{-1}, \wedge \mathcal{R}), \quad \mathcal{F} := (\mathcal{M}, \cdot, \wedge, +) \quad \text{and} \quad \hat{\mathcal{F}} := (\hat{\mathcal{M}}, \cdot, \vee, *).$$

All these operators $\mathcal{M}$, $\hat{\mathcal{M}}$, $\mathcal{F}$, $\hat{\mathcal{F}}$ are again closure operators. The *wedge operation* $\wedge$ is the natural generalization of language intersection to families of languages $\mathcal{L}_1$, $\mathcal{L}_2$ and is defined by

$$\mathcal{L}_1 \wedge \mathcal{L}_2 := \{L_1 \cap L_2 \mid L_1 \in \mathcal{L}_1, L_2 \in \mathcal{L}_2\}.$$

The closure of a family $\mathcal{L}$ under the wedge operation is consequently denoted by $\wedge(\mathcal{L})$, but, as it is usually done in AFL-theory, we then say that $\wedge(\mathcal{L})$ is closed under intersection instead of saying that $\wedge(\mathcal{L})$ is closed under wedge.

Similar to the wedge operation, $\vee$ is the natural *generalization of union* to families of languages $\mathcal{L}_1$ and $\mathcal{L}_2$:

$$\mathcal{L}_1 \vee \mathcal{L}_2 := \{L_1 \cup L_2 \mid L_1 \in \mathcal{L}_1, L_2 \in \mathcal{L}_2\}.$$

If $\mathcal{L} \vee \mathcal{L} \subseteq \mathcal{L}$, then $\mathcal{L}$ is called *union-closed.* The regular operations of product, Kleene plus, and Kleene star are generalized to families of languages in the obvious way.

Let λ denote the empty word. For any family of languages $\mathcal{L} \neq \{\{\lambda\}\}$ let $\mathcal{L}^{\Delta} := \{L - \{\lambda\} \mid L \in \mathcal{L}\}$ (note that $\Delta(\mathcal{L}) = \mathcal{L} \cup \mathcal{L}^{\Delta}$).

Let $\mathcal{F}in$ ($\mathcal{R}eg$, $\mathcal{CSS}$, $\mathcal{RE}$, respectively) be the family of finite sets (regular sets, computation sequence sets, recursively enumerable sets, respectively). For definitions of $\mathcal{CSS}$ in terms of Petri nets or counter machines see [4, 8, 11-13]. It is there

shown that $\mathscr{CSS} = (\mathscr{M}, \wedge)(\{D_1\})$, where D_1 is the semi-Dyck language over one pair of brackets. In [11, 12] it has been proved that $\mathscr{CSS} \neq \mathscr{H}^{dec}(\mathscr{CSS})$.

As in [14] the *shuffle operation* will be denoted by the symbol ш and is defined on languages L and K by:

$$L \text{ ш } K := \{w \in X^* \mid w = u_1 v_1 u_2 v_2 \dots u_n v_n,\ u_i, v_i \in X^*;$$
$$u_1 u_2 \dots u_n \in L,\ v_1 v_2 \dots v_n \in K\}.$$

The shuffle operation is generalized to families of languages in the obvious way.

The *iterated shuffle* was introduced in [20] and there denoted by a dagger. This symbol, however, can be confused with Kleene plus, so we choose the following notation from [1, 2, 23]. Let L be a language. Then

$$L^{\circledast} := \bigcup_{i \geq 0} L_i \quad \text{and} \quad L^{\oplus} := \bigcup_{i \geq 1} L_i,$$

where $L_0 := \{\lambda\}$ and $L_{i+1} := L_i \text{ ш } L$.

Let $\mathbb{N} := \{0, 1, 2, \dots\}$ be the set of nonnegative integers. If $X := \{x_1, x_2, \dots, x_n\}$ is an ordered finite alphabet then the *Parikh mapping* $\psi : X^* \to \mathbb{N}^n$ is defined by

$$\psi(w) := (\#_{x_1}(w), \#_{x_2}(w), \dots, \#_{x_n}(w)),$$

where $\#_{x_i}(w)$ is the number of occurrence of the symbol x_i in the word w.

2. Basic structural properties

In what follows we define classes of languages by using the operations union, product, Kleene star, shuffle, and iterated shuffle.

To denote the languages from these families we will use expressions without formally defining the corresponding classes of expressions, since this could be done in the standard way. Any such expression can be obtained from a finite set-theoretic description of a language mainly by replacing the braces { and } by the brackets (and), omitting superfluous brackets. Since no confusion seems possible we do not distinguish between expression and the languages defined by them. We define:

$$\mathscr{Shuf} := (\vee, \text{ш}, \circledast)(\mathscr{Fin}),$$
$$\mathscr{ER} := (\vee, \cdot, *, \circledast)(\mathscr{Fin}),$$
$$\mathscr{SE} := (\vee, \cdot, *, \text{ш}, \circledast)(\mathscr{Fin}).$$

The class $\mathscr{Shuf}$ has been studied in [11], some of the results will be mentioned in what follows. The class $\mathscr{SE}$ is the class of languages definable by shuffle expressions [23] or C-expressions [17]. The class $\mathscr{ER}$ may be defined through extended regular expressions and is the class mainly studied in this work. We solve the open problem in [19], which asks for the complexity of the class $\mathscr{ER}$.

For many of the proofs it is necessary to transform finite expressions into equivalent ones by means of certain transformation rules. The simplest ones are more or less known from the literature, but worth noticing.

Lemma 2.1. *The following equations are valid for arbitrary languages A, B, and C:*

(1) $A \sqcup B = B \sqcup A.$

(2) $(A \sqcup B) \sqcup C = A \sqcup (B \sqcup C).$

(3) $A \sqcup (B \cup C) = (A \sqcup B) \cup (A \sqcup C).$

(4) $(A \cup B)^{\circledast} = A^{\circledast} \sqcup B^{\circledast}.$

(5) $(A \cup B)^{\oplus} = (A^{\oplus} \sqcup B^{\oplus}) \cup A^{\oplus} \cup B^{\oplus}.$

(6) $(A^*)^{\circledast} = (A^{\circledast})^* = (A^{\circledast})^{\circledast} = (A^*)^{\oplus} = (A^{\circledast})^+ =$
$(A^+)^{\circledast} = (A^{\oplus})^* = (A^{\circledast})^{\oplus} = (A^{\oplus})^{\circledast} = A^{\circledast}.$

(7) $(A^{\oplus})^{\oplus} = (A^{\oplus})^+ = (A^+)^{\oplus} = A^{\oplus}.$

(8) $(A^{\circledast} \sqcup B)^{\circledast} = ((A \cup B)^{\circledast} \sqcup B) \cup \{\lambda\}.$

These equations were the basis for the normal form theorem for the family $\mathcal{Shuf}$ in [12], reported in [14], and used in [26]. Since [12] is not available everywhere and the result can easily be generalized we give an explicit proof here in Section 4. In order to do that, we first have to collect a few results which describe useful relations between the operators and which can then be used for inductive proofs quite easily.

Lemma 2.2. *The following relations are valid:*

(1) $(\vee)(\nabla) = (\nabla)(\vee).$

(2) $(*)(\nabla) = (\nabla)(*).$

(3) $(\circledast)(\nabla) = (\nabla)(\circledast).$

(4) $(\sqcup)(\vee) \leqslant (\vee)(\sqcup).$

(5) $(\cdot)(\vee) \leqslant (\vee)(\cdot).$

(6) $(\cdot)(\nabla) \leqslant (\nabla)(\vee)(\cdot).$

(7) $(\sqcup)(\nabla) \leqslant (\nabla)(\vee)(\sqcup).$

(8) $(\oplus)(\vee) \leqslant (\vee)(\sqcup)(\oplus).$

(9) $(\circledast)(\vee) \leqslant (\vee)(\sqcup)(\circledast).$

Proof. Recall that $(\nabla)\mathcal{L} = \mathcal{L} \cup \mathcal{L}^{\nabla} = \{L, L \cup \{\lambda\} \mid L \in \mathcal{L}\}$. Relations (1) to (7) are trivial consequences of Lemma 2.1. Relation (8) follows from Lemma 2.1, equations (5) and (7); and for the proof of relation (9), equations (4) and (6) have to be used. □

Theorem 2.3. *If $\mathcal{L}$ is a family of languages which contains the language $\{\lambda\}$ and is closed with respect to the operation ш, then*

$$(\vee, ш, \circledast)(\mathcal{L}) = (\vee)(ш)(\circledast)(\mathcal{L}).$$

Proof. The proof immediately follows from Theorem A.2. □

Corollary 2.4

(1) $\mathcal{Shuf} = (\vee)(ш)(\circledast)(\mathcal{Fin})$.

(2) $(\vee, ш, \circledast)(\mathcal{Reg}) = (\vee)(ш)(\circledast)(\mathcal{Reg})$.

(3) $(\vee, ш, \circledast)(\mathcal{CSS}) = (\vee)(ш)(\circledast)(\mathcal{CSS})$.

Notice that in a commutative monoid the operators ш and $\circledast$ are equivalent to product and star, respectively. Following [5], a subset of the free commutative monoid M (usually this is $\mathbb{N}^n$) is called *semilinear* if it is the finite union of linear sets $X = \{a\} + B^*$, with $a \in M$, $B \subseteq M$, and B finite. Each expression for a language from the class $(\vee)(ш)(\circledast)(\mathcal{Fin})$ can be transformed by using the equations of Lemma 2.1 so that one can see that this family is another notation for the class of semilinear sets (modulo the Parikh mapping). The family $(\vee, ш, \circledast)(\mathcal{Fin})$ then corresponds to the rational subsets of M. It is easy to verify that Corollary 2.4(1) describes the normal form result for the family $\mathcal{Shuf}$ obtained in [12].

Corollary 2.5

$$(\vee, ш, \circledast)(\mathcal{Reg}) \subseteq \mathcal{CSS} = (\mathcal{M}, \wedge)(D_1).$$

Proof. This follows from Corollary 2.4(2) and the fact that the family $\mathcal{CSS}$ is closed under union and shuffle, and contains the iterated shuffle of any regular set. $\mathcal{CSS} = (\mathcal{M}, \wedge)(D_1)$ has been proven in [8] and [11], where also the closure under $\vee$ and ш has been verified, see [9]. $R^{\circledast} \in \mathcal{CSS}$ for any regular set R is Lemma IV.6 in [7] and follows also from the proof of Theorem 5.1 in [26], where it has been shown that $R^{\circledast} \in \mathrm{P}$ by using the states of a finite automaton for R as counters for pebbles. □

If one is interested in the effect that the erasing of the empty word in languages has on the finite presentation of classes defined through operations from the set $\{\vee, \cdot, ш, *, \circledast\}$, the reader should consult [15], where the formal proofs of the following results can be found.

Theorem 2.6. *For any family of languages $\mathscr{L}$,*

$$[(\vee, \cdot, ш, *, \circledast)(\mathscr{L})]^{\Delta} = (\vee, ш, \cdot, +, \oplus)(\mathscr{L}^{\Delta}).$$

Theorem 2.7. *For any family of languages $\mathscr{L}$,*

$$[(\vee, ш, \circledast)(\mathscr{L})]^{\Delta} = (\vee)(ш)(\oplus)(ш)(\mathscr{L}^{\Delta}).$$

Corollary 2.8

(1) $\mathscr{SE}^{\Delta} = (\vee, \cdot, ш, +, \oplus)(\mathscr{Fin}^{\Delta})$.
(2) $\mathscr{Shuf}^{\Delta} = (\vee)(ш)(\oplus)(\mathscr{Fin}^{\Delta})$.
(3) $\mathscr{ER}^{\Delta} \subseteq (\vee, \cdot, +, \oplus)(\mathscr{Fin}^{\Delta})$.

The reason why equality cannot be proved for (3) is that one has to use the shuffle operation in order to represent a set $A^{\oplus} = A^{\circledast} ш A$.

We conjecture that the family $(\vee, \cdot, +, \oplus)(\mathscr{Fin}^{\Delta})$ is not contained in the family $\mathscr{ER}$. It would be sufficient to show that the set $(abc)^{\oplus}$ is not an element of $\mathscr{ER}$. Note, however, that the set $(ab)^{\oplus}$ is an element of $\mathscr{ER}$ since it can be written as $(ab)^{\oplus} = (a)(ab)^{\circledast}(b)(ab)^{\circledast}$.

It is easily seen from the definitions that the families $\mathscr{Shuf}$, $\mathscr{ER}$, and $\mathscr{SE}$ are closed under decreasing homomorphisms. Using the preceding results one also gets $[\mathscr{H}^{dec}(\mathscr{SE}^{\Delta})]^{\Delta} = \mathscr{SE}^{\Delta}$ and likewise the classes $\mathscr{Shuf}^{\Delta}$ and $\mathscr{ER}^{\Delta}$ are closed under decreasing homomorphisms, i.e. codings combined with erasing, modulo the empty word.

A special type of erasing which is very useful, and in fact needed to prove our main theorem in the next section, is the deletion of certain symbols by using so-called k-limited erasing. We give a definition.

Definition. A homomorphism $h \in \hat{\mathscr{H}}$, $h: X^* \to Y^*$, is called *k-limited erasing on a word* $w \in X^*$ if for each decomposition $w = v_1 u v_2$, $h(u) = \lambda$ implies $\lg(u) \leqslant k$.

h is called *k-limited erasing on a language* $L \subseteq X^*$ if h is k-limited erasing on each word $w \in L$.

Lemma 2.9. *Let $L \in (\vee, \cdot, ш, +, *, \oplus, \circledast)(\mathscr{Fin})$ and $h \in \mathscr{H}^{dec}$ then there exists a constant c, depending only on L, such that for each $w \in L$ there exists a $v \in L$ with $h(w) = h(v)$ and h is c-limited erasing on v.*

Proof. Let us define the constant $c := c(L)$ inductively as follows:

(1) For $L \in \mathscr{Fin}$ let $c(L) := \max\{\lg(w) \mid w \in L\}$.
(2) $c(L_1 \cup L_2) := \max\{c(L_1), c(L_2)\}$.
(3) $c(L_1 \cdot L_2) := c(L_1 ш L_2) := c(L_1) + c(L_2)$.
(4) $c(L_1^+) := c(L_1^*) := c(L_1^{\oplus}) := c(L_1^{\circledast}) := 2 \cdot c(L_1)$.

We prove the lemma by structural induction.

Basis. $L \in \mathscr{Fin}$ is trivially true.

Induction steps:

(1) If $L = L_1 \cup L_2$, then $w \in L$ implies $w \in L_1$ (or $w \in L_2$, which is symmetrical) and the induct. hyp. implies the existence of a $v \in L_1 \subseteq L$ such that $h(w) = h(v)$ and h is $c(L_1)$-limited erasing on v. Since $c(L_1) \leqslant c(L)$, h is also $c(L)$-limited erasing on $v \in L$.

(2) If $L = L_1 \cdot L_2$, then $w \in L$ implies $w = w_1 \cdot w_2$ with $w_1 \in L_1$, $w_2 \in L_2$. Then there exist $v_i \in L_i$ such that h is $c(L_1)$-limited erasing on v_1 and $c(L_2)$-limited erasing on v_2. Thus h is $c(L) = (c(L_1) + c(L_2))$-limited erasing on $v_1 v_2 \in L$.

(3) If $L = L_1 \sqcup\!\sqcup L_2$, the argumentation is similar to case (2) above and omitted.

(4) If $L = L_1^+$ (or L_1^*), then $w \neq \lambda$ (the case $w = \lambda$ is always trivially true) can be decomposed into $w = w_1 w_2 \ldots w_n$, $n \geqslant 1$, $w_i \in L_1$.

If $h(w_i) = \lambda$ for some i then one can iteratively delete these subwords, finally yielding a word $w' \in L_1^*$, $w' = w_1' w_2' \ldots w_m'$, with $h(w) = h(w')$ and $h(w_i') \neq \lambda$ for all $1 \leqslant i \leqslant m$, $w_i' \in L_1$. Now for each w_i' there exists a v_i such that $h(w_i') = h(v_i)$ and h is $c(L_1)$-limited erasing on each v_i. Since $h(v_i) \neq \lambda$ it is immediately seen that h is $2 \cdot c(L_1) = c(L_1^*)$-limited erasing on $v_1 \ldots v_m \in L_1^*$.

(5) $L = (L_1^{\oplus})$ or $L = (L_1)^{\circledast}$. In this case the argumentation is a bit more complicated, since by shuffling words $w_i \in L_1$ into each other one can obtain arbitrarily long subwords which are to be erased by h, even if $h(w_i) \neq \lambda$ for each i. However, if each w_i is decomposed into $w_i := w_{i_1} a_{i_1} w_{i_2} a_{i_2} \ldots w_{i_n} a_{i_n} w_{i_{n+1}}$, where $h(w_{i_j}) = \lambda$, $h(a_{i_j}) \neq \lambda$ then for each word $w \in w_1 \sqcup\!\sqcup \cdots \sqcup\!\sqcup w_k$, $k \in \mathbb{N}$ one can find another word $v \in w_1 \sqcup\!\sqcup \cdots \sqcup\!\sqcup w_k$ such that $h(w) = h(v)$ and h is $2 \cdot c(L_1)$-limited erasing on v just by first pasting together the subwords $w_{i_j} a_{i_j}$, $1 \leqslant j \leqslant n-1$, and $w_{i_n} a_{i_n} w_{i_{n+1}}$ and treating those as indivisible 'symbols'. A more formal proof is technical and omitted. □

3. The class $\mathscr{ER}$

The subject of this section is to prove the main theorem, i.e., that each language $L \in \mathscr{ER}$ can be accepted by some nondeterministic one-way multicounter machine in quasirealtime. In doing this, we shall not use counter machines, instead we will use the characterization of this class of languages as the least intersection closed AFL containing the semi-Dyck language $D_1 := (ab)^{\circledast}$. Consequently we denote this class by $(\mathscr{F}, \wedge)(D_1)$ (see [6, 8, 11]). The proof is quite involved and needs a number of definitions that we shall explain first.

The proof will be done by structural induction on the depth of the expressions that described languages from the class $\mathscr{ER}$. Roughly speaking we start with the sets of the family $\mathscr{Shuf}$ of which we know by Corollary 2.4 that they are elements of the family $(\mathscr{F}, \wedge)(D_1)$. Knowing that this family is not closed under iterated shuffle the following expressions of depth $k+1$ are of main interest: $(A^*)^{\circledast}$, $(A^{\circledast})^{\circledast}$, $(A \cup B)^{\circledast}$, and $(A \cdot B)^{\circledast}$. By the calculation rules of Lemma 2.1 the first three expressions do not cause problems.

However, to describe the set $(A \cdot B)^{\circledast}$ by some expressions of depth k that use only the sets A and B together with operations under which the family $(\mathscr{F}, \wedge)(D_1)$ is closed does create new problems that fall back to the other simpler cases. This is why the proof of our main result needs the definition of certain well structured expressions, called marked expressions, which can be used to describe a subclass of the family $\mathscr{ER}$ and which it is easier to deal with.

Definition. (1) If $F = \{w_1, \ldots, w_n\} \subseteq X^+$ is a finite set of nonempty strings, then $(w_1, \ldots, w_n)$ is the marked expression denoting F with $\text{depth}(F) := 0$.

(2) If E_1 and E_2 are marked expressions and x, y, $\bar{x}$, $\bar{y}$ are four different symbols not used in the expressions E_1 and E_2, then the following expressions E are marked expressions of $\text{depth}(E) := \max\{\text{depth}(E_1), \text{depth}(E_2)\} + 1$:

$$E := ((xE_1y) \cdot (\bar{x}E_2\bar{y})), \quad ((xE_1y) \cup (\bar{x}E_2\bar{y})), \quad ((xE_1y)^*) \quad \text{and} \quad ((xE_1y)^{\circledast}).$$

(3) Nothing else is a marked expression.

Let $m\mathscr{ER}$ be the class of all languages definable by marked expressions. If $L \in m\mathscr{ER}$ then $\text{depth}(L) := \min\{\text{depth}(E) \mid E \text{ is a marked expression denoting } L\}$.

Obviously $\mathscr{ER} = \mathscr{H}^{d\text{-}lec}(m\mathscr{ER})$.

We now show that any language definable by a marked expression can be accepted by a nondeterministic multicounter machine in quasirealtime, which is equivalent to showing $m\mathscr{ER} \subseteq (\mathscr{F}, \wedge)(D_1)$.

Theorem 3.1. $m\mathscr{ER} \subseteq (\mathscr{F}, \wedge)(D_1)$.

Proof. We use induction on $\text{depth}(L)$ for all $L \in m\mathscr{ER}$.

Basis. Clearly, if $L \in m\mathscr{ER}$ and $\text{depth}(L) = 0$, then L is a finite set and as such an element of $(\mathscr{F}, \wedge)(D_1)$.

If $L \in m\mathscr{ER}$ and the operation $\circledast$ is not used in the marked expression denoting L, then L is regular and thus an element of $(\mathscr{F}, \wedge)(D_1)$ for any $\text{depth}(L) \geq 0$. If $L \in m\mathscr{ER}$ is not regular and $\text{depth}(L) = 1$, then $L = (xEy)^{\circledast}$ for some finite set E. Also in this case $L \in (\mathscr{F}, \wedge)(D_1)$ since the latter family contains the iterated shuffle of any regular set (see Corollary 2.5 and its proof).

Induction step. Assume as induction hypothesis that the theorem is true for all $L \in m\mathscr{ER}$ of depth k or less for some $k \geq 1$. The two cases $k = 0$ and $k = 1$ are covered by the basic step. Let $L \in m\mathscr{ER}$ be such that $\text{depth}(L) = k + 1$. The following cases are possible.

Case 1: $L = ((xE_1y) \cdot (\bar{x}E_2\bar{y}))$.

Case 2: $L = ((xE_1y) \cup (\bar{x}E_2\bar{y}))$.

Case 3. $L = (xE_1y)^*$.

Case 4. $L = (xE_1y)^{\circledast}$.

In all these four cases E_1 and E_2 are expressions of depth k or less. Therefore in

Cases 1 to 3, L is in $(\mathscr{F}, \wedge)(D_1)$ since this family is closed under product, union, and Kleene star.

Since the family $(\mathscr{F}, \wedge)(D_1)$ is not closed with respect to iterated shuffle, it does not immediately follow that in Case 4 L is an element of $(\mathscr{F}, \wedge)(D_1)$.

Thus we have to consider Case 4 in more detail: since $k+1 \geq 2$ we find the following subcases of Case 4:

Subcase 4.1. $L = ((x((x_1E_1y_1) \cdot (x_2E_2y_2))y)^{\circledast}$, where E_1 and E_2 have depth $k-1$ or less. By Theorem 3.6 below we then find

$$L = ((x_1E_1y_1)^{\circledast} \sqcup\!\sqcup (x_2E_2y_2)^{\circledast} \sqcup\!\sqcup (xy)^{\circledast}) \cap ((xx_1y_1x_2y_2y)^{\circledast} \sqcup\!\sqcup X^*),$$

where X is some finite alphabet.

Now $\operatorname{depth}((x_iE_iy_i)^{\circledast}) \leq k$, for $1 \leq i \leq 2$, and the induction hypothesis shows $(x_iE_iy_i)^{\circledast}$, $(xx_1y_1x_2y_2y)^{\circledast}$, $(xy)^{\circledast}$, and $X^* \in (\mathscr{F}, \wedge)(D_1)$. L is obtained from these sets by operations, under which the family $(\mathscr{F}, \wedge)(D_1)$ is closed, therefore $L \in (\mathscr{F}, \wedge)(D_1)$, too.

Subcase 4.2. $L = (x((x_1E_1y_1) \cup (x_2E_2y_2))y)^{\circledast}$, where $\operatorname{depth}(E_i) \leq k-1$ for $1 \leq i \leq 2$. Then, by Theorem 3.8 below,

$$L = (((xy)^{\circledast} \sqcup\!\sqcup (x_1E_1y_1)^{\circledast}) \cap ((xx_1y_1y)^{\circledast} \sqcup\!\sqcup X^*))$$
$$\sqcup\!\sqcup (((xy)^{\circledast} \sqcup\!\sqcup (x_2E_2y_2)^{\circledast}) \cap ((xx_2y_2y)^{\circledast} \sqcup\!\sqcup X^*))$$

for some finite alphabet X.

As in Subcase 4.1 this shows $L \in (\mathscr{F}, \wedge)(D_1)$, since by the induction hypothesis $(x_iE_iy_i)^{\circledast}$, $1 \leq i \leq 2$, and all the other sets involved are in $(\mathscr{F}, \wedge)(D_1)$.

Subcase 4.3. $L = (x(x_1E_1y_1)^*y)^{\circledast}$, with $\operatorname{depth}(E_1) \leq k-1$. Then, by Theorem 3.9 below,

$$L = ((x_1E_1y_1)^{\circledast} \sqcup\!\sqcup (xy)^{\circledast}) \cap ((x(x_1y_1)^*y)^{\circledast} \sqcup\!\sqcup X^*),$$

where $(x_1E_1y_1)^{\circledast}$, $(xy)^{\circledast}$, X^*, and $(x(x_1y_1)^*y)^{\circledast} \in (\mathscr{F}, \wedge)(D_1)$ show $L \in (\mathscr{F}, \wedge)(D_1)$.

Subcase 4.4. $L = (x(x_1E_1y_1)^{\circledast}y)^{\circledast}$ with $\operatorname{depth}(E_1) \leq k-1$. Then, by Theorem 3.11 below,

$$L = (x(x_1E_1y_1)^{\circledast}y)^* \sqcup\!\sqcup (xy)^{\circledast},$$

and again $L \in (\mathscr{F}, \wedge)(D_1)$ by the induction hypothesis and the closure properties of $(\mathscr{F}, \wedge)(D_1)$.

Hence by induction we have $m\mathscr{ER} \subseteq (\mathscr{F}, \wedge)(D_1)$. □

Combining Theorem 3.1 and Lemma 2.9 we obtain the main result of this section.

Theorem 3.2. $\mathscr{ER} \subseteq (\mathscr{F}, \wedge)(D_1)$.

Proof. For each $L \in \mathscr{ER}$, $L \subseteq X^*$, there exists a language $L' \in m\mathscr{ER}$, $L' \subseteq Y^*$ and a homomorphism $h \in \mathscr{H}^{dec}$, $h: Y^* \to X^*$, such that $L = h(L')$. By Lemma 2.9 there is

a constant c such that $h(L') = h(L' \cap R)$ where R is the regular set

$$R := \{w \in Y^* \mid h \text{ is } c\text{-limited erasing on } w\}.$$

Thus h is c-limited erasing on $L' \cap R$.

By Theorem 3.1, $L' \in (\mathscr{F}, \wedge)(D_1)$ and since $(\mathscr{F}, \wedge)(D_1)$ is closed under intersection with regular sets and under c-limited erasing homomorphism (see [6]) finally $L = h(L' \cap R) \in (\mathscr{F}, \wedge)(D_1)$. □

Note that the family $(\mathscr{F}, \wedge)(D_1)$ is not closed under arbitrary decreasing homomorphisms, so that Lemma 2.9 is crucial for proving the main result.

Corollary 3.3. *The languages from the family $\mathscr{CR}$ are acceptable in nondeterministic space* $\log n$, *hence in deterministic space* $(\log n)^2$ *or in deterministic polynomial time.*

Theorem 3.1 is the basis for our main result: Theorem 3.2 and its Corollary 3.3. However, its proof is still not finished completely since it refers to four important and nontrivial technical results. The proofs for these, Theorems 3.6, 3.8, 3.9, and 3.11, are not obvious and we therefore introduce the new concept of shuffle schemes which shall be defined after some preliminary examples.

When dealing with the shuffle operation, string-oriented proofs about certain valid equalities tend to hide the main idea behind notational monstrosities unless they are informal and not precise. But very often proofs of the type 'it is easy to see ...' are risky and, especially for the shuffle operations, intuition may suggest the false. To give the reader examples, here are some equations of which the author thought for quite a while that they all were true. The reader is kindly invited to test his or her intuition about the validity of the following 'equations'.

Example 3.4. (1) For distinct symbols a, b, c:

$$(abc)^{\circledast} = ((ab)^{\circledast} \sqcup\!\sqcup (c)^{\circledast}) \cap ((bc)^{\circledast} \sqcup\!\sqcup (a)^{\circledast})?$$

(2) For *finite sets* A, B, C over *disjoint* alphabets:

$$(ABC)^{\circledast} = ((AB)^{\circledast} \sqcup\!\sqcup C^{\circledast}) \cap ((BC)^{\circledast} \sqcup\!\sqcup A^{\circledast})?$$

(3) *For distinct symbols* \$, ¢, a, b:

$$(\$(ab)^{\circledast}¢)^{\circledast} = (\$(ab)^{\circledast}¢)^* \sqcup\!\sqcup (\$c)^{\circledast}?$$

(4) For distinct symbols \$, ¢, a, b:

$$(\$[(ab)^{\circledast} \sqcup\!\sqcup ab]¢)^{\circledast} = (\$(ab)^{\circledast}¢)^* \sqcup\!\sqcup (\$ab¢)^{\circledast}?$$

Which of these four equations are true? It is not hard to show that equation (1) is true: "$\subseteq$" is seen as follows: One has $abc \in (ab)^{\circledast} \sqcup\!\sqcup (c)^{\circledast} = (ab, c)^{\circledast}$ and thus $(abc)^{\circledast} \subseteq (ab)^{\circledast} \sqcup\!\sqcup c^{\circledast}$. By the same reasoning $(abc)^{\circledast} \subseteq (bc)^{\circledast} \sqcup\!\sqcup (a)^{\circledast}$ thus the left-hand side of (1) is contained in the right-hand side of (1).

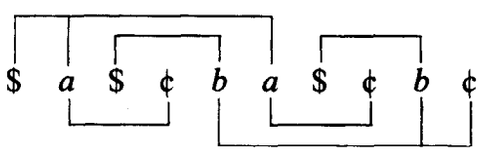
To see the converse, we do induction on the number of b's in a word $w \in (ab, c)^{\circledast} \cap (bc, a)^{\circledast}$. Obviously $\#_a(w) = \#_b(w) = \#_c(w)$. If $\#_b(w) = 1$, then obviously $w = abc \in (abc)^{\circledast}$; if $\#_b(w) = n$, then $w = u_1 a u_2 b u_3 c u_4$, since $w \in (ab, c)^{\circledast}$ (respectively $w \in (bc, a)^{\circledast}$) implies that to the left-hand (respectively right-hand) side of each occurrence of a symbol b in w there must exist an occurrence of an a in w (respectively c in w).

By induction $u_1 u_2 u_3 u_4 \in (abc)^{\circledast}$ so that $w \in (abc)^{\circledast}$, too.

Surprisingly, the similar equation (2) is false, as can be seen by the following example: Let

$$A := \{\$\}, \qquad B := \{a, aa, b, bb\}, \qquad C := \{\cent\}.$$

Then for $w := \$a\$\cent ba\$\cent b\cent$ one can easily show that $w \notin (ABC)^{\circledast}$. On the other hand, $w \in ((AB)^{\circledast} \sqcup C^{\circledast}) \cap ((BC)^{\circledast} \sqcup A^{\circledast})$, as can be seen by looking at the following decompositions of w into scattered subwords:



Equation (3) is less obvious but is true in the even more general form

$$(\$A^{\circledast}\cent)^{\circledast} = (\$A^{\circledast}\cent)^{*} \sqcup (\$\cent)^{\circledast},$$

where $A \subseteq X^{*}$ is an arbitrary set and $\$, \cent \notin X$. The proof of this result is given later in the text (Theorem 3.11).

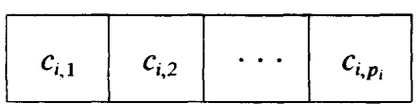
Equation (4) then is again wrong, as can be seen by looking at the following word:

$$w := \$aa\$bbaa\cent bb\cent,$$

for which $w \in (\$[(ab)^{\circledast} \sqcup ab]\cent)^{\circledast}$ but $w \notin (\$(ab)^{\circledast}\cent)^{*} \sqcup (\$ab\cent)^{\circledast}$.

In order to simplify proofs about equations similar to (1) and (3) of Example 3.4 we introduce a graphical notation, called *shuffle-schemes*, for words $v \in \{w_1\} \sqcup \cdots \sqcup \{w_n\}$, where the w_i are single words.

Definition. Let $w_1, \ldots, w_n \in X^{+}$ be given nonempty words. A *shuffle scheme S for* $v \in \{w_1\} \sqcup \cdots \sqcup \{w_n\}$ is a two-dimensional arrangement of $\#(S) := \sum_{i=1}^{n} \lg(w_i) = \lg(v)$ individual, square *cells* $c_{i,j}$, where $1 \leq i \leq n$, $1 \leq j \leq \lg(w_i)$, each of which has two different inscriptions: $S\text{-}cont(c_{i,j}) \in X$ and $S\text{-}num(c_{i,j}) \in \{1, \ldots, \#(S)\}$. These cells $c_{i,j}$ are arranged in n rows of the form



,

one directly underneath the other where

$$p_i = \lg(w_i), \qquad cont(c_{i,1})cont(c_{i,2}) \ldots cont(c_{i,p_i}) = w_i,$$

and

$$num(c_{i,1}) < num(c_{i,2}) < \cdots < num(c_{i,p_i}).$$

In addition we require that

$$\{num(c_{i,j}) \mid 1 \leq i \leq n, 1 \leq j \leq \lg(w_i)\} = \{1, 2, \ldots, \#(S)\},$$

which makes *num* to a bijection between the cells and their numberings. Instead of $num^{-1}(i)$ we shall write *S-cell*(i) (or shortly *cell*(i) if S is known) to denote the ith cell of S. (With respect to the numbering given by *num*.)

If a cell c of S has the inscription $x = cont(c)$, then we shall call c an x-cell. For any cell c let $\dot{c}$ (respectively $c\dot{}$) denote the cell immediately to the left (respectively right) of the cell c if it exists. If c_1 and c_2 are two cells of the very same row of S and $num(c_1) < num(c_2)$, then $[c_1 \ldots c_2]$ denotes that part of this row which contains all the cells $c_1, \dot{c}_1, \ddot{c}_1, \ldots, \dot{c}_2, c_2$. Likewise $[c_1 \ldots\ ]$ or $[\ \ldots c_2]$ shall denote the whole final (initial) part of this row including the cells $c_1, \dot{c}_1, \ddot{c}_1, \ldots$ (or $c_2, \dot{c}_2, \ddot{c}_2, \ldots$). The *unique word* $|S|$ determined by S is finally defined by $|S| := cont(cell(1))cont(cell(2)) \ldots cont(cell(\#(S)))$.

Example 3.5. Let $w_1 = aba$, $w_2 = aaba$, $w_3 = bab$ and $v := aababababa \in \{w_1\} \sqcup\!\sqcup \{w_2\} \sqcup\!\sqcup \{w_3\}$. Then v can be written in at least two different ways (see Fig. 1).

$S_1 :=$

a,1	b,3	a,4	
a,2	a,6	b,7	a,10
b,5	a,8	b,9	

$S_2 :=$

a,4	b,7	a,10	
a,1	a,2	b,5	a,6
b,3	a,8	b,9	

where $v = |S_1| = |S_2|$.

Fig. 1.

Now, we have the proper terminology available to formulate the proofs of the important technical results, necessary for the proof of Theorem 3.1.

Theorem 3.6. *Let* $L_1 := (\$(xAy) \cdot (\bar{x}B\bar{y})\phi)^{\circledast}$, *where* $A, B \subseteq X^*$ *are arbitrary sets and* $\$, \phi, x, y, \bar{x}, \bar{y} \notin X$. *Let*

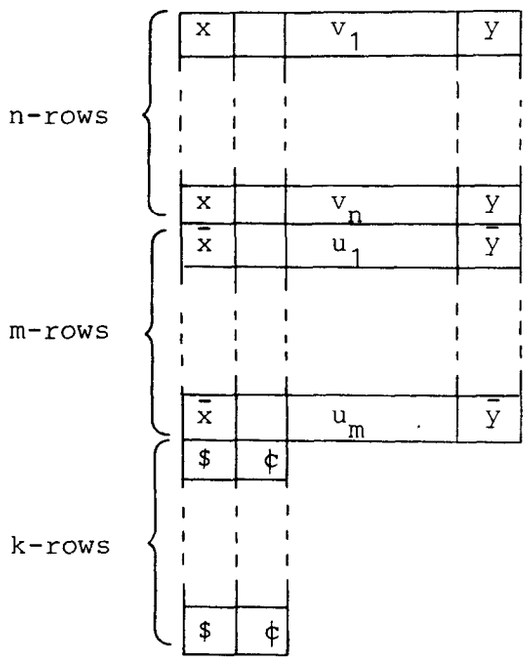
$$R_1 := (xAy)^{\circledast} \sqcup\!\sqcup (\bar{x}B\bar{y})^{\circledast} \sqcup\!\sqcup (\$\phi)^{\circledast}, \qquad R_2 := (\$(xy\bar{x}\bar{y})\phi)^{\circledast} \sqcup\!\sqcup X^{\circledast}.$$

Then

$$L_1 = R_1 \cap R_2.$$

Proof. Obviously $\$(xAy) \cdot (\bar{x}B\bar{y})\text{¢} \subseteq R_1$ (and $\subseteq R_2$). Since $R_1^{\circledast} = R_1$ (using the equation $(M_1 \cup M_2)^{\circledast} = M_1^{\circledast} \sqcup\!\sqcup M_2^{\circledast}$) and $R_2^{\circledast} = R_2$, one has $L_1 \subseteq R_1^{\circledast} = R_1$ and $L_1 \subseteq R_2^{\circledast} = R_2$ thus $L_1 \subseteq R_1 \cap R_2$.

To show $R_1 \cap R_2 \subseteq L_1$ we use the concept of shuffle-schemes. Let $w \in R_1$. Then there exists a shuffle-scheme S_1 with $w = |S_1|$, which is of the form depicted in Fig. 2, where only $cont(c)$ is written into each cell c.



where $v_i \in A$ for $1 \leq i \leq n$ and $u_j \in B$ for $1 \leq j \leq m$ for some $n, m, k \geq 0$.

Fig. 2.

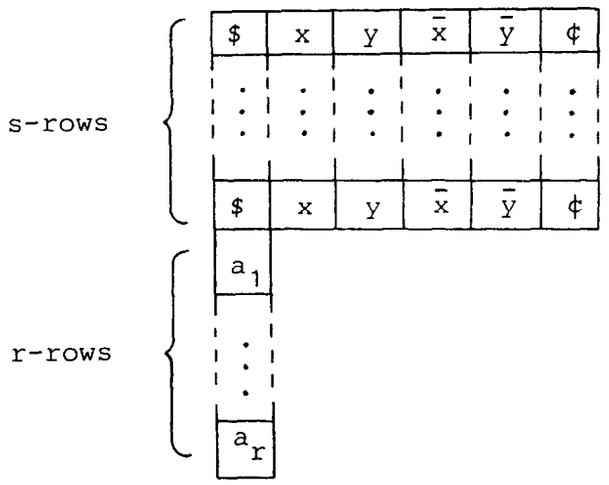
Let $w \in R_2$, then there exists a shuffle-scheme S_2 with $w = |S_2|$ of the form shown in Fig. 3.

Now $w \in R_1 \cap R_2$ implies $n = m = s = k$, $r = \sum_{i=1}^{n} (\lg(v_i) + \lg(u_i))$, and the number of z-cells in S_1 equals the number of z-cells in S_2 for each $z \in X \cup \{\$, \text{¢}, x, y, \bar{x}, \bar{y}\}$. More specifically: $S_1\text{-}cont(cell(i)) = S_2\text{-}cont(cell(i))$ for each $1 \leq i \leq \#(S_1) = \#(S_2)$.

We shall show that one can always construct a new shuffle-scheme S such that still $|S| = w$ and S has the form depicted in Fig. 4.

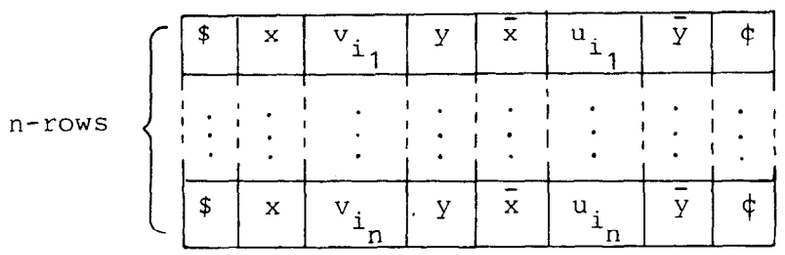
If this is possible, then obviously $w \in L = (\$(xAy) \cdot (\bar{x}B\bar{y})\text{¢})^{\circledast}$.

To construct the new shuffle-scheme S we shall use a *cut and paste* method, applied iteratively to the shuffle-scheme S_1, that will end up with the shuffle-scheme S. The cut and paste steps will uniquely be determined by the shuffle-scheme S_2.



where $a_i \in X$ for $1 \leq i \leq r$, $r \geq 0$.

Fig. 3.



where $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$, hence $v_{i_j} \in A$, $u_{i_j} \in B$.

Fig. 4.

Algorithm 3.7

Step 1. Start with the shuffle-scheme S_1 and call it S.

Step 2. **For** some \$-cell c_2 in S_2 not yet considered,
until all of them have been used,
do begin
Take the \$-cell c of S for which
$S_2\text{-}num(c_2) = S\text{-}num(c)$, cut it out and paste it immediately to the left of that x-cell c' in S for which $S\text{-}num(c') = S_2\text{-}num(c_2')$.
end.
Since $S_2\text{-}num(c_2) < S_2\text{-}num(c_2')$ this yields a new shuffle-scheme, again called S, for which still $w = |S|$ holds.

Step 3. **For** some y-cell c_2 in S_2 not yet considered,
until all y-cells of S_2 have been used,
do begin
Take the total row $[c \ldots]$ of S for which $S\text{-}num(c) = S_2\text{-}num(c_2')$, which implies $S\text{-}cont(c) = \bar{x}$, cut it out and paste it immediately to the right of that y-cell c'' in S for which $S\text{-}num(c'') = S_2\text{-}num(c_2)$.
end.

Since $S_2\text{-}num(c_2) < S_2\text{-}num(c_2')$ this yields a new shuffle-scheme, again called S, for which still $w = |S|$ holds.

Step 4. **For** some ¢-cell c_2 in S_2 not yet considered,
until all of them have been used,
do begin
Take that ¢-cell c in S for which $S\text{-}num(c) = S_2\text{-}num(c_2)$, cut it out and paste it immediately to the right of that $\bar{y}$-cell c' in S for which $S\text{-}num(c') = S_2\text{-}num(c_2')$.
end.

The final resulting shuffle-scheme S still has $w = |S|$ and is of the form depicted in Fig. 4. This completes the proof of Theorem 3.6. □

Theorem 3.8. *Let* $L_1 := (\$((xAy) \cup (\bar{x}B\bar{y}))¢)^{\circledast}$, *where* $A, B \subseteq X^*$ *are arbitrary sets and* $\$, ¢, x, y, \bar{w}, \bar{y} \notin X$. *Let*

$$R_1 := (((\$¢)^{\circledast} \sqcup (xAy)^{\circledast}) \cap ((\$xy¢)^{\circledast} \sqcup X^{\circledast}),$$

$$R_2 := (((\$¢)^{\circledast} \sqcup (\bar{x}B\bar{y})^{\circledast}) \cap ((\$\bar{x}\bar{y}¢)^{\circledast} \sqcup X^{\circledast}).$$

Then

$$L_1 = R_1 \sqcup R_2.$$

Proof. By Lemma 2.1(4) one gets $L_1 = (\$xAy¢)^{\circledast} \sqcup (\$\bar{x}B\bar{y}¢)^{\circledast}$. Hence it is sufficient to show $R_1 = (\$xAy¢)^{\circledast}$ and likewise $R_2 = (\$\bar{x}B\bar{y}¢)^{\circledast}$. The inclusion $(\$xAy¢)^{\circledast} \subseteq R_1$ is trivial.

The reverse inclusion follows by slightly modifying Algorithm 3.7 in the proof of Theorem 3.6 and can be omitted. □

Theorem 3.9. *Let* $L_1 := (\$(xAy)^*¢)^{\circledast}$, *where* $A \subseteq X^*$ *is arbitrary and* $\$, x, y, ¢ \notin X$. *Let*

$$R_1 := (xAy)^{\circledast} \sqcup (\$¢)^{\circledast}, \qquad R_2 := (\$(xy)^*¢)^{\circledast} \sqcup X^{\circledast}.$$

Then

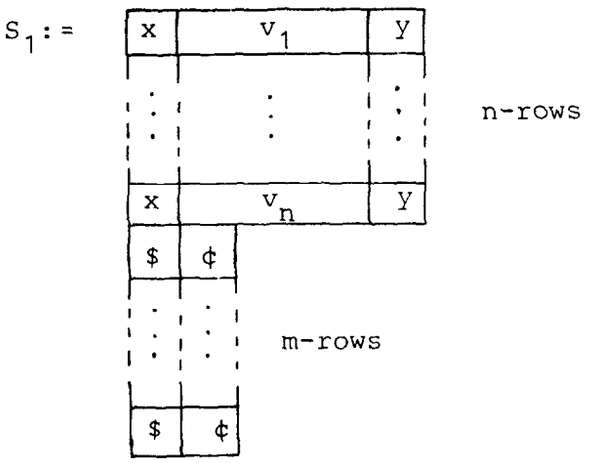
$$L_1 = R_1 \cap R_2.$$

Proof. First observe that $R_1 = (xAy)^{\circledast} \sqcup (\$¢)^{\circledast} = ((xAy) \cup (\$¢))^{\circledast}$ implies $R_1 = R_1^{\circledast}$ and similarly $R_2 = R_2^{\circledast}$. Hence $\$(xAy)^*¢ \subseteq R_1$ implies $L_1 = (\$(xAy)^*¢)^{\circledast} \subseteq R_1^{\circledast} = R_1$ and likewise $L_1 \subseteq R_2$, thus $L_1 \subseteq R_1 \cap R_2$.

For the proof of the reverse inclusion we again use a cut and paste algorithm on shuffle-schemes. Let $w \in R_1 \cap R_2$, then there exist shuffle-schemes S_1 and S_2 of the forms depicted in Figs. 5 and 6 such that $w = |S_1| = |S_2|$.

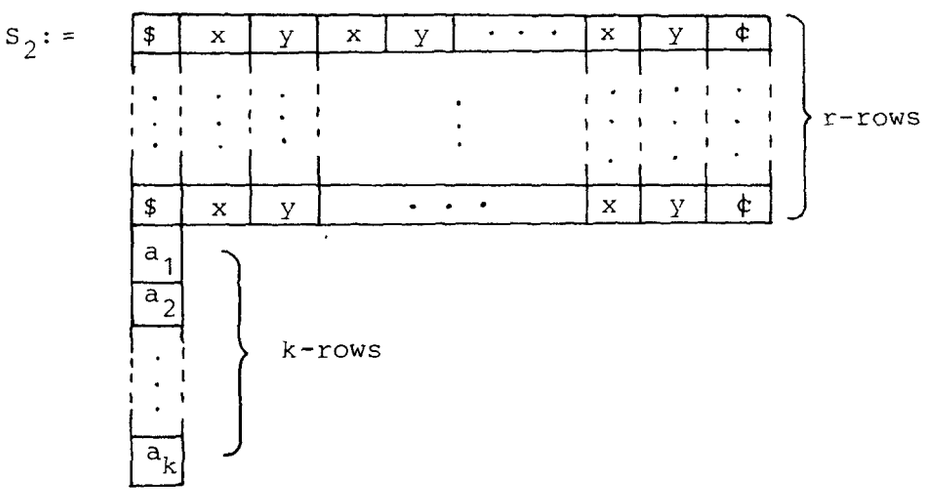
Since $w = |S_1| = |S_2|$ one has $r = m$, $k = \sum_{a \in X} \#_a(w) = \sum_{i=1}^{n} \lg(v_i)$, and more precisely

$$cont(S_1\text{-cell(i)}) = cont(S_2\text{-cell(i)}) \quad \text{for each } 1 \leq i \leq \#(S_1) = \#(S_2).$$



where $v_i \in A$ for $1 \leq i \leq n$, and $n, m \geq 0$.

Fig. 5.



where $a_i \in X$ for $1 \leq i \leq k$, $k \geq 0$.

Fig. 6.

Now the following cut and paste algorithm iteratively changes the shuffle-scheme S_1 according to the information contained in S_2 until a new shuffle-scheme S is found from which the desired inclusion follows.

Algorithm 3.10

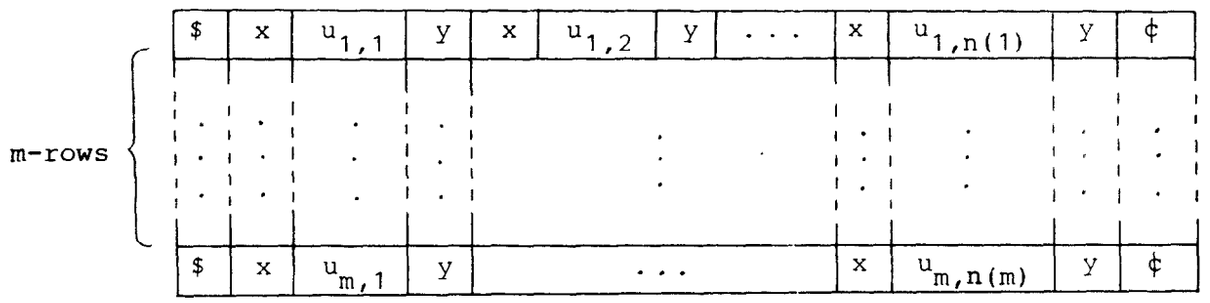
Step 1. Start with the shuffle-scheme S_1 and call it S.

Step 2. **For** some \$-cell c_2 in S_2 not yet considered,
until all of them have been used,
do begin
Take that \$-cell c of S for which $S\text{-}num(c) = S_2\text{-}num(c_2)$, cut it out and paste it immediately to the left of that x-cell c' in S for which $S\text{-}num(c') = S_2\text{-}num(c_2)$.
end.

Step 3. **For** some y-cell c_2 in S_2 with $S_2\text{-}cont(c_2) = x$ not yet considered,
until all such y-cells in S_2 have been used,
do begin
Take the total row $[c\ldots]$ in S for which $S\text{-}num(c) = S_2\text{-}num(c_2)$, cut it out and paste it immediately to the right of that y-cell c'' in S for which $S\text{-}num(c'') = S_2\text{-}num(c_2)$.
end.

Step 4. **For** some ¢-cell c_2 in S_2 not yet considered,
until all ¢-cells in S_2 have been used,
do begin
Take that ¢-cell c in S for which $S\text{-}num(c) = S_2\text{-}num(c_2)$, cut it out and paste it immediately to the right of that y-cell c' in S for which $S\text{-}num(c') = S_2\text{-}num(c_2)$.
end.

It is easy to verify that at any step in Algorithm 3.10 one has $|S| = w$ and the shuffle scheme S obtained finally has the form depicted in Fig. 7.



where $\{u_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n(j),\ n = \sum_{j=1}^{m} n(j)\} = \{v_i \mid 1 \leq i \leq n\} \subseteq A$.

Fig. 7.

From this it is clear that $w = |S| \in (\$(xAy)^*¢)^{\circledast} = L_1$ and therefore $(R_1 \cap R_2) \subseteq L_1$ which proves the theorem. □

The next result is most important for proving our main theorem, since it allows to reduce the nesting depth of $\circledast$ in certain $\mathscr{ER}$ languages at the cost of other operations.

Theorem 3.11. *For any set* $A \subseteq X^*$ *and symbols* \$, ¢ $\notin X$,

$$(\$A^{\circledast}¢)^{\circledast} = (\$A^{\circledast}¢)^* \sqcup (\$¢)^{\circledast}.$$

Proof. Let us use the abbreviations $L := (\$A^{\circledast}¢)^{\circledast}$ and $R := (\$A^{\circledast}¢)^* \sqcup (\$¢)^{\circledast}$. First let us show $R \subseteq L$. Obviously $(\$¢)^{\circledast} \subseteq L$ (since $\lambda \in (A)^{\circledast}$) and $\$A^{\circledast}¢ \subseteq L$.

Thus $(\$A^{\circledast}\¢)^* \subseteq L^{\circledast}$ and $R \subseteq L^{\circledast} \sqcup L$. By definition of $\circledast$ and L we get $L^{\circledast} \subseteq L$, hence $L^{\circledast} \sqcup L \subseteq L$ and $R \subseteq L$.

For the reverse inclusion $L \subseteq R$ the use of shuffle schemes is not appropriate.

Fact 1. $(\$A^{\circledast}\¢)^* \sqcup \$A^{\circledast}\¢ \subseteq R$ implies $R \sqcup \$A^{\circledast}\¢ \subseteq R$.

This is simply proved as follows:

$$R \sqcup \$A^{\circledast}\¢ = (\$A^{\circledast}\¢)^* \sqcup (\$¢)^{\circledast} \sqcup \$A^{\circledast}\¢ \quad \text{(definition of } R\text{)}$$

$$\subseteq R \sqcup (\$¢)^{\circledast} = R \quad \text{(using the assumption and the definition of } R\text{)}.$$

Fact 2. $R \sqcup (\$A^{\circledast}\¢) \subseteq R$ implies $L \subseteq R$.

This follows by induction from the definition of L:

$$L := (\$A^{\circledast}\¢)^{\circledast} = \bigcup_{i \geqslant 0} X_i, \quad \text{where } X_0 := \{\lambda\} \text{ and } X_{i+1} := X_i \sqcup (\$A^{\circledast}\¢).$$

Using these facts it is therefore enough to prove

$$(\$A^{\circledast}\¢)^* \sqcup \$A^{\circledast}\¢ \subseteq R.$$

To do this, let $w \in (\$A^{\circledast}\¢)^+$, $v \in (\$A^{\circledast}\¢)$, and $u \in \{w\} \sqcup \{v\}$. We shall show $u \in R$ by considering three cases. (The cases $w = \lambda$ or $v = \$¢$ trivially imply $\{w\} \sqcup \{v\} \subseteq R$.)

Case 1. Let $w = w_1 w_2$ with $w_1, w_2 \in (\$A^{\circledast}\¢)^*$. Then $u = w_1 v w_2$ trivially implies $u \in R$.

Case 2. Let $w = w_0 w_1 w_2$ with

$$w_0, w_2 \in (\$A^{\circledast}\¢)^*, \quad w_1 = \$w_1' w_1'' w_1'''¢ \in (\$A^{\circledast}\¢), \quad v = \$v'¢,$$

and

$$u \in \{w_0\$w_1'\$\}(\{v'\} \sqcup \{w_1''\})\{¢w_1'''¢w_2\}.$$

Then

$$u \in \{w_0\$\}(\{v'\} \sqcup \{w_1' w_1'' w_1'''\})\{¢w_2\} \sqcup \{\$¢\} \subseteq R \sqcup \{\$¢\} \subseteq R,$$

since $\{v'\} \sqcup \{w_1' w_1'' w_1'''\} \subseteq A^{\circledast}$.

Case 3. Let $w = w_0 w_1 w_2 \ldots w_n w_{n+1}$, $n \geqslant 2$, with $w_i = \$w_i'¢$ and $w_i' \in A^{\circledast}$ for $1 \leqslant i \leqslant n$,

$$w_1' = w_1'' w_1''', \quad w_n' = w_n'' w_n'''; \quad w_0, w_{n+1} \in (\$A^{\circledast}\¢)^*, \quad v = \$v'¢,$$

$$u \in \{w_0\$w_1''\$\}(\{v'\} \sqcup \{w_1'''¢\$w_2'¢ \cdots \$w_{n-1}'¢\$w_n''\})\{¢w_n'''¢w_{n+1}\}.$$

Then

$$u \in \{w_0\$w_1''\}(\{v'\} \sqcup \{w_1''' w_2' \cdots w_{n-1}' w_n''\})\{w_n'''¢w_{n+1}\} \sqcup (\$¢)^n$$

$$\subseteq \{w_0\$\}(\{v'\} \sqcup \{w_1' w_2' \cdots w_n'\})\{¢w_{n+1}\} \sqcup (\$¢)^n.$$

Since v and $w_i' \in A^{\circledast}$ for each w_i', $1 \leqslant i \leqslant n$, we thus get

$$u \in \{w_0\}(\$A^{\circledast}\¢)\{w_{n+1}\} \sqcup (\$¢)^n \subseteq (\$A^{\circledast}\¢)^* \sqcup (\$¢)^{\circledast} = R.$$

This finishes the proof of Theorem 3.11. □

(That these three cases really cover all possible cases can be seen by looking at Fig. 8, where all possible shufflings of w and v are vizualized informally.)

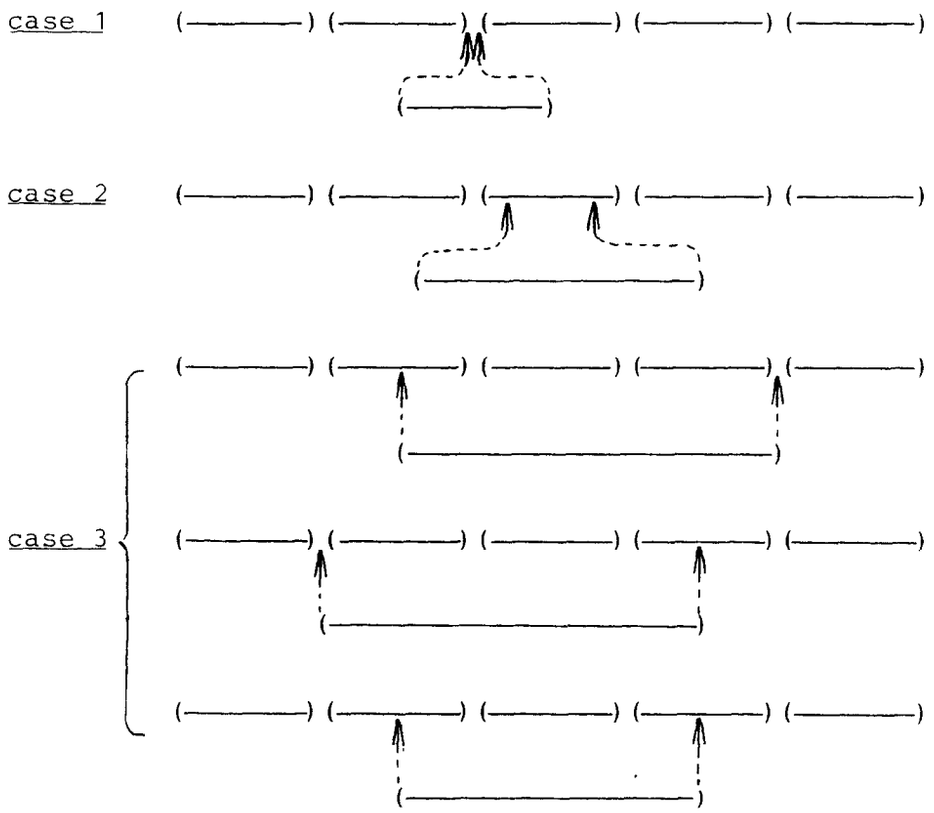


Fig. 8. Any horizontal line ——— stands for a word in $A^{\circledast}$, the \$ symbol is represented by an opening bracket (and the ¢ by the closing bracket). The dashed arcs point to the positions in w to which the \$ and ¢ symbols from the word v are shuffled.

It is interesting to compare Corollary 3.3 with the following proposition from [26].

Proposition 3.12. *The problem to decide* $v \in (w)^{\circledast}$ *for arbitrary words* $v, w \in X^*$, $\mathrm{card}(X) \geq 2$, *is* NP-*complete.*

The obvious question to be asked is whether or not every language in the larger class $\mathscr{SE}$ is nondeterministically acceptable by a multicounter machine in quasireal-time too, or is at least in P. In [14] it has been shown that the language $L := \{ab^n cde^n f \mid n \geq 0\}^{\circledast}$ is not in $(\mathscr{F}, \wedge)(D_1)$. However, the set $\{ab^n cde^n f \mid n \geq 0\}$ is not in $\mathscr{SE}$ since it does not contain an infinite regular subset. By now it is not at all clear whether $\mathscr{SE}$ contains some NP-complete set. In order to find out how one could eventually construct an NP-complete language in $\mathscr{SE}$ the following generalization of Theorem 3.2 gives some more hints.

Theorem 3.13. $(\vee, \sqcup\!\sqcup, \circledast)(\vee, \cdot, *, \circledast)(\mathscr{F}in) \subseteq (\mathscr{F}, \wedge)(D_1)$.

Proof. From Theorem 2.3 we first derive

$$(\vee, \sqcup, \circledast)(\vee, \cdot, *, \circledast)(\mathscr{F}in) = (\vee, \sqcup, \circledast)(\mathscr{CR}) \subseteq (\nabla)(\sqcup)(\circledast)(\sqcup)(\mathscr{CR}).$$

By the closure properties of $(\mathscr{F}, \wedge)(D_1)$ it is sufficient to show $(\circledast)(\sqcup)(\mathscr{CR}) \subseteq (\mathscr{F}, \wedge)(D_1)$ and moreover it is enough to prove the following:

For arbitrary languages $L_1, \ldots, L_n \in \mathscr{CR}$, $L_i \subseteq X^*$, one has $L := (L_1 \sqcup L_2 \sqcup \cdots \sqcup L_n)^{\circledast} \in (\mathscr{F}, \wedge)(D_1)$. To finally see this, let $x_1, \ldots, x_n \notin X$ be new symbols not used in the languages L_i, $1 \leq i \leq n$. Now construct $L' := (x_1 \cdot L_1 \sqcup x_2 \cdot L_2 \sqcup \cdots \sqcup x_n L_n)^{\circledast}$ and observe $L = h(L')$ for some $h \in \mathscr{H}^{dec}$.

Now

$$L' = (x_1 L_1 \cup x_2 L_2 \cup \cdots \cup x_n L_n)^{\circledast} \cap [(x_1 \ldots x_n)^{\circledast} \sqcup X^*]$$

is easy to see. But this already proves the theorem, since with $L_i \in \mathscr{CR}$ also $L'' := (x_1 L_1 \cup x_2 L_2 \cup \cdots \cup x_n L_n)^{\circledast} \in \mathscr{CR}$. Thus $L = h(L') = h(L'' \cap M)$, where $L'' \in \mathscr{CR}$, $M \in (\circledast)(\mathscr{F}in) \subseteq (\mathscr{F}, \wedge)(D_1)$, by Corollary 2.5.

Again by Lemma 2.9 there exists a regular set R such that h is c-limited erasing on $L' \cap R$ and therefore $L = h(L'' \cap M \cap R)$ where, by Theorem 3.2, $L'', M, R \in (\mathscr{F}, \wedge)(D_1)$ and h is c-limited erasing on $L'' \cap M \cap R$. Therefore, $L \in (\mathscr{F}, \wedge)(D_1)$. □

It is surprising that we were not able to generalize the ideas of the last proof to show that the typical language

$$(\$[(ab)^{\circledast} \sqcup (cd)]\not{c})^{\circledast} \in (\circledast)(\cdot)(\sqcup)(\circledast)(\mathscr{F}in),$$

is an element of $(\mathscr{F}, \wedge)(D_1)$ or at least acceptable in deterministic polynomial time. Note, that the similar looking sets

$$(\$((ab)^{\circledast} \sqcup (ab))\not{c})^{\circledast} = (\$(a(ab)^{\circledast} b(ab)^{\circledast})\not{c})^{\circledast}$$

and

$$(\$((ab)^{\circledast} \sqcup (cd)^{\circledast})\not{c})^{\circledast} = (\$(ab, cd)^{\circledast}\not{c})^{\circledast},$$

are both elements of the class $\mathscr{CR}$.

Appendix A

We shall now present the result necessary to prove Theorem 2.3 in all details. Also we shall list some open problems which have a more technical appeal.

Lemma A.1. $(\circledast)(\sqcup)(\circledast) \leq (\nabla)(\sqcup)(\circledast)(\sqcup)$.

Proof. Lemma 2.1 shows that

$$(A_1^{\circledast} \sqcup A_2^{\circledast} \sqcup \cdots \sqcup A_n^{\circledast} \sqcup B_1 \sqcup \cdots \sqcup B_m)^{\circledast}$$
$$= A_1^{\circledast} \sqcup \cdots \sqcup A_n^{\circledast} \sqcup (B_1 \sqcup \cdots \sqcup B_m)^{\circledast} \sqcup (B_1 \sqcup \cdots \sqcup B_m) \cup \{\lambda\}. \quad \square$$

Theorem A.2. $(\vee, ш, \circledast) \leqslant (\nabla)(\vee)(ш)(\circledast)(ш)$.

Proof. Clearly

$$(\vee, ш, \circledast) = \bigcup_{n \geqslant 0} (\vee, ш)[(\circledast)(\vee, ш)]^n.$$

We show $(\vee, ш)[(\circledast)(\vee, ш)]^n \leqslant (\nabla)(\vee)(ш)(\circledast)(ш)$ by induction on n.

Basis: $n = 0$ is trivial. For $n = 1$ we have

$$\begin{aligned}
&(\vee, ш)(\circledast)(\vee, ш)\\
&\quad \leqslant (\vee)(ш)(\circledast)(\vee)(ш) && \text{(by Lemma 2.2(4))}\\
&\quad \leqslant (\vee)(ш)(\vee)(ш)(\circledast)(ш) && \text{(by Lemma 2.2(9))}\\
&\quad \leqslant (\vee)(ш)(\circledast)(ш) && \text{(by Lemma 2.2(4))}\\
&\quad \leqslant (\nabla)(\vee)(ш)(\circledast)(ш).
\end{aligned}$$

Induction step:

$$\begin{aligned}
&(\vee, ш)[(\circledast)(\vee, ш)]^{n+1}\\
&\quad = (\vee, ш)(\circledast)(\vee, ш)[(\circledast)(\vee, ш)]^n\\
&\quad \leqslant (\vee, ш)(\circledast)(\nabla)(\vee)(ш)(\circledast)(ш) && \text{(by the induction hypothesis)}\\
&\quad \leqslant (\vee, ш)(\nabla)(\circledast)(\vee)(ш)(\circledast)(ш) && \text{(by Lemma 2.2(2))}\\
&\quad \leqslant (\vee, ш)(\nabla)(\vee)(ш)(\circledast)(ш)(\circledast)(ш) && \text{(by Lemma 2.2(9))}\\
&\quad \leqslant (\vee, ш)(\nabla)(\vee)(\nabla)(ш)(\circledast)(ш)(ш) && \text{(by Lemma 2.3)}\\
&\quad \leqslant (\nabla)(\vee)(ш)(\circledast)(ш) && \text{(by Lemma 2.2(1) and (4)).}
\end{aligned}$$

□

Now that we have finished the proof of Theorem 2.3 we formulate some of the open questions.

Problem 1. Is the family $\mathscr{ER}^{\Delta}$ a proper subclass of the family $(\vee, \cdot, +, \oplus)(\mathscr{Fin}^{\Delta})$, or not?

This problem arose in the context of Corollary 2.8.

Problem 2. In [15] it has been shown that the families $\mathscr{Shuf}$ and $\mathscr{SE}$ are closed with respect to quotient by finite sets. We conjecture that the family $\mathscr{ER}$ may not be closed under this operation. To prove this it would be sufficient to show that $(abc)^{\circledast}/c = (abc)^{\circledast} ш (ab) \notin \mathscr{ER}$.

Problem 3. Is $\mathscr{SE} \subset (\mathscr{F}, \wedge)(D_1)$?

Problem 4. Does the family $\mathscr{SC}$ contain some NP-complete set?

For this question, consult [26], too.

References

[1] T. Araki, T. Kagimasa and N. Tokura, Relations of flow languages to Petri net languages, *Theoret. Comput. Sci.* **15** (1981) 51-75.

[2] T. Araki and N. Tokura, Flow languages equal recursively enumerable languages, *Acta Inform.* **15** (1981) 209-217.

[3] J. Berstel, *Transductions and Context-Free Languages* (Teubner, Stuttgart, 1979).

[4] L. Czaja, Parallel systems schemas and their relation to automata, *Inform. Process. Lett.* **10** (1980) 153-158.

[5] S. Eilenberg and M.P. Schützenberger, Rational sets in commutative monoids, *J. Algebra* **13** (1969) 173-191.

[6] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages* (North-Holland, Amsterdam, 1975).

[7] J. Gisher, Shuffle languages, Petri nets, and context-sensitive grammars, *Comm. ACM* **24** (1981) 597-605.

[8] S.A. Greibach, Remarks on blind and partially blind one-way multicounter machines, *Theoret. Comput. Sci.* 7 (1978) 311-324.

[9] M. Hack, Petri net languages, Computation Structures Group Memo 124, Project MAC, M.I.T., 1975.

[10] D. Haussler and H.P. Zeiger, Very special languages and representations of recursively enumerable languages via computation histories, *Inform. Control* **47** (1980) 201-212.

[11] M. Jantzen, On the hierarchy of Petri net languages, R.A.I.R.O. *Inform. Théor.* **13** (1979) 19-30.

[12] M. Jantzen, Eigenschaften von Petrinetzsprachen, Dissertation, Universität Hamburg, FB Informatik, 1979.

[13] M. Jantzen, On zerotesting—bounded multicounter machines, *Lectures Notes in Computer Science* **67** (Springer, Berlin, 1979) 158-169.

[14] M. Jantzen, The power of synchronizing operations on strings, *Theoret. Comput. Sci.* **14** (1981) 127-154.

[15] M. Jantzen, Extending regular expressions with iterated shuffle, Tech. Rept., FB-Informatik, Univ. Hamburg, IFI-HH-B-99/84, 1984.

[16] J. Jedrzejowicz, On the enlargement of the class of regular languages by the shuffle closure, *Inform. Process. Lett.* **16** (1983) 51-54.

[17] T. Kimura, An algebraic system for process structuring and interprocess communication, *Proc. 8th Ann. ACM Symp. on Theory of Computing* (1976) 92-100.

[18] T. Kimura, Formal description of communication behaviour, *Proc. Johns Hopkins Conf. on Information Sciences and Systems*, 1979.

[19] W.F. Ogden, W.E. Riddle and W.C. Rounds, Complexity of expressions allowing concurrency, *Proc. 5th Ann. ACM Symp. on Principles of Programming Languages* (1978) 185-194.

[20] W.E. Riddle, Modelling and analysis of supervisor systems, Ph.D. Thesis, Computer Science Dept., Stanford Univ., 1972.

[21] W.E. Riddle, Software systems modelling and analysis, RSSM/25, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, 1976.

[22] A.C. Shaw, Systems design and documentation using path descriptions, *Proc. Sagamore Computer Conf. on Parallel Processing* (IEEE Computer Society, 1975) 180-181.

[23] A.C. Shaw, Software description with flow expressions, *IEEE Trans. Software Engrg.* **3** (4) (1978) 242-254.

[24] G. Slutzki, Non-synchronizing concurrent processes and their languages, unpublished manuscript, Dept. of Computer and Information Sciences, Univ. of Delaware, Newark, 1979.

[25] G. Slutzki, Descriptional complexity of concurrent processes, *Lecture Notes in Computer Science* **88** (Springer, Berlin, 1980) 601-611.

[26] M.K. Warmuth and D. Haussler, On the complexity of iterated shuffle, *J. Comput. Syst. Sci.* **28** (1984) 345-358.