Theoretical Computer Science 60 (1988) 1-82
North-Holland

THEORY OF TRACES

IJsbrand Jan AALBERSBERG* and Grzegorz ROZENBERG
Department of Computer Science, University of Leiden, 2300 RA Leiden, The Netherlands

Communicated by A. Salomaa
Received October 1986
Revised July 1987

Abstract. The *theory of traces*, originated by A. Mazurkiewicz in 1977, is an attempt to provide a mathematical description of the behavior of concurrent systems. Its aim is to reconcile the *sequential nature of observations* of the system behavior on the one hand and the *nonsequential nature of causality* between the actions of the system on the other hand.

One can see the theory of traces to be rooted in formal string language theory with the notion of partial commutativity playing the central role. Alternatively one can see the theory of traces to be rooted in the theory of labeled acyclic directed graphs (or even in the theory of labeled partial orders).

This paper attempts to present a major portion of the theory of traces in a unified way. However, it is not a survey in the sense that a number of new notions are introduced and a number of new results are proved. Although traditionally most of the development in the theory of traces follows the string-language-theoretic line, we try to demonstrate to the reader that the graph-theoretic point of view may be more appropriate.

The paper essentially consists of two parts. The first one (Sections 1 through 4) is concerned with the basic theory of traces. The second one (Section 5) presents applications of the theory of traces to the theory of the behavior of concurrent systems, where the basic system model we have chosen is the condition/event system introduced by C.A. Petri.

Contents

* Present affiliation: Philips Research Laboratories Eindhoven, 5600 JA Eindhoven, The Netherlands.

Introduction

The theory of traces was originated by Mazurkiewicz (in [31]) as an attempt to provide a mathematical description of the behavior of concurrent systems. The basic aim of this attempt is to provide tools for converting the sequential descriptions of a system into nonsequential descriptions. More specifically, the idea can be explained as follows. A natural way to describe the behavior of a (concurrent) system is through sequential observers. Here, a record of the behavior of a system is given as a linear sequence (string) of actions as observed by a sequential observer. The set of all such records constitutes the (sequential) description of the system behavior. The disadvantage of this approach is that such a record does not necessarily give faithful information about the system: two actions a and b may appear adjacent within a sequential record, while they are really performed concurrently within the system. Thus, in order to extract faithful information about the system behavior from the set of records by sequential observers, we have to have additional information about the system itself. Trace theory solves this problem in an elegant way: the information about the system is given as a binary relation (over the set of all actions)—called the independence relation of the system. A pair (a, b) belongs to ths relation if there is no direct causal relationship between the actions a and b within the system. Now, given a sequential record x of the form x_1abx_2, where the pair (a, b) belongs to the independence relation associated with the system, one may commute the given occurrences of a and b obtaining in this way another valid (equivalent) sequential record (namely x_1bax_2) of the system behavior. All this reflects the fact that the sequencing of ab within x results from the sequential nature of the observations rather than from the system properties. Thus within this approach one deals with equivalence classes of observations rather than with single observations only.

The independence relation as considered in the theory of traces is assumed to be symmetric and irreflexive—both assumptions express specific axioms concerning the phenomenon of concurrency (irreflexivity represents the assumption that no action in a system can occur concurrently with itself and symmetry represents the assumption that concurrency of actions is always mutual).

It is evident that, following the above philosophy, the behavior of a system S can be formalized now as a pair (L, I), where L is a set of strings representing all sequential records of S and I is the independence relation providing commutation

rules for symbols of the alphabet of L. In this way one enters the area of partially commutative string languages and hence the area of partially commutative monoids (this was first observed in [4]). It is interesting to know that the theory of partially commutative monoids was initiated much earlier than the theory of traces. The first work in this area is by Foata (in [12])—the main motivation here comes from combinatorial problems arising from the rearrangements of strings. These purely combinatorial considerations led to a by now quite rich theory (see, e.g., [27] and [30, Chapter 10]). As always it is exciting to observe how considerations arising from fundamental problems in computer science and purely mathematical considerations merge in a common framework.

There were other developments in computer science which led to the same framework.

First of all, already in 1971 Keller has investigated (in [26]) the notion of partial commutation within the theory of program schemes.

Secondly, in a number of papers (see, e.g., [14, 15, 16]) Flé and Roucairol considered a formalism for reasoning about transactions in database systems. These considerations led, again, to the framework of partially commutative monoids.

This paper attempts to present a major portion of the theory of traces in a unified way. Although it surveys quite a large part of the literature, it is not a survey in the usual sense because

(i) it is somewhat selective in the choice of the material, and

(ii) a number of new notions are introduced and a number of new results are proved.

It will be clear to the reader that this work strongly reflects our belief that graph-theoretic considerations are natural and fruitful in the theory of traces. As a matter of fact we try to emphasize the duality of the string- and the graph-oriented points of view: while for some considerations viewing a trace as a set of strings is useful, for other considerations looking at a trace via its so-called dependence graph is more appropriate.

The paper is addressed to both formal language theorists and to researchers in the theory of concurrency, especially to those interested in the theory of Petri nets. We feel that both communities can benefit from learning about the theory of traces: for language theorists it is a source of new, interesting and challenging problems, while for a researcher in Petri nets it provides natural formal tools to describe their behavior.

Basic knowledge of formal language theory is needed in reading this paper; in reading Section 5 (which provides applications of trace theory to Petri nets), basic knowledge about Petri nets is needed. However, since the knowledge of the basic theory of Petri nets is less common than that of formal languages, Subsection 5.1 gives a brief introduction to those aspects of the theory of Petri nets that are needed in Section 5.

The 'mixed' character of this paper (surveying known as well as providing new notions and results) is reflected in the style of writing. Known results are given

without proofs (except when we provide a new proof). At the end of each section we provide bibliographical comments. Notions and results not pointed out in the bibliographical comments are new. The only exception is Section 2 concerning the basic results; many of the basic observations here were either stated at the same time in a number of papers or they were a part of the folklore—for this reason they are not explicitly pointed out in the bibliographical comments.

Independently of the list of references which is used throughout the paper, we provide an annotated bibliography of the theory of traces, which makes it easier for the interested reader to get a more complete picture of the theory.

0. Preliminaries

We assume the reader to be familiar with basic formal string language theory, in particular with the Chomsky hierarchy of string languages (see, e.g., [24, 44]), and with basic notions of graph theory (e.g., as presented in [22]).

The empty set will be denoted by $\emptyset$ and the set of all nonnegative integers will be denoted by N. For a set A,

(i) $\#A$ denotes the cardinality of A,
(ii) $\mathrm{id}(A)$ denotes the set $\{(a, a): a \in A\}$,
(iii) $\mathcal{P}(A)$ denotes the set of all subsets of A, and
(iv) for a subset B of $\mathcal{P}(A)$, $\bigcup B$ denotes the set $\bigcup_{b \in B} b$.

Furthermore, for sets A and B,

(i) $A - B$ denotes the difference of A and B,
(ii) $A \subseteq B$ denotes the inclusion of A in B, and
(iii) $A \subset B$ denotes the strict inclusion of A in B.

For a relation R over a set A,

(i) $\mathrm{dom}(R)$ denotes the domain of R,
(ii) $\mathrm{ran}(R)$ denotes the range of R,
(iii) R^+ denotes the transitive closure of R, and
(iv) R^* denotes the reflexive and transitive closure of R.

When we deal with an irreflexive relation R over a set A, we will use the terms 'transitive' and 'complete' in the following sense:

(i) R is *transitive* if its reflexive closure is transitive, and
(ii) R is *complete* if its reflexive closure is complete.

When we specify a symmetric relation R over a set A, we often give it as a set of unordered pairs rather than as a set of ordered pairs (i.e., we write $\{a, b\} \in R$ rather than $(a, b) \in R$ and $(b, a) \in R$). Finally, for a symmetric relation R over a set A, a *clique* of R is a subset B of A such that R is complete on B.

Unless stated otherwise, we consider only finite, nonempty alphabets. The empty string will be denoted by λ. For a string x,

(i) $|x|$ denotes the length of x,
(ii) $\#_a(x)$ denotes the number of occurrences of symbol a in x,

(iii) alph(x) denotes the set of symbols occurring in x,

(iv) for $1 \leq k \leq |x|$, $x(k)$ denotes the symbol occurring on the kth position in x and, for $k > |x|$, $x(k)$ denotes the empty string, and

(v) for an alphabet A, $\phi_A(x)$ denotes the projection of x on A (i.e., the string resulting from x by erasing from it all the symbols that are not in A).

For two strings x and y, $x \| y$ denotes the set of shuffles of x and y, while, for two string languages K and L, $K \| L$ denotes the union of all sets of shuffles of x and y with $x \in K$ and $y \in L$. The Parikh image of a string x (a string language L) is denoted by $\psi(x)$ ($\psi(L)$ respectively). The string concatenation of two string languages K and L is denoted by $K \cdot L$; mostly we simply write KL. For an alphabet A, a linear ordering $<$ on A (which is extended to the linear ordering on A^* in the usual way), and a string language L over A, min(L) denotes the unique string x in L such that $y < x$ does not hold for any y in L. Finally, REG, CF, and CS denote the classes of regular string languages, context-free string languages, and context-sensitive string languages (over the alphabet considered), respectively.

A (*directed*) *node-labeled graph* will be specified in the form (V, E, Σ, l), where V is its set of nodes, $E \subseteq V \times V$ its set of edges, Σ its label alphabet and $l: V \to \Sigma$ its node-labeling function (occasionally we will specify E in the form of an incidence matrix). Analogously, a (*directed*) *graph* will be specified in the form (V, E), where V and E are as described above. To avoid set-theoretical difficulties, we will assume that we are given a 'universal' countable set of elements, from which the nodes of all the (directed) (node-labeled) graphs we consider are taken. We consider (directed) (node-labeled) graphs without loops i.e., without edges of the form (v, v) where v is a node, only. The *empty graph* (i.e., the graph with the empty set of nodes) is denoted by Λ. The set of all directed node-labeled graphs with label alphabet Σ is denoted by $\Gamma(\Sigma)$. Two (directed) node-labeled graphs α and β are *isomorphic*, denoted by $\alpha \cong \beta$, if there exists a node-label-preserving and adjacency-preserving isomorphism between the corresponding sets of nodes. Two (directed) graphs α and β are *isomorphic*, denoted by $\alpha \cong \beta$, if there exists an adjacency-preserving isomorphism between the corresponding sets of nodes. As customary, in order to facilitate our notation, we will notationally not distinguish very carefully between a 'concrete' (directed) (node-labeled) graph and an 'abstract' (directed) (node-labeled) graph; an 'abstract' (directed) (node-labeled) graph is the class of all graphs isomorphic to a 'concrete' (directed) (node-labeled) graph. For a directed node-labeled acyclic graph $\alpha = (V, E, \Sigma, l)$ with $n = \# V$, the (*linear extension*) *language of* α, denoted by $L(\alpha)$, is the string language $\{x \in \Sigma^n : v_1, \ldots, v_n$ is a sequence without repetitions of all elements of V such that, for all $1 \leq i \leq n$, $l(v_i) = x(i)$, and, for all $1 \leq i, j \leq n$, $(v_i, v_j) \in E$ implies that $i < j\}$. For a directed (node-labeled) acyclic graph α with V as its set of nodes and E as its set of edges

(i) a *node cut of* α is a subset S of V, such that, for all $v, w \in S$, $(v, w) \notin E^+$, and, for all $v \in V - S$, there exists a $w \in S$ with either $(v, w) \in E^+$ or $(w, v) \in E^+$,

(ii) an *edge cut of* α is a subset S of E such that, for all $(v, w), (y, z) \in S$, neither $(w, y) \in E^+$ nor $(z, v) \in E^+$, and such that the directed (node-labeled) acyclic graph

resulting from α by removing all edges in S consists of more connected components than α does,

(iii) $\min(\alpha)$ ($\max(\alpha)$) denotes the set of those nodes of α which do not have incoming (outgoing respectively) edges,

(iv) a *transitive edge of* α is an edge (v, w) of α such that there exists a path of length at least 2 from v to w,

(v) $\mathrm{red}(\alpha)$ ($\mathrm{trans}(\alpha)$) denotes the directed (node-labeled) acyclic graph resulting from α by removing (adding respectively) all transitive edges, and

(vi) for a subset S of V, the (reduced, transitive) *S-contracted version of* α is the directed (node-labeled) acyclic graph β ($\mathrm{red}(\beta)$, $\mathrm{trans}(\beta)$, respectively), where β results from α by removing all nodes in S together with all edges adjacent with a node in S and by adding an edge from a node v (not in S) to a node w (not in S) whenever there exists a node z in S such that $(v, z) \in E$ and $(z, w) \in E$.

Unless stated explicitly otherwise, all functions considered will be total. For a set V, a subset S of V, and a function f on V, $f|_S$ denotes the function f restricted to S. The notation $O(f(n))$, with $f: \mathbf{N} \to \mathbf{N}$, has the reserved meaning for indicating the complexity of algorithms.

Finally, for the basics of the theory of (free partially commutative) monoids (this is needed in Subsections 1.2 and 3.2 only), we refer the reader to [27]; for the basics of the theory of semilinear sets (this is needed in Subsections 3.2 and 3.3 only), we refer the reader to [19]; and for the basics of the theory of NP-completeness (this is needed in Subsection 3.3 only), we refer the reader to [18].

1. Basic notions

1.1. Traces and dependence graphs—basics

The most fundamental notion of trace theory is that of independent and dependent actions. In order to formalize the (in)dependence of actions, we need the following definition.

1.1. Definition. (1) Let Σ be an alphabet. An *independence relation* (*over* Σ) is a symmetric and irreflexive binary relation over Σ. A *dependence relation* (*over* Σ) is a symmetric and reflexive binary relation over Σ.

(2) A *reliance alphabet* is a triple (Σ, I, D), where Σ is an alphabet, I is an independence relation over Σ and D is a dependence relation over Σ such that $I \cup D = \Sigma \times \Sigma$ and $I \cap D = \emptyset$.

(3) Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $a, b \in \Sigma$. Whenever $\{a, b\} \in I$, we say that a and b are *independent* (*in* C) and whenever $\{a, b\} \in D$, we say that a and b are *dependent* (*in* C).

For a given reliance alphabet C, we will denote its alphabet by $\mathrm{alph}(C)$, its independence relation by $\mathrm{ind}(C)$ and its dependence relation by $\mathrm{dep}(C)$.

1.2. Remark. It is easily seen that, for a given reliance alphabet C, $\mathrm{ind}(C)$ and $\mathrm{dep}(C)$ are each others complement in $\mathrm{alph}(C) \times \mathrm{alph}(C)$. Hence, given an alphabet and an independence (dependence) relation over that alphabet, the corresponding dependence (independence respectively) relation is uniquely determined. Consequently, it would suffice to specify one of them only. However, while in some considerations in the theory of traces the independence relation plays the central role, in other considerations one concentrates on the dependence relation; thus it seems to be convenient to include both an independence and a dependence relation in the specification of a reliance alphabet.

Since symmetric binary relations can be represented by undirected graphs, we can give a reliance alphabet C by either giving an undirected graph which represents $\mathrm{ind}(C)$ or giving an undirected graph which represents $\mathrm{dep}(C)$. This leads us to the following definition.

1.3. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) The *independence graph of* C, denoted by $I(C)$, is the undirected graph (Σ, I).

(2) The *dependence graph of* C, denoted by $D(C)$, is the undirected graph $(\Sigma, D - \mathrm{id}(\Sigma))$.

Thus, in the terminology as above, both the independence graph of C and the dependence graph of C are constructed by considering every symbol in Σ as a node—in the independence graph an edge between two nodes is established if and only if they are independent in C (and hence different) and in the dependence graph an edge between two nodes is established if and only if they are dependent in C and different (hence, the reflexive part of $\mathrm{dep}(C)$ is not represented in $D(C)$).

1.4. Example. Let $\Sigma = \{a, b, c, d, e\}$, let

$$I = \{\{a, b\}, \{b, c\}, \{b, d\}, \{d, e\}\}$$

and let

$$D = \{\{a, c\}, \{a, d\}, \{a, e\}, \{b, e\}, \{c, d\}, \{c, e\}\} \cup \mathrm{id}(\Sigma).$$

Then,

(i) $C = (\Sigma, I, D)$ is a reliance alphabet,

(ii) $I(C)$ is as depicted in Fig. 1(a), and

(iii) $D(C)$ is as depicted in Fig. 1(b).

Considering adjacent independent symbols in a string to be commuting, one can relate different strings as follows.

1.5. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) The relation $\doteq_C \subseteq \Sigma^* \times \Sigma^*$ is defined as follows: for $x, y \in \Sigma^*$, $x \doteq_C y$ if and only if there exist $x_1, x_2 \in \Sigma^*$ and $\{a, b\} \in I$ such that $x = x_1 abx_2$ and $y = x_1 bax_2$.

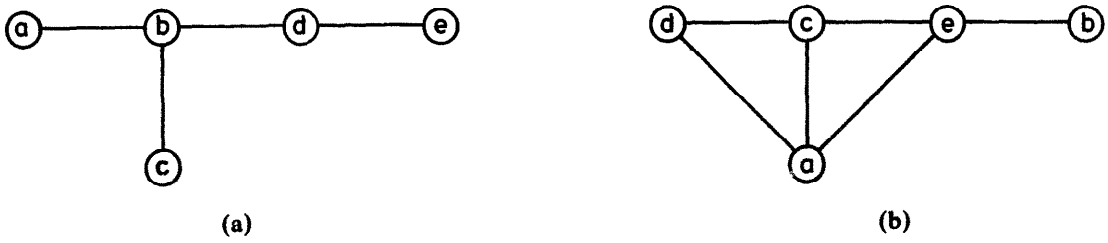


Fig. 1

(2) The *C-equivalence relation* $\equiv_C \subseteq \Sigma^* \times \Sigma^*$ is defined as the least equivalence relation over Σ^* containing $\doteq_C$. For $x, y \in \Sigma^*$, x and y are called *C-equivalent* if and only if $x \equiv_C y$.

Thus, for a given reliance alphabet C, two strings over alph(C) are C-equivalent if and only if they can be obtained from each other by commuting (a finite number of times) adjacent independent symbols. In other words, $\equiv_C$ identifies 'commutatively similar' strings—each such group of strings (an equivalence class of $\equiv_C$) is called a trace.

1.6. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) For an $x \in \Sigma^*$, the *trace of* x *(over* C*)*, denoted by $[x]_C$, is the equivalence class of $\equiv_C$ containing x.

(2) A *trace (over* C*)* is a set t of strings over Σ such that $t = [x]_C$ for some $x \in \Sigma^*$. A set of traces (over C) is called a *trace language (over* C*)*.

(3) For an $x \in \Sigma^*$ and a trace t over C, x is called a *representative of* t if and only if $t = [x]_C$.

For a reliance alphabet C, the trace $[\lambda]_C$ over C will be denoted by 1 and the set of all traces over C will be denoted by $\Theta(C)$.

1.7. Example. Let C be the reliance alphabet given in Example 1.4. Then,

(i) *bade* $\doteq_C$ *abde*, *abde* $\doteq_C$ *adbe* and hence, *bade* $\equiv_C$ *abde*, *abde* $\equiv_C$ *adbe* and *bade* $\equiv_C$ *adbe*, and

(ii) $[abde]_C = \{bade, abde, adbe, baed, abed\}$ is the trace of *abde* (over C).

While adjacent independent symbols in a string are commuting, dependent symbols in a string are ordered—in this way each trace yields a directed node-labeled acyclic graph as follows.

1.8. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) Let $x = \sigma_1 \ldots \sigma_n \in \Sigma^*$ for some $n \geq 0$ and $\sigma_1, \ldots, \sigma_n \in \Sigma$. The *canonical dependence graph of* x *(over* C*)*, denoted by $\langle x \rangle_C$, is the directed node-labeled acyclic

graph (V, E, Σ, l), where:

(i) $V = \{1, \ldots, n\}$,

(ii) for all $1 \leq i \leq n$, $l(i) = \sigma_i$, and

(iii) $E \subseteq V \times V$ is such that, for all $1 \leq i, j \leq n$, $(i, j) \in E$ if and only if $[i < j$ and $\{\sigma_i, \sigma_j\} \in D]$.

(2) For an $x \in \Sigma^*$, a *dependence graph of* x *(over* C*)* is a directed node-labeled acyclic graph d over Σ such that $d \cong \langle x \rangle_C$.

(3) A *dependence graph (over* C*)* is a directed node-labeled acyclic graph d over Σ such that $d \cong \langle x \rangle_C$ for some $x \in \Sigma^*$. A set of dependence graphs (over C) is called a *dependence graph language (over* C*)*.

(4) For $x, y \in \Sigma^*$, x and y are called *C-isomorphic* if and only if $\langle x \rangle_C \cong \langle y \rangle_C$.

Thus, for a given reliance alphabet C and a string x over alph(C), the canonical dependence graph of x over C is obtained by drawing arrows 'from left to right' between every two nodes that correspond to dependent (occurrences of) symbols in x.

For a given reliance alphabet C, the set of all dependence graphs over C will be denoted by $\Delta(C)$. Furthermore, although we will elaborate more on the notations used in this paper at the end of this section, we now already want to note that the term 'dependence graph (over C)' will often be abbreviated as 'dep-graph (over C)'.

1.9. Example. Let C be the reliance alphabet given in Example 1.4. The graph depicted in Fig. 2 is a dependence graph of *dbcee* (over C).

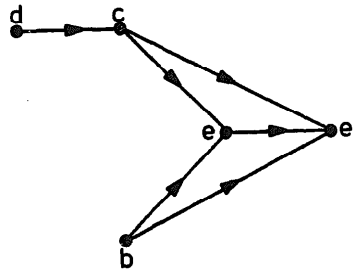


Fig. 2

1.2. Traces—more basics

In this subsection we will provide more basic notions and notations concerning traces and trace languages.

The following easy to prove properties are essential for defining some basic notions later on.

1.10. Lemma. *Let* $C = (\Sigma, I, D)$ *be a reliance alphabet.*

(1) *For* $x, y \in \Sigma^*$, $x \equiv_C y$ *implies* $|x| = |y|$.

(2) *For* $x_1, x_2, y_1, y_2 \in \Sigma^*$, $x_1 \equiv_C x_2$ *and* $y_1 \equiv_C y_2$ *imply that* $x_1 y_1 \equiv_C x_2 y_2$.

Hence, for a given reliance alphabet C,

(i) two C-equivalent strings over alph(C) have the same length, and

(ii) the C-equivalence relation is a congruence with respect to the operation of string concatenation.

Now we move to basic operations on traces and trace languages.

1.11. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet.

(1) For a $t\in\Theta(C)$, the *length of* t, denoted by $|t|$, is defined by $|t|=|x|$, where $x\in\Sigma^*$ is a representative of t.

(2) For $t_1, t_2\in\Theta(C)$, the *trace composition of* t_1 *and* t_2, denoted by $t_1\circ t_2$, is defined by $t_1\circ t_2=[x_1x_2]_C$, where $x_1, x_2\in\Sigma^*$ are representatives of t_1 and t_2 respectively.

(3) For $T_1, T_2\subseteq\Theta(C)$, the *trace composition of* T_1 *and* T_2, denoted by $T_1\circ T_2$, is defined by $T_1\circ T_2=\{t_1\circ t_2: t_1\in T_1 \text{ and } t_2\in T_2\}$.

(4) For a $T\subseteq\Theta(C)$ and $n\geqslant 0$, the *n-th trace power of* T, denoted by T^n, is defined inductively as follows: $T^0=\{1\}$ and $T^{n+1}=T\circ T^n$.

(5) For a $T\subseteq\Theta(C)$, the *trace iteration of* T, denoted by T^*, is defined by $T^*=\bigcup_{n=0}^{\infty} T^n$.

1.12. Remark. (1) It is easily seen that Lemma 1.10 implies that both the length of a trace and the trace composition operation are well-defined notions.

(2) The reader should not confuse the trace composition of traces with the string concatenation of traces. While it is true that one gets the trace composition of two traces by taking the trace of an arbitrary element of the string concatenation of these traces, the trace composition and the string concatenation may yield different sets. So, for a given reliance alphabet C and $t_1, t_2\in\Theta(C)$, the trace composition of t_1 and t_2 is the trace $t_1\circ t_2=[x_1x_2]_C$, where $x_1, x_2\in(\text{alph}(C))^*$ are arbitrary representatives of t_1 and t_2 respectively, while the string concatenation of t_1 and t_2 is the set of strings $t_1t_2=\{x_1x_2: x_1\in t_1 \text{ and } x_2\in t_2\}$.

(3) It is easily seen that the trace composition operation is associative, i.e., for a given reliance alphabet C and for $t_1, t_2, t_3\in\Theta(C)$, $t_1\circ(t_2\circ t_3)=(t_1\circ t_2)\circ t_3$. Hence, for a given reliance alphabet C, $(\Theta(C), \circ, 1)$ is the quotient monoid of $((\text{alph}(C))^*, \cdot, \lambda)$ with respect to $\equiv_C$. This monoid is usually termed the *free partially commutative monoid generated by C.*

1.13. Example. Let C be the reliance alphabet given in Example 1.4. Then,

(i) $|[abde]_C|=4$;

(ii) for the traces $t_1=[ad]_C=\{ad\}$ and $t_2=[be]_C=\{be\}$ over C the string concatenation t_1t_2 equals $\{adbe\}$ while the trace composition $t_1\circ t_2$ equals $t=[abde]_C=\{bade, abde, adbe, baed, abed\}$; and

(iii) for the trace language $T=\{[ab]_C\}$ over C, $T^0=\{1\}$, $T^1=T$, $T^2=\{[ab]_C\circ[ab]_C\}=\{[abab]_C\}=\{\{aabb, abab, abba, baab, baba, bbaa\}\}$, $T^n=\{[a^nb^n]_C\}$ for $n\geqslant 0$, and $T^*=\{[a^kb^k]_C: k\geqslant 0\}$.

Trace languages may be specified using string languages. As a matter of fact, there are several methods of using string languages to specify trace languages.

1.14. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K \subseteq \Sigma^*$.

(1) The *existential trace language of* K *(over* C*)*, denoted by $[K]_C^{\exists}$, is the set $\{t \in \Theta(C): t \cap K \neq \emptyset\}$.

(2) The *universal trace language of* K *(over* C*)*, denoted by $[K]_C^{\forall}$, is the set $\{t \in \Theta(C): t \subseteq K\}$.

(3) The *consistent trace language of* K *(over* C*)*, denoted by $[K]_C$, equals $[K]_C^{\exists}$ if $[K]_C^{\exists}=[K]_C^{\forall}$ and is undefined otherwise.

1.15. Remark. Note that, in the notation as above,

$$[K]_C^{\exists}=\{t \in \Theta(C): \textit{there exists } x \in t \text{ such that } x \in K\} \quad \text{and}$$

$$[K]_C^{\forall}=\{t \in \Theta(C): \textit{for all } x \in t,\ x \in K\};$$

this explains the names 'existential' and 'universal' respectively.

The above defined notions can be illustrated as follows.

Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K \subseteq \Sigma^*$. Then the C-equivalence relation $\equiv_C$ implies the partition Π_C of Σ^* into an infinite number of equivalence classes (see Fig. 3(a)), while K implies the partition Π_K of Σ^* into two subsets of Σ^*, namely K and Σ^*-K (see Fig. 3(b)).

Superimposing (showing at the same time) the two above partitions, we get the picture which is shown in Fig. 3(c). Now all elements of Π_C that have a nonempty intersection with K form the existential trace language of K (see Fig. 3(d)); thus $[K]_C^{\exists}$ consists of all traces which contain at least one string from K. Furthermore, all elements of Π_C that are included in K form the universal trace language of K (see Fig. 3(e)); thus $[K]_C^{\forall}$ consists of all traces which contain only strings from K. If $\Pi_C \subseteq \Pi_K$ (i.e., superimposing Π_C and Π_K yields the situation as depicted in Fig. 3(f)), then the set of all elements of Π_C having a nonempty intersection with K equals the set of all elements of Π_C that are included in K, and thus all these elements form the consistent trace language of K (see Fig. 3(g)).

1.16. Remark. Note that, for a given reliance alphabet $C=(\Sigma, I, D)$,

$$[\Sigma]_C^{\exists}=[\Sigma]_C^{\forall}=[\Sigma]_C, \qquad [\Sigma^*]_C^{\exists}=[\Sigma^*]_C^{\forall}=[\Sigma^*]_C,$$

and moreover, $([\Sigma]_C)^*=[\Sigma^*]_C=\Theta(C)$.

1.17. Example. Let $C=(\Sigma, I, D)$ be the reliance alphabet such that $\Sigma=\{a, b\}$ and $I=\{\{a, b\}\}$ (hence, $D=\{(a, a), (b, b)\}$).

(1) $\Theta(C)=\{t \subseteq \Sigma^*:$ there exist $m, n \geqslant 0$ such that $t=\{x \in \Sigma^*: \#_a(x)=m$ and $\#_b(x)=n\}\}$.

(2) Let $K=\{ab\}^*\{ba\}^*$. Then,

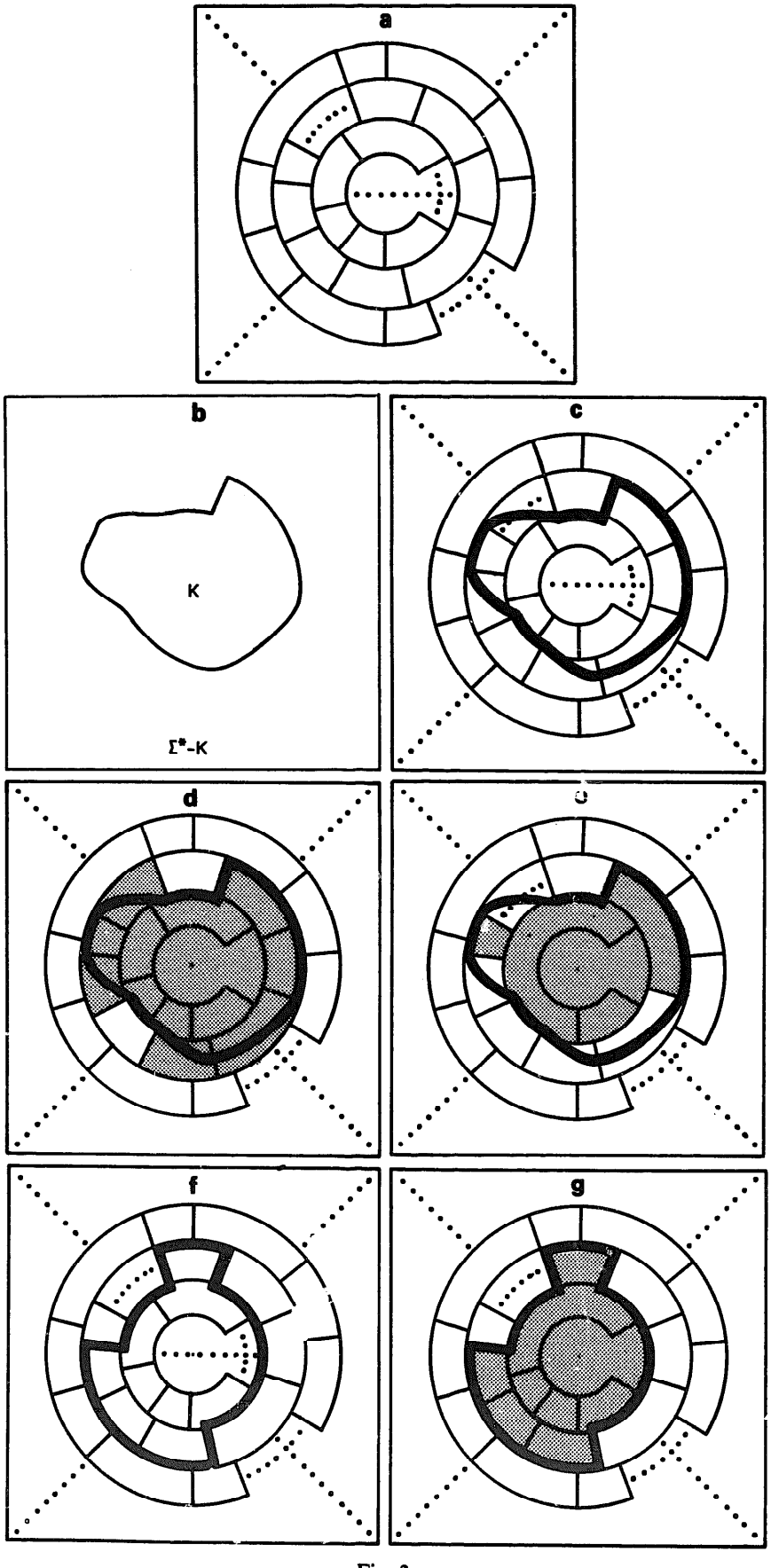


Fig. 3

(i) $[K]_C^{\exists} = \{t \subseteq \Sigma^* : \text{there exists an } m \geq 0 \text{ such that } t = \{x \in \Sigma^* : \#_a(x) = \#_b(x) = m\}\}$;
(ii) $[K]_C^{\forall} = \{1, [ab]\}$; and
(iii) $[K]_C$ is undefined, because $[K]_C^{\exists} \neq [K]_C^{\forall}$.
(3) Let $K = \{a\}^*\{b\}\{a\}^*\{b\}\{a\}^*$. Then,
(i) $[K]_C^{\exists} = \{t \subseteq \Sigma^* : \text{there exists an } m \geq 0 \text{ such that } t = \{x \in \Sigma^* : \#_a(x) = m \text{ and } \#_b(x) = 2\}\}$;
(ii) $[K]_C^{\forall} = [K]_C^{\exists}$; and, consequently,
(iii) $[K]_C$ is defined and $[K]_C = [K]_C^{\exists} = [K]_C^{\forall}$.

Since we use string languages to specify trace languages, we may use classes of string languages to define classes of trace languages. Thus, e.g., the classical Chomsky hierarchy yields the following classification of trace languages.

1.18. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $T \subseteq \Theta(C)$.

(1) T is *existentially* regular (context-free, context-sensitive) if $T = [K]_C^{\exists}$ for a regular (context-free, context-sensitive, respectively) string language $K \subseteq \Sigma^*$.

(2) T is *universally* regular (context-free, context-sensitive) if $T = [K]_C^{\forall}$ for a regular (context-free, context-sensitive, respectively) string language $K \subseteq \Sigma^*$.

(3) T is *consistently* regular (context-free, context-sensitive) if $T = [K]_C$ for a regular (context-free, context-sensitive, respectively) string language $K \subseteq \Sigma^*$.

Given a reliance alphabet C, we will use the following notations to denote the classes of trace languages defined above. $T_C^{\exists}(\mathrm{REG})$, $T_C^{\forall}(\mathrm{REG})$ and $T_C(\mathrm{REG})$ denote the classes of existentially, universally and consistently regular trace languages (over C), respectively; analogously we have the notations $T_C^{\exists}(\mathrm{CF})$, $T_C^{\forall}(\mathrm{CF})$, $T_C(\mathrm{CF})$, $T_C^{\exists}(\mathrm{CS})$, $T_C^{\forall}(\mathrm{CS})$, and $T_C(\mathrm{CS})$.

1.3. Dependence graphs—more basics

In this subsection we will introduce more basic notions and notations concerning dependence graphs and dependence graph languages.

First we introduce some slightly different types of dependence graphs.

1.19. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) Let $x \in \Sigma^*$. *The canonical reduced* (*transitive*) *dependence graph of* x (*over* C), denoted by $\langle x \rangle_C^{\mathrm{r}}$ ($\langle x \rangle_C^{\mathrm{t}}$ respectively), is the directed node-labeled acyclic graph $\mathrm{red}(\langle x \rangle_C)$ ($\mathrm{trans}(\langle x \rangle_C)$ respectively).

(2) Let $x \in \Sigma^*$. A *reduced* (*transitive*) *dependence graph of* x (*over* C) is a directed node-labeled acyclic graph d over Σ such that $d \cong \langle x \rangle_C^{\mathrm{r}}$ ($d \cong \langle x \rangle_C^{\mathrm{t}}$ respectively).

(3) A *reduced* (*transitive*) *dependence graph* (*over* C) is a directed node-labeled acyclic graph d over Σ such that $d \cong \langle x \rangle_C^{\mathrm{r}}$ ($d \cong \langle x \rangle_C^{\mathrm{t}}$ respectively) for some $x \in \Sigma^*$. A set of reduced (transitive) dependence graphs (over C) is called a *reduced* (*transitive* respectively) *dependence graph language* (*over* C).

Hence, in the notation as above, $\langle x\rangle_C^r$ is obtained from $\langle x\rangle_C$ by omitting from it all 'transitive' edges and $\langle x\rangle_C^t$ is obtained from $\langle x\rangle_C$ by adding to it all possible 'transitive' edges.

For a given reliance alphabet C, the set of all reduced (transitive) dependence graphs over C will be denoted by $\Delta^r(C)$ ($\Delta^t(C)$ respectively).

1.20. Remark. Note that, for a given reliance alphabet C and an $x \in (\text{alph}(C))^*$, $\langle x\rangle_C^r$ and $\langle x\rangle_C^t$ are unique because $\langle x\rangle_C$ is unique. Furthermore, note that, for a given reliance alphabet C and an $x \in (\text{alph}(C))^*$, all of $\langle x\rangle_C$, $\langle x\rangle_C^r$ and $\langle x\rangle_C^t$ represent the same partial order on the occurrences of symbols of x: $\langle x\rangle_C^r$ is the *covering graph* of the partial order in question, while $\langle x\rangle_C^t$ *fully represents* it. It is clear that, while working with these (different types of) dependence graphs which are convenient mathematical objects, we actually work with, and take the point of view of, labeled partial orders.

1.21. Example. Let C be the reliance alphabet given in Example 1.4. The graph depicted in Fig. 4(a) is a reduced dependence graph of *dbcee* (over C) and the graph depicted in Fig. 4(b) is a transitive dependence graph of *dbcee* (over C).

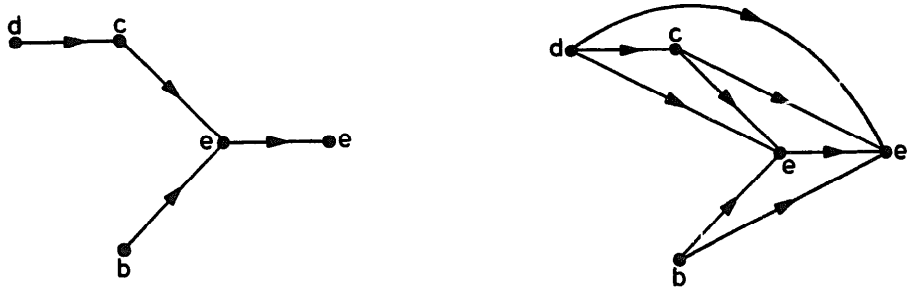


Fig. 4

The different types of dependence graph languages may be specified using (different sorts of) string languages. This leads us to the following notions.

1.22. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) Let $K \subseteq \Sigma^*$. The (*reduced, transitive*) *dependence graph language of* K (*over* C), denoted by $\langle K\rangle_C$ ($\langle K\rangle_C^r$, $\langle K\rangle_C^t$, respectively), is the set $\{d \in \Delta(C)$: there exists an $x \in K$ such that $d \cong \langle x\rangle_C\}$ ($\{d \in \Delta^r(C)$: there exists an $x \in K$ such that $d \cong \langle x\rangle_C^r\}$, $\{d \in \Delta^t(C)$: there exists an $x \in K$ such that $d \cong \langle x\rangle_C^t\}$, respectively).

(2) Let $W \subseteq \Gamma(\Sigma)$. W is called *regular* (*context-free*) if $W = \langle K\rangle_C$ for a regular (context-free respectively) string language $K \subseteq \Sigma^*$.

Given a reliance alphabet C, $D_C(\text{REG})$ and $D_C(\text{CF})$ will denote the classes of regular and context-free dependence graph languages (over C) respectively.

We now move to consider graph composition operations on graphs. Later on, they will turn out to be useful in dealing with different types of dependence graphs.

1.23. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet.

(1) Let $g_1=(V_1, E_1, \Sigma, l_1)$ and $g_2=(V_2, E_2, \Sigma, l_2)$ be two directed node-labeled graphs, such that $V_1 \cap V_2=\emptyset$. The (*reduced, transitive*) *graph composition of* g_1 *and* g_2 (*over* C), denoted by $g_1 \square g_2$ ($g_1 \overset{r}{\square} g_2$, $g_1 \overset{t}{\square} g_2$, respectively), is the directed node-labeled graph $g=(V, E, \Sigma, l)$ ($g=\mathrm{red}(V, E, \Sigma, l)$, $g=\mathrm{trans}(V, E, \Sigma, l)$, respectively), where:

(i) $V=V_1 \cup V_2$;

(ii) $E=E_1 \cup E_2 \cup \{(v_1, v_2) \in V_1 \times V_2 : \{l_1(v_1), l_2(v_2)\} \in D\}$; and

(iii) for all $v \in V$, $l(v)=l_1(v)$ if $v \in V_1$ and $l(v)=l_2(v)$ if $v \in V_2$.

(2) Let g_1 and g_2 be two directed node-labeled graphs over Σ. A directed node-labeled graph g over Σ is called a (*reduced, transitive*) *graph composition of* g_1 *and* g_2 (*over* C) if there exist $g_1'=(V_1, E_1, \Sigma, l_1)$ and $g_2'=(V_2, E_2, \Sigma, l_2)$, such that $g_1' \cong g_1$, $g_2' \cong g_2$, $V_1 \cap V_2=\emptyset$, and $g \cong g_1' \square g_2'$ ($g \cong g_1' \overset{r}{\square} g_2'$, $g \cong g_1' \overset{t}{\square} g_2'$, respectively).

1.24. Remark. In order to facilitate our notation, we have not included the subscript C in the graph composition symbols ($\square$, $\overset{r}{\square}$, and $\overset{t}{\square}$). Still it should be clear that the different types of graph compositions of directed node-labeled graphs are dependent on the reliance alphabet involved.

1.25. Example. Let C be the reliance alphabet given in Example 1.4, let d_1 be the dependence graph over C as depicted in Fig. 5(a), and let d_2 be the dependence graph over C as depicted in Fig. 5(b). Then a graph composition of d_1 and d_2, a reduced graph composition of d_1 and d_2 and a transitive graph composition of d_1 and d_2 are shown in Figs. 2, 4(a) and 4(b) respectively.

Fig. 5

In order to simplify our notation, in the sequel of this paper we will skip subscripts referring to a reliance alphabet C, whenever it is understood from the context. Thus, e.g., we may write for a given reliance alphabet $C=(\Sigma, I, D)$, $x, y \in \Sigma^*$ and $K \subseteq \Sigma^*$, $x \doteq y$ rather than $x \doteq_C y$, $[x]$ rather than $[x]_C$, $\langle K \rangle$ rather than $\langle K \rangle_C$, etc. Furthermore, we often will abbreviate 'dependence graph' by 'dep-graph'.

Bibliographical comments

The notions of an independence relation, a reliance alphabet, a trace, and a trace language were introduced by Mazurkiewicz in [31]—this paper has initiated the theory of traces. The correspondence between the theory of traces and the theory of partially commutative monoids was established in [4]. Given this correspondence one may consider an alternative origin of the theory of traces (see Introduction).

The existential way of defining trace languages via string languages was indicated in [31]; however, the first extensive research on existentially regular, existentially context-free, and existentially context-sensitive trace languages is presented in [46]. Consistently regular trace languages were introduced in [4].

Finally, because of the correspondence between traces and dependence graphs (discussed in the next section) dependence graphs are considered either explicitly or implicitly in quite a number of papers (see, e.g., [31, 33, 46]).

2. Basic properties

In this section a number of basic properties of traces and trace languages (Subsection 2.1) as well as of dep-graphs and dep-graph languages (Subsection 2.2) are considered.

We begin by giving a result relating traces with dep-graphs.

2.1. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $x, y \in \Sigma^*$. Then $[x] = [y]$ if and only if $\langle x \rangle \cong \langle y \rangle$.*

The above result underlies much of the philosophy (and techniques) of trace theory as presented in this paper. It says that, for a given reliance alphabet C, identifying strings having the same trace (which is done on basis of $\mathrm{ind}(C)$) yields the same result as identifying strings having the same dep-graph (which is done on basis of $\mathrm{dep}(C)$); thus strings are C-equivalent if and only if they are C-isomorphic.

2.1. Traces and trace languages

In this subsection we provide basic properties concerning traces and trace languages and introduce a number of new notions and notations related to these properties.

Since, according to Theorem 2.1, a dep-graph is 'invariant' for all strings belonging to a given trace, we can associate a dep-graph with a whole trace. Consequently, we get a clear correspondence between traces and dep-graphs, which can be partly expressed through the following definition (the other part of this correspondence is expressed through Definition 2.25.

2.2. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet. For a $d \in \Delta(C)$ ($d \in \Delta^r(C)$, $d \in \Delta^t(C)$), the *trace of d (over C)*, denoted by $[\![d]\!]_C$, is the trace $[x]_C$, where $x \in \Sigma^*$ is such that $d \cong \langle x \rangle_C$ ($d \cong \langle x \rangle^r_C$, $d \cong \langle x \rangle^t_C$, respectively).

2.3. Remark. Viewing a dep-graph d over a reliance alphabet C as a representation of a labeled partial order π, $[\![d]\!]_C$ is nothing else as the set of strings corresponding to linear extensions of π.

Again, in the notation as above, we write $[\![d]\!]$ rather than $[\![d]\!]_C$, whenever C is understood from the context.

In understanding basic properties of traces, the following easy to prove results are quite useful.

2.4. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet. Let $a_1, a_2 \in \Sigma$, $u_1, u_2 \in \Sigma^*$ and $t_1, t_2, t_3, t_4 \in \Theta(C)$.*

(1) $[u_1]=1$ *if and only if* $u_1=\lambda$.

(2.1) $t_1 \circ [a_1] = t_2 \circ [a_2]$ *and* $a_1 \neq a_2$ *imply*:

(i) $\{a_1, a_2\} \in I$, *and*

(ii) *there exists* $s \in \Theta(C)$ *such that* $t_1 = s \circ [a_2]$ *and* $t_2 = s \circ [a_1]$.

(2.2) $[a_1] \circ t_1 = [a_2] \circ t_2$ *and* $a_1 \neq a_2$ *imply*:

(i) $\{a_1, a_2\} \in I$, *and*

(ii) *there exists* $s \in \Theta(C)$ *such that* $t_1 = [a_2] \circ s$ *and* $t_2 = [a_1] \circ s$.

(3) $t_1 \circ t_2 \circ t_3 = t_1 \circ t_4 \circ t_3$ *if and only if* $t_2 = t_4$.

(4) *If* $\{b_1, b_2\} \in I$ *for all* $b_1 \in \mathrm{alph}(u_1)$ *and* $b_2 \in \mathrm{alph}(u_2)$ *such that* $b_1 \neq b_2$, *then* $[u_1] \circ [u_2] = [u_1] \| [u_2]$.

2.5. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet.*

(1) *If* $I=\emptyset$, *then, for every* $t \in \Theta(C)$, $\#t=1$.

(2) *If* $\#\Sigma=1$, *then, for every* $t \in \Theta(C)$, $\#t=1$.

(3) *If* $D=\mathrm{id}(\Sigma)$, *then, for every* $x, y \in \Sigma^*$, $[x]=[y]$ *if and only if, for every* $a \in \Sigma$, $\#_a(x)=\#_a(y)$ (*i.e., if and only if* $\psi(x)=\psi(y)$).

Furthermore, one has the following more involved results concerning traces.

The first one provides a characterization of the C-equivalence relation. It is easy to see that, for a given reliance alphabet C and for $x, y \in (\mathrm{alph}(C))^*$, if $x \equiv y$, then, for each $a \in \mathrm{alph}(C)$, $\#_a(x)=\#_a(y)$. However, in general, two strings x and y over $\mathrm{alph}(C)$ having the same number of occurrences of each letter in $\mathrm{alph}(C)$ do not have to be C-equivalent. Thus, in order to characterize the C-equivalence relation, one has to add an additional condition.

2.6. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet. For every $x, y \in \Sigma^*$, $x \equiv y$ if and only if [for every $a \in \Sigma$, $\#_a(x)=\#_a(y)$, and, for every $\{a, b\} \in D$, $\phi_{\{a,b\}}(x)=\phi_{\{a,b\}}(y)$].*

The next result is useful in 'trace calculus' and can be seen as a generalization of the celebrated Levi Lemma for strings (see [29]).

2.7. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet. For every $x, y, z, w \in \Sigma^*$, $[xy]=[zw]$ if and only if there exist $u, v, r, s \in \Sigma^*$ such that $[x]=[uv]$, $[y]=[rs]$, $[z]=[ur]$, $[w]=[vs]$, and, for every $a \in \mathrm{alph}(v)$ and $b \in \mathrm{alph}(r)$, $\{a, b\} \in I$.*

In general, a trace can be obtained as a trace composition of other traces, but trace decompositions of the given trace do not have to be unique. This feature often complicates proofs of various properties of traces (and trace languages). Hence it is desirable to have 'normal form' decompositions of traces. One of such normal forms will be discussed now.

2.8. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet. Every $t \in \Theta(C)$, $t \neq 1$, has a unique decomposition $t = t_1 \circ \cdots \circ t_m$, $m \geq 1$, such that:*

(i) *for all $1 \leq i \leq m$, $t_i \neq 1$;*

(ii) *for all $1 \leq i \leq m$, t_i can be written as $[u_i]$, where $u_i \in \Sigma^*$, $\#_a(u_i) = 1$ for each $a \in \text{alph}(u_i)$, and $\{a, b\} \in I$ for all $a, b \in \text{alph}(u_i)$ such that $a \neq b$; and*

(iii) *for all $1 \leq i \leq m-1$, if $t_i = [u_i]$ and $t_{i+1} = [u_{i+1}]$, then, for each $a \in \text{alph}(u_{i+1})$, there exists $b \in \text{alph}(u_i)$ such that $\{a, b\} \in D$.*

Hence, by the above result, every trace can be uniquely decomposed into a minimal number of 'maximal independent' parts. This decomposition yields a unique representation for each trace t and we will refer to it as the *normal form of t*. Furthermore, since the string $u_1 \ldots u_m$ is a representative of t, $u_1 \ldots u_m$ will be referred to as a *normal representative of t*. Finally, if we also order the alphabet Σ involved and, for all $1 \leq i \leq m$, write the symbols in u_i in that order, then the normal form of t yields the unique string $u_1 \ldots u_m$ over Σ, called the *minimal normal representative of t*; in other words, the minimal normal representative of t is the string $\min(\{x \in \Sigma^* : x$ is a normal representative of $t\})$.

2.9. Example. Let C be the reliance alphabet given in Example 1.4. Consider $t = [adcbbed]$. We note that $t = t_1 \circ t_2 \circ t_3 \circ t_4$, where $t_1 = [ab]$, $t_2 = [db]$, $t_3 = [c]$ and $t_4 = [ed]$. Since t, t_1, t_2, t_3, and t_4 satisfy the conditions (i), (ii) and (iii) from the statement of Theorem 2.8, $t_1 \circ t_2 \circ t_3 \circ t_4$ is the (unique) normal form of t and *abdbced* is a normal representative of t.

If we asume now the order of Σ to be (a, b, c, d, e), then the string *abbdcde* is the minimal normal representative of t (for the given ordering of Σ).

The following (deterministic) algorithm yields a normal representative of a trace for a given representative of the trace.

First we give an intuitive description of the algorithm.

Let $C = (\Sigma, I, D)$ be a reliance alphabet, let t be a trace over C of which we want to find a normal representative, and assume that t is given through a representative $w \in \Sigma^+$; C and w are the inputs of the algorithm. Let, for some $n \geq 1$, $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and let $l = |w|$.

The algorithm will construct the decomposition $t = t_1 \circ \cdots \circ t_m$ for some $m \leq l$ by constructing strings $u_1, \ldots, u_m \in \Sigma^*$ such that $t_i = [u_i]$ for all $1 \leq i \leq m$. The algorithm uses variables $u(1), \ldots, u(l)$, where, for all $1 \leq i \leq m$, $u(i)$ stores the so-far computed prefix of u_i. Initially, $u(1), \ldots, u(l)$ are set to λ.

For all $1 \leq j \leq n$, there is a pointer $p(j)$ which indicates that the next scanned occurrence of σ_j will be appended to the current value of $u(p(j))$. Initially, $p(1), \ldots, p(n)$ are set to 1.

To construct the decomposition of t, the algorithm reads w from left to right, symbol by symbol in one pass. The values of prefixes $u(1), \ldots, u(m)$ and the positions of pointers $p(1), \ldots, p(n)$ are updated as follows. Assume that currently the algorithm reads the kth element of w, $1 \leq k \leq l$, which is an occurrence of σ_j for some $1 \leq j \leq n$. Then,

(i) σ_j is appended to the current value of $u(p(j))$;

(ii) for all $1 \leq i \leq n$ such that $i \neq j$, σ_i and σ_j are dependent, and $p(i) \leq p(j)$, the value of $p(i)$ is set to $p(j)+1$; and

(iii) the value of $p(j)$ is increased by 1.

((ii) and (iii) have to be performed because the next occurrence of each such σ_i in w as well as the next occurrence of σ_j in w will have to occur in the decomposition of t in a later 'block'.)

After (i) and (ii) are performed (in this order) upon the reading of the kth element of w, $1 \leq k \leq l-1$, the algorithm moves to scan the $(k+1)$st element of w.

After the lth element of w has been scanned, the algorithm sets m equal to the largest i such that $u(i) \neq \lambda$ and the algorithm outputs the string $u(1) \ldots u(m)$.

More formally, the algorithm is as follows.

2.10. Algorithm.

Input: A reliance alphabet $C = (\Sigma, I, D)$ and a string $w \in \Sigma^+$.

Declaration: Let $l = |w|$, $n = \#\Sigma$ and $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$; let k be a variable over $\{0, \ldots, l\}$ and let p be an array of length n over $\{1, \ldots, l+1\}$; let i and j be variables over $\{1, \ldots, n\}$; let u be an array of length l over $(\Sigma \cup \{\lambda\})^n$; let m be a variable over $\{1, \ldots, l\}$.

Computation:

(1) for all $1 \leq i \leq n$, $p(i) := 1$.
(2) for all $1 \leq k \leq l$, $u(k) := \lambda$.
(3) $k := 0$.
(4) $k := k+1$.
(5) set j such that $w(k) = \sigma_j$.
(6) $u(p(j)) := u(p(j)) \cdot \sigma_j$.
(7) for all $1 \leq i \leq n$ such that $i \neq j$, $p(i) \leq p(j)$ and $\{\sigma_i, \sigma_j\} \in D$, $p(i) := p(j)+1$.
(8) $p(j) := p(j)+1$.
(9) if $k < l$ then goto (4).
(10) set m such that $u(m) \neq \lambda$ and either $m < l$ and $u(m+1) = \lambda$, or $m = l$.
(11) stop.

Output: The string $u(1) \ldots u(m)$.

2.11. Example. Let C be the reliance alphabet given in Example 1.4 and let $w =$ *adcbbed*. Then the computation of Algorithm 2.10, starting with C and w, can be illustrated as in Fig. 6. Hence, the output of the algorithm is *abdbced*.

	adcbbed	u(1)	u(2)	u(3)	u(4)	u(5)	u(6)	u(7)
0	adcbbed	abcde						
1	dcbbed	a b	acde					
2	cbbed	a b	d e	acd				
3	bbed	a b	d	c	acde			
4	bed	ab	d b	c	acde			
5	ed	ab	db	c b	acde			
6	d	ab	db	c	e d	abce		
7		ab	db	c	ed	abcde		
		u(1)	u(2)	u(3)	u(4)	u(5)	u(6)	u(7)

Fig. 6

Formalizing the above we get the following result.

2.12. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $w \in \Sigma^+$. Let x be the output of Algorithm* 2.10 *for the input (C, w). Then x is a normal representative of $[w]$.*

The computation of a normal representative of a trace by Algorithm 2.10 is done quite efficiently.

2.13. Theorem. (1) *For a reliance alphabet $C = (\Sigma, I, D)$ with $n = \#\Sigma$ and a string $w \in \Sigma^+$ with $l = |w|$, a normal representative of $[w]$ is constructed by Algorithm* 2.10 *in time $O(nl)$.*

(2)' *Let $C = (\Sigma, I, D)$ be a reliance alphabet with $n = \#\Sigma$. For a string $w \in \Sigma^+$ with $l = |w|$, a normal representative of $[w]$ is constructed by Algorithm* 2.10 *in time $O(l)$.*

Proof. Consider Algorithm 2.10. The theorem follows directly from the following observations:

(i) the steps (1), (2), (3), and (10) are executed once;
(ii) the steps (4) through (9) are executed l times;
(iii) the steps (3), (4), (6), (8), and (9) have a constant time complexity;
(iv) the steps (1), (5), and (7) have a time complexity which is linear in n; and
(v) the steps (2) and (10) have a time complexity which is linear in l. □

We now move to basic properties of trace languages.

2.14. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K, L \subseteq \Sigma^*$.*

(1) $K=\emptyset$ *if and only if* $[K]^{\exists}=\emptyset$.
(2) $K \subseteq L$ *implies that* $[K]^{\exists} \subseteq [L]^{\exists}$.
(3) $[K]^{\exists} \circ [L]^{\exists} = [KL]^{\exists}$.
(4) $[K^*]^{\exists} = ([K]^{\exists})^*$.
(5) $[K]^{\exists} \cup [L]^{\exists} = [K \cup L]^{\exists}$.
(6) $[K \cap L]^{\exists} \subseteq [K]^{\exists} \cap [L]^{\exists}$.
(7) $K=\Sigma^*$ *implies that* $[K]^{\exists} = \Theta(C)$.

2.15. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K, L \subseteq \Sigma^*$.*

(1) $K=\emptyset$ *implies that* $[K]^{\forall}=\emptyset$.
(2) $K \subseteq L$ *implies that* $[K]^{\forall} \subseteq [L]^{\forall}$.
(3) $[K]^{\forall} \cup [L]^{\forall} \subseteq [K \cup L]^{\forall}$.
(4) $[K]^{\forall} \cap [L]^{\forall} = [K \cap L]^{\forall}$.
(5) $K=\Sigma^*$ *if and only if* $[K]^{\forall} = \Theta(C)$.

2.16. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K, L \subseteq \Sigma^*$.*

(1) $[K]^{\forall} \subseteq [K]^{\exists}$.
(2) $[K]^{\exists} = \Theta(C) - [\Sigma^* - K]^{\forall}$.
(3) $[K]^{\forall} = \Theta(C) - [\Sigma^* - K]^{\exists}$.
(4) $[K-L]^{\exists} \subseteq [K]^{\exists} - [L]^{\forall}$.
(5) $[K-L]^{\forall} = [K]^{\forall} - [L]^{\exists}$.

The following 'collective' example provides a ramification of the last three theorems. It provides examples needed to prove that inclusions (implications) from the above three results can be strict (cannot be turned into 'if-and-only-if' statements respectively). After each example we provide (in brackets) a pointer to the inclusion (implication) that can be proved to be strict (can be shown to be not convertible into an 'if-and-only-if' statement respectively) using this particular counter-example; exceptions are the points (7) through (10), which demonstrate why Theorem 2.15 does not state analogies of the points (3) and (4) from Theorem 2.14.

2.17. Example. Let $C=(\Sigma, I, D)$ be the reliance alphabet such that $\Sigma=\{a, b\}$ and $I=\{\{a, b\}\}$.

(1) $[\{ab\}]^{\exists}=\{\{ab, ba\}\}\subseteq\{\{ab, ba\}\}=[\{ba\}]^{\exists}$, while $\{ab\}\not\subseteq\{ba\}$ (Theorem 2.14(2)).

(2) $[\{ab\}]^{\exists}\cap[\{ba\}]^{\exists}=\{\{ab, ba\}\}\cap\{\{ab, ba\}\}=\{\{ab, ba\}\}\not\subseteq\emptyset=[\emptyset]^{\exists}$ $=[\{ab\}\cap\{ba\}]^{\exists}$ (Theorem 2.14(6)).

(3) Since $ab\equiv ba$ and $ab\in\Sigma^*-\{ba\}$, $[\Sigma^*-\{ba\}]^{\exists}=\Theta(C)$, while $\Sigma^*-\{ab\}\neq\Sigma^*$ (Theorem 2.14(7)).

(4) $[\{ab\}]^{\forall}=\emptyset$, while $\{ab\}\neq\emptyset$ (Theorem 2.15(1)).

(5) $[\{ab\}]^{\forall}=\emptyset\subseteq\emptyset=[\{ba\}]^{\forall}$, while $\{ab\}\not\subseteq\{ba\}$ (Theorem 2.15(2)).

(6) $[\{ab\}\cup\{ba\}]^{\forall}=[\{ab, ba\}]^{\forall}=\{\{ab, ba\}\}\not\subseteq\emptyset=[\{ab\}]^{\forall}\cup[\{ba\}]^{\forall}$ (Theorem 2.15(3)).

(7) $[\{a\}]^{\forall}\circ[\{b\}]^{\forall}=\{\{a\}\}\circ\{\{b\}\}=\{\{ab, ba\}\}\not\subseteq\emptyset=[\{ab\}]^{\forall}=[\{a\}\{b\}]^{\forall}$.

(8) $[\{\lambda, ab\}\{\lambda, ba\}]^{\forall}=[\{1, ab, ba, abba\}]^{\forall}=\{1, [ab]\}\not\subseteq\{1\}=[\{\lambda\}]^{\forall}\circ[\{\lambda\}]^{\forall}$ $=[\{\lambda, ab\}]^{\forall}\circ[\{\lambda, ba\}]^{\forall}$.

(9) If $K=\{ab, aabb, abba, baab, baba, bbaa\}$, then $abab\in K^*$ and consequently, $[aabb]\in[K^*]^{\forall}$. Hence, since $[K]^{\forall}=\emptyset$, $[aabb]\in[K^*]^{\forall}\not\subseteq\{1\}=\emptyset^*=([K]^{\forall})^*$.

(10) If $K=\{ab, ba\}$, then $([K]^{\forall})^*=\{\{ab, ba\}\}^*$ and consequently, $[abab]=[ab]\circ[ab]=\{ab, ba\}\circ\{ab, ba\}\in([K]^{\forall})^*$. Hence, since $[K^*]^{\forall}=\{\{ab, ba\}, 1\}$, $[abab]\notin[K^*]^{\forall}$ and thus, $([K]^{\forall})^*\not\subseteq[K^*]^{\forall}$.

(11) $[\{ab\}]^{\exists}=\{\{ab, ba\}\}\not\subseteq\emptyset=[\{ab\}]^{\forall}$ (Theorem 2.16(1)).

(12) $[\{ab\}]^{\exists}-[\{ab\}]^{\forall}=\{\{ab, ba\}\}-\emptyset\not\subseteq\emptyset=[\emptyset]^{\exists}=[\{ab\}-\{ab\}]^{\exists}$ (Theorem 2.16(4)).

Given a reliance alphabet C and a $K\subseteq(\mathrm{alph}(C))^*$, it is important (for defining $[K]^{\exists}$ and $[K]^{\forall}$) to understand the way in which the partition of Σ^* induced by $\equiv$ partitions K. To this aim we introduce now a number of technical notions and state some of their basic properties.

2.18. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K\subseteq\Sigma^*$.

(1) The *C-exterior of K*, denoted by $\uparrow_C K$, is the string language $\bigcup\{t\in\Theta(C): t\in[K]^{\exists}\}$.

(2) The *C-interior of K*, denoted by $\downarrow_C K$, is the string language $\bigcup\{t\in\Theta(C): t\in[K]^{\forall}\}$.

(3) K is *C-consistent* if and only if $[K]^{\exists}=[K]^{\forall}$.

2.19. Remark. Note that thus, for a reliance alphabet C and a $K\subseteq(\mathrm{alph}(C))^*$, the C-exterior of K is based on the existential trace language of K, while the C-interior of K is based on the universal trace language of K.

Again, to simplify our notation, we write $\uparrow K$ and $\downarrow K$ rather than $\uparrow_C K$ and $\downarrow_C K$, respectively, whenever C is understood from the context.

The basic relationships between the above notions and existentially and universally defined trace languages are given by the following results.

2.20. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K\subseteq\Sigma^*$.*

(1) $\downarrow K\subseteq K\subseteq\uparrow K$.

(2) $\uparrow K=\Sigma^*-\downarrow(\Sigma^*-K)$.

(3) $\downarrow K=\Sigma^*-\uparrow(\Sigma^*-K)$.

2.21. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet, let $T\subseteq\Theta(C)$ and let $K=\bigcup\{t\in\Theta(C): t\in T\}$. Then $T=[K]^{\exists}=[K]^{\forall}$.*

2.22. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K\subseteq\Sigma^*$.*

(1) *$\uparrow K$ and $\downarrow K$ are C-consistent.*

(2) $\uparrow K=\bigcap\{K'\subseteq\Sigma^*: K\subseteq K'$ *and* K' *is C-consistent}* and $\downarrow K=\bigcup\{K'\subseteq\Sigma^*: K'\subseteq K$ *and* K' *is C-consistent}*.

2.23. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K\subseteq\Sigma^*$. The following four statements are equivalent.*

(1) *K is C-consistent.*

(2) $K=\downarrow K$.

(3) $K=\uparrow K$.

(4) *For all $x, y\in\Sigma^*$, $[x]=[y]$ implies $[x\in K$ if and only if $y\in K]$.*

The above notions are closely related to our basic classification (see Definition 1.18) of trace languages.

2.24. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $K\subseteq\Sigma^*$ be regular (context-free, context-sensitive).*

(1) *$\uparrow K$ is a regular (context-free, context-sensitive respectively) string language over Σ if and only if $[K]^{\exists}$ is a consistently regular (context-free, context-sensitive respectively) trace language over C.*

(2) *$\downarrow K$ is a regular (context-free, context-sensitive respectively) string language over Σ if and only if $[K]^{\forall}$ is a consistently regular (context-free, context-sensitive respectively) trace language over C.*

2.2. Dep-graphs and dep-graph languages

In this subsection we will give basic properties of dep-graphs. First we provide some general results concerning dep-graphs, then we take a more detailed look into the different types of dep-graphs and the different types of graph compositions.

We start by formulating the second part of the correspondence between traces and dep-graphs (implied by Theorem 2.1)—the first part of this correspondence was already expressed through Definition 2.2.

2.25. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet, where Σ is ordered.

(1) For a $t\in\Theta(C)$, the *canonical (reduced, transitive) dependence graph of t (over C)*, denoted by $\langle\!\langle t\rangle\!\rangle_C$ ($\langle\!\langle t\rangle\!\rangle_C^{\mathrm{r}}$, $\langle\!\langle t\rangle\!\rangle_C^{\mathrm{t}}$, respectively), is the canonical (reduced, transitive, respectively) dependence graph $\langle x\rangle_C$ ($\langle x\rangle_C^{\mathrm{r}}$, $\langle x\rangle_C^{\mathrm{t}}$, respectively), where x is the minimal normal representative of t.

(2) For a $t \in \Theta(C)$, a (*reduced, transitive*) *dependence graph of* t (*over* C) is a node-labeled acyclic graph d over Σ such that $d \cong \langle\langle t \rangle\rangle_C$ ($d \cong \langle\langle t \rangle\rangle^r_C$, $d \cong \langle\langle t \rangle\rangle^t_C$, respectively).

Again, in the notation as above, we write $\langle\langle t \rangle\rangle$, $\langle\langle t \rangle\rangle^r$, and $\langle\langle t \rangle\rangle^t$ rather than $\langle\langle t \rangle\rangle_C$, $\langle\langle t \rangle\rangle^r_C$, and $\langle\langle t \rangle\rangle^t_C$, respectively, whenever C is understood from the context.

2.26. Remark. Note that the canonical (reduced, transitive) dep-graph of a trace can only be defined when the alphabet considered is ordered. However, since we always deal with finite alphabets, we may assume that, whenever we deal with the canonical (reduced, transitive) dep-graph of a trace, the alphabet considered is ordered.

We continue by giving a characterization of dep-graphs.

2.27. Theorem. *Let* $C = (\Sigma, I, D)$ *be a reliance alphabet and let* $g = (V, E, \Sigma, l)$ *be a directed node-labeled graph.* g *is a dep-graph over* C *if and only if* [g *is acyclic and, for every* $v', v'' \in V$ *with* $v' \neq v''$, $\{l(v'), l(v'')\} \in D$ *if and only if either* $(v', v'') \in E$ *or* $(v'', v') \in E$].

Proof. If g is a dep-graph over C, then, by definition, we may assume that, for some $n \geq 0$, $V = \{v_1, \ldots, v_n\}$ is such that, for all $1 \leq i, j \leq n$, $(v_i, v_j) \in E$ if and only if [$i < j$ and $\{l(v_i), l(v_j)\} \in D$]. Consequently, g is acyclic and, for every $v', v'' \in V$ with $v' \neq v''$, $\{l(v'), l(v'')\} \in D$ if and only if either $(v', v'') \in E$ or $(v'', v') \in E$.

If g is acyclic, then we may assume that, for some $n \geq 0$, $V = \{v_1, \ldots, v_n\}$ is such that, for all $1 \leq i, j \leq n$, $(v_i, v_j) \in E$ implies that $i < j$. Hence, if additionally g is such that, for all $1 \leq i, j \leq n$ with $i \neq j$, $\{l(v_i), l(v_j)\} \in D$ if and only if either $(v_i, v_j) \in E$ or $(v_j, v_i) \in E$, then, for all $1 \leq i, j \leq n$, $(v_i, v_j) \in E$ if and only if [$i < j$ and $\{l(v_i), l(v_j)\} \in D$]. Consequently, g is a dep-graph of the string $l(v_1) \ldots l(v_n)$. □

The following nondeterministic algorithm yields a representative of the trace of a dep-graph. The method used by the algorithm is often referred to as topological sorting (see, e.g., [35]).

Intuitively, given a dep-graph d over a reliance alphabet C, the algorithm iterates the following steps:

(i) choose a minimal node of d;
(ii) concatenate its label to the so far obtained sequence of labels; and
(iii) remove the chosen node (together with its outgoing edges).

This is done until no nodes are left.

2.28. Algorithm.

Input: A reliance alphabet $C = (\Sigma, I, D)$ and a dep-graph $d = (V, E, \Sigma, l) \neq \Lambda$ over C.

Declaration: *Let* $n = \# V$ and $V = \{v_1, \ldots, v_n\}$; let x be a variable over $(\Sigma \cup \{\lambda\})^n$; let i, j and k be variables over $\{0, \ldots, n\}$ and let W be a variable over $\mathcal{P}(\{1, \ldots, n\})$.

Computation:
(1) $x := \lambda$; $i := 0$; $W := \{1, \ldots, n\}$.
(2) $i := i+1$.
(3) set j such that $j \in W$ and, for all $k \in W$, $(v_k, v_j) \notin E$.
(4) $x := x \cdot l(v_j)$; $W := W - \{j\}$.
(5) if $i < n$ then goto (2).
(6) stop.
Output: The string x.

2.29. Example. For C being the reliance alphabet given in Example 1.4 and d being the dep-graph over C depicted in Fig. 2, a computation of the above algorithm is depicted in Fig. 7; in this case $x = dbcee$. Note that another computation would give a different result, e.g., $x = bdcee$.

Formalizing the above we get the following result, the obvious proof of which is left to the reader.

2.30. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $d \neq \Lambda$ be a dep-graph over C. Let X be the set of all outputs of Algorithm* 2.28 *for the input (C, d). Then, for every $x \in \Sigma^*$, $x \in X$ if and only if $[x] = [\![d]\!]$ (i.e., $[x]$ is the trace of d).*

Although the following deterministic algorithm (Algorithm 2.31) yielding a dep-graph of (a representative of) a trace is an obvious implementation of Definition 1.8, for the sake of completeness we provide it here; Algorithm 2.28 and Algorithm 2.31 together form a complete picture of translations (both ways) between dep-graphs and (representatives of) traces.

Intuitively, given a trace t over a reliance alphabet C via a representative x, the algorithm iterates the following steps:

(i) to the so-far obtained graph a node labeled by the first symbol of x (say b) is added;

(ii) this (occurrence of) b is erased from x; and

(iii) the new node gets incoming edges from all other nodes which are labeled by a symbol dependent of b.

This is done until no symbols of x are left.

2.31. Algorithm.

Input: A reliance alphabet $C = (\Sigma, I, D)$ and a string $x \in \Sigma^+$.

Declaration: Let $n = |x|$; let $V = \{v_1, \ldots, v_n\}$, let E be a variable over $\mathcal{P}(V \times V)$ and let l be a variable over $\{f : f$ is a partial function from V into $\Sigma\}$; let i and j be variables over $\{0, \ldots, n\}$.

Computation:
(1) $E := \emptyset$; $i := 0$.
(2) $i := i+1$.

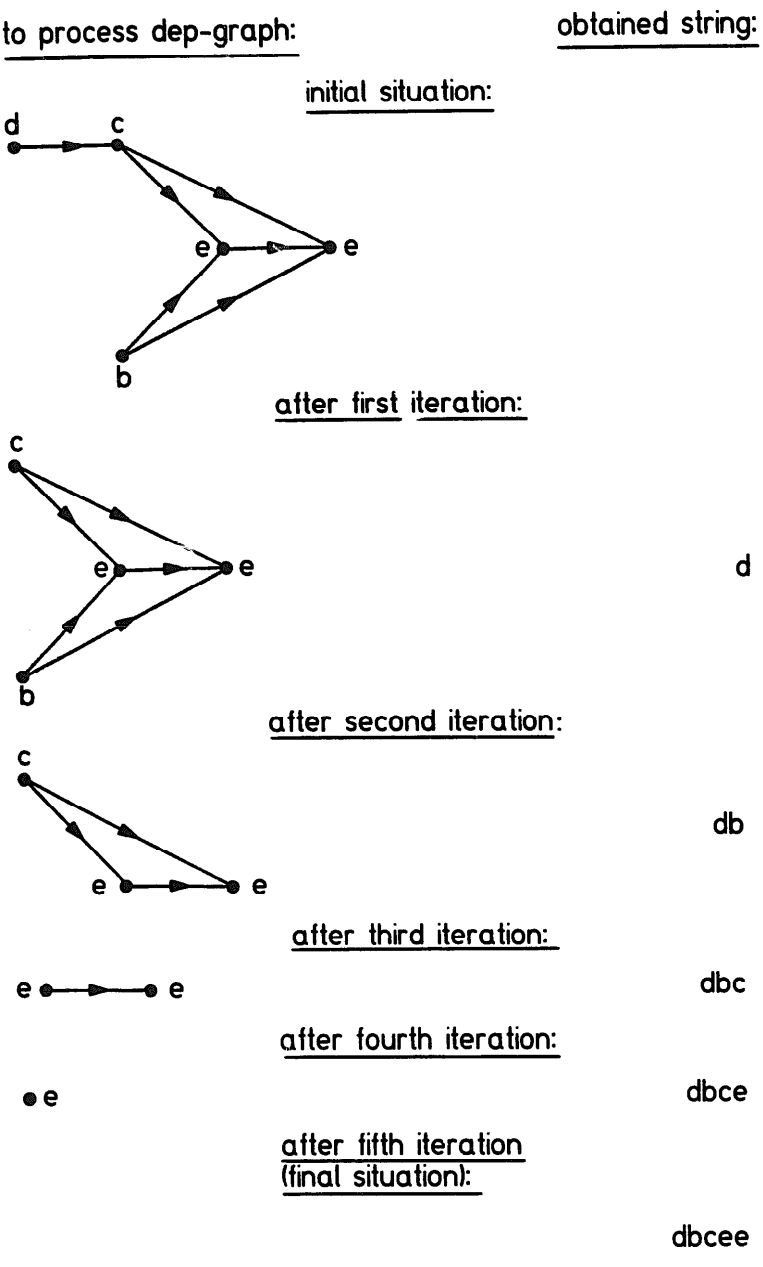


Fig. 7

(3) $l(v_i) := x(i)$.
(4) $E := E \cup \{(v_j, v_i) \in V \times V : 1 \leq j \leq i-1$ and $\{l(v_j), l(v_i)\} \in D\}$.
(5) if $i < n$ then goto (2).
(6) stop.

Output: The directed node-labeled graph (V, E, Σ, l).

2.32. Example. For C being the reliance alphabet given in Example 1.4 and $w =$ *dbcee*, the computation of the above algorithm is depicted in Fig. 8.

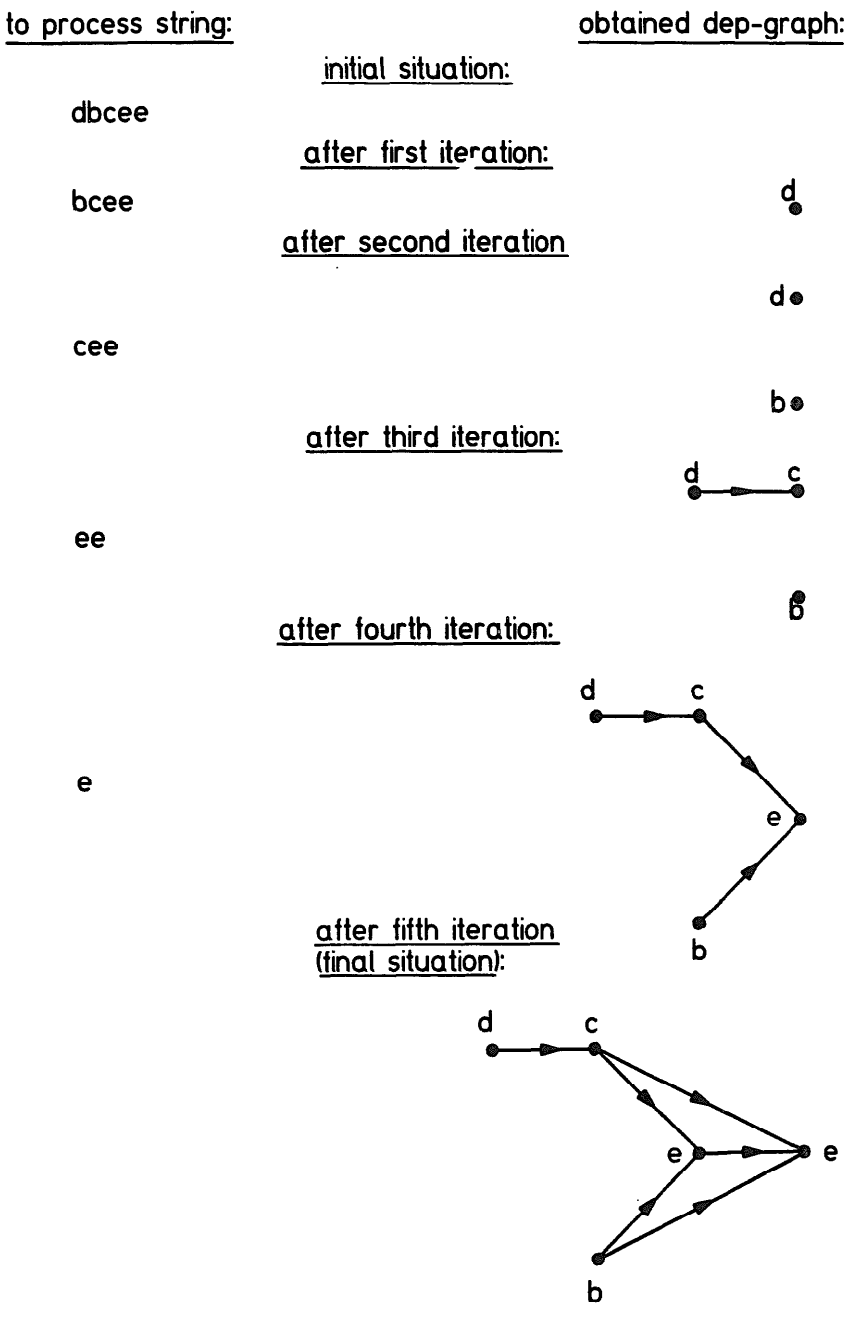


Fig. 8

Formalizing the above we get the following result, whose obvious proof is left to the reader.

2.33. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $x \in \Sigma^+$. Let g be the output of Algorithm* 2.31 *for the input (C, x). Then $g \cong \langle\langle t \rangle\rangle$ (i.e., g is a dep-graph of t), where t is the trace of x.*

Now we consider in more detail the different types of dep-graphs and the different types of graph compositions.

As we have seen in Definition 1.19, for a reliance alphabet C and a string x over alph(C), the canonical reduced (transitive) dep-graph of x can be directly constructed from the canonical dep-graph of x. The following theorem presents a reverse construction.

2.34. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $x\in\Sigma^*$.*

(1) *Let $\langle x\rangle^r=(V, E, \Sigma, l)$, let $E_1=\{(v_1, v_2)\in V\times V: \langle x\rangle^r$ contains a path of length at least 2 from v_1 to $v_2\}$ and let $E_2=\{(v_1, v_2)\in E_1: \{l(v_1), l(v_2)\}\in D\}$. Then $\langle x\rangle=(V, E\cup E_2, \Sigma, l)$ and $\langle x\rangle^t=(V, E\cup E_1, \Sigma, l)$.*

(2) *Let $\langle x\rangle^t=(V, E, \Sigma, l)$, let $E_1=\{(v_1, v_2)\in V\times V: \langle x\rangle^t$ contains a path of length at least 2 from v_1 to $v_2\}$ and let $E_2=\{(v_1, v_2)\in E_1: \{l(v_1), l(v_2)\}\in I\}$. Then $\langle x\rangle=(V, E-E_2, \Sigma, l)$ and $\langle x\rangle^r=(V, E-E_1, \Sigma, l)$.*

Proof. The theorem follows directly from the graph-theoretical observation that, for a directed node-labeled graph $g=(V, E, \Sigma, l)$ and for $v_1, v_2\in V$, g contains a path of length at least 2 from v_1 to v_2 if and only if red(g) contains a path of length at least 2 from v_1 to v_2 if and only if trans(g) contains a path of length at least 2 from v_1 to v_2. □

The above theorem directly implies the following extension of Theorem 2.1.

2.35. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $x, y\in\Sigma^*$. Then the following four statements are equivalent.*

(1) $[x]=[y]$.
(2) $\langle x\rangle\cong\langle y\rangle$.
(3) $\langle x\rangle^r\cong\langle y\rangle^r$.
(4) $\langle x\rangle^t\cong\langle y\rangle^t$.

The different types of dep-graphs are closely related to the operation of trace composition and to the different types of graph compositions as follows.

2.36. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $x, y_1, y_2\in\Sigma^*$. Then the following four statements are equivalent.*

(1) $[x]=[y_1]\circ[y_2]$.
(2) $\langle x\rangle$ *is a graph composition of* $\langle y_1\rangle$ *and* $\langle y_2\rangle$.
(3) $\langle x\rangle^r$ *is a reduced graph composition of* $\langle y_1\rangle^r$ *and* $\langle y_2\rangle^r$.
(4) $\langle x\rangle^t$ *is a transitive graph composition of* $\langle y_1\rangle^t$ *and* $\langle y_2\rangle^t$.

Proof. The theorem directly follows from Definition 1.23, Theorem 2.35, and the observation that, for every $z_1, z_2\in\Sigma^*$, $\langle z_1z_2\rangle$ is a graph composition of $\langle z_1\rangle$ and $\langle z_2\rangle$. □

The edge cuts in dep-graphs play the role of 'gluers in graph compositions'. This is formally expressed by the following result.

2.37. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet.*

(1) *Let $d'=(V', E', \Sigma, l')$ and $d''=(V'', E'', \Sigma, l'')$ be (reduced, transitive) dep-graphs over C such that $V' \cap V''=\emptyset$; let $d=(V, E, \Sigma, l)$ be the (reduced, transitive, respectively) graph composition of d' and d''. If $E-(E' \cup E'') \neq \emptyset$, then $E-(E' \cup E'')$ is an edge cut of d.*

(2) *Let $d=(V, E, \Sigma, l)$ be a (reduced, transitive) dep-graph over C and let F be an edge cut of d. Let $g'=(V', E', \Sigma, l')$ and $g''=(V'', E'', \Sigma, l'')$ be the directed node-labeled graphs such that $V' \cap V''=\emptyset$, $V' \cup V''=V$, $E=E' \cup E'' \cup F$, $F \subseteq V' \times V''$, $l'=l|_{V'}$ and $l''=l|_{V''}$. Then g' and g'' are (reduced, transitive, respectively) dep-graphs over C and d is the (reduced, transitive, respectively) graph composition of g' and g''.*

Proof. (1) directly follows from the fact that in the (reduced, transitive, respectively) graph composition of two graphs, new edges are only edges from the first graph to the second graph.

(2) directly follows from Theorem 2.27, Theorem 2.36 and (1) above. □

We end this subsection with a basic observation concerning node cuts of dep-graphs.

2.38. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet, let $d=(V, E, \Sigma, l)$ be a (transitive, reduced) dep-graph over C, let S be a node cut of d and let α be the size of the greatest clique of I.*

(1) *If $v, w \in S$ and $v \neq w$, then $\{l(v), l(w)\} \in I$.*

(2) $\#S \leq \alpha$.

Proof. (1) directly follows from the fact that in the (reduced, transitive, respectively) graph composition of two graphs, new edges are only edges from the first graph to the second graph.

(2) directly follows from Theorem 2.27, Theorem 2.36 and (1) above. □

Bibliographical comments

The correspondence between traces and dependence graphs (Theorem 2.1) is observed in many different papers, e.g., in [31, 33, 46]. However, with the exception of [33], none of these papers elaborates on this correspondence.

Since many of the results stated in Subsection 2.1 are basic, they can be found in different papers, e.g., in [34, 39, 46], or, in a different terminology, in [38] (it is interesting to notice the similarity of the basic methodologies of the theory of traces and the theory of rough sets). The characterization of the C-equivalence relation (Theorem 2.6) is proved in [39] and the generalization of Levi's Lemma for strings (Theorem 2.7) is proved in [33]. The ideas behind the decomposition of traces into normal forms (Theorem 2.8 and Algorithm 2.10) are from [12], although the proof of Theorem 2.8 can be found in [27]; also [4] and Algorithm 4.1 in Section 4 must be mentioned in this context. The notion of the C-exterior of a string language and

the notion of a C-consistent string language can be found in various papers, e.g., in [38, 46, 47]; in [38] one also finds the notion of the C-interior of a string language.

The graph composition of dependence graphs discussed in Subsection 2.2 is considered for the first time in [45].

3. Trace languages revisited

3.1. Equations on trace languages

As we have seen in the previous sections, we use string languages to define trace languages. One of the important methods of specifying string languages is to use fixed points of equations (see, e.g., [43]). In this subsection we will investigate how fixed points of equations on string languages carry over to fixed points of equations on trace languages. This allows one to use equations on trace languages, where in solving these equations techniques for solving equations on string languages may be used.

3.1. Definition. Let A be a set and let f be a function from $\mathcal{P}(A)$ into $\mathcal{P}(A)$.

(1) f is *monotone*, if, for every $X', X'' \subseteq A$, $X' \subseteq X''$ implies that $f(X') \subseteq f(X'')$.

(2) $X \subseteq A$ is called a *fixed point of* f if $f(X) = X$.

3.2. Example. Let Σ be an alphabet and let f and g be the functions from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$ defined as follows: for every $X \subseteq \Sigma^*$, $f(X) = X^*$ and $g(X) = \Sigma^* - f(X)$.

(1) Since, for every $X', X'' \subseteq \Sigma^*$, $X' \subseteq X''$ implies that $f(X') = (X')^* \subseteq (X'')^* = f(X'')$, f is monotone. However, since $\emptyset \subseteq \Sigma$ while $g(\emptyset) = \Sigma^+ \not\subseteq \emptyset = g(\Sigma)$, g is not monotone.

(2) Since, for every $X \subseteq \Sigma^*$, $X^* = (X^*)^* = f(X^*)$, every X^* with $X \subseteq \Sigma^*$ is a fixed point of f. However, since, for every $X \subseteq \Sigma^*$ and $x \in \Sigma^*$, $x \in X$ implies $x \notin \Sigma^* - X^* = g(X)$, and since $\emptyset \neq \Sigma^* - \emptyset^*$, g has no fixed points.

The following fundamental result concerning fixed points of monotone functions is due to Tarski and Knaster (see, e.g., [48]).

3.3. Theorem. *Let A be a set and let f be a monotone function from $\mathcal{P}(A)$ into $\mathcal{P}(A)$. Let P be the set of all fixed points of f.*

(1) $P \neq \emptyset$.

(2) $\bigcap \{X \subseteq A : X \in P\} = \bigcap \{X \subseteq A : f(X) \subseteq X\}$.

(3) $\bigcup \{X \subseteq A : X \in P\} = \bigcup \{X \subseteq A : X \subseteq f(X)\}$.

The above theorem justifies calling the set $\bigcap \{X \subseteq A : X \in P\}$ the *minimal fixed point of* f and the set $\bigcup \{X \subseteq A : X \in P\}$ the *maximal fixed poin of* f.

3.4. Example. Let Σ and f be as in Example 3.2. Since f is monotone, f has a minimal fixed point, which is $\{\lambda\}$, and a maximal fixed point, which is Σ^*. This is seen as follows.

Assume that $X \subseteq \Sigma^*$ is a fixed point of f, i.e., $f(X) = X^* = X$. Consequently, $\lambda \in X$. Thus, λ is an element of every fixed point of f. Moreover, since $f(\{\lambda\}) = \{\lambda\}^* = \{\lambda\}$, $\{\lambda\}$ is also a fixed point of f; thus, $\{\lambda\}$ is the minimal fixed point of f. Furthermore, since every fixed point of f is a subset of Σ^* and since $f(\Sigma^*) = (\Sigma^*)^* = \Sigma^*$ implies that Σ^* is also a fixed point of f, Σ^* is the maximal fixed point of f.

In order to establish the link between functions on string languages and functions on trace languages, we need the following definition.

3.5. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) Let f be a function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$.

(1.1) f is called *C-existential* (*C-universal*) if, for every $X', X'' \subseteq \Sigma^*$, $[X']^\exists = [X'']^\exists$ implies that $[f(X')]^\exists = [f(X'')]^\exists$ ($[X']^\forall = [X'']^\forall$ implies that $[f(X')]^\forall = [f(X'')]^\forall$ respectively).

(1.2) f is called *C-consistent* if, for every $X \subseteq \Sigma^*$, X is C-consistent implies that $f(X)$ is C-consistent.

(2) Let f be a C-existential (C-universal) function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$. The *C-existential* (*C-universal* respectively) *trace function of f*, denoted by $[f]_C^\exists$ ($[f]_C^\forall$ respectively), is the function from $\mathcal{P}(\Theta(C))$ into $\mathcal{P}(\Theta(C))$ defined as follows: for every $X \subseteq \Sigma^*$, $[f]_C^\exists([X]^\exists) = [f(X)]^\exists$ ($[f]_C^\forall([X]^\forall) = [f(X)]^\forall$ respectively).

Again, in the terminology as above, we write $[f]^\exists$ and $[f]^\forall$ rather than $[f]_C^\exists$ and $[f]_C^\forall$, respectively, whenever C is understood from the context.

3.6. Remark. (1) Note that, for a given reliance alphabet C, the assumption that f is C-existential (C-universal) used for the definition of the C-existential (C-universal respectively) trace function of f is needed to guarantee that $[f]^\exists$ ($[f]^\forall$ respectively) is well-defined. Furthermore, note that Theorem 2.21 implies that for every trace language T there exists a string language K such that $T = [K]^\exists = [K]^\forall$; thus the C-existential and C-universal trace functions are *really* functions on arbitrary trace languages.

(2) Note that in general it may happen that, for a reliance alphabet C and a function f as described above, both $[f]^\exists$ and $[f]^\forall$ are defined, while $[f]^\exists \neq [f]^\forall$ (see Example 3.7(2)).

3.7. Example. Let $C = (\Sigma, I, D)$ be the reliance alphabet such that $\Sigma = \{a, b\}$ and $I = \{\{a, b\}\}$.

(1) Let f be the function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$ defined as follows: for every $X \subseteq \Sigma^*$, $f(X) = \{\lambda, a\}X$.

(1.1) Since, by Theorem 2.14(3), for every $X', X'' \subseteq \Sigma^*$, $[X']^\exists = [X'']^\exists$ implies that

$$[f(X')]^\exists = [\{\lambda, a\}X']^\exists = [\{\lambda, a\}]^\exists \circ [X']^\exists = [\{\lambda, a\}]^\exists \circ [X'']^\exists$$
$$= [\{\lambda, a\}X'']^\exists = [f(X'')]^\exists,$$

f is C-existential. Furthermore, since $[\{b, ba\}]^{\forall} = \{[b]\} = [\{b\}]^{\forall}$, while

$$[f(\{b, ba\})]^{\forall} = [\{\lambda, a\}\{b, ba\}]^{\forall} = [\{b, ba, ab, aba\}]^{\forall} = \{[b], [ab]\}$$
$$\neq \{[b]\} = [\{b, ab\}]^{\forall} = [\{\lambda, a\}\{b\}]^{\forall} = [f(\{b\})]^{\forall},$$

f is not C-universal. Finally, since $\{b\}$ is C-consistent, while $[f(\{b\})]^{\forall} = \{\lambda, a\}\{b\} = \{b, ab\}$ is not C-consistent, f is not C-consistent.

(1.2) Since f is C-existential, $[f]^{\exists}$ is the function defined as follows: for every $T \subseteq \Theta(C)$, $[f]^{\exists}(T) = \{1, [a]\} \circ T$.

(2) Let f be the function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$ defined as follows: for every $X \subseteq \Sigma^*$, $f(X) = \{x \in \Sigma^* : [x] \in [X]^{\exists}$ and x is the minimal normal representative of $[x]\}$.

(2.1) f is C-existential which can be seen as follows. Assume that $X', X'' \subseteq \Sigma^*$ are such that $[X']^{\exists} = [X'']^{\exists}$. Furthermore, assume that $t \in [f(X')]^{\exists}$. Hence, there exists an $x \in f(X')$ such that $[x] = t$. Consequently, by the definition of f, $t = [x] \in [X']^{\exists} = [X'']^{\exists}$. Thus, $x \in f(X'')$ and $t = [x] \in [f(X'')]^{\exists}$, which implies that $[f(X')]^{\exists} \subseteq [f(X'')]^{\exists}$. The reverse inclusion is proven analogously.

(2.2) f is C-universal, which can be seen as follows. Assume that $X', X'' \subseteq \Sigma^*$ are such that $[X']^{\forall} = [X'']^{\forall}$. Furthermore, assume that $t \in [f(X')]^{\forall}$. Hence, there exists an $X \subseteq f(X')$ such that $X = t$. Consequently, by the definition of f, $\#t = 1$. Thus, $t \in [X']^{\forall} = [X'']^{\forall}$, and hence, by using arguments similar to the ones above, we get $t \in [f(X'')]^{\forall}$. Consequently, $[f(X')]^{\forall} \subseteq [f(X'')]^{\forall}$. The reverse inclusion is proven analogously.

(2.3) Reasoning as under (2.1) and (2.2) above we can easily show that $[f]^{\exists}$ and $[f]^{\forall}$ are defined as follows: for every $T \subseteq \Theta(C)$, $[f]^{\exists}(T) = T$ and $[f]^{\forall}(T) = \{t \in T : \#t = 1\}$.

The following result 'transfers' the basic properties of a function f, from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$, into the basic properties of the functions $[f]^{\exists}$ and $[f]^{\forall}$ (for a given reliance alphabet $C = (\Sigma, I, D)$).

3.8. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let f be a function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$.*

(1) *If f is C-existential and monotone, then $[f]^{\exists}$ is monotone.*

(2) *If f is C-universal and monotone, then $[f]^{\forall}$ is monotone.*

(3) *If f is C-existential and C-universal, then f is C-consistent if and only if $[f]^{\exists} = [f]^{\forall}$.*

Proof. (2): Assume that f is C-universal and monotone and let $T', T'' \subseteq \Theta(C)$ be such that $T' \subseteq T''$. If we set $X' = \{x \in \Sigma^* : [x] \in T'\}$ and $X'' = \{x \in \Sigma^* : [x] \in T''\}$, then $T' = [X']^{\forall}$, $T'' = [X'']^{\forall}$ and $X' \subseteq X''$. Since f is monotone, $f(X') \subseteq f(X'')$, and hence, by Theorem 2.15(2), $[f(X')]^{\forall} \subseteq [f(X'')]^{\forall}$. Thus,

$$[f]^{\forall}(T') = [f]^{\forall}([X']^{\forall}) = [f(X')]^{\forall}$$
$$\subseteq [f(X'')]^{\forall} = [f]^{\forall}([X'']^{\forall}) = [f]^{\forall}(T'')$$

and so $[f]^{\forall}$ is monotone.

(3): Assume that f is C-existential and C-universal.

(i) Assume that f is C-consistent. Let $T \subseteq \Theta(C)$. If we set $X = \{x \in \Sigma^* : [x] \in T\}$, then, clearly, X is C-consistent and $T = [X]^\exists = [X]^\forall$. Since f is C-existential and C-universal,

$$[f]^\exists(T) = [f]^\exists([X]^\exists) = [f(X)]^\exists \quad \text{and}$$

$$[f]^\forall(T) = [f]^\forall([X]^\forall) = [f(X)]^\forall.$$

Moreover, since X is C-consistent and f is C-consistent, $f(X)$ is C-consistent and so $[f(X)]^\exists = [f(X)]^\forall$. Thus $[f]^\exists(T) = [f]^\forall(T)$ and, consequently, $[f]^\exists = [f]^\forall$.

(ii) Assume that $[f]^\exists = [f]^\forall$. Let $X \subseteq \Sigma^*$ be C-consistent. Thus, $[X]^\exists = [X]^\forall$, and, consequently,

$$[f(X)]^\exists = [f]^\exists([X]^\exists) = [f]^\forall([X]^\forall) = [f(X)]^\forall.$$

Hence, $f(X)$ is C-consistent.

Now (3) follows from (i) and (ii) above. □

3.9. Example. Let C be as in Example 3.7.

(1) Let f be as in Example 3.7(1). Clearly, f and $[f]^\exists$ are monotone.

(2) Let f be as in Example 3.7(2). Clearly, f is not C-consistent.

The next result provides, for a given reliance alphabet $C = (\Sigma, I, D)$, the basic relationships between the minimal (maximal) fixed points of a function f, from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$, and the minimal (maximal respectively) fixed points of the functions $[f]^\exists$ and $[f]^\forall$.

3.10. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet, let f be a monotone function from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$ and let $X_{\min}$ ($X_{\max}$) be the minimal (maximal respectively) fixed point of f.*

(1) *If f is C-existential, then $[X_{\min}]^\exists$ is the minimal fixed point of $[f]^\exists$.*

(2) *If f is C-universal, then $[X_{\max}]^\forall$ is the maximal fixed point of $[f]^\forall$.*

(3) *If f is C-consistent, C-existential and C-universal, then $[X_{\min}]^\exists$ is the minimal fixed point of $[f]^\exists$ $(=[f]^\forall)$ and $[X_{\max}]^\forall$ is the maximal fixed point of $[f]^\forall$ $(=[f]^\exists)$.*

Proof. (2): Assume that f is C-universal. Then, since $f(X_{\max}) = X_{\max}$,

$$[f]^\forall([X_{\max}]^\forall) = [f(X_{\max})]^\forall = [X_{\max}]^\forall;$$

hence $[X_{\max}]^\forall$ is a fixed point of $[f]^\forall$.

To prove that $[X_{\max}]^\forall$ is maximal, assume that $T \subseteq \Theta(C)$ is an arbitrary fixed point of $[f]^\forall$. If we set $X = \{x \in \Sigma^* : [x] \in T\}$, then, clearly, $T = [X]^\forall$. Thus, for every $x \in \Sigma^*$, $x \in X$ if and only if $[x] \in T$. But $[x] \in T$ if and only if $[x] \in [f]^\forall(T)$

if and only if $[x] \in [f]^{\forall}([X]^{\forall})$ if and only if $[x] \in [f(X)]^{\forall}$. Hence, $x \in f(X)$ and so $X \subseteq f(X)$. Thus, by Theorem 3.3, $X \subseteq X_{\max}$ and, consequently, by Theorem 2.15(2), $T = [X]^{\forall} \subseteq [X_{\max}]^{\forall}$. Since T was assumed to be an arbitrary fixed point of $[f]^{\forall}$, $[X_{\max}]^{\forall}$ is the maximal fixed point of $[f]^{\forall}$.

(3) directly follows from (1) and (2) above and from Theorem 3.8(3). □

3.11 Example. Let $C = (\Sigma, I, D)$ be the reliance alphabet such that $\Sigma = \{a, b, c\}$ and $I = \{\{a, b\}, \{b, c\}\}$. Let g be the function from $\mathcal{P}(\Theta(C))$ into $\mathcal{P}(\Theta(C))$ defined as follows: for every $T \subseteq \Theta(C)$, $g(T) = (\{[ab]\} \circ T \circ \{[c]\}) \cup \{[abc]\}$.

In order to compute the minimal fixed point of g, we consider the function f from $\mathcal{P}(\Sigma^*)$ into $\mathcal{P}(\Sigma^*)$ defined by: for every $X \subseteq \Sigma^*$, $f(X) = (\{ab\}X\{c\}) \cup \{abc\}$. Clearly, f is monotone and C-existential and thus the C-existential trace function of f is defined as follows: for every $X \subseteq \Sigma^*$,

$$[f]^{\exists}([X]^{\exists}) = [(\{ab\}X\{c\}) \cup \{abc\}]^{\exists}.$$

Hence, by Theorem 2.14(3) and (5), for every $X \subseteq \Sigma^*$,

$$[f]^{\exists}([X]^{\exists}) = (\{[ab]\} \circ [X]^{\exists} \circ \{[c]\}) \cup \{[abc]\}$$

and thus, by Theorem 2.21, $g = [f]^{\exists}$. Consequently, since $\{(ab)^n c^n : n \geq 1\}$ is the minimal fixed point of f, Theorem 3.10(1) implies that $[\{(ab)^n c^n : n \geq 1\}]^{\exists}$ is the minimal fixed point of g.

3.2. *Relations between various classes of trace languages*

In this subsection we will investigate the relationship between various classes of trace languages.

First, the relationship between classes of trace languages defined by

(i) *different* types (classes) of string languages (from the Chomsky hierarchy), but

(ii) the *same* way of claiming (existential, universal or consistent) is considered.

Then the relationship between classes of trace languages defined by

(i) the *same* type (class) of string languages (from the Chomsky hierarchy), but

(ii) *different* ways of claiming (existential, universal or consistent) is investigated.

The following lemma is useful in the sequel of this section.

3.12. Lemma. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) $T_C^{\exists}(\mathrm{REG}) = T_C^{\forall}(\mathrm{REG})$ *if and only if I is transitive.*

(2) $T_C(\mathrm{REG})$ *is closed under union and trace composition.*

We now turn to the first result on the relationship between classes of trace languages, defined by different types of string languages but using the same way of claiming.

3.13. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) $T_C^{\exists}(\mathrm{REG}) \subseteq T_C^{\exists}(\mathrm{CF}) \subset T_C^{\exists}(\mathrm{CS})$, *where* $T_C^{\exists}(\mathrm{REG}) = T_C^{\exists}(\mathrm{CF})$ *if and only if* I *is complete.*

(2) $T_C^{\forall}(\mathrm{REG}) \subseteq T_C^{\forall}(\mathrm{CF}) \subset T_C^{\forall}(\mathrm{CS})$, *where* $T_C^{\forall}(\mathrm{REG}) = T_C^{\forall}(\mathrm{CF})$ *if and only if* $\#\Sigma = 1$.

(3) $T_C(\mathrm{REG}) \subseteq T_C(\mathrm{CF}) \subset T_C(\mathrm{CS})$, *where* $T_C(\mathrm{REG}) = T_C(\mathrm{CF})$ *if and only if* $\#\Sigma = 1$.

Proof. The above relationships $T_C^{\exists}(\mathrm{REG}) \subseteq T_C^{\exists}(\mathrm{CF}) \subset T_C^{\exists}(\mathrm{CS})$, $T_C^{\forall}(\mathrm{REG}) \subseteq T_C^{\forall}(\mathrm{CF}) \subset T_C^{\forall}(\mathrm{CS})$, and $T_C(\mathrm{REG}) \subseteq T_C(\mathrm{CF}) \subset T_C(\mathrm{CS})$ follow from the inclusions $\mathrm{REG} \subseteq \mathrm{CF} \subseteq \mathrm{CS}$, from the fact that over a one-letter alphabet $\mathrm{CF} \subset \mathrm{CS}$, and from Theorem 2.5(2).

The rest of the theorem is proved as follows.

(1): (i) Assume that I is complete. Then, from Theorem 2.5(3) and from the fact that for every context-free string language there exists a regular string language with the same Parikh image (see, e.g., [44]), it follows that $T_C^{\exists}(\mathrm{REG}) = T_C^{\exists}(\mathrm{CF})$.

(ii) Assume that I is not complete. Then Σ contains two different symbols a and b such that $\{a, b\} \notin I$. Since

$$[\{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}]^{\exists} \in T_C^{\exists}(\mathrm{CF}) - T_C^{\exists}(\mathrm{REG}),$$

$T_C^{\exists}(\mathrm{REG}) \neq T_C^{\exists}(\mathrm{CF})$.

(2): (i) Assume that $\#\Sigma = 1$. Then, because over a one-letter alphabet $\mathrm{REG} = \mathrm{CF}$, Theorem 2.5(2) implies that $T_C^{\forall}(\mathrm{REG}) = T_C^{\forall}(\mathrm{CF})$.

(ii) Assume that $\#\Sigma > 1$. If I is not complete, then, by arguments analogous to those under (1)(ii) above, $T_C^{\forall}(\mathrm{REG}) \neq T_C^{\forall}(\mathrm{CF})$. If I is complete, then $[L]^{\forall} \in T_C^{\forall}(\mathrm{CF}) - T_C^{\forall}(\mathrm{REG})$, where $L = L_1 \cup L_2 \cup L_3$, with

$$\begin{aligned} L_1 = {} & \{a, b\}^*\{b\}\{a^i b^j : i, j \geq 1, j \neq i+1\}\{a\}\{a, b\}^* \\ & \cup \{a^i b^j : i, j \geq 1, j \neq i+1\}(\{\lambda\} \cup \{a\}\{a, b\}^*) \\ & \cup (\{\lambda\} \cup \{a, b\}^*\{b\})\{a^i b^j : i, j \geq 1, j \neq i+1\}; \end{aligned}$$

$$L_2 = \{a, b\}^*\{a\}\{b^i a^j : i, j \geq 1, j \neq i+1\}\{b\}\{a, b\}^*, \quad \text{and}$$

$$L_3 = \{\lambda\} \cup \{aa, b\}\{a, b\}^* \cup \{a, b\}^*\{a\},$$

where a and b are two different symbols in Σ. This can be seen as follows.

(a) First we observe that L is a finite union of context-free string languages, and thus, $[L]^{\forall} \in T_C^{\forall}(\mathrm{CF})$.

(b) Next we note that every string of $\{a, b\}^* - L$ is of the form $ab^2a^3b^4 \ldots a^{2n-1}b^{2n}$, where $n \geq 1$, and thus, if we order the strings of $\{a, b\}^* - L$ according to their length, then the length of the kth string, $k \geq 1$, of $\{a, b\}^* - L$ is $k(2k+1)$. Consequently, $\psi(\{a, b\}^* - L)$ and $\psi(\Sigma^* - L)$ are not semilinear sets.

(c) We now claim that $[L]^{\forall} \notin T_C^{\forall}(\mathrm{REG})$, which can be seen as follows. Assume to the contrary that $[L]^{\forall} \in T_C^{\forall}(\mathrm{REG})$. Then, since I is complete, it follows from Lemma 3.12(1) that $[L]^{\forall} \in T_C^{\exists}(\mathrm{REG})$. Hence, there exists a regular $K \subseteq \Sigma^*$ such

that $[L]^{\forall} = [K]^{\exists}$. From Theorem 2.16(3) it now follows that $\Theta(C) - [\Sigma^* - L]^{\exists} = [K]^{\exists}$. Hence, from Theorem 2.5(3) it follows that $\psi(\Sigma^* - L)$ is the complement of $\psi(K)$. Furthermore, because the Parikh image of a regular string language is a semilinear set and the complement of a semilinear set is again a semilinear set (see, e.g., [19]), $\psi(K)$ and $\psi(\Sigma^* - L)$ are semilinear sets. This however contradicts the conclusion from (b) above and, consequently, $[L]^{\forall} \notin T_C^{\forall}(\mathrm{REG})$.

From (a) and (c) above it now follows that $T_C^{\forall}(\mathrm{REG}) \neq T_C^{\forall}(\mathrm{CF})$.

(3): (i) Assume that $\#\Sigma = 1$. Then, because over a one-letter alphabet REG = CF, Theorem 2.5(2) implies that $T_C(\mathrm{REG}) = T_C(\mathrm{CF})$.

(ii) Assume that $\#\Sigma > 1$. Then, using the trace language given in (1)(ii) above, one shows that $T_C(\mathrm{REG}) \neq T_C(\mathrm{CF})$. □

We now proceed to study classes of trace languages defined by using regular string languages.

First of all it can be shown that the class of consistently regular trace languages is the class of those languages for which the set of minimal representatives (for a given ordering on the alphabet involved) is regular.

3.14. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet, where Σ is ordered. Then, for every $T \subseteq \Theta(C)$, $T \in T_C(\mathrm{REG})$ if and only if $\{\min(t) : t \in T\} \in \mathrm{REG}$.*

Secondly, it can be shown that one can use finite automata to define the class of consistently regular trace languages.

3.15. Definition. (1) Let M be a monoid. A *finite M-automaton* is a quintuple $A = (Q, M, \delta, q_{\mathrm{in}}, F)$, where

(i) Q is a finite set of *states*;

(ii) $\delta : Q \times M \to Q$ is a transition mapping such that, for every $q \in Q$, $\delta(q, e) = q$, where e is the identity of M, and, for every $q \in Q$ and for every $m_1, m_2 \in M$, $\delta(q, m_1 \cdot m_2) = \delta(\delta(q, m_1), m_2)$, where $\cdot$ is the operation of M;

(iii) q_{in} is the *initial state of A*; and

(iv) $F \subseteq Q$ is the set of *final states of A*.

(2) Let M be a monoid and let $A = (Q, M, \delta, q_{\mathrm{in}}, F)$ be a finite M-automaton. The *language of A*, denoted by $L(A)$, is the set $\{m \in M : \delta(q_{\mathrm{in}}, m) \in F\}$.

(3) Let C be a reliance alphabet and let $T \subseteq \Theta(C)$. T is *recognizable* if there exists a finite $(\Theta(C), \circ, 1)$-automaton A such that $T = L(A)$.

Given a reliance alphabet C, $T_C(FA)$ denotes the class of all recognizable trace languages over C.

3.16. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet. Then $T_C(FA) = T_C(\mathrm{REG})$.*

Thirdly, it can be shown that the class of consistently regular trace languages can be built up recursively, using three elementary trace language operations (union,

trace composition, and concurrent trace iteration) and starting with elementary trace languages. The operation of concurrent trace iteration is defined as follows.

3.17. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet.

(1) Let $t \in \Theta(C)$. The *concurrent decomposition of t*, denoted by $\mathrm{cd}(t)$, is $\{1\}$ if $t = 1$ and for $t \neq 1$ it is the set $\{t_1, \ldots, t_n\} \subseteq \Theta(C)$, where $n \geq 1$ and:

(i) $t = t_1 \circ \cdots \circ t_n$;

(ii) for every $1 \leq i \leq n$, $\langle\langle t_i \rangle\rangle$ is connected; and

(iii) $\langle\langle t \rangle\rangle$ consists of n maximal connected components.

(2) Let $T \subseteq \Theta(C)$. The *concurrent decomposition of T*, denoted by $\mathrm{cd}(T)$, is the trace language $\bigcup \{\mathrm{cd}(t) : t \in T\}$.

(3) Let $T \subseteq \Theta(C)$. The *concurrent trace iteration of T*, denoted by $T^{\dagger}$, is the trace language $(\mathrm{cd}(T))^*$.

Thus, in the terminology as above, the concurrent decomposition of the trace t is obtained as follows: one considers $\langle\langle t_i \rangle\rangle$ and for each maximal connected component d of $\langle\langle t_i \rangle\rangle$ one takes $[\![d]\!]$; the set of all such $[\![d]\!]$ forms the concurrent decomposition of t. Consequently, the difference between the trace iteration of a trace language T and the concurrent trace iteration of T is that, in the former, one considers arbitrary compositions of whole traces in T, while, in the latter, one considers only compositions of those parts of traces t in T which correspond to maximal connected components of the dep-graphs of t.

3.18. Example. Let $C = (\Sigma, I, D)$ be the reliance alphabet such that

$$\Sigma = \{a, b, c, d\} \quad \text{and} \quad I = \{\{a, b\}, \{b, c\}, \{b, d\}\}.$$

Let $t = [adbcb]$ and $T = \{[adbcb]\}$. From Fig. 9 which represents a dep-graph of t, it directly follows that $\mathrm{cd}(t) = \{[adc], [bb]\}$ and thus that $\mathrm{cd}(T) = \{[adc], [bb]\}$. Hence,

$$T^{\dagger} = (\mathrm{cd}(T))^* = \{[adc], [bb]\}^* = [\{adc, bb\}^*]^{\exists},$$

while $T^* = \{[adbcb]\}^* = [\{adbcb\}^*]^{\exists}$.

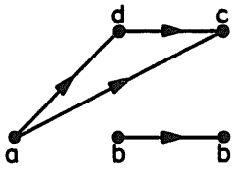


Fig. 9

We now turn to a characterization of consistently regular trace languages.

3.19. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet. Then $T_C(\mathrm{REG})$ is the least class of trace languages containing $\emptyset$, $\{1\}$, and $\{[a]\}$ for all $a \in \Sigma$, and closed under union, trace composition, and concurrent trace iteration.*

The above characterization is quite analogous to the famous Kleene characterization of regular string languages (see, e.g., [44]). The basic difference is that we now use the concurrent trace iteration rather than the trace iteration. That this is necessary is illustrated by the following example.

3.20. Example. Let $C=(\Sigma, I, D)$ be the reliance alphabet such that $\Sigma=\{a, b\}$ and $I=\{\{a, b\}\}$. Let $T=\{[ab]\}$; then, clearly, T is a consistently regular trace language over C. Furthermore, it is not difficult to see that $T^*=[K]^{\exists}$, where $K=\{x\in\Sigma^*: \#_a(x)=\#_b(x)\}$ is such that $K=\uparrow K$. Consequently, since K is not a regular string language over Σ, it follows from Theorem 2.24(1) that T^* is not a consistently regular trace language over C.

We now show an interesting application of Theorem 3.19.

3.21. Lemma. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $X\subseteq\Sigma^*$ be regular. If, for all $x, y, z\in\Sigma^*$, $\{x\}\{y\}^*\{z\}\subseteq X$ implies that $\langle y\rangle$ is connected, then $[X]^{\exists}\in T_C(\mathrm{REG})$.*

Proof. Let us recall (see, e.g., [24]) that the class of regular string languages over Σ is the least class of languages Ξ over Σ satisfying the following two properties:

(i) $\emptyset\in\Xi$ and, for all $a\in\Sigma$, $\{a\}\in\Xi$; and

(ii) if $K, L\in\Xi$, then $K\cup L\in\Xi$, $KL\in\Xi$, and $K^*\in\Xi$.

We need the following two notions.

A regular string language Y over Σ has degree n, where $n\geq 0$, if and only if n is the minimal number m such that Y can be obtained from the languages stated under (i) above, by applying the operations stated under (ii) above m times.

A string language Y over Σ satisfies property P if and only if, for all $x, y, z\in\Sigma^*$, $\{x\}\{y\}^*\{z\}\subseteq Y$ implies that $\langle y\rangle$ is connected. Thus, in order to prove the theorem, we have to prove that if X satisfies property P, then $[X]^{\exists}\in T_C(\mathrm{REG})$.

The proof now goes by induction on n, the degree of X.

Induction basis: Assume that $n=0$. Hence, either $X=\emptyset$ or $X=\{a\}$ for some $a\in\Sigma$, and thus, clearly, X satisfies P and $[X]^{\exists}\in T_C(\mathrm{REG})$.

Induction hypothesis: Let $k\geq 0$ be such that, for all $0\leq m\leq k$ and for all regular string languages Y over Σ satisfying P, Y has a degree smaller than or equal to m implies that $[Y]^{\exists}\in T_C(\mathrm{REG})$.

Induction step: Assume that $n=k+1$ and that X satisfies property P. Now we can distinguish the following three cases.

Case 1: $X=K\cup L$, where K and L are string regular languages over Σ of degree smaller than or equal to $n-1$. Then, since X satisfies property P and, for all $x, y, z\in\Sigma^*$, $\{x\}\{y\}^*\{z\}\subseteq K$ implies that

$$\{x\}\{y\}^*\{z\}\subseteq K\cup L=X$$

and $\{x\}\{y\}^*\{z\}\subseteq L$ implies that

$$\{x\}\{y\}^*\{z\}\subseteq K\cup L=X,$$

K and L satisfy property P. Hence, by the induction hypothesis, $[K]^{\exists} \in T_C(\mathrm{REG})$ and $[L]^{\exists} \in T_C(\mathrm{REG})$, and thus, by Lemma 3.12(2), $[K]^{\exists} \cup [L]^{\exists} \in T_C(\mathrm{REG})$. Consequently, since it follows from Theorem 2.14(5) that $[K]^{\exists} \cup [L]^{\exists} = [K \cup L]^{\exists}$, $[X]^{\exists} = [K \cup L]^{\exists} \in T_C(\mathrm{REG})$.

Case 2: $X = KL$, where K and L are regular string languages over Σ of degree smaller than or equal to $n-1$. If either $K = \emptyset$ or $L = \emptyset$, then $X = \emptyset$ and thus the degree of X would be 0; hence, neither $K = \emptyset$ nor $L = \emptyset$. Then, since X satisfies property P and, for all $x, y, z \in \Sigma^*$, $\{x\}\{y\}^*\{z\} \subseteq K$ implies that

$$\{x\}\{y\}^*\{zv\} \subseteq X \quad \text{for some } v \in L$$

and $\{x\}\{y\}^*\{z\} \subseteq L$ implies that

$$\{wx\}\{y\}^*\{z\} \subseteq X \quad \text{for some } w \in K,$$

K and L satisfy property P. Hence, by the induction hypothesis, $[K]^{\exists} \in T_C(\mathrm{REG})$ and $[L]^{\exists} \in T_C(\mathrm{REG})$, and thus, by Lemma 3.12(2), $[K]^{\exists} \circ [L]^{\exists} \in T_C(\mathrm{REG})$. Consequently, since it follows from Theorem 2.14(3) that $[K]^{\exists} \circ [L]^{\exists} = [KL]^{\exists}$, $[X]^{\exists} = [KL]^{\exists} \in T_C(\mathrm{REG})$.

Case 3: $X = K^*$, where K is a regular string language over Σ of degree smaller than or equal to $n-1$. Then, since X satisfies property P and, for all $x, y, z \in \Sigma^*$, $\{x\}\{y\}^*\{z\} \subseteq K$ implies that

$$\{x\}\{y\}^*\{z\} \subseteq K^* = X,$$

K satisfies property P. Hence, by the induction hypothesis, $[K]^{\exists} \in T_C(\mathrm{REG})$. Furthermore, for every $x \in K$, $\{x\}^* \subseteq K^* \subseteq X$ and thus, since X satisfies property P, for every $x \in K$, $\langle x \rangle$ is connected and $\mathrm{cd}([x]) = \{[x]\}$. Hence, $\mathrm{cd}([K]^{\exists}) = [K]^{\exists}$ and thus, $([K]^{\exists})^{\dagger} = ([K]^{\exists})^*$. Since it follows from Theorem 2.14(4) that $([K]^{\exists})^* = [K^*]^{\exists}$,

$$[X]^{\exists} = [K^*]^{\exists} = ([K]^{\exists})^{\dagger}$$

and, consequently, since $[K]^{\exists} \in T_C(\mathrm{REG})$ and since, by Theorem 3.19, $T_C(\mathrm{REG})$ is closed under concurrent trace iteration, $[X]^{\exists} \in T_C(\mathrm{REG})$. □

Returning to our study of classes of trace languages defined by using regular string languages, we can prove, assuming certain restrictions on the reliance alphabet C, that $T_C(\mathrm{REG}) \subset T_C^{\exists}(\mathrm{REG})$. Actually, one gets an infinite hierarchy of classes of trace languages between $T_C(\mathrm{REG})$ and $T_C^{\exists}(\mathrm{REG})$. The classes of languages from this hierarchy are obtained by considering in more detail the way that a trace is 'claimed' in the existential way by a string language.

3.22. Definition. Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $n \geq 1$.

(1) Let $K \subseteq \Sigma^*$ be such that $\#([x] \cap K) \leq n$ for each $x \in K$. Then $[K]^{\exists}$ is called the (n, K)-*existential trace language* (*over* C).

(2) Let $T \subseteq \Theta(C)$. We say that T is *n-existentially regular* if T is the (n, K)-existential trace language for some regular $K \subseteq \Sigma^*$.

For a reliance alphabet $C = (\Sigma, I, D)$, $K \subseteq \Sigma^*$ and $n \geq 0$, we use the notation $T_C^{\exists,n}(\mathrm{REG})$ to denote the class of all n-existentially regular trace languages over C.

3.23. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet. Then*

$$T_C(\mathrm{REG}) \subseteq T_C^{\exists,1}(\mathrm{REG}) \subseteq T_C^{\exists,2}(\mathrm{REG}) \subseteq \cdots$$

$$\subseteq \bigcup_{n=1}^{\infty} T_C^{\exists,n}(\mathrm{REG}) \subseteq T_C^{\exists}(\mathrm{REG});$$

moreover,

$$T_C(\mathrm{REG}) \subset T_C^{\exists,1}(\mathrm{REG}) \subset T_C^{\exists,2}(\mathrm{REG}) \subset \cdots$$

$$\subset \bigcup_{n=1}^{\infty} T_C^{\exists,n}(\mathrm{REG}) \subset T_C^{\exists}(\mathrm{REG})$$

if I is not transitive.

By Theorem 3.19 the smallest class from the above hierarchy ($T_C(\mathrm{REG})$) is characterized by a Kleene-like theorem, using the operation of concurrent trace iteration. Replacing this operation of concurrent trace iteration by the operation of trace iteration changes the class of trace languages characterized (see the comment after Theorem 3.19). Namely, it turns out that such a replacement yields the characterization of the largest class from the above hierarchy ($T_C^{\exists}(\mathrm{REG})$).

3.24. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet. Then $T_C^{\exists}(\mathrm{REG})$ is the least class of trace languages containing $\emptyset$, $\{1\}$, and $\{[a]\}$ for all $a \in \Sigma$, and closed under union, trace composition, and trace iteration.*

Proof. This directly follows from the definition of REG, Theorem 2.14(3), (4) and (5). □

The last result concerning the relationship between different types of regular trace languages is the following one.

3.25. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) $T_C^{\exists}(\mathrm{REG}) \subseteq T_C^{\forall}(\mathrm{REG})$ *if and only if* $T_C^{\forall}(\mathrm{REG}) \subseteq T_C^{\exists}(\mathrm{REG})$ *if and only if* $T_C^{\exists}(\mathrm{REG}) = T_C^{\forall}(\mathrm{REG})$ *if and only if I is transitive.*

(2) $T_C(\mathrm{REG}) \subseteq T_C^{\exists}(\mathrm{REG}) \cap T_C^{\forall}(\mathrm{REG})$; *moreover,* $T_C(\mathrm{REG}) = T_C^{\exists}(\mathrm{REG}) \cap T_C^{\forall}(\mathrm{REG})$ *if and only if* $I = \emptyset$.

Proof. (2): (i) Obviously the inclusion holds.

(ii) Assume that $I=\emptyset$. Then, from Theorem 2.5(1) it follows that $T_C(\mathrm{REG})=T_C^{\exists}(\mathrm{REG})=T_C^{\forall}(\mathrm{REG})$. Consequently, $T_C(\mathrm{REG})=T_C^{\exists}(\mathrm{REG})\cap T_C^{\forall}(\mathrm{REG})$.

(iii) Assume that $I\neq\emptyset$. Hence, Σ contains two different symbols a and b such that $\{a,b\}\in I$. Let $K=\{ab\}^*$ and let

$$L=\{a,b\}^*-(\{a\}\{a\}^*\{ab\}^*\cup\{b\}\{b\}^*\{ab\}^*).$$

Then K and L are regular string languages and $[K]^{\exists}=[L]^{\forall}\notin T_C(\mathrm{REG})$. Consequently, $T_C(\mathrm{REG})\neq T_C^{\exists}(\mathrm{REG})\cap T_C^{\forall}(\mathrm{REG})$.

Now (2) follows from (i), (ii) and (iii) above. □

We will now consider classes of trace languages defined by using context-free string languages.

3.26. Theorem. *Let $C=(\Sigma,I,D)$ be a reliance alphabet.*

(1) $T_C^{\forall}(\mathrm{CF})\subseteq T_C^{\exists}(CF)$ *if and only if* $T_C^{\forall}(\mathrm{CF})=T_C^{\exists}(\mathrm{CF})$ *if and only if* $I=\emptyset$.

(2) *If $I=\emptyset$ or I is complete, then* $T_C^{\exists}(\mathrm{CF})\subseteq T_C^{\forall}(\mathrm{CF})$.

(3) $T_C(\mathrm{CF})\subseteq T_C^{\exists}(CF)\cap T_C^{\forall}(\mathrm{CF})$; *moreover,* $T_C(\mathrm{CF})=T_C^{\exists}(\mathrm{CF})\cap T_C^{\forall}(\mathrm{CF})$ *if and only if either* $\#\Sigma\leq 2$ or $I=\emptyset$.

Finally, we consider classes of trace languages defined by using context-sensitive string languages. In this case the following result turns out to be useful.

3.27. Theorem. *Let $C=(\Sigma,I,D)$ be a reliance alphabet. Let $K\subseteq\Sigma^*$ be context-sensitive.*

(1) $\downarrow K$ *is context-sensitive.*

(2) $\uparrow K$ *is context-sensitive.*

Proof. We will sketch the main idea of the proof, leaving the obvious but tedious technical details to the reader.

(1) Since K is context-sensitive, K is accepted by a linear bounded automaton A. We construct a linear bounded automaton A', based on A, as follows. Given a string $x\in\Sigma^*$, A' starts by simulating A on x. If x is not accepted by A, then x is also not accepted by A'. If x is accepted by A, then A' switches to the following mode. It generates, one by one, all strings $y\in\Sigma^*$ such that $|y|=|x|$, and, for each such y, A' checks whether or not $y\in[x]$ implies that y is accepted by A (note that, by Theorem 2.8, to check whether or not $y\in[x]$, it suffices to check whether or not the minimal normal representatives of $[y]$ and $[x]$ are equal). Then x is accepted by A' if and only if for all y's as above all these tests turn out to be positive.

Clearly, such a linear bounded automaton A' exists and A' accepts $\downarrow K$; consequently, $\downarrow K$ is context-sensitive.

(2) Since K is context-sensitive, K is accepted by a linear bounded automaton A. Let A' be a linear bounded automaton such that A' accepts a string $x\in\Sigma^*$ if and only if A accepts a string $y\in\Sigma^*$ for which $[y]=[x]$ (note that, by Theorem 2.8, to check whether or not $[y]=[x]$, it suffices to check whether or not the minimal normal representative of $[y]$ and $[x]$ are equal).

Clearly, such an A' exists and A' accepts $\uparrow K$; consequently, $\uparrow K$ is context-sensitive. □

From the above result, the following theorem easily follows.

3.28. Theorem. *Let C be a reliance alphabet. Then $T_C^{\exists}(\mathrm{CS}) = T_C^{\forall}(\mathrm{CS}) = T_C(\mathrm{CS})$.*

3.3. Closure properties of, decision problems for, and the complexity of various classes of trace languages

In this subsection we will first briefly consider closure properties of the different types of regular, context-free and context-sensitive trace languages and then we discuss (the complexity of) some decision problems concerning these classes of trace languages.

We start with the closure properties of the different types of regular trace languages.

3.29. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) *$T_C^{\exists}(\mathrm{REG})$ is closed under union, trace composition and trace iteration.*

(2) *$T_C^{\exists}(\mathrm{REG})$ is closed under intersection if and only if $T_C^{\exists}(\mathrm{REG})$ is closed under complement if and only if I is transitive.*

3.30. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) *$T_C^{\forall}(\mathrm{REG})$ is closed under intersection.*

(2) *$T_C^{\forall}(\mathrm{REG})$ is closed under union if and only if $T_C^{\forall}(\mathrm{REG})$ is closed under complement if and only if $T_C^{\forall}(\mathrm{REG})$ is closed under trace composition if and only if $T_C^{\forall}(\mathrm{REG})$ is closed under trace iteration if and only if I is transitive.*

3.31. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) *$T_C(\mathrm{REG})$ is closed under union, intersection, complement and trace composition.*

(2) *$T_C(\mathrm{REG})$ is closed under trace iteration if and only if $I = \emptyset$.*

Next we consider the closure properties of two different types of context-free trace languages.

3.32. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) *$T_C^{\exists}(\mathrm{CF})$ is closed under union, trace composition and trace iteration.*

(2) *$T_C^{\exists}(\mathrm{CF})$ is closed under intersection if and only if $T_C^{\exists}(\mathrm{CF})$ is closed under complement if and only if I is complete.*

3.33. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet.*

(1) *$T_C(\mathrm{CF})$ is closed under union.*

(2) *$T_C(\mathrm{CF})$ is closed under intersection if and only if $T_C(\mathrm{CF})$ is closed under complement if and only if $\#\Sigma \leq 2$ and I is complete.*

(3) *$T_C(\mathrm{CF})$ is closed under trace iteration if and only if either $\#\Sigma \leq 2$ or $I = \emptyset$.*

(4) $T_C(\mathrm{CF})$ *is closed under trace composition if and only if either* $\#\Sigma \leq 2$ *or* $I = \tilde{\emptyset}$ *or* $[\#\Sigma \leq 3$ *and* I *is complete*].

Finally, we consider the closure properties of the different types of context-sensitive trace languages.

3.34. Theorem. *Let* $C = (\Sigma, I, D)$ *be a reliance alphabet.* $T_C^{\exists}(\mathrm{CS})$, $T_C^{\forall}(\mathrm{CS})$ *and* $T_C(\mathrm{CS})$ *are closed under union, intersection, trace composition and trace iteration.*

Proof. We first note that, by Theorem 3.28, $T_C^{\exists}(\mathrm{CS}) = T_C^{\forall}(\mathrm{CS}) = T_C(\mathrm{CS})$. Next, because CS is closed under union, intersection, string concatenation and Kleene star, we observe that Theorem 2.14 implies that $T_C^{\exists}(\mathrm{CS})$ is closed under union, trace composition and trace iteration. Finally, we observe that Theorem 2.15 implies that $T_C^{\forall}(\mathrm{CS})$ is closed under intersection. Consequently, the theorem holds. □

We now turn to (the complexity of) some decision problems concerning different types of regular and context-free trace languages (from now on, whenever we deal with a regular string language L, we assume that L is given by a right-linear string grammar, and whenever we deal with a context-free string language L, we assume that L is given by a context-free string grammar).

Given a string language, we can use it to define the ('associated') trace language by either the existential or the universal method of claiming. Actually, both methods coincide if the string language we consider is C-consistent (where C is the given reliance alphabet). Hence, the question of deciding whether or not a given string language is C-consistent is quite important.

3.35. Theorem. (1) *It is decidable whether or not, for a reliance alphabet* $C = (\Sigma, I, D)$ *and a regular* $L \subseteq \Sigma^*$, L *is* C*-consistent.*

(2) *It is undecidable whether or not, for a reliance alphabet* $C = (\Sigma, I, D)$ *and a context-free* $L \subseteq \Sigma^*$, L *is* C*-consistent.*

Proof. (1): Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $L \subseteq \Sigma^*$ be regular. Let $A = (\Sigma, Q, \delta, q_{\mathrm{in}}, F)$ be the minimal deterministic finite automaton such that $L = L(A)$ (it can be effectively constructed from the right-linear grammar which generates L; see, e.g., [24]).

Clearly, L is C-consistent if and only if, for every $\{a, b\} \in I$ and for every $p, q \in Q$, $[\delta(p, ab) = q$ if and only if $\delta(p, ba) = q]$. However, this can be checked in a finite number of steps and, consequently, it is decidable whether or not L is C-consistent.

(2): Let $C = (\Sigma, I, D)$ be the reliance alphabet such that $\Sigma = \{a, b\}$ and $I = \{\{a, b\}\}$. Let $K \subseteq \{a, b\}^*$ be context-free and let $L = \{ab\}K \cup \{aa, ba, bb\}\{a, b\}^*$. Clearly, L is context-free and can be effectively constructed from K. Moreover, L is C-consistent if and only if $K = \{a, b\}^*$. Since it is undecidable for an arbitrary context-free string language K whether or not $K = \{a, b\}^*$ (see, e.g., [44]), it is undecidable for an arbitrary context-free string language L whether or not L is C-consistent. □

We now turn to the membership problem. However, first we observe the following.

Let $C = (\Sigma, I, D)$ be a reliance alphabet, let $x \in \Sigma^*$ and let $L \subseteq \Sigma^*$. Then,

(i) $[x]$ is a *finite* subset of Σ^*;

(ii) $[x] \in [L]^\exists$ if and only if $[x] \cap L \neq \emptyset$;

(iii) $[x] \in [L]^\forall$ if and only if $[x] \subseteq L$; and

(iv) (if L is C-consistent) $[x] \in [L]$ if and only if $x \in L$.

Thus, if the membership problem is decidable for L, then it is also decidable for $[L]^\exists$, for $[L]^\forall$, and (if L is C-consistent) for $[L]$. Consequently, we have the following result.

3.36. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet. Then the membership problem for $T_C^\exists(\mathrm{REG})$, $T_C^\forall(\mathrm{REG})$, $T_C(\mathrm{REG})$, $T_C^\exists(\mathrm{CF})$, $T_C^\forall(\mathrm{CF})$, $T_C(\mathrm{CF})$, $T_C^\exists(\mathrm{CS})$, $T_C^\forall(\mathrm{CS})$, and $T_C(\mathrm{CS})$ is decidable.*

As far as the complexity of the membership problem for various classes of trace languages is concerned, we have the following results.

We start by discussing classes of trace languages defined by using regular string languages.

3.37. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let α be the size of the greatest clique of I. Let $L \subseteq \Sigma^*$ be regular.*

(1) *There is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]^\exists$ in $O(|x|^\alpha)$ time.*

(2) *There is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]^\forall$ in $O(|x|^\alpha)$ time.*

(3) *If L is C-consistent, then there is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]$ in $O(|x|)$ time.*

Proof. (2) directly follows from (1), because, for every $x \in \Sigma^*$, $[x] \in [L]^\forall$ if and only if $[x] \notin [\Sigma^* - L]^\exists$ and $\Sigma^* - L$ is a regular string language.

(3): Obvious, because the membership problem for regular string languages is decidable in linear time. □

For the case when trace languages are specified using context-free string languages, we have the following result.

3.38. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let α be the size of the greatest clique of I. Let $L \subseteq \Sigma^*$ be context-free.*

(1) *There is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]^\exists$ in $O((|x|^{3\alpha})/(\log|x|))$ time.*

(2) *There is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]^\exists$ in $O(\mathrm{BM}(|x|^\alpha))$ time, where $\mathrm{BM}(n)$ is the time needed for multiplying two boolean $n \times n$ matrices.*

(3) *If L is C-consistent, then there is an algorithm which, given an arbitrary string $x \in \Sigma^*$, decides whether or not $[x] \in [L]$ in $f(|x|)$ time, where f is the time complexity of the membership problem for context-free string languages.*

Proof. (3): Obvious. □

Concerning the general membership problem for existentially regular and context-free trace languages, one has the following result.

3.39. Theorem. *The problem whether or not given an arbitrary reliance alphabet $C = (\Sigma, I, D)$, an arbitrary regular or context-free $L \subseteq \Sigma^*$, and an arbitrary $x \in \Sigma^*$, $[x] \in [L]^\exists$ is* NP*-complete.*

Imposing various restrictions on the (in)dependence relation leads to the following results.

3.40. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet, where I is transitive. The following problems are decidable for arbitrary regular $L_1, L_2 \subseteq \Sigma^*$.*

(1)(a) "$[L_1]^\exists = \emptyset$?" *and* "$[L_1]^\exists = \Theta(C)$?", *and*

(b) "$[L_1]^\forall = \emptyset$?" *and* "$[L_1]^\forall = \Theta(C)$?".

(2)(a) "$[L_1]^\exists \subseteq [L_2]^\exists$?" *and*

(b) "$[L_1]^\forall \subseteq [L_2]^\forall$?".

(3)(a) "$[L_1]^\exists = [L_2]^\exists$?" *and*

(b) "$[L_1]^\forall = [L_2]^\forall$?".

(4)(a) "$[L_1]^\exists \cap [L_2]^\exists = \emptyset$?" *and*

(b) "$[L_1]^\forall \cap [L_2]^\forall = \emptyset$?".

3.41. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet, where I is complete. The following problems are decidable for arbitrary context-free $L_1, L_2 \subseteq \Sigma^*$.*

(1) "$[L_1]^\exists = \emptyset$?" *and* "$[L_1]^\exists = \Theta(C)$?".

(2) "$[L_1]^\exists \subseteq [L_2]^\exists$?".

(3) "$[L_1]^\exists = [L_2]^\exists$?".

(4) "$[L_1]^\exists \cap [L_2]^\exists = \emptyset$?".

Proof. This result follows from Theorem 3.40, Theorem 3.13(1) and the following two well-known (see, e.g., [44]) facts:

(i) the Parikh image of a context-free string language can be effectively computed, and

(ii) given a semilinear set S, a regular string language with S as its Parikh image can be effectively computed. □

We conclude this subsection by discussing some fundamental problems which turn out to be undecidable already on the level of trace languages defined by regular string languages.

3.42. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet where, for some a, b, c, and d in Σ, $I(C)$ has an induced dep-graph of the form depicted in Fig. 10. Then the following problems are recursively undecidable for arbitrary regular $L_1, L_2 \subseteq \Sigma^*$.*

(1)(a) "$[L_1]^{\exists}=[L_2]^{\exists}$?" *and*

(b) "$[L_1]^{\forall}=[L_2]^{\forall}$?".

(2)(a) "$[L_1]^{\exists}\subseteq[L_2]^{\exists}$?" *and*

(b) "$[L_1]^{\forall}\subseteq[L_2]^{\forall}$?".

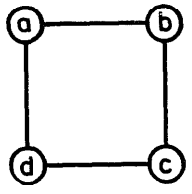


Fig. 10

Proof. (1)(b): From Theorem 2.16(3) it follows that, for $L_1, L_2 \subseteq \Sigma^*$, $[L_1]^{\exists}=[L_2]^{\exists}$ if and only if $[\Sigma^*-L_1]^{\forall}=[\Sigma^*-L_2]^{\forall}$. Hence, (1)(b) directly follows from (1)(a).

(2)(a) and (b) directly follow from (1)(a) and (1)(b) respectively. □

Bibliographical comments

Subsection 3.1 is more or less an overview and an extension of the investigation of equations on trace languages initiated in [32]. There the notion of a C-existential (trace) function is introduced and proofs of Theorem 3.8(1) and Theorem 3.10(1) can be found.

Basic papers on the subject of Subsection 3.2 are [2, 3, 5, 46]. Theorem 3.13(1) is proved in [46]. Both the first and the third characterization of consistently regular trace languages (Theorem 3.14 and Theorem 3.19, respectively) are proved in [37], where also the notions of concurrent decomposition and concurrent trace iteration are introduced. Lemma 3.21 is proved in [36], although the proof provided there is completely different from the one we have given. Both the second characterization of consistently regular trace languages (Theorem 3.16) and the infinite hierarchy of existentially regular trace languages (Theorem 3.19) are proved in [5], where also the notion of an (n, K)-existential trace language is introduced. Theorem 3.25(1) (also stated as Lemma 3.12) is proved in [3] and Theorem 3.26 is proved in [2].

Basic papers on the subject of closure properties of trace languages (first part of Subsection 3.3) are [2, 3, 46]. Theorem 3.29(1) is proved in [3] and in [47], while Theorem 3.29(2) is proved in [3] and in [42]; Theorem 3.29(2) is also stated in [8]. Theorem 3.30 and Theorem 3.31 (except for the closure of consistently regular trace languages under trace composition) are proved in [3]. The fact that consistently regular trace languages are closed under trace composition is proved in both [13] and [17]. Theorem 3.32 and Theorem 3.33 are proved in [2].

The basic paper on the subject of decidability problems of trace languages (second part of Subsection 3.3) is [6], where Theorems 3.37(1), 3.38(1), 3.39, 3.40(3)(a), and Theorem 3.42(1)(a) are proved. Finally, Theorem 3.38(2) is proved in [9] and Theorem 3.40 is proved in [3].

4. Dep-graphs and dep-graph languages revisited

4.1. Using dep-graphs to reason about traces

In this subsection we will demonstrate the use of dep-graphs in the analysis of various properties of traces. We will reconsider Algorithm 2.10—this time in terms of dep-graphs—and provide some other basic results on traces using the theory of dep-graphs. We hope that in this way we can demonstrate the usefulness of dep-graphs in both getting a good understanding of (an intuitive feeling for) various properties of traces and getting elegant proofs.

We begin by translating Algorithm 2.10 into the framework of dep-graphs. The result of this translation is Algorithm 4.1, which works as follows.

Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $t \neq 1$ be a trace over C, of which we want to find a normal representative; let d be a dep-graph of t. The algorithm will construct the decomposition $t = t_1 \circ \cdots \circ t_k$ for some $k \geq 1$, by constructing strings $u(1), \ldots, u(k) \in \Sigma^*$ such that $t_i = [u(i)]$ for all $1 \leq i \leq k$. The construction of these $u(i)$'s is by iterating the following steps (starting with the graph $g = d$ and $i = 1$):

(i) $u(i)$ is a string built up by concatenating the labels of the minimal elements of the current graph g;

(ii) the set of minimal elements and their adjacent edges are removed from g (and so a new current graph results); and

(iii) i is increased by 1.

This iteration stops when g becomes the empty graph.

Formally, the algorithm is as follows.

4.1. Algorithm.

Input: A reliance alphabet $C = (\Sigma, I, D)$ and a dep-graph $d = (V, E, \Sigma, l) \neq \Lambda$ over C.

Declaration: Let $p = \#V$ and $n = \#\Sigma$; let k be a variable over $\{0, \ldots, p\}$; let u be an array of length p over $(\Sigma \cup \{\lambda\})^n$.

Computation:

(1) for all $1 \leq k \leq p$, $u(k) := \lambda$.

(2) $k := 0$.

(3) $k := k+1$; $g := (V, E, \Sigma, l)$.

(4) $u(k) := \sigma_1 \ldots \sigma_r$, where $r \geq 1$ and $\sigma_1, \ldots, \sigma_r \in \Sigma$ are such that $\{\sigma_1, \ldots, \sigma_r\} = \{l(v) : v \in \min(g)\}$.

(5) $V := V - \min(g)$.

(6) $E := E \cap (V \times V)$; $l := l|_V$.

(7) if $V \neq \emptyset$ then goto (3).

(8) stop.

Output: The string $u(1) \ldots u(k)$.

4.2. Example. Let C be the reliance alphabet given in Example 1.4 and let $d = \langle adcbbed \rangle$. Then the computation of Algorithm 4.1 starting with C and d, can be illustrated as in Fig. 11. Hence, the output of the algorithm is *abdbced*.

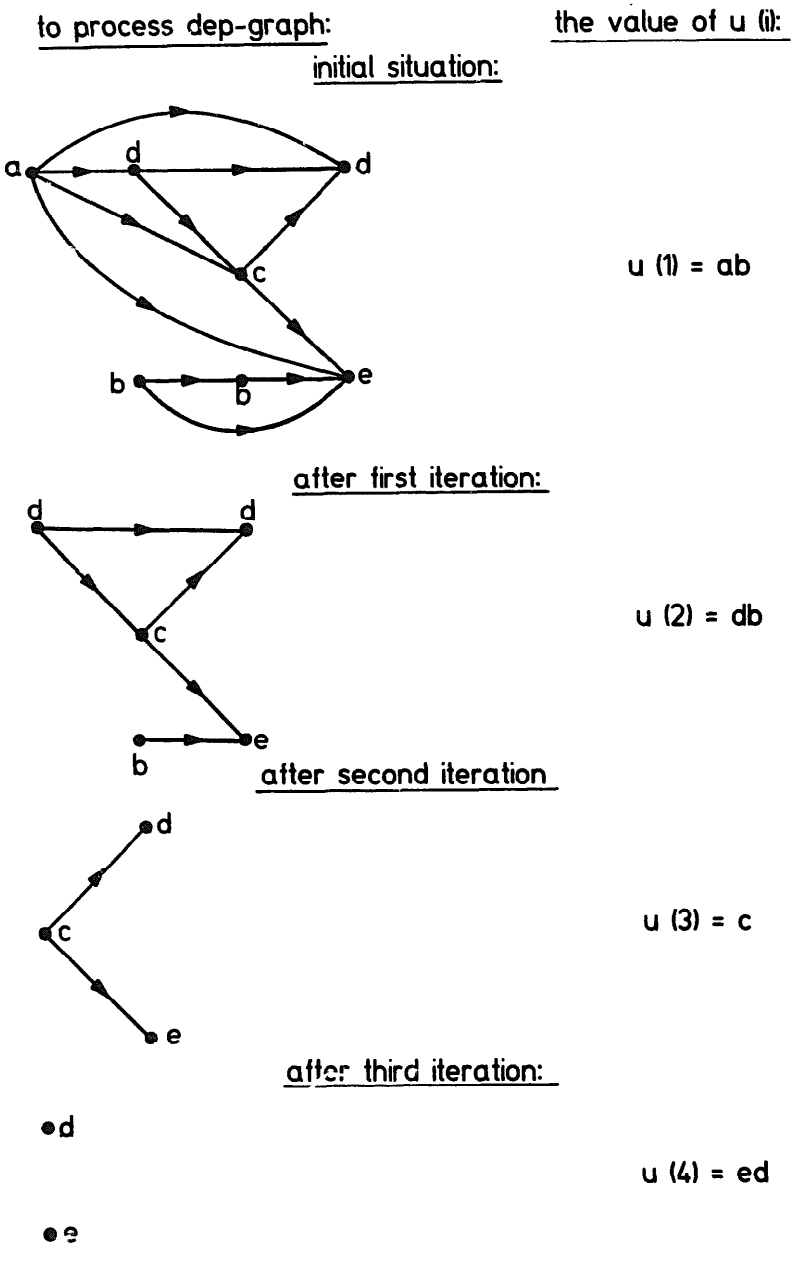


Fig. 11

Formalizing the above we get the following result.

4.3. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $d=(V, E, \Sigma, l)\neq\Lambda$ be a dep-graph over C; let $t=[\![d]\!]$. Let X be the set of outputs of Algorithm* 4.1 *for the input (C, d). Then, for every $x\in\Sigma^*$, $x\in X$ if and only if x is a normal representative of t.*

Proof. Let $x\in X$. Hence, there exist a $k\geq 1$ and $u(1),\ldots,u(k)\in\Sigma^*$ as described in Algorithm 4.1 such that $x=u(1)\ldots u(k)$. Then, in order to prove that x is a

normal representative of t, we have to show that

(i) for all $1 \leq i \leq k$, $u(i) \neq \lambda$;

(ii) for all $1 \leq i \leq k$, $\#_a(u(i)) = 1$ for each $a \in \text{alph}(u(i))$, and $\{a, b\} \in I$ for each $a, b \in \text{alph}(u(i))$, $a \neq b$; and

(iii) for all $1 \leq i \leq k-1$, for each $a \in \text{alph}(u(i+1))$, there exists a $b \in \text{alph}(u(i))$ such that $\{a, b\} \in D$.

Clearly, (i) holds. Furthermore, since, at every moment in the computation of x, $\min(g)$ is a subset of a node cut of d, Theorem 2.38(1) implies that (ii) holds. Finally, since, for every $2 \leq i \leq k$ and for every symbol b occurring in $u(i)$, b corresponds, at some moment in the computation of x, to a minimal node in g which was not a minimal node in g one iteration before, d contains an edge from a node corresponding to a symbol occurring in $u(i-1)$ to the node corresponding to b; thus, for every $2 \leq i \leq k$ and for every $b \in \text{alph}(u(i))$, there exists an $a \in \text{alph}(u(i-1))$ such that $\{a, b\} \in D$, and, hence, (iii) holds. Consequently, x is a normal representative of t.

Let $x \in \Sigma^*$ be a normal representative of t. Hence, there exist $k \geq 1$ and $u(1), \ldots, u(k) \in \Sigma^*$, satisfying (i), (ii), and (iii) above, such that $x = u(1) \ldots u(k)$. Then, by the construction of a dep-graph and by requirements (ii) and (iii) above, a symbol b occurs in $u(1)$ if and only if the node corresponding to b in d has no incoming edges in d; thus, $\text{alph}(u(1)) = \{l(v) : v \in \min(d)\}$. Hence, $u(1)$ can be obtained in the first iteration of Algorithm 4.1. Moreover, it is now easily seen that, repeating the above considerations, $u(1), \ldots, u(k)$ can be successively obtained by Algorithm 4.1. Consequently, $x = u(1) \ldots u(k) \in X$. □

For a reliance alphabet C, the following result provides means for counting the number of C-equivalent strings with a given string $x \in (\text{alph}(C))^*$.

4.4. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let α be the size of the greatest clique of I; let $t \in \Theta(C)$. Then $\#t \leq \min((|t|)!, \alpha^{|t|})$.*

Proof. If $|t| = 0$, then the theorem obviously holds.

Assume that $|t| \neq 0$.

On the one hand, if $x, y \in t$, then x is a permutation of y. Since a string of length n has at most $n!$ different permutations, we get $\#t \leq (|t|)!$.

On the other hand, if $x \in t$, then $x(1)$ is the label of a minimal node of $\langle\langle t \rangle\rangle$. Since the set of minimal nodes of $\langle\langle t \rangle\rangle$ is a node cut of $\langle\langle t \rangle\rangle$ and since, by Theorem 2.38(2), every node cut of $\langle\langle t \rangle\rangle$ has at most α elements, there are at most α candidates for being $x(1)$. Moreover, making a choice of a minimal node of $\langle\langle t \rangle\rangle$ and reducing $\langle\langle t \rangle\rangle$ by this node, there exist, for every $1 \leq i \leq |t|$, after iterating this choice and reducing $i-1$ times, at most α candidates for being $x(i)$. In this way we get $\#t \leq \alpha^{|t|}$.

Hence, $|t| \neq 0$ implies that both $\#t \leq (|t|)!$ and $\#t \leq \alpha^{|t|}$. Consequently, $\#t \leq \min((|t|)!, \alpha^{|t|})$. □

The last result in this subsection is a result on splitting up traces in prefixes and suffixes. Although a proof of it is already given in [6], we give here a different proof based on the analysis of dep-graphs. We believe that our proof provides a better intuitive insight into the nature of the result.

4.5. Theorem. *Let $C=(\Sigma, I, D)$ be a reliance alphabet and let α be the size of the greatest clique of I. Let $t \in \Theta(C)$ and let $P_t = \{t' \in \Theta(C)$: there exists a $t'' \in \Theta(C)$ such that $t = t' \circ t''\}$. Then $\#P_t \leq (1+|t|/\alpha)^{\alpha}$.*

Proof. From Theorems 2.36 and 2.37 it follows that the number of prefixes of t equals the number of possibilities to split up $\langle\langle t \rangle\rangle$ in two dep-graphs d_1 and d_2, such that $d = d_1 \square d_2$. Thus, $\#P_t$ equals the number of edge cuts of the directed node-labeled acyclic graph g, which is obtained from $\langle\langle t \rangle\rangle$ by adding the new (arbitrarily labeled) nodes $v_{\min}$ and $v_{\max}$ and the sets of new edges $\{(v_{\min}, v): v \in \min(\langle\langle t \rangle\rangle)\}$ and $\{(v, v_{\max}): v \in \max(\langle\langle t \rangle\rangle)\}$.

The proof now continues by counting the number of edge cuts in the graph g described above. In order to do this, we need the following fact, which can be proved by standard methods from the theory of partial orders (see, e.g., [11]). If $g' = (V', E')$ is a directed acyclic graph and $n \geq 1$ is such that, for every node cut S of g', $\#S \leq n$, then there exist n lines $g_i = (V_i, E_i)$ of g', $1 \leq i \leq n$, such that $V' = \bigcup_{i=1}^{n} V_i$.

Let us assume that $\langle\langle t \rangle\rangle = (V, E, \Sigma, l)$ and that $g = (V', E', \Sigma', l')$. From Theorem 2.38(2) it follows that, for every node cut S of $\langle\langle t \rangle\rangle$ and thus clearly also for every node cut S of g, $\#S \leq \alpha$. Hence, by the above fact and by the construction of g, we can find α lines $g_i = (V_i \cup \{v_{\min}, v_{\max}\}, E_i)$ of g, $1 \leq i \leq \alpha$, such that $V = \bigcup_{i=1}^{\alpha} V_i$. Since the intersection of an arbitrary line and an arbitrary edge cut of a directed acyclic graph consists of at most, in the case of g even exactly, one element, in order to 'edge cut' g, we have $\#V_1 + 1$ possibilities to 'edge cut' g_1. Moreover, whenever a choice for an edge cut of g_1 is made, in order to 'edge cut' g_2 consistently with this edge cut, we have at most $\#(V_2 - V_1) + 1$ possibilities to do so. In general, whenever, for some $1 \leq i \leq \alpha$, consistent choices of edge cuts of $g_1, \ldots, g_{i-1}$ are made, in order to 'edge cut' g_i consistently with these edge cuts, we have at most $\#(V_i - \bigcup_{j=1}^{i-1} V_j) + 1$ possibilities to do so.

Hence, the number of edge cuts of g is smaller than or equal to

$$\prod_{i=1}^{\alpha} \left(\#\left(V_i - \bigcup_{j=1}^{i-1} V_j \right) + 1 \right).$$

Now we observe that

(i) for every $1 \leq p, q \leq \alpha$, $p \neq q$,

$$\left(V_p - \bigcup_{j=1}^{p-1} V_i \right) \cap \left(V_q - \bigcup_{j=1}^{q-1} V_j \right) = \emptyset; \quad \text{and}$$

(ii) $\bigcup_{i=1}^{\alpha}\left(V_i-\bigcup_{j=1}^{i-1}V_j\right)=V.$

Thus, the number of edge cuts of g is smaller than or equal to $(\prod_{i=1}^{\alpha}(n_i+1))$, where $n_1,\ldots,n_\alpha\geqslant 0$ are such that $\sum_{i=1}^{\alpha}n_i=\#V=|t|$. Hence, since the geometric mean of a set of numbers is smaller than or equal to its arithmetic mean, the number of edge cuts of g is smaller than or equal to $(1+|t|/\alpha)^\alpha$. Consequently, the cardinality of P_t is bounded by $(1+|t|/\alpha)^\alpha$. □

4.2. Dep-graph languages and graph grammars

In this subsection we will introduce classes of graph grammars that generate dep-graph languages. Moreover, we demonstrate the correspondence between generating dep-graph languages by graph grammars and defining dep-graph languages by string languages in the usual way. The classes of graph grammars are natural—they fit well in the existing framework in the theory of graph grammars.

We start by recalling the notion of a directed node-label controlled graph grammar.

4.6. Definition. A *directed node-label controlled graph grammar*, abbreviated a *DNLC grammar*, is a system $G=(\Omega,\Xi,P,J,\xi)$, where

(i) Ω is an alphabet, called the *total alphabet of* G;
(ii) $\Xi\subseteq\Omega$ is the *terminal alphabet of* G;
(iii) $P\subseteq(\Omega-\Xi)\times\Gamma(\Omega)$ is the *set of productions of* G;
(iv) $J\subseteq\Omega\times\Omega$ is the *connection relation of* G; and
(v) ξ, called the *axiom of* G, is a graph over $\Omega-\Xi$ consisting of one node only.

Informally speaking, a DNLC grammar $G=(\Omega,\Xi,P,J,\xi)$ generates a set of graphs as follows.

Given a graph γ to be rewritten and a production of the form (X,β), where $X\in\Omega-\Xi$ and $\beta\in\Gamma(\Omega)$, one chooses a node v of γ labeled by X and replaces it by β (actually, by an isomorphic copy of β such that its set of nodes is disjoint with the set of nodes of γ). Then, in order to embed β in "the remainder of γ" (i.e., the graph resulting from γ by removing v), one uses the relation J as follows. For *every* pair $(b,c)\in J$ one establishes an edge from *each* node w in "the remainder of γ" labeled by c to each node of β labeled by b, provided that there was an edge from w to v in γ. Analogously, for *every* pair $(b,c)\in J$ one establishes an edge from *each* node of β labeled by b to each node w in "the remainder of γ" labeled by c, provided that there was an edge from v to w in γ. Every graph γ' isomorphic to the resulting graph is said to be *directly derived from* γ *in* G; we write then $\gamma\Rightarrow_{(G)}\gamma'$. Iterating the *direct derivation step* (starting with the axiom ξ of G) and choosing only those derived graphs that are labeled by labels from the terminal alphabet Ξ, one gets the (*graph*) *language* $L(G)$ *of* G.

4.7. Remark. The notion of a DNLC grammar discussed above differs somewhat from the standard notion of a DNLC grammar formally defined in [25]. First of all, we allow also erasing productions. Secondly, rather than to use separate incoming and outgoing connection relations, we use one common connection relation.

We now define two subclasses of the class of DNLC grammars (the reader will notice the obvious analogy with two basic classes of string grammars).

4.8. Definition. Let $G=(\Omega, \Xi, P, J, \xi)$ be a DNLC grammar.

(1) G is called a *regular DNLC grammar* if every production of G is
- either of the form $(X, {}^{\sigma}\bullet\!\!\rightarrow\!\!\bullet^{Y})$,
- or of the form $(X, \bullet^{\sigma})$,
- or of the form (X, Λ),

where $\sigma \in \Xi$ and $X, Y \in \Omega - \Xi$.

(2) G is called a *context-free DNLC grammar* if every production of G is
- either of the form $(X, {}^{Y}\bullet\!\!\rightarrow\!\!\bullet^{Z})$,
- or of the form $(X, \bullet^{\sigma})$,
- or of the form (X, Λ),

where $\sigma \in \Xi$ and $X, Y, Z \in \Omega - \Xi$.

4.9. Example. Let $\Xi=\{a, b, c\}$, $\Omega=\Xi \cup \{A, B, D, E, F, S\}$, $\xi \cong \bullet^{S}$, $J=(\Omega \times \Omega)-\{(a, b), (b, a), (b, c), (c, b)\}$, and

$$P=\{(S, \overset{D}{\bullet}\!\!\rightarrow\!\!\overset{F}{\bullet}), (S, \overset{E}{\bullet}\!\!\rightarrow\!\!\overset{F}{\bullet}), (D, \overset{E}{\bullet}\!\!\rightarrow\!\!\overset{S}{\bullet}), (E, \overset{A}{\bullet}\!\!\rightarrow\!\!\overset{B}{\bullet}), (A, \overset{a}{\bullet}), (B, \overset{b}{\bullet}), (F, \overset{c}{\bullet})\}.$$

Then $G=(\Omega, \Xi, P, J, \xi)$ is a context-free DNLC grammar.

Furthermore, Fig. 12 depicts an example of a derivation (in G) of a graph from $L(G)$.

The following notion is crucial in considering a DNLC grammar as a generator of dep-graphs.

4.10. Definition. Let $C=(\Sigma, I, D)$ be a reliance alphabet and let $G=(\Omega, \Xi, P, J, \xi)$ be a DNLC grammar. G is called *C-uniform* if $\Sigma=\Xi$ and $J=(\Omega \times \Omega)-I$.

4.11. Remark. Note that in a derivation by a C-uniform DNLC grammar, where C is a reliance alphabet, breaking of edges between nodes happens only between nodes labeled by terminal symbols and, moreover, this breaking up of edges happens according to ind(C) (labels independent of each other in C get disconnected).

4.12. Example. Let $C=(\Sigma, I, D)$ be the reliance alphabet such that $\Sigma=\{a, b, c\}$ and $I=\{\{a, b\}, \{b, c\}\}$. Then the DNLC grammar of Example 4.9 is C-uniform.

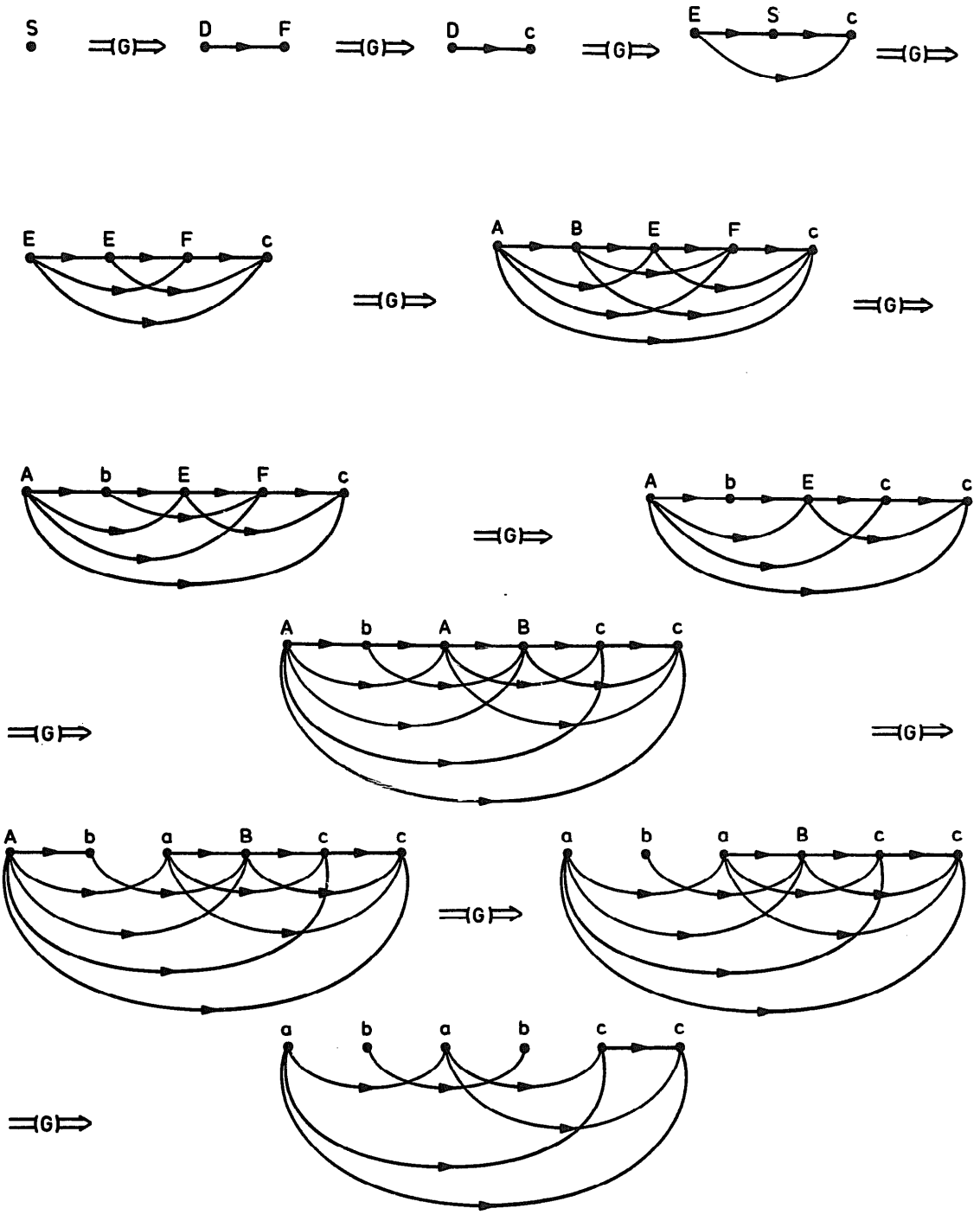


Fig. 12

We now demonstrate that regular C-uniform DNLC grammars, where C is a reliance alphabet, generate *precisely* the class of regular dep-graph languages over C.

4.13. Theorem. Let $C=(\Sigma, I, D)$ *be a reliance alphabet and let* $W \subseteq \Gamma(\Sigma)$. *Then* $W \in D_C(\mathrm{REG})$ *if and only if* $W=L(G)$ *for some regular* C*-uniform* DNLC *grammar* G.

Proof. The proof goes in two steps, each proving the implication in one direction.

(i) Let $W \in D_C(\mathrm{REG})$. Hence there exists a right-linear string grammar $G_1 = (\Omega, \Sigma, P_1, S)$ such that $W = \langle L(G) \rangle$.

Consider the regular C-uniform DNLC grammar $G_2 = (\Omega, \Sigma, P_2, J, \xi)$, where

$$P_2 = \{(X, \overset{\sigma}{\bullet}\!\!\rightarrow\!\!\overset{Y}{\bullet}) : \sigma \in \Sigma,\ X, Y \in \Omega - \Sigma, \text{ and } (X, \sigma Y) \in P_1\}$$

$$\cup \{(X, \overset{\sigma}{\bullet}) : \sigma \in \Sigma,\ X \in \Omega - \Sigma, \text{ and } (X, \sigma) \in P_1\}$$

$$\cup \{(X, \Lambda) : X \in \Omega - \Sigma, \text{ and } (X, \lambda) \in P_1\}, \quad \text{and}$$

$$\xi \cong \overset{S}{\bullet}.$$

It is easily seen that $W = L(G_2)$.

(ii) Let $G_1 = (\Omega, \Sigma, P_1, J, \xi)$ be a regular C-uniform DNLC grammar.

Consider the right-linear string grammar $G_2 = (\Omega, \Sigma, P_2, S)$, where

$$P_2 = \{(X, \sigma Y) : \sigma \in \Sigma,\ X, Y \in \Omega - \Sigma, \text{ and } (X, \overset{\sigma}{\bullet}\!\!\rightarrow\!\!\overset{Y}{\bullet}) \in P_1\}$$

$$\cup \{(X, \sigma) : \sigma \in \Sigma,\ X \in \Omega - \Sigma, \text{ and } (X, \overset{\sigma}{\bullet}) \in P_1\}$$

$$\cup \{(X, \lambda) : X \in \Omega - \Sigma, \text{ and } (X, \Lambda) \in P_1\},$$

and S is such that $\xi \cong \bullet^S$.

It is easily seen that $L(G_1) = \langle L(G_2) \rangle$.

The theorem follows from (i) and (ii) above. □

Now we move to the context-free dep-graph languages over a reliance alphabet C. It turns out that this class of dep-graph languages is *precisely* the class of graph languages generated by context-free C-uniform DNLC grammars.

The proof of the following result can be made completely analogous (using the Chomsky Normal Form for context-free string languages) to the proof of Theorem 4.13 and so it is left to the reader.

4.14. Theorem. *Let $C = (\Sigma, I, D)$ be a reliance alphabet and let $W \subseteq \Gamma(\Sigma)$. Then $W \in D_C(\mathrm{CF})$ if and only if $W = L(G)$ for some context-free C-uniform* DNLC *grammar G.*

4.15. Remark. A context-free DNLC grammar has productions of the form $(X, {}^{Y}\bullet\!\!\rightarrow\!\!\bullet^{Z})$, $(X, \bullet^{\sigma})$, and (X, Λ), where σ is a terminal and X, Y, and Z are nonterminals. Clearly, this form of productions (except for the erasing productions (X, Λ)) comes from the analogy with context-free string grammars in *Chomsky Normal Form* (see, e.g., [44]).

Alternatively, we could have defined context-free DNLC grammars by also allowing productions of the form depicted in Fig. 13, where $n \geq 1$ and $X, Y_1, \ldots, Y_n$ are

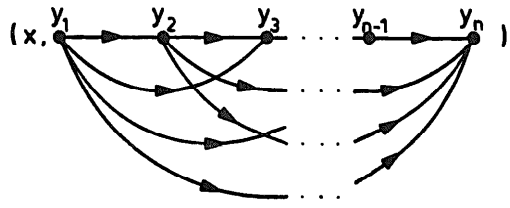


Fig. 13

nonterminals (now the right-hand side of a production does not have to be at most binary as long as all the 'transitive closure edges' are also present). In this way, one gets a closer analogy to (the form of) arbitrary context-free string grammars (except that in all productions with more than one node at the right-hand side all the labels are nonterminals).

In manipulating DNLC grammars the first form (the form that we use in this paper) is more convenient. However, it turns out that the above characterization result (Theorem 4.14) holds for both variants of the definition.

Bibliographical comments

Using graph grammars to generate regular dependence graph languages was initiated in [1], where also a proof of a theorem closely related to Theorem 4.13 is given.

5. Trace languages, dep-graph languages, and Petri nets

5.1. C/E structures, C/E systems, firing sequences and processes

The theory of *Petri nets* deals with distributed systems and processes. Within this theory many important concepts and tools have been developed which aid in the design and analysis of distributed systems.

The basic system model underlying the theory of Petri nets is a *condition/event system*, abbreviated a *C/E system*. In this section we are going to relate the theory of traces and dep-graphs to the theory of C/E systems. We start by recalling in this subsection the notion of a C/E system which we will base on the notion of a C/E structure introduced here. Then we discuss the firing sequence approach to represent the behavior of C/E structures and of C/E systems and the process representation of the behavior of C/E systems. Our exposition in this section is very 'dense' (and rather 'dry') in the sense that it gives all the basic notions (and terminology) only to the extent we will need them.

The basic construct underlying the whole area of Petri nets is that of a (directed) net.

5.1. Definition. (1) A *net* is a 3-tuple $N=(S, T, F)$, where

(i) S and T are finite sets such that $S \cap T=\emptyset$ and $S \cup T \neq \emptyset$, and

(ii) $F \subseteq (S \times T) \cup (T \times S)$ is such that $T \subseteq \mathrm{dom}(F) \cup \mathrm{ran}(F)$.

(2) Let $N=(S, T, F)$ be a net. The elements of S are called *S-elements* and the elements of T are called *T-elements*. For every $x \in S \cup T$, the *input set of* x, denoted by ${}^\bullet x$, is the set $\{y \in S \cup T : (y, x) \in F\}$; the *output set of* x, denoted by $x^\bullet$, is the set $\{y \in S \cup T : (x, y) \in F\}$; and the *neighborhood of* x, denoted by ${}^\bullet x^\bullet$, is the set ${}^\bullet x \cup x^\bullet$. For every $X \subseteq S \cup T$, the *input set of* X, denoted by ${}^\bullet X$, is the set $\bigcup \{{}^\bullet x : x \in X\}$; the *output set of* X, denoted by $X^\bullet$, is the set $\bigcup \{x^\bullet : x \in X\}$; and the *neighborhood of* X, denoted by ${}^\bullet X^\bullet$, is the set ${}^\bullet X \cup X^\bullet$.

5.2. Remark. In the above definition of a net we require that $T \subseteq \mathrm{dom}(F) \cup \mathrm{ran}(F)$—this differs from the standard definition of a net in the literature. However, nets as used in condition/event structures, condition/event systems, and occurrence nets (to be defined later on in this subsection) satisfy this property and thus introducing nets in this way facilitates our considerations later on.

In order to talk about dynamic behavior of nets we need the following notion.

5.3. Definition. Let $N=(S, T, F)$ be a net.

(1) Let $t \in T$ and let $\mu \subseteq S$. t *is enabled in* μ, denoted by $\mu[t\rangle$, if ${}^\bullet t \subseteq \mu$ and $t^\bullet \subseteq S - \mu$.

(2) Let $t \in T$ and let $\mu, \mu' \subseteq S$. t *fires from* μ *to* μ', denoted by $\mu[t\rangle\mu'$, if $\mu[t\rangle$ and $\mu' = (\mu - {}^\bullet t) \cup t^\bullet$.

(3) Let $\mu \subseteq S$. The *μ-reachability set*, denoted by $\leqslant \mu \geqslant$, is the smallest subset of $\mathcal{P}(S)$, satisfying

(i) $\mu \in \leqslant \mu \geqslant$, and

(ii) if either $\mu'[t\rangle\mu''$ or $\mu''[t\rangle\mu'$, for a $t \in T$, a $\mu' \in \leqslant \mu \geqslant$, and a $\mu'' \subseteq S$, then $\mu'' \in \leqslant \mu \geqslant$.

A net $N=(S, T, F)$ has a convenient representation as a bipartite graph, where the S-elements and the T-elements form the bipartition of the set of nodes, and F corresponds to the set of edges. Usually, S-elements are represented by circles and T-elements are represented by boxes.

5.4. Example. Consider the net $N=(S, T, F)$, where $S=\{s_1, s_2, s_3, s_4\}$, $T=\{t_1, t_2, t_3\}$, and

$$F=\{(s_1, t_1), (s_2, t_2), (s_3, t_1), (s_4, t_3), (t_1, s_2), (t_2, s_3), (t_2, s_4), (t_3, s_1)\}.$$

The graphical representation of N is depicted in Fig. 14.

Furthermore, t_1 is enabled in $\{s_1, s_3\}$ and t_1 fires from $\{s_1, s_3\}$ to $\{s_2\}$. Finally, $\leqslant \{s_1, s_3\} \geqslant = \{\{s_1, s_3\}, \{s_2\}, \{s_3, s_4\}\}$.

We are now ready to define the notion of a condition/event structure and to recall the notion of a condition/event system.

5.5. Definition. (1) A *condition/event structure, C/E structure* for short, is a net $N=(P, Q, R)$ such that $P \subseteq \mathrm{dom}(R) \cup \mathrm{ran}(R)$.

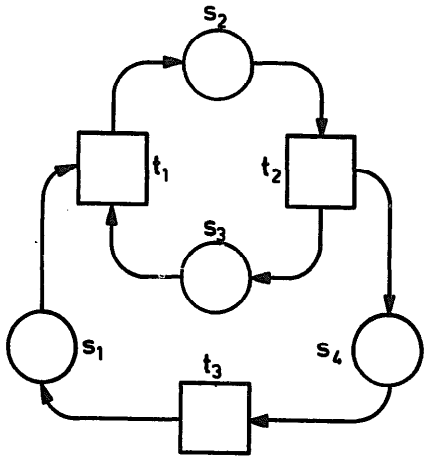


Fig. 14

(2) A C/E structure $N=(P, Q, R)$ is *simple* if, for all $x, y \in P \cup Q$, ${}^\bullet x = {}^\bullet y$ and $x^\bullet = y^\bullet$ imply that $x = y$.

(3) Let $N=(P, Q, R)$ be a C/E structure.

(3.1) Let $\alpha \in Q^*$, $\mu_0 \subseteq P$, and $\mu_{|\alpha|} \in \leqslant \mu_0 \geqslant$. α is called a *firing sequence (of N) from μ_0 to $\mu_{|\alpha|}$*, denoted by $\mu_0[\alpha\rangle\mu_{|\alpha|}$ if there exist $\mu_1, \ldots, \mu_{|\alpha|-1} \in \leqslant \mu_0 \geqslant$ such that, for all $1 \leqslant i \leqslant |\alpha|$, $\mu_{i-1}[\alpha(i)\rangle\mu_i$.

(3.2) Let $\alpha \in Q^*$ and $\mu \subseteq P$. α is called a *firing sequence (of N) from μ*, denoted by $\mu[\alpha\rangle$ if there exists a $\mu' \in \leqslant \mu \geqslant$ such that $\mu[\alpha\rangle\mu'$.

(3.3) Let $\alpha \in Q^*$. α is called a *firing sequence (of N)* if there exists a $\mu \subseteq P$ such that $\mu[\alpha\rangle$.

(3.4) Let $\mu \subseteq P$ and $\mu' \in \leqslant \mu \geqslant$. The *$(\mu, \mu')$-language of N*, denoted by $L(\mu, \mu', N)$, is the set of all firing sequences of N from μ to μ'. The *(μ)-language of N*, denoted by $L(\mu, N)$, is the set of all firing sequences of N from μ. The *(firing sequence) language of N*, denoted by $L(N)$, is the set of all firing sequences of N.

(4) A *condition/event system, C/E system* for short, is a system $M=(P, Q, R, \mathscr{C})$, where

(i) (P, Q, R) is a simple C/E structure; and

(ii) $\mathscr{C} \subseteq \mathscr{P}(P)$ is such that

(a) $\mathscr{C} = \leqslant \mu \geqslant$ for some $\mu \subseteq P$, and

(b) for every $q \in Q$, there exist $\mu', \mu'' \in \mathscr{C}$ such that $\mu'[q\rangle\mu''$.

(5) Let $M=(P, Q, R, \mathscr{C})$ be a C/E system.

(5.1) M is *contact-free* if, for every $q \in Q$ and for every $\mu \in \mathscr{C}$, ${}^\bullet q \subseteq \mu$ implies that $q^\bullet \subseteq P - \mu$ and $q^\bullet \subseteq \mu$ implies that ${}^\bullet q \subseteq P - \mu$.

(5.2) Let $\alpha \in Q^*$. α is called a *firing sequence (of M)* if there exists a $\mu \in \mathscr{C}$ such that $\alpha \in L(\mu, N)$, where $N=(P, Q, R)$.

(5.3) The *(firing sequence) language of M*, denoted by $L(M)$, is the set of all firing sequences of M.

In the above definition, each element of P is called a *condition* (*of* N or *of* M), each element of Q is called an *event* (*of* N or *of* M), and each element of $\mathscr{C}$ is called a *case* (*of* M).

5.6. Remark. (1) In Definition 5.5 and in the sequel it is assumed that notions and terminology concerning nets carry over in the obvious way to C/E structures and C/E systems (and other systems based on nets that we will consider).

(2) Note that one can consider a C/E system $(P, Q, R, \mathscr{C})$ to consist of two components: the first one specifying the underlying static structure of the system and the second one specifying the dynamic behavior of the system. The first component is given by the simple C/E structure (P, Q, R) and the second one by the distributed state space $\mathscr{C}$.

(3) In the set-up as above the notion of contact-freeness of a C/E system is crucial for the investigation of the behavior of C/E systems. We do not have enough room here to elaborate on this issue, but it turns out that defining processes of C/E systems (later on in this subsection) is possible for contact-free C/E systems only.

In order to represent a C/E system $M = (P, Q, R, \mathscr{C})$ graphically, we use the graph representation of the net (P, Q, R) and additionally, in order to represent $\mathscr{C}$, we choose a $\mu \subseteq P$ such that $\mathscr{C} = \leqslant \mu \geqslant$ and then represent this μ by putting a token in each circle representing an element of μ.

5.7. Example. Consider the net $N = (S, T, F)$ of Example 5.4. Let $\mu' = \{s_1, s_3, s_4\}$, $\mu'' = \{s_2\}$ and $M = (S, T, F, \leqslant \mu'' \geqslant)$. Then N is a C/E structure, $L(\mu', N) = \{x \in T^* : \text{there exists a } z \in T^* \text{ such that } xz \in \{t_1 t_3 t_2\}^*\}$, M is a contact-free C/E system and $L(M) = \{x \in T^* : \text{there exist } y, z \in T^* \text{ such that } yxz \in \{t_1 t_2 t_3\}^*\}$.

The following is a fundamental result concerning the languages of C/E structures and C/E systems.

5.8. Theorem. (1) *Let N be a C/E structure. Then $L(N)$ is regular.*

(2) *Let M be a C/E system. Then $L(M)$ is regular.*

A way of defining the behavior of a C/E system M is to consider $L(M)$—this corresponds to the so called 'interleaved' point of view, where the behavior of the concurrent system is given through a set of its linear histories. An alternative (and perhaps more appropriate for concurrent systems) point of view is to consider partially ordered sets of events which may take place in the system. In particular, recording such sets of events *together* with appropriate condition holdings leads one to the notion of a process in a C/E system.

In order to formalize this, one needs a special subclass of nets.

5.9. Definition. (1) An *occurrence net* is a net $K = (S, T, F)$, where

(i) for every $x, y \in S \cup T$, $(x, y) \in F^+$ implies that $(y, x) \notin F^+$; and

(ii) for every $s \in S$, $\#({}^\bullet s) \leq 1$ and $\#(s^\bullet) \leq 1$.

(2) Let $K = (S, T, F)$ be an occurrence net. A *slice of* K is a subset S' of S such that, for every $s_1, s_2 \in S'$, $(s_1, s_2) \notin F^+$ and for every $s_3 \in S - S'$, there exists an $s_4 \in S'$ such that either $(s_3, s_4) \in F^+$ or $(s_4, s_3) \in F^+$.

5.10. Example. The graph depicted in Fig. 15 represents an occurrence net K. The set of slices of K consists of $\{s_1\}$, $\{s_2, s_3\}$, $\{s_4\}$, and $\{s_5\}$.

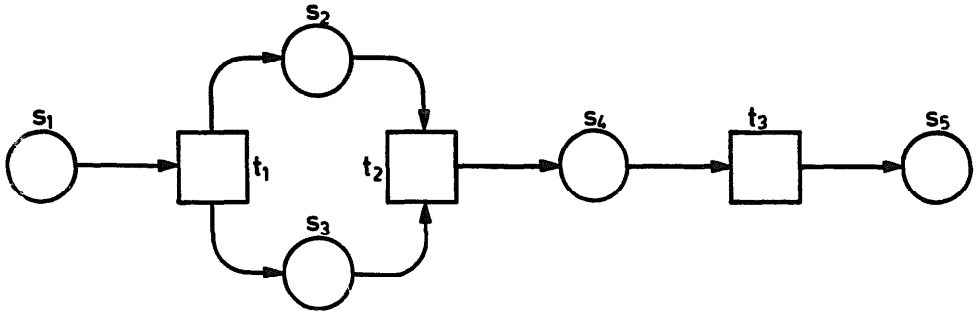


Fig. 15

Labeling occurrence nets will be a convenient way to use them for expressing the (concurrent) behavior of C/E systems.

5.11. Definition. (1) A *labeled occurrence net* is a system $K = (S, T, F, \Sigma, \phi)$, where

(i) (S, T, F) is an occurrence net;

(ii) Σ is an alphabet; and

(iii) $\phi : S \cup T \to \Sigma$ is a function such that $\phi(S) \cap \phi(T) = \emptyset$.

(2) Let $M = (P, Q, R, \mathscr{C})$ be a contact-free C/E system. A *process of* M is a labeled occurrence net $K = (S, T, F, P \cup Q, \phi)$, such that

(i) $\phi(S) \subseteq P$ and $\phi(T) \subseteq Q$;

(ii) for every $t \in T$, $\phi({}^\bullet t) = {}^\bullet\phi(t)$ and $\phi(t^\bullet) = \phi(t)^\bullet$; and

(iii) for every slice S' of K, ϕ is injective on S' and $\phi(S') \in \mathscr{C}$.

5.12. Remark. In the above definition it is assumed that the notion of a slice of an occurrence net carries over to a labeled occurrence net in the obvious way.

5.13. Example. Consider the contact-free C/E system M depicted in Fig. 16(a). Then the labeled occurrence net K depicted in Fig. 16(b) is a process of M.

The following result on processes is fundamental in the sense that it carries 'dependences' in C/E systems to 'dependences' in their processes.

5.14. Theorem. *Let $M = (P, Q, R, \mathscr{C})$ be a C/E system and let $K = (S, T, F, P \cup Q, \phi)$ be a process of M. Let $q_1, q_2 \in Q$ be such that $q_1 \neq q_2$ and ${}^\bullet q_1 \cap {}^\bullet q_2 \neq \emptyset$, and let $t_1, t_2 \in T$ be such that $\phi(t_1) = q_1$ and $\phi(t_2) = q_2$. Then either $(t_1, t_2) \in F^+$ or $(t_2, t_1) \in F^+$.*

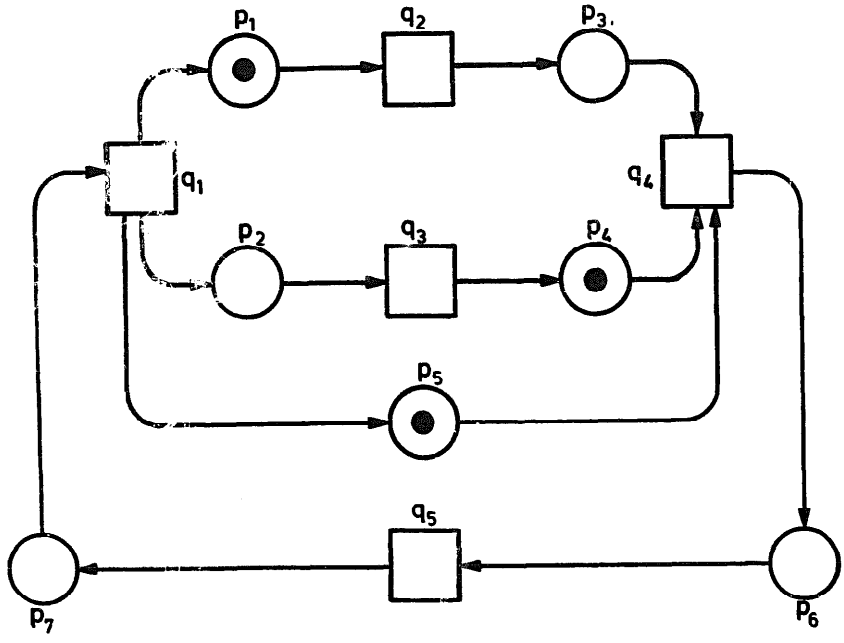


Fig. 16(a)

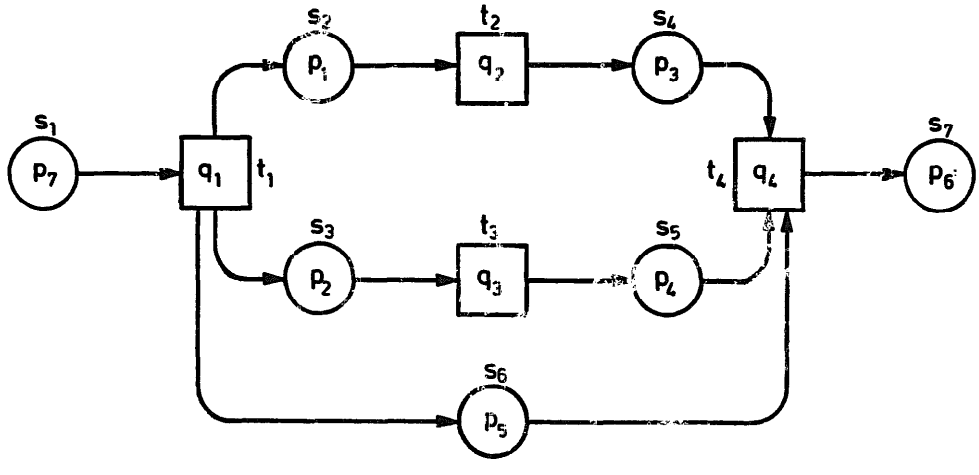


Fig. 16(b)

Proof. Assume that neither $(t_1, t_2) \in F^+$ nor $(t_2, t_1) \in F^+$ and assume that $p \in {}^\bullet q_1{}^\bullet \cap {}^\bullet q_2{}^\bullet$. Then there are $s_1, s_2 \in S$ such that $s_1 \in {}^\bullet t_1{}^\bullet$, $s_2 \in {}^\bullet t_2{}^\bullet$, $\phi(s_1) = p$, and $\phi(s_2) = p$. If either $s_1 = s_2$ or $(s_1, s_2) \in F^+$ or $(s_2, s_1) \in F^+$, then, since, for every $s \in S$, $\#({}^\bullet s) \leq 1$ and $\#(s^\bullet) \leq 1$, $(t_1, t_2) \in F^+$ or $(t_2, t_1) \in F^+$, which contradicts our assumption. Hence s_1 and s_2 are different elements of a slice and thus, since ϕ is injective on slices, $\phi(s_1) \neq \phi(s_2)$. However, this contradicts the facts that $\phi(s_1) = p$ and $\phi(s_2) = p$. Consequently, either $(t_1, t_2) \in F^+$ or $(t_2, t_1) \in F^+$. □

Often in recording the dynamic behavior of a C/E system, one is interested only in occurrences of events (*without* the associated condition holdings). This leads us to the following definition.

5.15. Definition. Let $M = (P, Q, R, \mathscr{C})$ be a contact-free C/E system. A node-labeled directed graph γ is a (*reduced, transitive*) *contracted process of* M if there exists a process $K = (S, T, F, \Sigma, \phi)$ of M such that γ is the (reduced, transitive, respectively) S-contracted version of K.

For a contact-free C/E system M, $P(M)$ will denote its set of processes and $CP(M)$ ($RCP(M)$, $TCP(M)$) will denote its set of (reduced, transitive, respectively) contracted processes.

5.16. Example. Consider the contact-free C/E system M depicted in Fig. 16(a). Then the graph depicted in Fig. 17(a) is both a contracted process and transitive contracted process of M, while the graph depicted in Fig. 17(b) is a reduced contracted process of M.

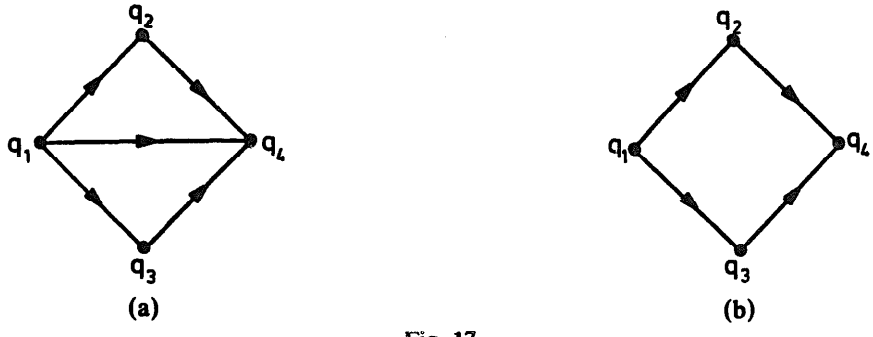


Fig. 17

We now turn to a basic result, which relates the (firing sequence) language of a C/E system to the 'firing sequence languages' of all its processes.

First we need the definition of the language of a labeled occurrence net, followed by an observation on the defined notion.

5.17. Definition. Let $K = (S, T, F, \Sigma, \phi)$ be a labeled occurrence net with $n = \#T$. The (*firing sequence*) *language of* K, denoted by $L(K)$, is the set

$\{x \in \Sigma^n$: there exist sets $S_0, \ldots, S_n$ of S-elements of K and a sequence $t_1, \ldots, t_n$ without repetitions of all T-elements of K such that $S_0 = \{s \in S : {}^\bullet s = \emptyset\}$, $S_n = \{s \in S : s^\bullet = \emptyset\}$, and, for every $1 \leq i \leq n$, $S_{i-1}[t_i\rangle S_i$ and $\phi(t_i) = x(i)\}$.

The following result expresses the basic property of the sets $S_0, \ldots, S_n$ of S-elements from the above definition.

5.18. Theorem. *Let $K = (S, T, F)$ be an occurrence net with $n = \#T$, let $S_0, \ldots, S_n$ be sets of S-elements of K, and let $t_1, \ldots, t_n$ be a sequence without repetitions of all T-elements of K such that $S_0 = \{s \in S : {}^\bullet s = \emptyset\}$, $S_n = \{s \in S : s^\bullet = \emptyset\}$, and, for every $1 \leq i \leq n$, $S_{i-1}[t_i\rangle S_i$. Then, for every $0 \leq i \leq n$, S_i is a slice of K.*

Proof. Observe first that, for every $0 \leq i \leq n-1$ and $s_1, s_2 \in S$, $s_1 \in S_i - S_{i+1}$ and $s_2 \in S_{i+1} - S_i$ imply that $(s_1, t_{i+1}) \in F$ and $(t_{i+1}, s_2) \in F$. This fact will be extensively used in the proof.

Clearly, S_0 is a slice of K. We will now show that, for every $0 \leq i \leq n-1$, S_i is a slice of K implies that S_{i+1} is also a slice of K. From this the theorem follows.

Assume that, for some $0 \leq i \leq n-1$, S_i is a slice of K.

Let $s_1, s_2 \in S_{i+1}$. If $s_1, s_2 \in S_i$, then, since S_i is a slice of K, $(s_1, s_2) \notin F^+$. If $s_1 \in S_i$ and $s_2 \notin S_i$, then $(s_1, s_2) \in F^+$ implies that there exists an $s_3 \in {}^\bullet t_{i+1} \cap S_i$ such that $(s_1, s_3) \in F^+$; this contradicts the fact that S_i is a slice of K and hence, $(s_1, s_2) \notin F^+$. If $s_1 \notin S_i$ and $s_2 \in S_i$, then $(s_1, s_2) \in F^+$ implies that there exists an $s_3 \in {}^\bullet t_{i+1} \cap S_i$ such that $(s_3, s_2) \in F^+$; this again contradicts the fact that S_i is a slice of K and hence, $(s_1, s_2) \notin F^+$. Finally, if $s_1, s_2 \notin S_i$, then $(s_1, s_2) \in F^+$ implies that either $(s_1, s_1) \in F^+$ or $\#({}^\bullet s_2) \geq 2$; both contradict the fact that K is an occurrence net and hence $(s_1, s_2) \notin F^+$. Consequently, for every $s_1, s_2 \in S_{i+1}$, $(s_1, s_2) \notin F^+$.

Let $s_3 \in S - S_{i+1}$. If $s_3 \in S_i$, then, clearly, there exists an $s_4 \in (t_{i+1})^\bullet \cap S_{i+1}$ such that $(s_3, s_4) \in F^+$. If $s_3 \in S - S_i$, then, since S_i is a slice of K, there exists $s_5 \in S_i$ such that either $(s_3, s_5) \in F^+$ or $(s_5, s_3) \in F^+$. In the case that $s_5 \in S_{i+1}$, clearly, there exists an $s_4 = s_5 \in S_{i+1}$ such that either $(s_3, s_4) \in F^+$ or $(s_4, s_3) \in F^+$. In the case that $s_5 \notin S_{i+1}$, $(s_3, s_5) \in F^+$ implies that there exists an $s_4 \in (t_{i+1})^\bullet \cap S_{i+1}$ such that $(s_3, s_4) \in F^+$ and $(s_5, s_3) \in F^+$ implies that there exists an $s_4 \in (t_{i+1})^\bullet \cap S_{i+1}$ such that $(s_4, s_3) \in F^+$. Consequently, for every $s_3 \in S - S_{i+1}$, there exists an $s_4 \in S_{i+1}$ such that either $(s_3, s_4) \in F^+$ or $(s_4, s_3) \in F^+$.

From the above it follows that S_{i+1} is also a slice of K, which concludes the proof of the theorem. □

Now we are ready to relate the language of a C/E system to the languages of all its processes.

5.19. Theorem. *For every contact-free C/E system M,*

$$L(M) = \bigcup \{L(K) : K \in P(M)\}.$$

Analogous results for (reduced, transitive) contracted processes can also be obtained. For this purpose, we first express the relationship between the languages of the processes of a contact-free C/E system M and the languages of the different types of contracted processes of M.

5.20. Theorem. (1) *Let M be a contact-free C/E system, let K be a process of M, and let γ be the S-contracted version of K. Then $L(K) = L(\gamma)$.*

(2) *Let M be a contact-free C/E system, let K be a process of M, and let γ be the reduced (transitive) S-contracted version of K. Then $L(K) = L(\gamma)$.*

Proof. (1): Let $M = (P, Q, R, \mathscr{C})$, $K = (S, T, F, P \cup Q, \phi)$, and $\gamma = (T', F', Q', \phi')$. Note now that, since K is a labeled occurrence net,

(a) K is acyclic and

(b) for every $s \in S$, $\#({}^\bullet s) \leq 1$ and $\#(s^\bullet) \leq 1$;

these facts will be extensively used in the proof. Furthermore, note that $T' = T$, $F' = \{(t', t'') \in T' : \text{there exists an } s \in S \text{ such that } (t', s) \in F \text{ and } (s, t'') \in F\}$, $Q' = Q$, and $\phi' = \phi|_{T'}$. The proof now continues in two steps, each of them proving one part of the equality.

(i) Let $x \in L(K)$ and $n = \#T$. Then there exist sets $S_0, \ldots, S_n$ of S-elements of K and a sequence $t_1, \ldots, t_n$ without repetitions of all T-elements of K such that $S_0 = \{s \in S : {}^\bullet s = \emptyset\}$, $S_n = \{s \in S : s^\bullet = \emptyset\}$, and, for every $1 \leq i \leq n$, $S_{i-1}[t_i\rangle S_i$ and $\phi(t_i) = x(i)$. We will now show that, for every $1 \leq i, j \leq n$, $(t_i, t_j) \in F'$ implies that $i < j$. From this it then follows that $x \in L(\gamma)$.

Let $1 \leq i, j \leq n$ be such that $(t_i, t_j) \in F'$. Hence, there exists an $s \in S$ such that $(t_i, s) \in F$ and $(s, t_j) \in F$; thus, $s \in (t_i)^\bullet \cap {}^\bullet(t_j)$. If $i > j$, then, by (b) above, for every $0 \leq k \leq j-1$, $s \in S_k$; thus, $i > j$ implies that ${}^\bullet s = \emptyset$, which yields a contradiction. Consequently, since $i = j$ contradicts (a) above, we have $i < j$.

(ii) Let $x \in L(\gamma)$ and $n = \#T'$. Then there exists a sequence $t_1, \ldots, t_n$ without repetitions of all elements of T' such that, for every $1 \leq i \leq n$, $\phi'(t_i) = x(i)$ and, for every $1 \leq i, j \leq n$, $(t_i, t_j) \in F'$ implies that $i < j$. Let $S_0 = \{s \in S : {}^\bullet s = \emptyset\}$ and let, for every $1 \leq i \leq n$, $S_i = (S_{i-1} - {}^\bullet(t_i)) \cup (t_i)^\bullet$. We will now show that, for every $1 \leq i \leq n$, $S_{i-1}[t_i\rangle S_i$ and that $S_n = \{s \in S : s^\bullet = \emptyset\}$. From this it then follows that $x \in L(K)$.

Let $1 \leq i \leq n$ and let $s \in {}^\bullet(t_i)$. If ${}^\bullet s = \emptyset$, then $s \in S_0$ and thus, by (b) above, for every $0 \leq k \leq i-1$, $s \in S_k$. If ${}^\bullet s \neq \emptyset$, then, by (a) and (b) above and by the given property of the sequence $t_1, \ldots, t_n$, there exists $1 \leq j \leq i-1$ such that ${}^\bullet s = \{t_j\}$ and thus, again by (b) above, for every $j \leq k \leq i-1$, $s \in S_k$. Hence, for every $1 \leq i \leq n$, $s \in {}^\bullet(t_i)$ implies that $s \in S_{i-1}$. Let $1 \leq i \leq n$ and let $s \in (t_i)^\bullet$. This implies that $s \notin S_0$ and thus, by (b) above, for every $0 \leq k \leq i-1$, $s \notin S_k$. Hence, for every $1 \leq i \leq n$, $s \in (t_i)^\bullet$ implies that $s \notin S_{i-1}$. Consequently, for every $1 \leq i \leq n$, it follows from the construction of S_i that $S_{i-1}[t_i\rangle S_i$.

In order to prove that $S_n = \{s \in S : s^\bullet = \emptyset\}$, first assume that there exist $s \in S_n$ and $1 \leq i \leq n$ such that $(s, t_i) \in F$. This then implies that $s \notin S_i$ and thus there exists $i+1 \leq j \leq n$ such that $(t_j, s) \in F$, which contradicts the given property of the sequence $t_1, \ldots, t_n$. Hence, $S_n \subseteq \{s \in S : s^\bullet = \emptyset\}$. To prove the inverse inclusion, assume that there exist $s \in S - S_n$ such that $s^\bullet = \emptyset$. If ${}^\bullet s = \emptyset$, then $s \in S_0$ and thus, for every $0 \leq k \leq n$, $s \in S_k$, which contradicts the fact that $s \in S - S_n$. If ${}^\bullet s \neq \emptyset$, then there exists $1 \leq i \leq n$ such that $(t_i, s) \in F$ and thus, by (b) above, for every $i \leq k \leq n$, $s \in S_k$, which contradicts the fact that $s \in S - S_n$. Hence, $\{s \in S : s^\bullet = \emptyset\} \subseteq S_n$.

(2) directly follows from (1) above and from the graph-theoretical observation that, for a directed node-labeled graph $g = (V, E, \Sigma, l)$ and for $v_1, v_2 \in V$, g contains a path of length at least 2 from v_1 to v_2 if and only if red(g) (trans(g) respectively) contains a path of length at least 2 from v_1 to v_2. □

Now we are ready to provide the analogy of Theorem 5.19 in terms of (reduced, transitive) contracted processes.

5.21. Theorem. (1) *For every contact-free C/E system M,*

$$L(M) = \bigcup \{L(\gamma) : \gamma \in \mathrm{CP}(M)\}.$$

(2) *For every contact-free C/E system M,*

$$L(M) = \bigcup \{L(\gamma) : \gamma \in \mathrm{RCP}(M)\} = \bigcup \{L(\gamma) : \gamma \in \mathrm{TCP}(M)\}.$$

Proof. This directly follows from Theorems 5.19 and 5.20. □

We conclude this subsection by providing the following notion, which is crucial in our considerations concerning traces and nets.

5.22. Definition. Let $N = (S, T, F)$ be a net. The *reliance alphabet induced by N*, denoted by $C(N)$, equals (T, I, D), where

$$I = \{(t_1, t_2) \in T \times T : {}^\bullet t_1{}^\bullet \cap {}^\bullet t_2{}^\bullet = \emptyset\} \quad \text{and}$$

$$D = \{(t_1, t_2) \in T \times T : {}^\bullet t_1{}^\bullet \cap {}^\bullet t_2{}^\bullet \neq \emptyset\}.$$

5.23. Remark. As usual, the notion of a reliance alphabet induced by a net carries over to a C/E structure and a C/E system (through their underlying nets).

5.24. Example. Consider the C/E system $M = (P, Q, R, \mathscr{C})$ depicted in Fig. 18. Then $C(M) = (Q, I, D)$, where

$$I = \{\{q_1, q_4\}, \{q_1, q_5\}, \{q_1, q_6\}, \{q_2, q_3\}, \{q_2, q_4\}, \{q_3, q_5\},$$

$$\{q_3, q_6\}, \{q_4, q_6\}\}$$

and

$$D = \{\{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_5\}, \{q_2, q_6\}, \{q_3, q_4\},$$

$$\{q_4, q_5\}, \{q_5, q_6\}\} \cup \mathrm{id}(Q).$$

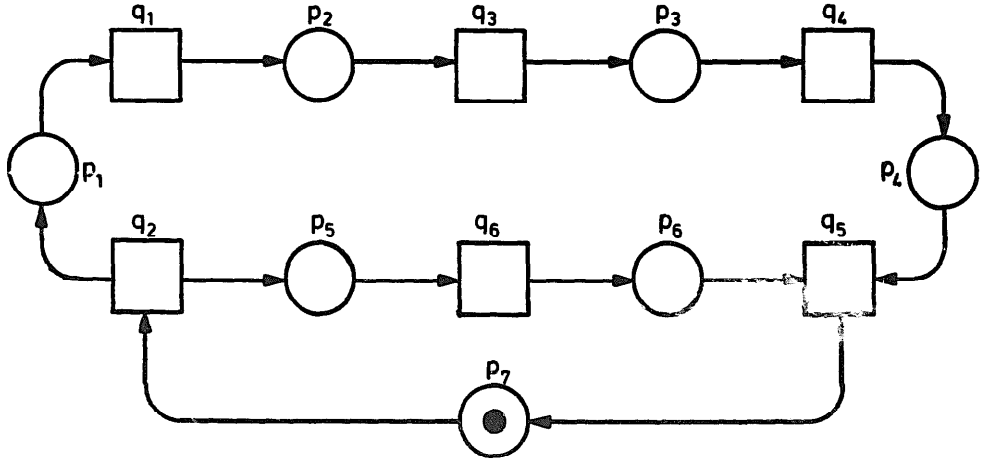


Fig. 18

5.2. Decomposing C/E structures using traces

In this subsection a method is presented for finding the trace language associated to a C/E structure; this method makes a considerable use of the theory of traces.

The following notions on traces and trace languages are needed in order to deal with different reliance alphabets (induced by different C/E structures).

5.25. Definition. Let $C' = (\Sigma', I', D')$ and $C'' = (\Sigma'', I'', D'')$ be two reliance alphabets.

(1) The *projection of* C' *onto* C'', denoted by $C' | C''$, is the reliance alphabet $C = (\Sigma, I, D)$ such that $\Sigma = \Sigma' \cap \Sigma''$ and $D = D' \cap D''$.

(2) The *synchronization of* C' *and* C'', denoted by $C' \| C''$, is the reliance alphabet $C''' = (\Sigma''', I''', D''')$ such that $\Sigma''' = \Sigma' \cup \Sigma''$ and $D''' = D' \cup D''$.

Hence, the projection of a reliance alphabet onto another yields a reliance alphabet with a *smaller* symbol alphabet (the intersection of both alphabets), where the inherited dependence relation is the *intersection* of both dependence relations, while the synchronization of two reliance alphabets yields a reliance alphabet with a *bigger* symbol alphabet (the union of both alphabets), where the inherited dependence relation is the *union* of both dependence relations.

5.26. Example. For the reliance alphabets C' and C'', such that $D(C')$ and $D(C'')$ are as depicted in Fig. 19(a) and Fig. 19(b), respectively; $D(C' | C'')$ and $D(C' \| C'')$ are depicted in Fig. 19(c) and Fig. 19(d), respectively.

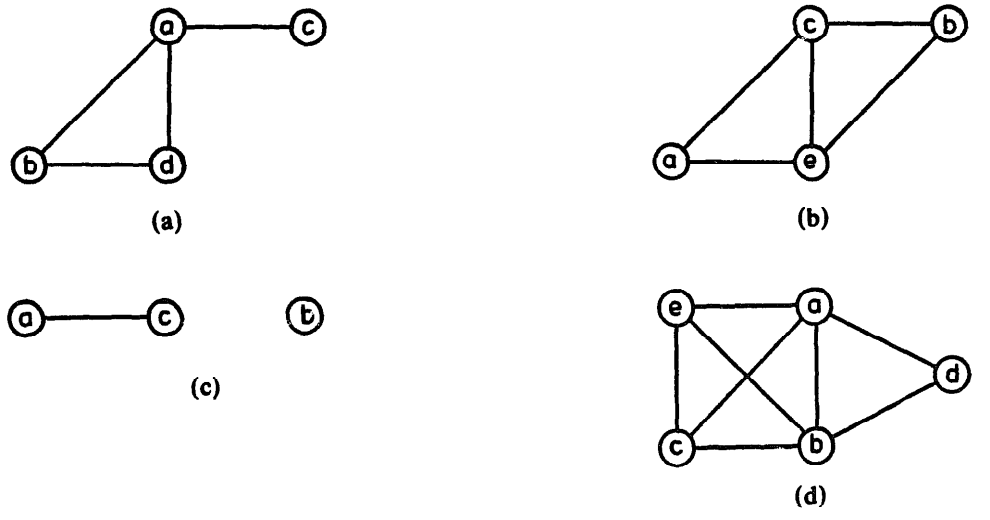


Fig. 19

The operations 'projection' and 'synchronization' are dual in the following sense.

5.27. Theorem. (1) *Let* $C' = (\Sigma', I', D')$, $C'' = (\Sigma'', I'', D'')$, *and* $C''' = (\Sigma''', I''', D''')$ *be reliance alphabets such that* $C''' = C' \| C''$. *Then* $C' = C''' | C'$ *and* $C'' = C''' | C''$.

(2) *Let* $C=(\Sigma, I, D)$, $C'=(\Sigma', I', D')$, *and* $C''=(\Sigma'', I'', D'')$ *be reliance alphabets such that* $C=C'|C''$. *Then* $C'=C\|C'$ *and* $C''=C\|C''$.

Proof. (1) Let $\hat{C}=(\hat{\Sigma}, \hat{I}, \hat{D})$ be the reliance alphabet such that $\hat{C}=C'''|C'$. Then

$$\hat{\Sigma}=\Sigma'''\cap\Sigma'=(\Sigma'\cup\Sigma'')\cap\Sigma'=\Sigma' \quad \text{and}$$

$$\hat{D}=D'''\cap D'=(D'\cup D'')\cap D'=D'.$$

Thus $\hat{\Sigma}=\Sigma'$ and $\hat{D}=D'$, and, consequently, $C'=C'''|C'$.

Analogously it can be be shown that $C''=C'''|C''$.

(2) Let $\hat{C}=(\hat{\Sigma}, \hat{I}, \hat{D})$ be the reliance alphabet such that $\hat{C}=C\|C'$. Then

$$\hat{\Sigma}=\Sigma\cup\Sigma'=(\Sigma'\cap\Sigma'')\cup\Sigma'=\Sigma' \quad \text{and}$$

$$\hat{D}=D\cup D'=(D'\cap D'')\cup D'=D'.$$

Thus $\hat{\Sigma}=\Sigma'$ and $\hat{D}=D'$, and, consequently, $C'=C\|C'$.

Analogously it can be shown that $C''=C\|C''$. □

In order to extend the operation of projection to traces, we need the following result.

5.28. Theorem. *Let* $C=(\Sigma, I, D)$, $C'=(\Sigma', I', D')$, *and* $C''=(\Sigma'', I'', D'')$ *be reliance alphabets such that* $C=C'|C''$. *Let* $x', y'\in(\Sigma')^*$ *be such that* $x'\equiv_{C'} y'$. *Then* $\phi_{\Sigma''}(x')\equiv_C \phi_{\Sigma''}(y')$.

Proof. The proof goes in two steps, each step proving one requirement of the sufficient condition from the statement of Theorem 2.6.

Let $a\in\Sigma$. Thus, since $\Sigma=\Sigma'\cap\Sigma''$, $a\in\Sigma''$. Hence,

$$\#_a(\phi_{\Sigma''}(x'))=\#_a(x') \quad \text{and} \quad \#_a(\phi_{\Sigma''}(y'))=\#_a(y').$$

Furthermore, once again since $\Sigma=\Sigma'\cap\Sigma''$, $a\in\Sigma'$. Hence, by Theorem 2.6, $\#_a(x')=\#_a(y')$. Consequently, $\#_a(\phi_{\Sigma''}(x'))=\#_a(\phi_{\Sigma''}(y'))$.

Let $\{a, b\}\in D$. Thus, since $D=D'\cap D''$, $\{a, b\}\in D''$ and moreover, $\{a, b\}\subseteq\Sigma''$. Hence,

$$\phi_{\{a,b\}}(\phi_{\Sigma''}(x'))=\phi_{\{a,b\}}(x') \quad \text{and} \quad \phi_{\{a,b\}}(\phi_{\Sigma''}(y'))=\phi_{\{a,b\}}(y').$$

Furthermore, once again since $D=D'\cap D''$, $\{a, b\}\in D'$. Hence, by Theorem 2.6, $\phi_{\{a,b\}}(x')=\phi_{\{a,b\}}(y')$. Consequently, $\phi_{\{a,b\}}(\phi_{\Sigma''}(x'))=\phi_{\{a,b\}}(\phi_{\Sigma''}(y'))$. From Theorem 2.6 and the above, it now follows directly that $\phi_{\Sigma''}(x')\equiv_C\phi_{\Sigma''}(y')$. □

The above theorem enables us to define the operation of projection of a trace t through an arbitrary representative of t.

5.29. Definition. Let $C'=(\Sigma', I', D')$ and $C''=(\Sigma'', I'', D'')$ be two reliance alphabets. Let $t'\in\Theta(C')$. The *projection of* t' *onto* C'', denoted by $t'|C''$, is the trace $[\phi_{\Sigma''}(x')]_C$, where $x'\in(\Sigma')^*$ is such that $t'=[x']_{C'}$ and $C=C'|C''$.

Intuitively speaking, the projection of a trace t' (over a reliance alphabet $C'=(\Sigma', I', D')$) onto a reliance alphabet $C''=(\Sigma'', I'', D'')$, results by erasing all symbols in t' that are not in Σ'' and then using only those dependences from D' that are also in D''.

5.30. Example. Let C' and C'' be as in Example 5.26. Let $t'=[acbcd]_{C'}$ and let $C=C'|C''$. Then $t'\in\Theta(C')$ and a dep-graph of t' is depicted in Fig. 20(a). Furthermore, $(t'|C'')=[acbc]_C$ and a dep-graph of $t'|C''$ is depicted in Fig. 20(b).

Fig. 20

In order to extend the operation of synchronization to traces, we need the following results.

5.31. Theorem. *Let* $C'=(\Sigma', I', D')$ *and* $C''=(\Sigma'', I'', D'')$ *be two reliance alphabets.*

(1) *Let* $C'''=C'\|C''$ *and let* $t'''\in\Theta(C''')$. *Then* $(t'''|C')\in\Theta(C')$ *and* $(t'''|C'')\in\Theta(C'')$.

(2) *Let* $C=(\Sigma, I, D)=C'|C''$ *and let* $x'\in(\Sigma')^*$ *and* $x\in\Sigma^*$ *be such that* $([x']_{C'}|C'')=[x]_C$. *Then*

(i) *for every* $a\in\Sigma$, $\#_a(x')=\#_a(x)$; *and*

(ii) *for every* $\{a, b\}\in D$, $\phi_{\{a,b\}}(x')=\phi_{\{a,b\}}(x)$.

(3) *Let* $C'''=(\Sigma''', I''', D''')=C'\|C''$ *and let* $t'''_1, t'''_2\in\Theta(C''')$ *be such that* $(t'''_1|C')=(t'''_2|C')$ *and* $(t'''_1|C'')=(t'''_2|C'')$. *Then* $t'''_1=t'''_2$.

Proof. (1) directly follows from Definition 5.29 and Theorem 5.27.

(2): From Definition 5.29 it follows that $([x']_{C'}|C'')=[\phi_{\Sigma''}(x')]_C$, and thus, $[\phi_{\Sigma''}(x')]_C=[x]_C$.

Let $a\in\Sigma$. Then, by Theorem 2.6, $\#_a(\phi_{\Sigma''}(x'))=\#_a(x)$. Consequently, since $\Sigma=\Sigma'\cap\Sigma''$ implies that $a\in\Sigma''$, $\#_a(x')=\#_a(x)$.

Let $\{a, b\} \in D$. Then, by Theorem 2.6, $\phi_{\{a,b\}}(\phi_{\Sigma''}(x')) = \phi_{\{a,b\}}(x)$. Consequently, since $D = D' \cap D''$ implies that $\{a, b\} \in \Sigma'$, $\phi_{\{a,b\}}(x') = \phi_{\{a,b\}}(x)$.

(3): Let $x_1''', x_2''' \in (\Sigma''')^*$, $x' \in (\Sigma')^*$ and $x'' \in (\Sigma'')^*$ be such that

$$t_1''' = [x_1''']_{C'''}, \qquad t_2''' = [x_2''']_{C'''},$$

$$(t_1''' \mid C') = [x']_{C'} = (t_2''' \mid C'), \quad \text{and} \quad (t_1''' \mid C'') = [x'']_{C'} = (t_2''' \mid C'').$$

Now it follows from Theorem 2.6 that, in order to prove $t_1''' = t_2'''$, it suffices to prove that, for every $a \in \Sigma'''$, $\#_a(x_1''') = \#_a(x_2''')$, and, for every $\{a, b\} \in D'''$, $\phi_{\{a,b\}}(x_1''') = \phi_{\{a,b\}}(x_2''')$.

Let $a \in \Sigma'''$. Then, since $\Sigma''' = \Sigma' \cup \Sigma''$, either $a \in \Sigma'$ or $a \in \Sigma''$. In the first case, (2) above implies that $\#_a(x_1''') = \#_a(x') = \#_a(x_2''')$, and in the second case, (2) above implies that $\#_a(x_1''') = \#_a(x'') = \#_a(x_2''')$. Consequently, $\#_a(x_1''') = \#_a(x_2''')$.

Let $\{a, b\} \in D'''$. Then, since $D''' = D' \cup D''$, either $\{a, b\} \in D'$ or $\{a, b\} \in D''$. In the first case, (2) above implies that

$$\phi_{\{a,b\}}(x_1''') = \phi_{\{a,b\}}(x') = \phi_{\{a,b\}}(x_2'''),$$

and in the second case, (2) above implies that

$$\phi_{\{a,b\}}(x_1''') = \phi_{\{a,b\}}(x'') = \phi_{\{a,b\}}(x_2''').$$

Consequently, $\phi_{\{a,b\}}(x_1''') = \phi_{\{a,b\}}(x_2''')$. □

The above theorem enables us to define the operation of synchronization of two traces as follows (Theorem 5.31(3) guarantees the uniqueness of the obtained trace).

5.32. Definition. Let $C' = (\Sigma', I', D')$ and $C'' = (\Sigma'', I'', D'')$ be two reliance alphabets.

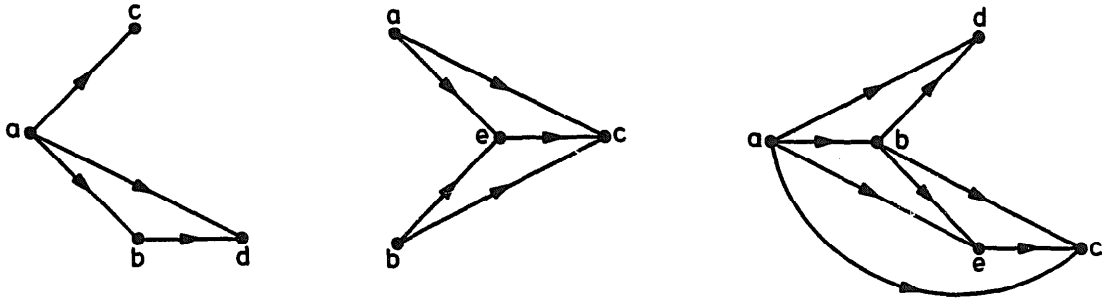
(1) Let $t' \in \Theta(C')$, $t'' \in \Theta(C'')$, $C''' = C' \| C''$, and $t''' \in \Theta(C''')$ be such that $(t''' \mid C') = t'$ and $(t''' \mid C'') = t''$. Then t''' is called the *synchronization of* t' *and* t'', and is denoted by $t' \| t''$.

(2) Let $T' \subseteq \Theta(C')$ and $T'' \subseteq \Theta(C'')$. The *synchronization of* T' *and* T'', denoted by $T' \| T''$, is the trace language $T''' \subseteq \Theta(C''')$, where $C''' = C' \| C''$, such that

$$T''' = \{t''' \in \Theta(C''') : \text{there exist } t' \in T' \text{ and } t'' \in T'' \text{ with } t''' = t' \| t''\}.$$

Intuitively speaking, the synchronization of a trace t' (over a reliance alphabet C') and a trace t'' (over a reliance alphabet C''), is the trace t''' (over the reliance alphabet $C' \| C''$), which yields t' when projected on C' and which yields t'' when projected on C''.

5.33. Example. Let C' and C'' be as in Example 5.26. Let $t' = [abdc]_{C'}$ and $t'' = [baec]_{C''}$; let $C''' = C' \| C''$. Then a dep-graph of t' is depicted in Fig. 21(a) and a dep-graph of t'' is depicted in Fig. 21(b). Furthermore, $(t' \| t'') = [abecd]_{C'''}$ and the graph depicted in Fig. 21(c) is a dep-graph of $t' \| t''$.



(a) (b) (c)

Fig. 21

The following result provides a valuable insight into the nature of the operation of synchronization.

5.34. Theorem. *Let* $C' = (\Sigma', I', D')$ *and* $C'' = (\Sigma'', I'', D'')$ *be two reliance alphabets and let* $C''' = C' \| C''$. *Let* $x' \in (\Sigma')^*$, $a' \in \Sigma'$, *and* $x'' \in (\Sigma'')^*$.

(1) $([\lambda]_{C'} \| [\lambda]_{C''}) = [\lambda]_{C'''}$.

(2) *If* $([x']_{C'} | C'') = ([x'']_{C''} | C')$, *then* $([x']_{C'} \| [x'']_{C''}) = ([x'']_{C''} \| [x']_{C'})$.

(3) *If* $a' \notin \Sigma''$ *and* $([a'x']_{C'} | C'') = ([x'']_{C''} | C')$, *then* $([x']_{C'} | C'') = ([x'']_{C''} | C')$ *and*

$$([a'x']_{C'} \| [x'']_{C''}) = [a']_{C'''} \circ ([x']_{C'} \| [x'']_{C''}).$$

(4) *If* $a' \in \Sigma''$ *and* $([a'x']_{C'} | C'') = ([a'x'']_{C''} | C')$, *then* $([x']_{C'} | C'') = ([x'']_{C''} | C')$ *and*

$$([a'x']_{C'} \| [a'x'']_{C''}) = [a']_{C'''} \circ ([x']_{C'} \| [x'']_{C''}).$$

In order to obtain the main result of this subsection, we need the following notion on decomposing C/E structures.

5.35. Definition. Let $N = (P, Q, R)$, $N' = (P', Q', R')$, and $N'' = (P'', Q'', R'')$ be C/E structures. N is the *composition of* N' *and* N'' (N is *decomposed into* N' *and* N''), denoted by $N = N' \| N''$, if $P = P' \cup P''$, $P' \cap P'' = \emptyset$, $Q = Q' \cup Q''$, and $R = R' \cup R''$.

5.36. Example. The C/E structures N', N'', N''', and N, depicted in Figs. 22, 23, 24, and 25, respectively, are such that $N = (N' \| N'') \| N'''$.

The main result of this subsection is the following, which shows that the trace languages associated to a C/E structure can be built up from the trace languages associated with its parts.

5.37. Theorem. *Let* $N = (P, Q, R)$ *be a C/E structure, let* $C(N) = (Q, I, D)$, *and let* $\mu_1, \mu_2 \subseteq P$. *Let* $N' = (P', Q', R')$ *and* $N'' = (P'', Q'', R'')$ *be two C/E structures such that* $N = N' \| N''$ *and let* $\mu_1' = \mu_1 \cap P'$, $\mu_1'' = \mu_1 \cap P''$, $\mu_2' = \mu_2 \cap P'$, *and* $\mu_2'' = \mu_2 \cap P''$.

(1) $[L(\mu_1, \mu_2, N)]^{\exists}_{C(N)} = [L(\mu_1', \mu_2', N')]^{\exists}_{C(N')} \| [L(\mu_1'', \mu_2'', N'')]^{\exists}_{C(N'')}$.

(2) $[L(\mu_1, N)]^{\exists}_{C(N)} = [L(\mu_1', N')]^{\exists}_{C(N')} \| [L(\mu_1'', N'')]^{\exists}_{C(N'')}$.

(3) $[L(N)]^{\exists}_{C(N)} = [L(N')]^{\exists}_{C(N')} \| [L(N'')]^{\exists}_{C(N'')}$.

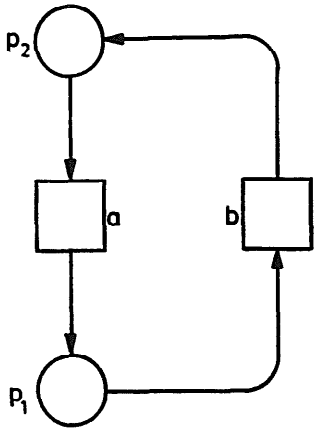


Fig. 22

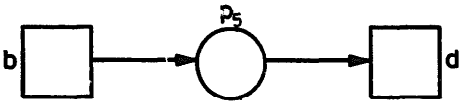


Fig. 23

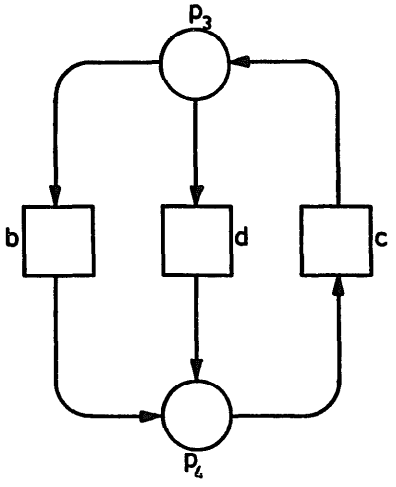


Fig. 24

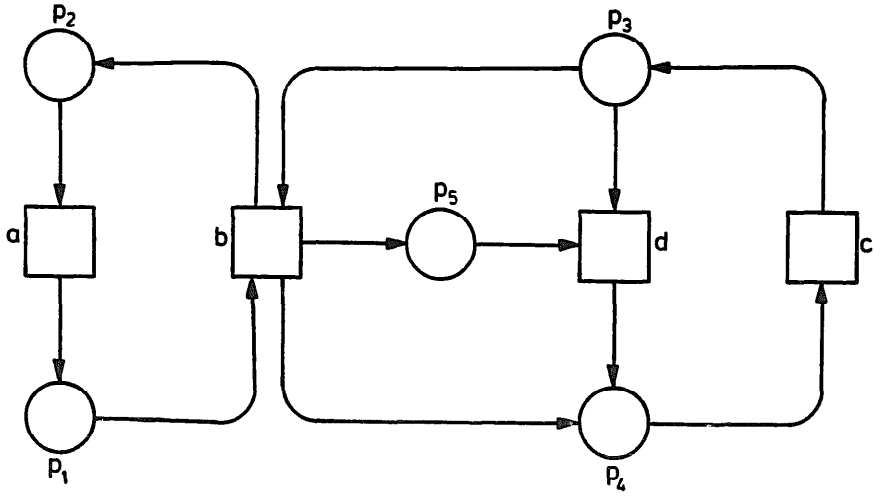


Fig. 25

In this way, a C/E structure can be decomposed into smaller ones, for which the associated trace languages are either known or easy to find. Then one finds the trace languages of the given C/E structure by synchronizing these smaller trace languages. In particular, decomposing the given C/E structure into atoms (i.e., C/E structures containing one condition only) yields simple trace languages.

5.38. Example. Let N', N'', N''', and N as in Example 5.36. Then, $L(\{p_1\}, \{p_1\}, N') = \{ba\}^*$, $L(\emptyset, \emptyset, N'') = \{bd\}^*$, and $L(\{p_3\}, \{p_3\}, N''') = \{bc, dc\}^*$. Hence, by the theorem above,

$$\begin{aligned}&[L(\{p_1, p_3\}, \{p_1, p_3\}, N)]^{\exists}_{C(N)}\\ &\quad = ([\{ba\}^*]^{\exists}_{C(N')} \| [\{bd\}^*]^{\exists}_{C(N'')}) \| [\{bc, dc\}^*]^{\exists}_{C(N''')}\\ &\quad = [\{bacdc\}^*]^{\exists}_{C(N)}.\end{aligned}$$

5.5. Traces, dep-graphs, and processes of C/E systems

In this subsection we relate the theory of traces and the theory of dep-graphs to the theory of (processes of) C/E systems. We start by observing that considering C/E systems as generators of trace languages fits well in our framework.

5.39. Theorem. *Let M be a C/E system.*

(1) *$L(M)$ is $C(M)$-consistent.*

(2) $[L(M)]_{C(M)} \in T_{C(M)}(\mathrm{REG})$.

Proof. (2) directly follows from (1) above and from Theorem 5.8(2). □

Next we consider C/E systems as generators of directed node-labeled graphs. This point of view brings (the behavior of) C/E systems and dep-graph languages close together.

We begin with the following observation.

5.40. Theorem. *There exists a contact-free C/E system M such that* $\mathrm{CP}(M) \neq \langle L(M)\rangle_{C(M)}$.

Proof. Consider the C/E system M which is depicted in Fig. 26(Then every contracted process of M is of the form which is depicted in Fig. 26(b) (where, for the sake of convenience, the labels of the nodes are omitted), while every element of $\langle L(M)\rangle_{C(M)}$ is of the form depicted in Fig. 26(c) (where, again for the sake of convenience, the labels of the nodes are omitted).

Clearly, M satisfies the statement of the theorem. □

To the contrary, if we turn to *reduced* and *transitive* contracted processes and *reduced* and *transitive* dep-graphs, then the two approaches coincide (yield the same result).

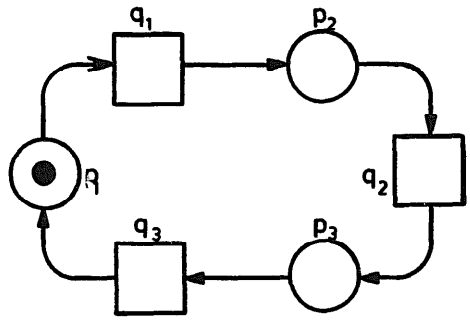
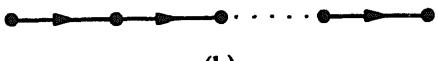
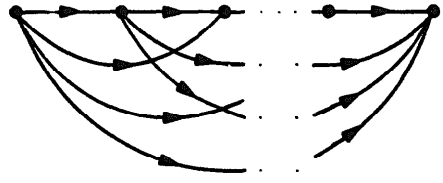


Fig. 26(a)



(b)



(c)

Fig. 26

5.41. Theorem. (1) *For every contact-free C/E system M, for every* $\gamma \in \mathrm{RCP}(M)$ ($\gamma \in \mathrm{TCP}(M)$) *and for every* $x \in L(\gamma)$, $\langle x \rangle^{\mathrm{r}}_{C(M)} \cong \gamma$ ($\langle x \rangle^{\mathrm{t}}_{C(M)} \cong \gamma$ *respectively*).

(2) *For every contact-free C/E system M,* $\mathrm{RCP}(M) = \langle L(M) \rangle^{\mathrm{r}}_{C(M)}$ *and* $\mathrm{TCP}(M) = \langle L(M) \rangle^{\mathrm{t}}_{C(M)}$.

Proof. (1) Let $M = (P, Q, R, \mathscr{C})$ be a contact-free C/E system and let $C(M) = (Q, I, D)$. Let $\gamma = (T, E, Q, l) \in \mathrm{CP}(M)$ with $n = \#T$ and let $x \in L(\gamma)$. Let $K = (S, T, F, P \cup Q, \phi) \in P(M)$ be such that γ is the S-contracted version of K and let $\langle x \rangle_{C(M)} = (V', E', Q, l')$ (note that, by definition, $V' = \{1, \ldots, n\}$).

Observe now the following.

First, there exists a sequence without repetitions $t_1, \ldots, t_n$ of all elements of T such that, for every $1 \leq i \leq n$, $l(t_i) = x(i)$, and, for every $1 \leq i, j \leq n$, $(t_i, t_j) \in E$ implies that $i < j$.

Second, for every $1 \leq i \leq n$, $l'(i) = l(t_i) = \phi(t_i) = x(i)$.

Third, for every $1 \leq i, j \leq n$, $(t_i, t_j) \in E^+$ if and only if $(i, j) \in (E')^+$. This can be seen as follows.

Assume that $1 \leq i, j \leq n$ are such that $(t_i, t_j) \in E^+$. Thus there exist, for some $r \geq 2$, $1 \leq k(1), \ldots, k(r) \leq n$, such that $k(1) = i$, $k(r) = j$, and, for every $2 \leq p \leq r$, $(t_{k(p-1)}, t_{k(p)}) \in E$. Hence,

(i) for every $2 \leq p \leq r$, $k(p-1) < k(p)$; and

(ii) for every $2 \leq p \leq r$, $(t_{k(p-1)}, t_{k(p)}) \in F^2$ and thus ${}^\bullet(t_{k(p-1)})^\bullet \cap {}^\bullet(t_{k(p)})^\bullet \neq \emptyset$, which, by the definition of a process, implies that, for every $2 \leq p \leq r$,

$${}^\bullet(\phi(t_{k(p-1)}))^\bullet \cap {}^\bullet(\phi(t_{k(p)}))^\bullet \neq \emptyset$$

and thus

$$\{l'(k(p-1)), l'(k(p))\} = \{\phi(t_{k(p-1)}), \phi(t_{k(p)})\} \in D.$$

Consequently, for every $2 \leq p \leq r$, $(k(p-1), k(p)) \in E'$, and hence $(i, j) = (k(1), k(r)) \in (E')^+$.

Assume that $1 \leq i, j \leq n$ are such that $(i, j) \in (E')^+$. Thus $i < j$ and there exist, for some $r \geq 2$, $i = k(1) < \cdots < k(r) = j$ such that, for every $2 \leq p \leq r$, $(k(p-1), k(p)) \in E'$. Hence, for every $2 \leq p \leq r$,

$$\{\phi(t_{k(p-1)}), \phi(t_{k(p)})\} = \{l'(k(p-1)), l'(k(p))\} \in D$$

and thus ${}^\bullet(\phi(t_{k(p-1)}))^\bullet \cap {}^\bullet(\phi(t_{k(p)}))^\bullet \neq \emptyset$. It now follows from Theorem 5.14 that, for every $2 \leq p \leq r$, either

$$(t_{k(p-1)}, t_{k(p)}) \in F^+ \quad \text{or} \quad (t_{k(p)}, t_{k(p-1)}) \in F^+,$$

and hence either

$$(t_{k(p-1)}, t_{k(p)}) \in E^+ \quad \text{or} \quad (t_{k(p)}, t_{k(p-1)}) \in E^+.$$

However, since $k(p-1) < k(p)$ for every $2 \leq p \leq r$, the above implies that, for every $2 \leq p \leq r$, $(t_{k(p-1)}, t_{k(p)}) \in E^+$. Consequently, $(t_i, t_j) = (t_{k(1)}, t_{k(r)}) \in E^+$.

From the observations above, it now follows that, for every $1 \leq i, j \leq n$, $l(t_i) = l'(i)$ and γ contains a path from t_i to t_j if and only if $\langle x \rangle_{C(M)}$ contains a path from i to j. Hence, by standard graph-theoretical arguments, $\mathrm{red}(\gamma) \cong \langle x \rangle^{\mathrm{r}}_{C(M)}$ and $\mathrm{trans}(\gamma) \cong \langle x \rangle^{\mathrm{t}}_{C(M)}$.

(2) directly follows from (1) above and from Theorems 5.21(2) and (3) respectively. □

The above theorem yields a direct relationship between (the different types of) dep-graphs of firing sequences of C/E systems and (the different types of) processes of C/E systems. It turns out that this relationship can easily be extended to traces of firing sequences of C/E systems.

5.42. Theorem. (1) *For every contact-free C/E system M, for every $\gamma \in \mathrm{RCP}(M)$ ($\gamma \in \mathrm{TCP}(M)$), and for every $x, y \in L(\gamma)$, $x \equiv_{C(M)} y$.*

(2) *For every contact-free C/E system M, for every $\gamma \in \mathrm{CP}(M)$ ($K \in P(M)$), and for every $x, y \in L(\gamma)$ ($x, y \in L(K)$ respectively) $x \equiv_{C(M)} y$.*

Proof. (1): Let M be a contact-free C/E system, let $\gamma \in \mathrm{RCP}(M)$ ($\gamma \in \mathrm{TCP}(M)$), and let $x, y \in L(\gamma)$. Then, by Theorem 5.41(1),

$$\langle x \rangle^{\mathrm{r}}_{C(M)} \cong \gamma \cong \langle y \rangle^{\mathrm{r}}_{C(M)} \qquad (\langle x \rangle^{\mathrm{t}}_{C(M)} \cong \gamma \cong \langle y \rangle^{\mathrm{t}}_{C(M)} \text{ respectively}).$$

Consequently, by Theorem 2.35, $x \equiv_{C(M)} y$.

(2) directly follows from (1) above and from Theorem 5.20. □

5.43. Theorem. (1) *For every contact-free C/E system M and for every* $\gamma \in \mathrm{RCP}(M)$ $(\gamma \in \mathrm{TCP}(M))$, $L(\gamma) \in \Theta(C(M))$.

(2) *For every contact-free C/E system M and for every* $\gamma \in \mathrm{CP}(M)$ $(K \in P(M))$, $L(\gamma) \in \Theta(C(M))$ $(L(K) \in \Theta(C(M))$ *respectively*).

Proof. (1) Let $M = (P, Q, R, \mathscr{C})$ be a contact-free C/E system and let $\gamma \in \mathrm{RCP}(M)$ $(\gamma \in \mathrm{TCP}(M))$. In order to prove that $L(\gamma) \in \Theta(C(M))$, one has to show that, for every $x, y \in L(\gamma)$, $x \equiv_{C(M)} y$, and that, for every $x \in L(\gamma)$ and $y \in Q^*$, $x \equiv_{C(M)} y$ implies that $y \in L(\gamma)$. Since the first requirement directly follows from Theorem 5.42(1), it suffices to prove the second requirement.

Let $x \in L(\gamma)$ and let $y \in Q^*$ such that $x \equiv_{C(M)} y$. From Theorem 5.21(2) it now follows that $x \in L(M)$, and thus, by Theorem 5.39(1), $y \in L(M)$. Hence, again by Theorem 5.21(2), there exists $\gamma' \in \mathrm{RCP}(M)$ ($\gamma' \in \mathrm{TCP}(M)$ respectively) such that $y \in L(\gamma')$. However, since by Theorems 2.35 and 5.41(1),

$$\gamma \cong \langle x \rangle^{\mathrm{r}}_{C(M)} \cong \langle y \rangle^{\mathrm{r}}_{C(M)} \cong \gamma' \qquad (\gamma \cong \langle x \rangle^{\mathrm{t}}_{C(M)} \cong \langle y \rangle^{\mathrm{t}}_{C(M)} \cong \gamma' \text{ respectively}),$$

$\gamma \cong \gamma'$ and thus $y \in L(\gamma)$.

(2) directly follows from (1) above and from Theorem 5.20. □

5.4. Appendix (proof of Theorem 5.19)

In this appendix we give a proof of Theorem 5.19. For this purpose we need a number of notions and results which are rather standard in the theory of C/E systems and which can be found, e.g., in [41]. To avoid recalling all of these notions and results, we simply refer to the appropriate places in the standard text book [41]. Whenever we make such a reference to [41], we indicate this by appending the letter "R" to the reference index; thus, e.g., "Lemma R 3.4(b)" stands for "Lemma 3.4(b) in [41]".

The notions we need in the proof of the theorem are the following:

(i) for a process K of a C/E system, the set of *minimal elements of* K, denoted by ${}^\circ K$, and the set of *maximal elements of* K, denoted by K° (see Definition R 3.1(i));

(ii) for two processes $K_1 = (S_1, T_1, F_1, \Sigma_1, \phi_1)$ and $K_2 = (S_2, T_2, F_2, \Sigma_2, \phi_2)$ of a C/E system such that $\phi_1((K_1)^\circ) = \phi_2({}^\circ(K_2))$, the *composition of* K_1 *and* K_2, denoted by $K_1 \circ K_2$ (see Definition R 3.4(c));

(iii) the *case graph of* a C/E system M, denoted by Φ_M (see Definition R 2.6(a)); and

(iv) for an edge e in the case graph of a C/E system, the *process of* e (Definition R 3.5(b)).

Now we are ready to prove Theorem 5.19.

5.19. Theorem. *For every contact-free C/E system M,*

$$L(M) = \bigcup \{L(K) : K \in P(M)\}.$$

Proof. Let $M=(P, Q, R, \mathscr{C})$ be a contact-free C/E system.

Let $x \in L(M)$ with $n=|x|$. Hence, there exist $q_1, \ldots, q_n \in Q$ and $\mu_0, \ldots, \mu_n \in \mathscr{C}$ such that $x=q_1 \ldots q_n$ and, for every $1 \leq i \leq n$, $\mu_{i-1}[q_i\rangle\mu_i$; thus, by Definition R 2.6(a), for every $1 \leq i \leq n$, (μ_{i-1}, q_i, μ_i) is an edge in Φ_M. Let, for every $1 \leq i \leq n$, $K_i=(S_i, T_i, F_i, P \cup Q, \phi_i)$ be the process of the edge (μ_{i-1}, q_i, μ_i) in Φ_M; thus,

(i) for every $1 \leq i \leq n$, K_i is a process of M such that the set of T-elements of K_i consists of only one T-element t_i' with $\phi_i(t_i')=q_i$;

(ii) $\phi_1({}^\circ(K_1))=\mu_0$ and $\phi_n((K_n)^\circ)=\mu_n$; and

(iii) for every $1 \leq i \leq n-1$,

$$\phi_i((K_i)^\circ)=\phi_{i+1}({}^\circ(K_{i+1}))=\mu_i.$$

Hence, by Lemma R 3.4(b), there exists a process $K=(S, T, F, P \cup Q, \phi)$ of M isomorphic to $K_1 \circ \cdots \circ K_n$ and slices $S_0, \ldots, S_n$ of K such that

(i) $T=\{t_1, \ldots, t_n\}$, where, for every $1 \leq i, j \leq n$, $i \neq j$ implies that $t_i \neq t_j$;

(ii) $S_0=\{s \in S : {}^\bullet s=\emptyset\}={}^\circ K$ and $S_n=\{s \in S : s^\bullet=\emptyset\}=K^\circ$;

(iii) $\phi(S_0)=\phi_1({}^\circ(K_1))=\mu_0$ and $\phi(S_n)=\phi_n((K_n)^\circ)=\mu_n$;

(iv) for every $1 \leq i \leq n-1$, $\phi(S_i)=\phi_i((K_i)^\circ)=\phi_{i+1}({}^\circ(K_{i+1}))=\mu_i$; and

(v) for every $1 \leq i \leq n$, $\phi(t_i)=\phi_i(t_i')=q_i$ and $S_{i-1}[t_i\rangle S_i$. Consequently, for this process K of M, $x=q_1 \ldots q_n=\phi(t_1) \ldots \phi(t_n) \in L(K)$, which proves the inclusion $L(M) \subseteq \bigcup\{L(K) : K \in P(M)\}$.

Now, let $K=(S, T, F, P \cup Q, \phi)$ be a process of M and let $x \in L(K)$ with $n=|x|$. Hence, there exist sets $S_0, \ldots, S_n$ of S-elements of K and a sequence $t_1, \ldots, t_n$ without repetitions of all T-elements of K, such that $S_0=\{s \in S : {}^\bullet s=\emptyset\}$, $S_n=\{s \in S : s^\bullet=\emptyset\}$, and, for every $1 \leq i \leq n$, $S_{i-1}[t_i\rangle S_i$ and $\phi(t_i)=x(i)$. We will now show that, for every $1 \leq i \leq n$, $\phi(S_{i-1})[\phi(t_i)\rangle\phi(S_i)$. From this it directly follows that $x=\phi(t_1) \ldots \phi(t_n) \in L(M)$, which proves the inclusion $\bigcup\{L(K) : K \in P(M)\} \subseteq L(M)$.

Let $1 \leq i \leq n$. Clearly, $S_{i-1}[t_i\rangle S_i$ implies that ${}^\bullet(t_i) \subseteq S_{i-1}$ and thus, by the definition of a process,

$${}^\bullet(\phi(t_i))=\phi({}^\bullet(t_i)) \subseteq \phi(S_{i-1});$$

hence, since M is contact-free, $(\phi(t_i))^\bullet \subseteq P-\phi(S_{i-1})$. Consequently, $\phi(S_{i-1})[\phi(t_i)\rangle$. Also, $S_{i-1}[t_i\rangle S_i$ implies that $S_i=(S_{i-1}-{}^\bullet(t_i)) \cup (t_i)^\bullet$ and thus, by the definition of a process and by Theorem 5.18,

$$\begin{aligned}\phi(S_i) &= \phi((S_{i-1}-{}^\bullet(t_i)) \cup (t_i)^\bullet) \\ &= (\phi(S_{i-1})-{}^\bullet(\phi(t_i))) \cup (\phi(t_i))^\bullet.\end{aligned}$$

Consequently, $\phi(S_{i-1})[\phi(t_i)\rangle\phi(S_i)$. □

Bibliographical comments

The theory of Petri nets was initiated by Petri in [40]. A basic text book on Petri nets is [41], where many of the notions that we have recalled in Subsection 5.1 can

be found. The relationship between the theory of Petri nets and the theory of traces was established already in [31]. In fact, the original motivation of [31] was to study the concurrent behavior of Petri nets. The fact that the firing sequence languages of C/E structures and C/E systems are regular (Theorem 5.8) is a well-known 'folklore' result—it can be traced to [21]. The notion of a reliance alphabet induced by a net is introduced in [7] (although a different terminology is used there).

Subsection 5.2 is based essentially on [34]. The notion of projection was introduced in [20] and the notion of synchronization is close to the synchronization mechanism discussed in [23] and [28]. The proof of Theorem 5.34 can be found in [34], where also the notion of the composition of C/E structures is introduced. The proof of the main theorem of Subsection 5.2 (Theorem 5.37) can also be found in [34].

As far as Subsection 5.3 is concerned, the proof of Theorem 5.39(1) can be found in [7]. The idea of relating Petri net languages to partial order languages is also discussed in [10], although the approach there is quite different from the approach we have taken.

Acknowledgment

The authors are indebted to the following persons for valuable comments and suggestions concerning the preliminary version of this paper: V. Diekert, J. Engelfriet, A.M. Fishscale, M.P. Flé, H. Goeman, P. Graubmann, H.J. Hoogeboom, R. Janicki, N. Keesmaat, A. Kiehn, H. Kleijn, A. Mazurkiewicz, E. Ochmanski, D. Perrin, N. Sabadini, R. Verraedt, H. Vogler, and E. Welzl. Moreover, the authors are indebted to the referees for their constructive comments.

References

[1] IJ.J. Aalbersberg and G. Rozenberg, Traces, dependency graphs and DNLC grammars, *Discr. Appl. Math.* **11** (1985) 299-306.

[2] IJ.J. Aalbersberg and G. Rozenberg, Trace languages defined by context-free string languages, Manuscript, Dept. of Computer Science, Univ. of Leiden, 1985.

[3] IJ.J. Aalbersberg and E. Welzl, Trace languages defined by regular string languages, *RAIRO Inform. Théor. et Applic.* **20** (1986) 103-119.

[4] A. Bertoni, M. Brambilla, G. Mauri and N. Sabadini, An application of the theory of free partially commutative monoids: asymptotic densities of trace languages, in: Lecture Notes in Computer Science **118** (Springer, Berlin, 1981) 205-215.

[5] A. Bertoni, G. Mauri and N. Sabadini, A hierarchy of regular trace languages and some combinatorial applications, in: *Proc. 2nd. World Conf. on Mathematics at the Service of Men*, Las Palmas (1982) 146-153.

[6] A. Bertoni, G. Mauri and N. Sabadini, Equivalence and membership problems for regular and context-free trace languages, Intern. Rept., Inst. Cibern., Univ. of Milan, 1982.

[7] A. Bertoni, G. Mauri and N. Sabadini, Concurrency and commutativity, Intern. Rept., Inst. Cibern., Univ. of Milan, 1982 (presented at *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, 1982).

[8] A. Bertoni, G. Mauri and N. Sabadini, Unambiguous regular trace languages, in: Colloquia Mathematica Societatis János Bolyai **42** (North-Holland, Amsterdam, 1985) 113-123.

[9] A. Bertoni, G. Mauri and N. Sabadini, Representations of prefixes of a trace and membership problem for context-free trace languages, Intern. Rept., Inst. Cibern., Univ. of Milan, 1985.

[10] E. Best, C. Fernandez and H. Plünnecke, Concurrent systems and processes, *GMD Studien 104*, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin, 1985.
[11] K.P. Bogart, Decomposing partial orderings into chains, *J. Combin. Theor.* **9** (1970) 97-99.
[12] P. Cartier and D. Foata, *Problèmes Combinatoires de Commutation et Rearrangements*, Lecture Notes in Mathematics **85** (Springer, Berlin 1969).
[13] R. Cori and D. Perrin, Automates et commutations partielles, *RAIRO Inform. Théor.* **19** (1985) 21-32.
[14] M.P. Flé and G. Roucairol, On serializability of iterated transactions, in: *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, Ottawa (1982) 194-200.
[15] M.P. Flé and G. Roucairol, Multiserialization of iterated transactions, *Inform. Process. Lett.* **18** (1984) 243-247.
[16] M.P. Flé and G. Roucairol, Fair serializability of iterated transactions using FIFO-nets, in: Lecture Notes in Computer Science **188** (Springer, Berlin, 1985) 154-168.
[17] M. Fliess, Matrices de Hankel, *J. Math. Pures et Appl.* **53** (1974) 197-224.
[18] M.R. Garey and D.S. Johnson, *Computers and Intractability* (Freeman, San Francisco, CA, 1979).
[19] S. Ginsburg, *The Mathematical Theory of Context-Free Languages* (McGraw-Hill, New York, 1966).
[20] G. Györy, E. Knuth and L. Romai, Grammatical projections, Working paper of Comput. and Autom. Institute, Hungarian Academy of Sciences, 1979.
[21] M. Hack, Petri net languages, Comput. Struct. Group Memo 124, Project MAC, MIT, Cambridge, MA, 1975.
[22] F. Harary, *Graph Theory* (Addison-Wesley, Reading, MA, 1969).
[23] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21** (1978) 666-677.
[24] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[25] D. Janssens and G. Rozenberg, A characterization of context-free string languages by directed node-label controlled graph grammars, *Acta Inform.* **16** (1981) 63-85.
[26] R.M. Keller, A solvable program-schema equivalence problem, in: *Proc. 5th. Ann. Princeton Conf. on Information Sciences and Systems*, Princeton, NJ (1971) 301-306.
[27] G. Lallement, *Semigroups and Combinatorial Applications* (Wiley, New York, 1979).
[28] P.E. Lauer, M.W. Shields and E. Best, Design and analysis of highly parallel and distributed systems, in: Lecture Notes in Computer Science **86** (Springer, Berlin, 1979) 451-503.
[29] F.W. Levi, On semigroups, *Bull. Calcutta Math. Soc.* **36** (1944) 141-146.
[30] M. Lothaire, *Combinatorics on Words* (Addison-Wesley, Reading, MA, 1983).
[31] A. Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI Rept. PB-78, Aarhus Univ., Aarhus, 1977.
[32] A. Mazurkiewicz, A calculus of execution traces for concurrent systems, Manuscript, Inst. of Computer Science, Polish Acad of Science, Warsaw, 1983.
[33] A. Mazurkiewicz, Traces, histories, graphs: instances of a process monoid, in: Lecture Notes in Computer Science **176** (Springer, Berlin, 1984) 115-133.
[34] A. Mazurkiewicz, Semantics of concurrent systems: a modular fixed-point trace approach, Lecture Notes in Computer Science **188** (Springer, Berlin, 1985) 353-375.
[35] K. Mehlhorn, *Data Structures and Algorithms 2* (Springer, Berlin, 1984).
[36] Y. Metivier, On recognizable subsets of free partially commutative monoids, in: Lecture Notes in Computer Science **226** (Springer, Berlin, 1986) 254-264.
[37] E. Ochmanski, Regular trace languages, Ph.D. Thesis, Inst. of Computer Science, Polish Acad. of Science, Warsaw, 1985.
[38] Z. Pawlak, Rough sets, *Internat. J. Comput. Inform. Sci.* **11** (1982) 341-356.
[39] D. Perrin, Words over a partially commutative alphabet, NATO ASI Series F12 (1985) 329-340.
[40] C.A. Petri, Kommunikation mit Automaten, Schrift Nr. 2, Institut fur Instrumentelle Mathematik, 1962.
[41] W. Reisig, *Petri Nets, an Introduction* (Springer, Berlin, 1985).
[42] J. Sakarovitch, On regular trace languages, *Theoret. Comput. Sci.* **52** (1987) 59-75.
[43] A. Salomaa, *Theory of Automata* (Pergamon, New York, 1969).
[44] A. Salomaa, *Formal Languages* (Academic Press, New York, 1973).
[45] M. Szijarto, Trace languages and closure operations, Tech. Rept., Dept. of Numerical and Computational Mathematics, L. Eötvos Univ., Budapest, 1979.
[46] M. Szijarto, A classification and closure properties of languages for describing concurrent system behaviours, *Fund. Inform.* **4** (1981) 531-549.

[47] A. Tarlecki, Notes on the implementability of formal languages by concurrent systems, ICS PAS Rept. 481, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1982.

[48] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific. J. Math.* **5** (1955) 285-309.

Annotated bibliography

[A1] IJ.J. Aalbersberg and G. Rozenberg, Traces, dependency graphs and DNLC grammars, *Discr. Appl. Math.* **11** (1985) 299-306.

Using dep-graphs a relationship between trace theory and the theory of graph grammars is established. In praticular, it is demonstrated how regular dep-graph languages can be generated by regular DNLC grammars.

[A2] IJ.J. Aalbersberg and G. Rozenberg, Trace languages defined by context-free string languages, Manuscript, Dept. of Computer Science, University of Leiden, 1985.

The classes of existentially, universally and consistently context-free trace languages are investigated.

[A3] IJ.J. Aalbersberg and E. Welzl, Trace languages defined by regular string languages, *RAIRO Inform. Théor. Applic.* **20** (1986) 103-119.

The relationship between existentially and universally regular trace languages is investigated as well as decidability problems for and closure properties of existentially, universally and consistently regular trace languages are considered.

[A4] A. Bertoni, M. Brambilla, G. Mauri and N. Sabadini, An application of the theory of free partially commutative monoids: asymptotic densities of trace languages, in: Lecture Notes in Computer Science **118** (1981) 205-215.

The connection between the theory of traces and the theory of partially commutative monoids is established. Some basic properties of traces and trace languages implied by this connection are given.

[A5] A. Bertoni and M. Goldwurm, Average analysis of an algorithm for a membership problem on trace languages, Intern. Rept., Dept. of Information Science, Univ. of Milan, 1986.

The average behavior of an algorithm for deciding the membership problem of existentially regular trace languages is investigated and the influence of different assumptions on the set of inputs is considered.

[A6] A. Bertoni, G. Mauri and N. Sabadini, A hierarchy of regular trace languages and some combinatorial applications, in: *Proc. 2nd. World Conf. on Mathematics at the Service of Men*, Las Palmas (1982) 146-153.

Using regular string languages different classes of existentially regular trace languages are defined via different methods of existential claiming. It is demonstrated that in this manner a (possibly strict) hierarchy of existentially regular trace languages is obtained.

[A7] A. Bertoni, G. Mauri and N. Sabadini, Equivalence and membership problems for regular trace languages, in: Lecture Notes in Computer Science **140** (1982) 61-71.

Various types of the equivalence and membership problems for existentially regular trace languages are considered (this paper is extended in [A9]).

[A8] A. Bertoni, G. Mauri and N. Sabadini, Context-free trace languages, in: *Proc. 7th. CAAP*, Lille (1982) 32-42.

An algebraic characterization of existentially context-free trace languages is given and the complexity of various types of membership problems for existentially context-free trace languages is considered (this paper is extended in both [A9] and [A12]).

[A9] A. Bertoni, G. Mauri and N. Sabadini, Equivalence and membership problems for regular and context-free trace languages, Intern. Rept., Institute Cibernetica, Univ. of Milan, 1982.

An overview and an extension of the results from [A7] and [A8]: various types of equivalence and membership problems for existentially regular and context-free trace languages are considered (this paper is partly extended in [A12]).

[A10] A. Bertoni, G. Mauri and N. Sabadini, Concurrency and commutativity, Intern. Rept., Institute Cibernetica, Univ. of Milan, 1982 (presented at *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, 1982).

Consistently regular trace languages are related to Petri net languages. In particular, it is demonstrated that safe Petri nets generate only consistently regular trace languages, where the independence relation in question is induced by the Petri net considered.

[A11] A. Bertoni, G. Mauri and N. Sabadini, Unambiguous regular trace languages, in: Colloquia Mathematica Societas Janos Bolyai **42** (North-Holland, Amsterdam, 1985) 113-123.

The class of the so-called unambiguous regular trace languages is investigated and its position between the class of consistently regular trace languages and the class of existentially regular trace languages is established.

[A12] A. Bertoni, G. Mauri and N. Sabadini, Representations of prefixes of a trace and membership problem for context-free trace languages, Intern. Rept. Institute Cibernetica, Univ. of Milan, 1985.

The upper bound on the complexity of the membership problem for existentially context-free trace languages, as given in [A8], is improved; a representation technique for prefixes of a trace in terms of arrays is used (this paper is an extension of [A9]).

[A13] E. Best, C. Fernandez and H. Plünnecke, Concurrent systems and processes, *GMD Studien 104*, Gesellschaft für Mathematik und Datenverarbeitung, Sankt Augustin, 1985.

A small part of this extensive book-like manuscript deals with the relationship between the theory of behavior of various types of Petri nets and the theory of traces.

[A14] A. Carpi and A. De Luca, Square-free words on partially commutative free monoids, *Inform. Process. Lett.* **22** (1986) 125-132.

A characterization of the partially commutative free monoids having an infinite number of square-free elements is given. It is proved that it is decidable whether a given partially commutative free monoid contains infinitely many square-free words.

[A15] P. Cartier and D. Foata, *Problèmes Combinatoires de Commutation et Réarrangements*, Lecture Notes in Mathematics **85** (1969).

This paper originates the theory of partially commutative monoids, introduced here as a framework to consider rearrangements of words. The idea behind the decomposition of traces into normal forms can also be found in this paper.

[A16] M. Clerbout and M. Latteux, Partial commutations and faithful rational transductions, *Theoret. Comput. Sci.* **34** (1984) 241-254.

The relationship between the operation of partial commutativity (i.e., taking the exterior) and various well-known language-theoretic operations are considered. Using the obtained results, a number of families of languages is characterized.

[A17] M. Clerbout and M. Latteux, Semi-commutations, Tech. Rept. IT-63-84, Equipe Lilloise d'Informatique Théorique, Univ. de Lille 1, Villeneuve d'Ascq, 1984.

The nonsymmetric version of the concurrency relation is considered; this leads to semicommutations rather than to commutations.

[A18] R. Cori and Y. Metivier, Recognizable subsets of some partially abelian monoids, *Theoret. Comput. Sci.* **35** (1985) 179-189.

It is demonstrated that, for a finite set of strings L over an alphabet Σ such that each string of L contains at least one occurrence of any letter from Σ, the exterior of L^* is regular. It is assumed that the dependence graph of the reliance alphabet involved is connected.

[A19] R. Cori and D. Perrin, Automates et commutations partielles, *RAIRO Inform. Théor.* **19** (1985) 21-32.

Besides some basics on traces, it is demonstrated that the class of consistently regular trace languages is closed under trace-concatenation. Furthermore, it is proved that if L is a regular string language such that no pair of independent symbols occurs in the same string of L, then the exterior of L^* is regular.

[A20] C. Duboc, Some properties of commutation in free partially commutative monoids, *Inform. Process. Lett.* **20** (1985) 1-4.

The paper investigates the equation $t \circ u = u \circ t$, where t and u are traces. Among other results, it is demonstrated that powers of two traces t and u are equal if and only if t and u are powers of a third trace.

[A21] C. Duboc, Equations in free partially commutative monoids, in: Lecture Notes in Computer Science **210** (1986) 192-202.

A characterization of the solutions of all the equations in two unknowns in free partially commutative monoids is given; it is shown that the solutions are basically cyclic. The paper ends with some results on the transposition and the conjugacy relation.

[A22] M.P. Flé and G. Roucairol, On serializability of iterated transactions, in: *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, Ottawa (1982) 194–200.

The notion of serializability considered in database systems is formalized in the framework of traces. A synchronization algorithm is provided, which allows only those behaviors which are correct and fair.

[A23] M.P. Flé and G. Roucairol, Multiserialization of iterated transactions, *Inform. Process. Lett.* **18** (1984) 243–247.

The notion of serializability from [A22] is generalized to multiserializability and the new notion is used to investigate deadlock freedom for transaction systems.

[A24] M.P. Flé and G. Roucairol, Fair serializability of iterated transactions using FIFO-nets, in: Lecture Notes in Computer Science **188** (1985) 154–168.

This paper deals with infinite traces. An algorithm is given which controls the serializability condition; the algorithm yields the maximal parallellism and guarantees fairness.

[A25] M.P. Flé and G. Roucairol, Maximal serializability of iterated transactions, *Theoret. Comput. Sci.* **38** (1985) 1–16.

This paper deals with infinite traces. The behavior of a system of (synchronized) transactions is investigated; a characterization of the prefixes of behaviors satisfying the serializability condition is given and it is shown that the set of all these behaviors can be controlled by a finite automaton.

[A26] D. Foata, Rearrangements of words, in: M. Lothaire, ed., *Combinatorics on Words* (Addison-Wesley, Reading, MA, 1983) Chapter 10.

This is an exposition of the theory of rearrangements of words, leading, among others, to partially commutative monoids.

[A27] J. Grabowski, On partial languages, *Fund. Inform.* **4** (1981) 427–498.

Various closure properties of the so-called partial languages of (safe) Petri nets are investigated; the theory of partial languages is related to the theory of traces.

[A28] R. Janicki, Synthesis of concurrent schemes, in: Lecture Notes in Computer Science **64** (1978) 298–307.

A method is given, which yields for a fixed set of traces (of a special kind) an appropriate concurrent scheme 'realizing' this set.

[A29] R. Janicki, On the design of concurrent systems, in: *Proc. 2nd Internat. Conf. on Distributed Computing Systems*, Paris (1981) 455–466.

A method based on both the theory of Petri nets and the theory of traces is presented in order to construct concurrent systems out of sequential systems.

[A30] R. Janicki, Trace semantics for communicating sequential processes, Tech. Rept. R-85-12, Institute for Elektr. Syst., Univ. Aalborg, Aalborg, 1985.

A non-interleaving semantics for Hoare's CSP is proposed. As a medium to describe the partial orders of event occurrences defined by CSP systems, traces and generalizations of traces are used.

[A31] R.M. Keller, A solvable program-schema equivalence problem, in: *Proc. 5th. Ann. Princeton Conf. on Information Sciences and Systems*, Princeton (1971) 301–306.

An equivalence problem for a class of program schemas is considered. A symmetric binary relation on a set of operations, expressing their independence, forms the basis of the definition of the equivalence used.

[A32] E. Knuth, Petri nets and regular trace languages, Tech. Rept. ASM/47, Computing Laboratory, Univ. Newcastle upon Tyne, 1978.

[A33] E. Knuth, Petri nets and trace languages, in: *Proc. 1st. Europ. Conf. on Parallel and Distributed Processing*, Toulouse (1979) 51–56.

The relationship between path expressions and traces is investigated (this paper is closely related to [A34]).

[A34] E. Knuth and G. Györy, Paths and traces, *Comput. Linguis. Comput. Languages* **13** (1979) 31–42.

The relationship between path expressions and traces is investigated (this paper is closely related to [A33]).

[A35] A. Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI Rept. PB-78, Aarhus Univ., Aarhus, 1977.

The notions of a trace and a trace language are introduced and investigated. It is shown how to solve trace equations by solving string equations. A relationship between the theory of traces

and Petri nets is established. This paper initiates the theory of traces in the way it is considered in our paper.

[A36] A. Mazurkiewicz, A calculus of execution traces for concurrent systems, Manuscript, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1983.

[A37] A. Mazurkiewicz, Traces, histories, graphs: instances of a process monoid, in: Lecture Notes in Computer Science **176** (1984) 115-133.

It is demonstrated how various theories (among them the theory of traces) can be expressed in the framework of process algebra introduced in this paper.

[A38] A. Mazurkiewicz, Semantics of concurrent systems: a modular fixed-point trace approach, in: Lecture Notes in Computer Science **188** (1985) 353-375.

A method for describing the behavior (the language) of Petri nets using trace theory is given. Operations on nets, such as composition and decomposition, are transformed into operations on traces, such as synchronization and projection.

[A39] Y. Metivier, Une condition suffisante de reconnaissabilité dans un monoïde partiellement commutatif, *RAIRO Inform. Théor. et Applic.* **20** (1986) 121-127.

It is shown that the trace iteration of a consistently regular trace language T is (again) a consistently regular trace language, provided that the dep-graphs of the traces of T are connected.

[A40] Y. Metivier, On recognizable subsets of free partially commutative monoids, in: Lecture Notes in Computer Science **226** (Springer, Berlin, 1986) 254-264.

It is shown that the exterior of a string language L is regular if L is regular and if the dep-graph of each iterative factor of every string in L is connected.

[A41] E. Ochmanski, Regular trace languages, Ph.D. Thesis, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 1985.

The class of consistently regular trace languages are characterized in two different ways. First a characterization is given using the operation of concurrent iteration and then a characterization is given by means of the lexicographical minimal string in a trace.

[A42] D. Perrin, Words over a partially commutative alphabet, NATO ASI Series F12 (1985) 329-340.

The paper has the character of an introductory survey. Some basic results on traces and trace languages are discussed. In particular, a section is dedicated to two different normal forms of traces.

[A43] W. Rytter, Some properties of trace languages, *Fund. Inform.* **7** (1984) 117-127.

The membership problem for existentially context-free trace language is considered, as far as time and space complexity are concerned. Furthermore, some results concerning the classes of exteriors of regular and context-free string languages are given.

[A44] J. Sakarovitch, On regular trace languages, *Theoret. Comput. Sci.* **52** (1987) 59-75.

Using the theory of partially commutative monoids, it is proved that the existentially regular trace languages form a boolean algebra if and only if the existentially regular trace languages are (so-called) unambiguously regular (see also [A11]) if and only if the considered concurrency relation is transitive.

[A45] P.H. Starke, Traces and semiwords, in: Lecture Notes in Computer Science **208** (1985) 332-349.

The paper discusses the relationship between the theory of traces and the theory of semiwords.

[A46] M. Szijarto, Trace languages and closure operations, Tech. Rept., Dept. of Numerical and Computational Mathematics, L. Eötvos Univ., Budapest, 1979.

This is a preliminary version of [A47]. The paper investigates several closure properties of various classes of trace languages defined existentially using string languages from the Chomsky hierarchy. Also, the notion of a graph composition of dep-graphs can be found here.

[A47] M. Szijarto, A classification and closure properties of languages for describing concurrent system behaviors, *Fund. Inform.* **4** (1981) 531-549.

One of the first serious language theoretic investigations of the theory of traces (this paper is the final version of [A46]).

[A48] A. Tarlecki, Notes on the implementability of formal languages by concurrent systems, ICS PAS Rept. 481, Institute of Computer Science, Polish Academy of Sciences, Warsawa, 1982.

Petri nets and functions on symbols (their extensions to sets of symbols and to sets of sets of symbols) are used to investigate existentially and consistently regular trace languages.

[A49] G.X. Viennot, Heaps of pieces 1: basic definitions and combinatorial lemmas, Tech. Rept. I-8614, U.E.R. de Mathématiques et d'Informatique, Univ. de Bordeaux 1, Bordeaux, 1986.

The introduction of the combinatorial notion of heaps of pieces leads to a geometrical interpretation of the free partially commutative monoid. Several results based on the theory of these monoids are related to the theory of heaps.

[A50] W. Zielonka, Proving assertions about parallel programs by means of traces, ICS PAS Rept. 424, Institute of Computer Science, Polish Academy of Sciences, Warsawa, 1980.

The paper uses the theory of Petri nets as well as the theory of traces in order to prove assertions of a great and rather complex airline reservation system. Conclusions on the method used are drawn and difficulties are indicated.

[A51] W. Zielonka, Notes on finite asynchronous automata and trace languages, *RAIRO Inform. Théor. et Applic.* **21** (1987) 99-135.

Finite-state asynchronous automata are defined and then it is claimed that they accept exactly consistently regular trace languages.