INFORMATIQUE THÉORIQUE ET APPLICATIONS

A. EHRENFEUCHT
H. J. HOOGEBOOM
G. ROZENBERG

On the structure of recognizable languages of dependence graphs

Informatique théorique et applications, tome 27, nº 1 (1993), p. 7-22.
<http://www.numdam.org/item?id=ITA_1993__27_1_7_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 27, n° 1, 1993, p. 7 à 22)

ON THE STRUCTURE OF RECOGNIZABLE LANGUAGES OF DEPENDENCE GRAPHS (*)

by A. Ehrenfeucht [1], H. J. Hoogeboom [2] and G. Rozenberg [1] [2]

Abstract. – *Within the* theory of traces *a* dependence graph *represents a behaviour of a concurrent system (e.g., a safe Petri net) in a very much the same way that a string represents a behaviour of a sequential system (e.g., a finite automaton). A* recognizable language of dependence graphs, RecDG language *for short, represents the set of all behaviours of a concurrent system (with a "regular behaviour"). In this paper we characterize* naked RecDG languages, i.e., *the sets of unlabelled graphs obtained by erasing labels from graphs of RecDG languages.*

Résumé. – *Dans le cadre de la théorie des traces, un graphe de dépendance représente le comportement d'un système distribué (par exemple un réseau de Petri) par analogie avec la relation mot-automate dans le cas séquentiel. Un langage reconnaissable de graphes de dépendances représente ainsi l'ensemble de tous les comportements d'un système distribué satisfaisant des conditions de régularité. Dans cet article nous caractérisons les langages de graphes obtenus à partir des précédents par suppression des étiquettes sur les sommets.*

1. INTRODUCTION

In the *theory of traces* (as introduced in [M1], *see* also [M2] and [AR1]) a concurrent system is represented by an alphabet Σ (representing the *events* of the system), a string language K (representing all *sequential observations* of the system), and a binary relation I on the set of events of the system (representing the *independence* between events). It is required that I is symmetric. In this paper we will consider regular string languages K. Given $w \in K$ such that $w = w_1 abw_2$ for events a, b with $(a, b) \in I$, one infers that the order "first a then b" in w is a feature of the *sequential* observation and *not* of the system (because a and b are independent in the system). Hence $w' = w_1 baw_2$

(*) Submitted March 1990, Revised form June 1992.
[1] University of Colorado at Boulder, Department of Computer Science Colorado, U.S.A.
[2] Leiden University, Department of Computer Science P.O. Box 9512, 2300 RA Leiden, The Netherlands.
All correspondence to the second author.

represents also a sequential observation of the system – actually an observation of "the same behaviour". We say then that w is *I-related* to w' and write $w =_I w'$. Then, words x and y are *I-equivalent*, written $x \equiv_I y$, if one may obtain y from x via a chain of I-related words (*i. e.*, $\equiv_I$ is the congruence relation generated by the equalities $ab \equiv_I ba$ with $(a, b) \in I$). An equivalence class of $\equiv_I$ is referred to as a *trace* – hence a trace consists of *all* sequential observations of the same (concurrent) behaviour. We recall that the quotient of Σ^* under $\equiv_I$ is a monoid (with concatenation as operation).

If one considers $w = a_1 \ldots a_n \in K$, where $n \geq 1$ and $a_1 \ldots, a_n$ are events of the system, and constructs a node labelled directed graph by creating a node i for each $1 \leq i \leq n$, labelling node i by the letter a_i, and introducing an edge from node i to node j if and only if $i < j$ and $(a_i, a_j) \notin I$, then one gets the *dependence graph* of w. It represents the partial order of events in the system that corresponds to the sequential observation w.

It turns out that a dependence graph (a single object) may be identified with a trace (a *set* of strings) in the following sense: two strings from K have the same (*i. e.*, isomorphic) dependence graphs if and only if they belong to the same trace. One can also reverse the situation and say that given a dependence graph, all (words corresponding to all) linear extensions of it constitute a trace.

The set of all dependence graphs corresponding to all words from K (the *dependence graph language* corresponding to K) represents then the *behaviour* of the system in much the same way as the language (of strings) of a finite automaton (modelling a sequential system) represents its behaviour.

In this way dependence graphs constitute fundamental objects in the theory of concurrent systems (*see* also [M2] and [AR1] where dependence graphs are used to represent the behaviour of condition-event systems in the sense of Petri).

Apart from [M1], independent origins of the theory of traces are [CF], [FR] and [K], while, e. g., [AW], [BMS], [CP], [Me] and [O] represent various research aspects of the theory; [AR1] and [P] are survey type papers covering (parts of) the theory of traces.

Another motivation for considering dependence graphs (and their languages) comes from the *theory of graph grammars* (*see*, e. g., [ENR]). One of the research areas of this theory is the search for graph grammars analogous to right-linear string grammars; one hopes in this way to isolate a class of graph languages as simple and fundamental for the theory of graph grammars as right-linear grammars (and regular languages) are for the theory of string grammars.

Within the *node label control* (NCL) *grammars* this effort leads to the so-called BNLC grammars (*see*, e. g., [RW]) for generating undirected graphs and to RDNLC grammars (*see*, e. g., [AR2]) for generating directed graphs. In particular, it turns out that one can use a subclass of the class of RDNLC grammars (*see* [AR2]) to characterize rational languages of dependence graphs as used in the theory of traces. In this way dependence graphs are fundamental objects in the theory of graph grammars.

This paper investigates the structure of recognizable languages of dependence graphs, abbreviated *RecDG languages*, which are languages of dependence graphs as explained above. By the "structure" of a dependence graph g we understand its "naked" version, *i. e.*, the unlabelled graph obtained from g by erasing the labels of nodes of g.

In [ER] a characterization of naked dependence graphs is given – here were extend it to provide a characterization of *naked RecDG languages*.

The paper is organized as follows. In the preliminaries and in Section 3 we settle the basic notation and terminology (especially concerning graphs) to be used in this paper. In Section 4 we recall from [ER] basic notions and results concerning (special kinds of) labellings of graphs. These are then used in Section 5 to obtain a characterization of naked RecDG languages.

2. PRELIMINARIES

In this section we introduce some basic notation and terminology to be used in the paper.

For an equivalence relation $\equiv$ on a set X, $[x]_\equiv$ denotes the equivalence class of $\equiv$ containing $x \in X$. The set of equivalence classes of $\equiv$ is called the *partition induced by* $\equiv$ *on* X, and is denoted by $\mathbf{P}(\equiv)$.

For a function $\varphi : X \to Y$, and $Z \subseteq X$, $\varphi(Z) = \{y \in Y : \varphi(z) = y$ for some $z \in Z\}$. The *range of* φ is the set $\varphi(X)$. The composition of functions φ and ψ (first apply φ, then ψ) is denoted by $\psi \circ \varphi$.

For a word w, *alph*(w) denotes the set of letters occurring in w.

We move now to consider notations and terminology concerning graphs. We assume the reader to be familiar with basic notions of graph theory – the aim of this section is to settle the specific notation and terminology concerning graphs that we will use in this paper.

A (finite) directed graph is referred to simply as a *graph*. A graph g is specified in the form (V, E) where V is the set of nodes of g and $E \subseteq V \times V$ is the set of (directed) edges of g. We also use V_g and E_g to denote the sets of nodes and edges of a given graph g.

Let $g=(V, E)$ be a graph.
For $U \subseteq V$, the *subgraph of g induced by U* is the graph $(U, E \cap (U \times U))$; in this paper a *subgraph* of g is always a subgraph of g induced by a subset of V.

The *symmetric and reflexive closure of g*, denoted $symr(g)$, is the graph (V, E'), where for all $u, v \in V$, $(u, v) \in E'$ if and only if either $u=v$ or $(u, v) \in E$ or $(v, u) \in E$. We say that g is a *symmetric and reflexive graph*, a *symr graph* for short, if $g=symr(g)$. Note that $symr(g)$ can be seen as the undirected version of the graph g, with a loop for each node. These loops are convenient for a technical reason: their existence simplifies the definition of a homomorphism between graphs.

Let $g_1=(V_1, E_1)$, $g_2=(V_2, E_2)$ be graphs.
A function $\psi: V_1 \to V_2$ is a *homomorphism of* g_1 *into* g_2 if, for all $u, v \in V_1$, $(u, v) \in E_1$ if and only if $(\psi(u), \psi(v)) \in E_2$; ψ is an *isomorphism* if ψ is also bijective. If g_1 and g_2 are *isomorphic*, then we write $g_1 =_{is} g_2$. An *automorphism of* g_1 is an isomorphism of g_1 into itself. We will use $AUT(g)$ to denote the set of automorphisms of a graph g.

A *graph language* is a set of graphs. For a graph g and a graph language K we write $g \in_{is} K$ if there exists an $h \in K$ such that $g =_{is} h$. For graph languages K_1, K_2 we write $K_1 =_{is} K_2$ if for every graph g, $g \in_{is} K_1$ if and only $g \in_{is} K_2$.

A *labelling* of a graph g is a (total) function on V_g. A *node labelled graph, nl graph* for short, is a system (V, E, φ) where (V, E) is a graph and φ is a labelling of (V, E). For a nl graph $h=(V, E, \varphi)$ we use V_h, E_h and φ_h to denote V, E, and φ respectively; the range $\varphi(V)$ of φ is called the *label alphabet* of h. The graph (V, E), called the *underlying graph of h*, is denoted by $und(h)$.

Let h be a nl graph. The *string language of h*, denoted $lan(h)$, is the set $\{\varphi_h(v_1) \ldots \varphi_h(v_n) : v_1, \ldots, v_n$ is a topological ordering of $h\}$, where a topological ordering of the graph h is an ordering $v_1, \ldots, v_n$ on the set of nodes V_h, such that there is no edge $(v_i, v_j) \in E_h$ with $i \geqq j$. Note that such an ordering exists if and only if the graph is acyclic. Moreover, note that the string languages of isomorphic nl graphs are equal.

A *nl graph language* is a set of nl graphs. Most derived notions for nl graphs can be extended in a natural way to their languages. In particular, for a nl graph language K, we set $und(K)=\{und(h) : h \in K\}$, and the *string language of K*, denoted $lan(K)$, is the set $\bigcup_{g \in K} \text{lan}(g)$.

3. DEPENDENCE GRAPHS AND LANGUAGES

In this section we recall the definition of dependence graphs and their languages.

Given an alphabet Σ, a *dependence alphabet* (*over* Σ) is a symr graph $\Gamma=(\Sigma, D)$; D is called the *dependence relation of* Γ. If $(a, b)\in D$, then a and b are called *dependent*. For a word $w=a_1 \ldots a_n$ over Σ, $n\geqq 0$ and $a_i\in\Sigma$ for $i\in\{1, \ldots, n\}$, the *canonical* Γ-*dependence graph of* w, denoted $\langle w\rangle_\Gamma$, is the nl graph $g=(V, E, \varphi)$ such that $V=\{1, \ldots, n\}$, for all $i, j\in V$, $(i, j)\in E$ if and only if $i<j$ and $(a_i, a_j)\in D$, and $\varphi(i)=a_i$. For $w\in\Sigma^+$, any nl graph isomorphic with $\langle w\rangle_\Gamma$ is called a Γ-*dependence graph* (*of* w).

A nl graph is a *dependence graph* if it is a Γ-dependence graph for a dependence alphabet Γ. A *naked dependence graph* is a (unlabelled) graph g such that $g=und(h)$ for a dependence graph h.

3.1. *Example:* Let $\Sigma=\{a, b, c, d, e\}$ and let $\Gamma=(\Sigma, D)$ be given by:

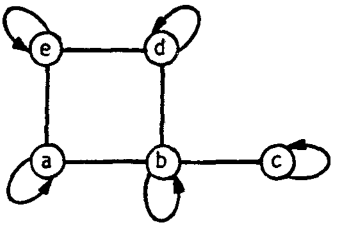

Figure 3.1

i.e., $\Gamma=symr(\Sigma, \{(a, b), (a, e), (b, c), (b, d), (d, e)\})$.

Then $\langle adedcba\rangle_\Gamma$ is the following nl graph (for a node v we write the label of v inside the circle representing v):

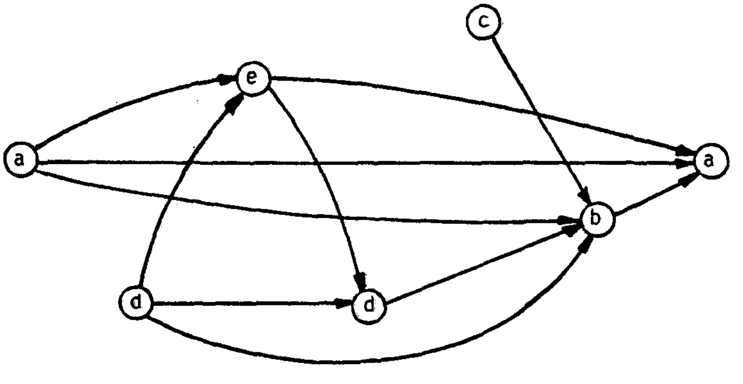

Figure 3.2

Hence

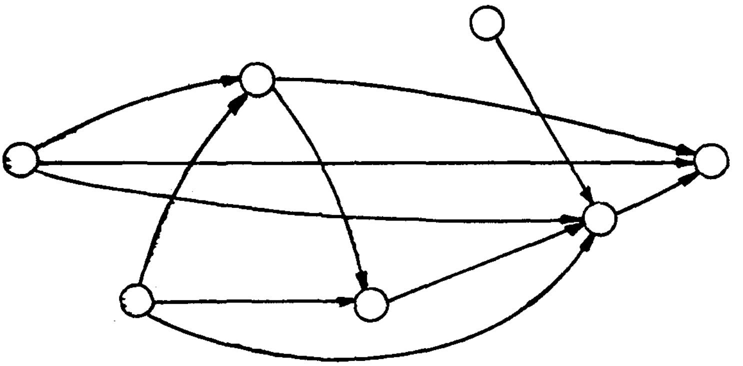

Figure 3.3

is a naked dependence graph. ■

Let Γ be a dependence alphabet over Σ, and let $W \subseteq \Sigma^+$. $\langle W \rangle_\Gamma$ denotes the nl graph language $\{\langle w \rangle_\Gamma : w \in W\}$. A language K of dependence graphs is *rational* (or *regular*) if $K =_{is} \langle W \rangle_\Gamma$ for a dependence alphabet Γ and a regular language W. If moreover $lan(K)$ is regular, then K is *recognizable* (or *consistent regular*, as in [AR1]). The acronym a *RecDG language* stands for a recognizable language of dependence graphs. A language K of graphs is a *naked RecDG language* if $K =_{is} und(L)$ for a RecDG language L.

3.2. *Example:* Let $\Sigma = \{a, b\}$ and let $\Gamma = (\Sigma, D)$ be as follows:

Figure 3.4

Let $K = \{ab\}^+$. Consider $w = (ab)^3 \in K$. Then $\langle w \rangle_\Gamma$ is as follows:

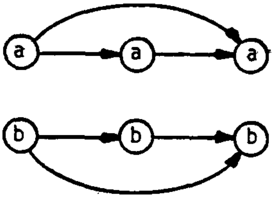

Figure 3.5

$und(\langle K \rangle_\Gamma)$ is the following set of graphs:

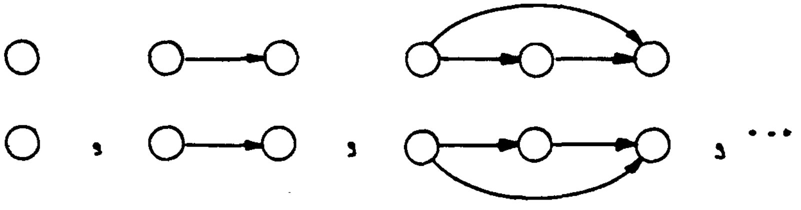

Figure 3.6

$\langle K \rangle_\Gamma$ is *not* a RecDG language because $lan(\langle K \rangle_\Gamma)$

$$= \{ w \in \Sigma^+ : w \text{ contains an equal number of } a\text{'s and } b\text{'s} \}$$

is not regular. ■

3.3. *Remark:* We like to stress here once again the "natural regularity" (within the setting of dependence graphs) of RecDG languages. They can also be characterized by means of a congruence, yielding a "Nerode-type" theorem analogous to the one for regular string languages. Given a dependence alphabet $\Gamma = (\Sigma, D)$ one defines the Γ-concatenation $g_1 \circ_\Gamma g_2$ of Γ-dependence graphs g_1 and g_2 as follows. The graph $g_1 \circ_\Gamma g_2$ consists of the (disjoint) union of the graphs g_1 and g_2 together with all edges leading from a node x in g_1 to a node y in g_2 such that the labels of x and y are dependent (according to Γ). Then we have for all $w_1, w_2 \in \Sigma^*$, $\langle w_1 \rangle_\Gamma \circ_\Gamma \langle w_2 \rangle_\Gamma =_{is} \langle w_1 . w_2 \rangle_\Gamma$ (*see* [AR2]). For a dependence graph language K over Γ we define the congruence $\equiv_K$ by: $g_1 \equiv_K g_2$ if and only if [for all dependence graphs g, g' over Γ, $g \circ_\Gamma g_1 \circ_\Gamma g' \in_{is} K$ iff $g \circ_\Gamma g_2 \circ_\Gamma g' \in_{is} K$].

Then one obtains the following characterization:

K is recognizable if and only if $\equiv_K$ is of finite index.

The Nerode-congruence and characterization can be carried over from the string case to the dependence graph case like the concatenation of words can be carried over from strings to their dependence graphs; in particular, one has

$$w . w_1 . w' \in lan(K) \qquad \text{iff} \quad \langle w \rangle_\Gamma \circ_\Gamma \langle w_1 \rangle_\Gamma \circ_\Gamma \langle w' \rangle_\Gamma \in_{is} K. \quad ■$$

4. LABELLINGS OF GRAPHS

In this section we recall from [ER] the basic notions and results concerning various kinds of labellings of (symr) graphs. We also prove some new results that will turn out to be useful in the sequel of this paper.

Obviously, two nodes in a dependence graph that have the same label are either both connected or both not connected to any other node in the graph; if they are both connected then the directions of the connecting arcs may differ. Hence two nodes, for which a third node can be found such that exactly one of the two nodes is connected to the third one, can't be both labelled by the same letter. Also two nodes that are not connected by an edge cannot be labelled by the same label. Using these observations it is possible to infer information concerning the structure of the dependence alphabet Γ, by looking at the *structure* of a Γ-dependence graph g, *i.e.* the graph that results after removing labels and directions of edges. Consequently, we will use the unlabelled, undirected version *symr* (*und* (g)) of a dependence graph to extract information about the dependence alphabet.

This leads to the following fundamental relation between nodes of a graph, which was introduced in [ER] (in a somewhat different notation).

4.1. DEFINITION: Let g be a symr graph. The *resemblance relation of* g, denoted res_g, is the binary relation on V defined by $(v_1, v_2) \in res_g$ if and only if

(1) $(v_1, v_2) \in E_g$, and

(2) for all $v \in V$, $(v, v_1) \in E_g$ if and only if $(v, v_2) \in E_g$. ∎

Actually the first requirement in the above definition follows from the second by taking $v = v_1$ and the observation that (v, v_1) is an edge in the reflexive graph g. We have chosen to keep (1) in the definition because it makes explicit the fact that two unconnected nodes are not allowed to resemble each other.

In a straigthforward way one may verify that for a graph g, res_g is an equivalence relation (*cf.* [ER]). Moreover connections in g are the same for equivalent nodes. More formally, if $v_1 \, res_g \, v_2$ and $u_1 \, res_g \, u_2$, then $(v_1, u_1) \in E_g$. Hence it is possible to define a quotient structure of a graph g with respect to the equivalence res_g.

4.2. DEFINITION: Let g be a symr graph. The *type of* g, denoted $\hat{g}$, is the graph (V, E) such that $V = \mathbf{P}(res_g)$, and for all $u, v \in V_g$, $([u]_{res_g}, [v]_{res_g}) \in E$ if and only if $(u, v) \in E_{symr\,(g)}$. ∎

As indicated above, $E_{\hat{g}}$ is well defined: the definition is independent of the representative u and v of the equivalence classes of res_g.

Note that, for each graph g, $\hat{g}$ is a symr graph with set of nodes $\mathbf{P}(res_g)$, and so it is a dependence alphabet over $\mathbf{P}(res_g)$.

A surjective homomorphism of one symr graph onto another one can be seen as a "contraction" of the original graph: nodes that have the same connections to the other nodes in the graph (and are connected to each other) may be mapped by the homomorphism onto the same node in the image. Intuitively the type of a symr graph formalizes the maximal contraction possible. Indeed, the next result implies that the type $\hat{g}$ of a symr graph g is the smallest (with respect to the number of nodes) symr such that g can be "contracted" onto that graph.

For a symr graph g let $cal_g: V_g \to V_{\hat{g}}$, defined by $cal_g(u) = [u]_{res_g}$, be the canonical homorphism from g onto $\hat{g}$. In [ER] it is called the *canonical labelling of* g.

4.3. LEMMA: *Let g, h be* symr *graphs, and let φ be a homomorphism from g onto h. Then there exists an homomorphism ψ from h onto $\hat{g}$ such that $cal_g = \psi \circ \varphi$.*

Proof: It is rather straighforward to verify that, for $u_1, u_2 \in V_g$, $\varphi(u_1) = \varphi(u_2)$ implies $u_1 \, res_g \, u_2$. This, together with the fact that φ is surjective, makes $\psi: V_h \to \mathbf{P}(res_g)$, defined by $\psi(v) = [u]_{res_g}$ for $v = \varphi(u)$ a well-defined mapping from V_h onto $V_{\hat{g}}$. Using the fact that both cal_g and φ are homomorphisms, one easily shows that ψ is an homomorphism; it satisfies $cal_g = \psi \circ \varphi$ by definition. ■

We give now two elementary consequences of the above result. First, homomorphisms of a symr graph into its type are all equal, up to symmetries of the type. Secondly, if one symr graph can be "contracted" onto another one, then their types are equal.

4.4. COROLLARY: *Let g be* symr *graph, and let φ_1 and φ_2 be homomorphisms from g into $\hat{g}$. Then there exists an automorphism ψ of $\hat{g}$ such that $\varphi_1 = \psi \circ \varphi_2$.*

Proof: Let h_i be the subgraph of $\hat{g}$ that is the image of g under φ_i. By Lemma 4.3 there exists an homomorphism ψ_i from h_i onto $\hat{g}$. Obviously, this implies that h_i cannot have less nodes than $\hat{g}$. Hence h_i equals $\hat{g}$; φ_i is surjective. Thus ψ_1 and ψ_2 are automorphisms of $\hat{g}$ that satisfy $\psi_1 \circ \varphi_1 = cal_g = \psi_2 \circ \varphi_2$. Consequently, $\varphi_1 = (\psi_1^{-1} \circ \psi_2) \circ \varphi_2$. ■

4.5. COROLLARY: *Let g, h be* symr *graphs, such that there exists a homomorphism from g onto h. Then $\hat{g} =_{is} \hat{h}$.*

Proof: Applying Lemma 4.3 two consecutive times to the homomorphism from g onto h, one obtains first a homomorphism from h onto $\hat{g}$, and then one from $\hat{g}$ onto $\hat{h}$. On the other hand, the homomorphism from g onto h can easily be transformed into one mapping g onto $\hat{h}$. Applying Lemma 4.3 to the latter homomorphism yields a homomorphism from $\hat{h}$ onto $\hat{g}$. Thus $\hat{g}$ and $\hat{h}$ are of equal size; consequently they are isomorphic. ■

In the remainder of this section we will more directly focus on dependence graphs. This means that the graphs that we will consider are directed rather than symr (*i. e.*, undirected) graphs. The notion of *type* is extended to these graphs by defining the type of an arbitrary graph g to be the type of *symr*(g).

We will recall now the main result of [ER] which characterizes the naked dependence graphs for a given dependence alphabet. First we will need a couple of additional notions.

Generalizing the notion of a dependence graph, for an unlabelled graph g we consider the set of nl graphs with g as underlying graph, and having a labelling "compatible" with a given dependence alphabet (*see* also the remark following the definition).

4.6. DEFINITION: Let $\Gamma = (\Sigma, D)$ be a dependence alphabet.

(1) A labelling φ of a graph g is called a Γ*-labelling* if φ is a homomorphism from *symr*(g) into Γ.

(2) A nl graph h is Γ*-labelled* if φ_h is a Γ-labelling of *und*(h).

(3) Let $g = (V, E)$ be a graph. The set of Γ*-labelled versions of* g, denoted $g[\Gamma]$, is the set $\{(V, E, \varphi) : \varphi$ is a Γ-labelling of $g\}$. ■

Usually (as in [AR]), a Γ-labelling of a graph is defined in an "explicit way" by saying that two nodes are connected by an edge in *symr*(g) if and only if their labels are dependent in Γ. It can be easily seen that this is equivalent to the definition we have given above. We have adopted our version of the definition because it seems to be more suitable in the present paper. Additionally, our definition of Γ-labelling is justified by the following, rather elementary, lemma. It formalizes the connection between the notions of Γ-labelling and of dependence graph.

Note that, for every word w over the alphabet (*i. e.* the set of nodes) of Γ, the dependence graph $\langle w \rangle_\Gamma$ is a Γ-labelled graph. Conversely, an *acyclic* Γ-labelled graph turns out to be a Γ-dependence graph.

4.7. OBSERVATION: Let g be an acyclic graph, and let $\Gamma=(\Sigma, D)$ be a dependence alphabet. Then every graph in $g[\Gamma]$ is a Γ-dependence graph.

Moreover, if $w \in \Sigma^*$, then $w \in lan(g[\Gamma])$ if and only if $\langle w \rangle_\Gamma \in_{is} g[\Gamma]$.

Proof:(1) Let $h \in g[\Gamma]$. Since h is acyclic there exists a topological ordering of h, and so $lan(h)$ is nonempty. Choose an arbitrary $w \in lan(h)$; we will show that h is a Γ-dependence graph of w. Consider a topological ordering $v_1, \ldots, v_n$ of the nodes of h such that $a_i=\varphi_h(v_i)$ for $i \in \{1, \ldots, n\}$, where $w=a_1 \ldots a_n$. Since φ_h is a Γ-labelling of h, v_i and $v_j (i<j)$ are connected by an edge in $symr(h)$ if and only if $(a_i, a_j) \in D$. The direction of this edge in h is determined by the fact that $v_1, \ldots, v_n$ is a topological ordering of h. Hence $(v_i, v_j) \in E_h$ if and only if $i<j$ and $(a_i, a_j) \in D$. Consequently $h={}_{is} \langle w \rangle_\Gamma$.

(2) Clearly the above reasoning shows that if $w \in lan(h)$ for some $h \in g[\Gamma]$, then $h={}_{is}\langle w \rangle_\Gamma$, and consequently $\langle w \rangle_\Gamma \in_{is} g[\Gamma]$. On the other hand, if $\langle w \rangle_\Gamma \in_{is} g[\Gamma]$, then $lan\langle w \rangle_\Gamma) \subseteq lan(g[\Gamma])$ follows from the observation that $w \in lan \langle w \rangle_\Gamma$. ■

The existence of Γ-labellings for a grah g (in terms of the structure of g) was investigated in [ER]. The following characterization was obtained – it shows that the type of a graph is the smallest (up to isomorphism) dependence alphabet Γ for which there exists a Γ-labelling of the graph. [Recall that the type of an arbitrary graph g is the type of its symmetric and reflexive closure $symr(g)$.]

4.8. PROPOSITION [ER]: *Let g be an acyclic graph and let Γ be a dependence alphabet. Then g is a naked Γ-dependence graph (i.e., $g[\Gamma] \neq \varnothing$) if and only if $\hat{g}$ is isomorphic to a subgraph of* Γ. ■

We would like to note that it is an easy exercise to reobtain the above characterization using Lemma 4.3.

4.9. THEOREM: *Let g be an acyclic graph, let Γ be a dependence alphabet such that $\Gamma={}_{is}\hat{g}$, and let $h \in g[\Gamma]$.*

(1) $g[\Gamma]=\{(V_g, E_g, \psi \circ \varphi_h) : \psi \in \mathrm{AUT}(\Gamma)\}$, *and*

(2) $lan(g[\Gamma]) = \bigcup_{\psi \in \mathrm{AUT}(\Gamma)} \psi\,(lan\,(h))$.

Proof: Note that, due to Proposition 4.8, $g[\Gamma]$ is not empty.

(1) Observe that φ is a Γ-labelling of g if and only if $\varphi=\psi \circ \varphi_h$ for some $\psi \in \mathrm{AUT}(\Gamma)$. The "if-part" of this statement is obvious: the composition of $\psi \in \mathrm{AUT}(\Gamma)$ and the Γ-labelling φ_h yields a Γ-labelling. In order to see the "only-if-part" we observe that Corollary 4.4 remains valid if one replaces

"homomorphisms into $\hat{g}$" by "homomorphisms into Γ" when Γ is isomorphic to $\hat{g}$.

(2) This follows from (1). Note that, for any "renaming" ψ of the label alphabet of a nl graph h, $lan(\psi(h)) = \psi(lan(h))$, where $\psi(h)$ is the graph $(V_h, E_h, \psi \circ \varphi_h)$. ■

5. LABELLINGS OF LANGUAGES

The notions, terminology, and results from the previous section extend to graph languages in a natural way.

5.1. Definition: Let K be a graph language.

(1) K is *typed* if and only if there exists a graph Γ such that for all $g \in K$, $\hat{g} =_{is} \Gamma$; Γ is called a *type* of K.

(2) Let Γ be a dependence alphabet. The set of *Γ-labelled versions of* K, denoted $K[\Gamma]$, is the set $\bigcup_{g \in K} g[\Gamma]$. ■

Note that it seems not be possible to extend Proposition 4.8 to sets of graphs in a straightforward way. Consider a graph language with graphs of two different types, as given in Figure 5.1 (where we have omitted the transitive edges for clearity). They can be labelled by a dependence alphabet similar to Figure 5.1 (*i.e.*, the *single* graph consisting of two components, but also by a dependence alphabet like the one in Figure 5.2. Note that neither of the two graphs is a subgraph of the other.

Figure 5.1

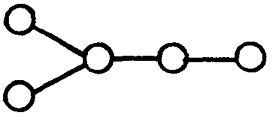

Figure 5.2

5.2. Lemma: Let K be a language of acyclic graphs, and let $\Gamma = (\Sigma, D)$ be a dependence alphabet.

(1) Let $W = lan(K[\Gamma])$. Then $\langle W \rangle_\Gamma =_{is} K[\Gamma]$.

(2) If K is a typed language and Γ is a type of K, then $K = und(K[\Gamma])$.

Proof: (1) According to Observation 4.7, every graph in $K[\Gamma]$ is a Γ-dependence graph. So if $h \in_{is} K[\Gamma]$, then $h =_{is} \langle w \rangle_\Gamma$ for some $w \in \Sigma^+$. Moreover, $w \in lan(K[\Gamma])$ if and only if $h \in_{is} K[\Gamma]$. Hence

$$\langle W \rangle_\Gamma = \{ \langle w \rangle_\Gamma : w \in lan(K[\Gamma]) \} =_{is} K[\Gamma].$$

(2) The inclusion $K \supseteq und(K[\Gamma])$ is obvious. The reverse inclusion follows from Proposition 4.8. Every graph g in K admits a Γ-labelling, *i.e.* $g[\Gamma] \neq \varnothing$, because $\hat{g} =_{is} \Gamma$. Note that the statement holds under a weaker requirement: the type of every graph in K should be isomorphic with a subgraph of Γ (rather than with Γ itself). ■

We now are able to prove our main result on naked recognizable dependence graph languages.

5.3. THEOREM: *Let K be a language of acyclic graphs. K is a naked RecDG language if and only if it is a finite union $K_1 \cup \ldots \cup K_n$ such that*

(*a*) *for each $1 \leq i \leq n$, K_i is typed, and*

(*b*) *for each $1 \leq i \leq n$, $lan(K_i[\Omega])$ is regular, where Ω is a type of K_i.*

Proof: (1) Assume that K is a naked RecDG language. Thus $K = und(\langle W \rangle_\Gamma)$ for a dependence alphabet $\Gamma = (\Sigma, D)$ and $W \subseteq \Sigma^*$ such that $lan(\langle W \rangle_\Gamma)$ is regular. Let Γ_Δ be the subgraph of Γ induced by Δ, *i.e.* the dependence alphabet $(\Delta, D \cap (\Delta \times \Delta))$, let $W_\Delta = \{ w \in W : alph(w) = \Delta \}$, and let $K_\Delta = und(\langle W_\Delta \rangle_\Gamma)$. It is clear that K is the (finite) union of the languages K_Δ, $\Delta \subseteq \Sigma$.

Each of these languages is typed; the type of K_Δ is $\hat{\Gamma}_\Delta$. This is seen as follows. Each graph g in K_Δ can be labelled to become the dependence graph of a word from W_Δ. This labelling is a (*surjective*) homomorphism from g onto Γ_Δ. Hence by Corollary 4.5, $\hat{\Gamma}_\Delta =_{is} \hat{g}$.

It remains to be shown that $lan(K_\Delta[\hat{\Gamma}_\Delta])$ is regular. To this aim, let $g = und(\langle w \rangle_\Gamma)$ for some $w \in W_\Delta$. As we have seen $\hat{g} =_{is} \hat{\Gamma}_\Delta$. Hence we may apply Theorem 4.9 to obtain

$$lan(g[\hat{\Gamma}_\Delta]) = \bigcup_{\psi \in \mathrm{AUT}(\hat{\Gamma}_\Delta)} \psi(lan(\langle w \rangle_\Gamma)).$$

Since the types of the underlying graphs of dependence graphs of the words in W_Δ are isomorphic to $\hat{\Gamma}_\Delta$, we may take the union over $w \in W_\Delta$ to obtain

$$lan(K_\Delta[\hat{\Gamma}_\Delta]) = \bigcup_{w \in W_\Delta} \bigcup_{\psi \in AUT(\hat{\Gamma}_\Delta)} \psi(lan(\langle w \rangle_\Gamma = \bigcup_{\psi \in AUT(\hat{\Gamma}_\Delta)} \psi(lan(\langle W_\Delta \rangle_\Gamma))$$

Being a finite union of homomorphic images of a regular language, this language is regular. This completes the proof of the "only-if-part" of the theorem.

(2) First assume that L is a typed language of acyclic graphs, such that $lan(L[\Omega])$ is regular, where Ω is a type of L. Let $W = lan(L[\Omega])$, then by Lemma 5.2 $\langle W \rangle_\Omega =_{is} L[\Omega]$. Hence $L[\Omega]$ is a RecDG language, and consequently $L = und(L[\Omega])$ is a naked RecDG language.

In order to prove the ("if-part" of the) statement it suffices to show that naked RecDG languages are closed under (finite) union. So assume that, for $i \in \{1, 2\}$, $K_i = und(\langle W_i \rangle_{\Gamma_i})$ for a dependence alphabet $\Gamma_i = (\Sigma_i, D_i)$ and $W_i \subseteq \Sigma_i^*$ such that $lan(\langle W_i \rangle_{\Gamma_i})$ is regular. We may assume that Σ_1 and Σ_2 are disjoint (perhaps after renaming some of the elements – this does not influence the naked langages K_i). Note that $\langle W_i \rangle_{\Gamma_i} = \langle W_i \rangle_\Gamma$ for $i \in \{1, 2\}$, where $\Gamma = (\Sigma_1 \cup \Sigma_2, D_1 \cup D_2)$. Also

$$lan(\langle W_1 \cup W_2 \rangle_\Gamma) = lan(\langle W_1 \rangle_{\Gamma_1}) \cup lan(\langle W_2 \rangle_{\Gamma_2})$$

is regular, and consequently $K_1 \cup K_2 = und(\langle W_1 \cup W_2 \rangle_\Gamma)$ is a naked RecDG language. ■

5.4. *Remark:* By deleting the references to regularity in the proof of Theorem 5.3 one obtains a (rather short) proof of the following characterization of naked dependence graph languages:

Let K be a language of acyclic graphs. K is a naked dependence graph language if and only if it is a finite union of typed languages. ■

DISCUSSION

In this paper we have succeeded in characterizing naked RecDG languages. As we have explained in the introduction, dependence graphs and their languages play a fundamental role in the theory of concurrent systems and in the theory of graph grammars. Hence we feel that the present paper together with [ER] (where naked dependence graphs where characterized) constitutes a contribution to both theories.

There are (at least) two important directions that one may take continuing the research presented in this paper.

(1) Clearly one would like to obtain a characterization of (labelled) RecDG languages. The difficulty in extending Theorem 3.2 (characterizing *naked* RecDG languages) in this direction is that one may have a partition of a language K of nl graphs into classes $K_1, \ldots, K_n$, $n \geqq 1$, where, for each $1 \leqq i \leqq n$, $und(K_i)$ is typed, *but*, while some of $lan(K_i)$ are not regular, their union will yield a regular language.

(2) In the theory of concurrent systems one is often interested in transitive closures of dependence graphs (hence partial orders) rather than in dependence graphs themselves—*see*, e. g., [AR1], where it is shown that after taking transitive closures dependence graphs of condition-event systems yield precisely their *elementary event structures*. Hence one should also aim at a characterization of languages of partial orders resulting from transitive closures of naked RecDG languages.

ACKNOWLEDGEMENTS

The work presented here was carried out within ESPRIT Basic Research Actions ASMICS and DEMON. The authors are indebted to V. Diekert and W. Thomas (also members of these projects), and to two anonymous referees, for useful comments on the previous version of this paper.

REFERENCES

[AR1] IJ. J. Aalbersberg and G. Rozenberg, Theory of Traces, *Theoretical Computer Science*, Vol. *60*, 1988, pp. 1-82.

[AR2] IJ. J. Aalbersberg and G. Rozenberg, Traces, Dependency Graphs and DNLC Grammars, *Discrete Applied Mathematics*, Vol. *11*, 1985, pp. 299-306.

[AW] IJ. J. Aalbersberg and E. Welzl, Trace Languages Defined by Regular String Languages, *RAIRO Informatique Théorique*, Vol. *20*, 1986, pp. 103-119.

[BMS] A. Bertoni, G. Mauri and N. Sabadini, Equivalence and Membership Problems for Regular Trace Languages, *Lecture Notes in Computer Science*, Vol. *140*, 1982, pp. 61-71.

[CF] P. Cartier and D. Foata, Problemes combinatoires de commutation et rearrangements, *Lecture Notes in Mathematics*, Vol. *85*, 1981.

[CP] R. Cori and D. Perrin, Automates et commutations partielles, *RAIRO Informatique Théorique*, Vol. *19*, 1985, pp. 21-32.

[ENR] H. Ehrig, M. Nagl and G. Rozenberg eds., Graph Grammars and their Applications to Computer Science, *Lecture Notes in Computer Science*, Vol. *153*, 1983.

[ER] A. Ehrenfeucht and G. Rozenberg, On the structure of dependency graphs, in: *Concurrency and nets*, K. Voss, H. J. Genrich, G. Rozenberg Eds., Springer Verlag, 1987, pp. 141-170.

[FR] M. P. FLÉ and G. ROUCAIROL, *On Serizlizability of Iterated Transactions*, Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, 1982, pp. 194-200.

[K] R. M. KELLER, *A Solvable Program-Schema Equivalence Problem*, Proc. 5th Annual Princeton Conference on Information Sciences and Systems, Princeton, 1971, pp. 301-306.

[M1] A. MAZURKIEWICZ, *Concurrent Program Schemes and their Interpretations*, Dept. of Computer Science, University of Aarhus, Technical Report No. PB-78, Aarhus, 1977.

[M2] A. MAZURKIEWICZ, Semantics of Concurrent Systems: a Modular Fixed Point Approach, *Lecture Notes in Computer Science*, Vol. *188*, 1985, pp. 353-375.

[Me] Y. METIVIER, Une condition suffisante de reconnaissabilité dans un monoïde partiellement commutatif, *RAIRO Informatique Théorique*, Vol. *20*, 1986, pp. 121-127.

[O] E. OCHMANSKI, *Regular Trace Languages*, Ph.D. Thesis, Dept. of Mathematics, University of Warsaw, 1985.

[P] D. PERRIN, Partial Commutations, *Lecture Notes in Computer Science*, Vol. *372*, pp. 637-651.

[RW] G. ROZENBERG and E. WELZL, Boundary NLC grammars. Basic Definitions, Normal Forms and Complexity, *Information and Control*, Vol. *69*, 1986, pp. 136-167.