TRACES, DEPENDENCY GRAPHS AND DNLC GRAMMARS

by

IJ.J. Aalbersberg* and G. Rozenberg*

CU-CS-296-85 April, 1985

*Insitute of Applied Mathematics and Computer Science, University of Leiden, Leiden, The Netherlands.

TRACES, DEPENDENCY GRAPHS AND DNLC GRAMMARS

IJ.J.Aalbersberg
G.Rozenberg

July 1984

Institute of Applied Mathematics and Computer Science
University of Leiden
Wassenaarseweg 80, 2333 AL LEIDEN
The Netherlands

ABSTRACT

We point out the use of graph grammars for specifying (generating) languages of dependency graphs that arise in theoretical studies of concurrent systems.

INTRODUCTION

The theory of traces has been introduced in [Mz1] and has become quite popular as an approach to the theory of concurrent systems (see, e.g., [Mz2], [Mz3], [BtBbMrSb], [AR] and [AW]). In this approach strings (corresponding to observations by sequential observers) are divided into equivalence classes according to an equivalence relation induced by an (symmetric and irreflexive) independence relation describing concurrency of events within a system (hence independently from observations). Each equivalence class of strings is referred to as a trace and a set of traces is referred to as a trace language. To specify trace languages one uses string languages (specification methods of which are very well understood).

All strings within one trace describe the same structure (of partially ordered events)-this structure is referred to as a dependency graph (d-graph for short). Hence each trace language specifies a set of d-graphs (a d-graph language). Thus within the theory of traces one is really interested in d-graph languages specified (defined) indirectly by string languages.

In this paper we demonstrate how graph grammars can be used for specifying directly d-graph languages.

0. PRELIMINARIES

We assume the reader to be familiar with basic formal (string) language theory, see, e.g., [Sl2].

We use mostly standard notation and terminology; perhaps only the following points require some additional attention.

Throughout the paper only finite nonempty alphabets will be considered. Furthermore, λ denotes the empty word.

A right-linear grammar in which each production is either of the form $A \to bB$ or $A \to b$, where A and B are nonterminals and b is a terminal, will be called here a regular grammar. A regular grammar is specified in the form $G = (\Gamma,\Sigma,P,S)$, where Γ is the total alphabet, Σ is the terminal alphabet, P is the set of productions and S is the axiom. Regular languages are string languages generated by regular grammars (hence we consider only λ-free regular (string) languages).

For sets A and B, A-B denotes their difference; ϕ denotes the empty set.

A directed node labeled graph, in the sequel called simply a graph, will be specified in the form $H=(V,E,\Sigma,\ell)$, where V is its set of nodes, $E \subseteq V \times V$ is its set of edges, Σ is its label alphabet and $\ell: V \to \Sigma$ is its (node) labeling function. (We consider only graphs without loops.)

$\mathcal{G}_\Sigma$ denotes the set of all graphs with the label alphabet Σ; Λ denotes the empty graph, i.e. the graph with the empty set of nodes.

If graphs $H,\bar{H}$ are isomorphic (and we consider only the node label preserving isomporphisms), then we write $H \simeq \bar{H}$.

1. TRACES AND DEPENDENCY GRAPHS

In this section we recall basic notions concerning traces and dependency graphs (see [Mz1], [Mz2] and also [AR]).

Definition 1.1. (1) Let Σ be an alphabet. A concurrency relation over Σ is a binary relation over Σ which is irreflexive and symmetric.

(2) Let Σ be an alphabet and let C be a concurrency relation over Σ.

(2.1) The pair $\langle\Sigma,C\rangle$ is called a concurrent alphabet.

(2.2) The relation $=_C \subseteq \Sigma^* \times \Sigma^*$ is defined by: for $x,y \in \Sigma^*$, $x =_C y$ if and only if there exist $x_1,x_2 \in \Sigma^*$ and $(a,b) \in C$ such that $x = x_1abx_2$ and $y = x_1bax_2$.

(2.3) The relation $\equiv_C \subseteq \Sigma^* \times \Sigma^*$ is defined as the least equivalence relation over Σ^*, containing $=_C$; for $w \in \Sigma^*$, $[w]_C$ denotes the equivalence class of $\equiv_C$ containing w.

(2.4) A trace (on $<\Sigma,C>$) is an element of the quotient monoid $\Sigma^*/\equiv_C$ (in this quotient monoid the catenation is defined by, for $x,y \in \Sigma^*$, $[x]_C\ [y]_C = [xy]_C$ and $[\lambda]_C$ is the identity of this monoid).

(2.5) A trace language (over $<\Sigma,C>$) is a subset of the quotient monoid $\Sigma^*/\equiv_C$. □

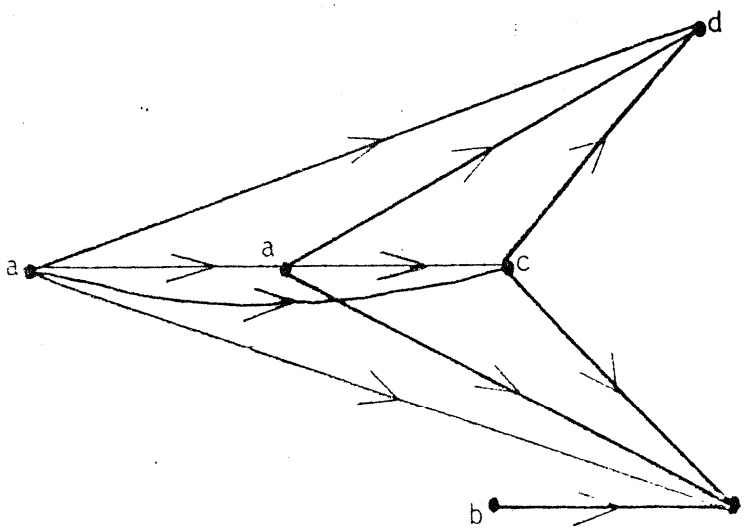
Remark. In order not to complicate our notation too much, throughout the paper we will identify, for a concurrent alphabet $<\Sigma,C>$, an element of $\Sigma^*/\equiv_C$ with the equivalence class it corresponds to. □

Example 1.1. Let $\Sigma=\{a,b,c,d,e\}$ and let $C=\{(a,b),(b,a),(b,c),(c,b),(b,d),(d,b),(d,e),(e,d)\}$. Then:
(1) $<\Sigma,C>$ is a concurrent alphabet,
(2) $adbb =_C abdb$,
(3) $acdbe \equiv_C abced$, and
(4) $[dbe]_C = \{bde,dbe,bed\}$ is a trace (on $<\Sigma,C>$). □

For a concurrent alphabet $<\Sigma,C>$, the quotient monoid $\Sigma^*/\equiv_C$ (with the catenation and identity defined as above) is referred to as the free partially commutative monoid generated by $<\Sigma,C>$, denoted by $F(\Sigma,C)$ (see, e.g., [BtBbMrSb] and [L]). Note that if $C= \Sigma\times\Sigma - \{(a,a)|a \in \Sigma\}$, then we deal with commutative monoids - language theory on commutative monoids has always been an important part of formal language theory (see, e.g., [Sl1]).

Definition 1.2. Let $<\Sigma,C>$ be a concurrent alphabet and let $x=\sigma_1....\sigma_n \in \Sigma^*$ for some $n\geq 0$ and $\sigma_1, ...,\sigma_n \in \Sigma$. The dependency graph of x (over $<\Sigma,C>$), abbreviated d-graph of x (over $<\Sigma,C>$) and denoted by $G_{<\Sigma,C>}(x)$ (or simply G(x) whenever $<\Sigma,C>$ is understood), is the graph (V,E,Σ,ℓ) where $V= \{v_1,...,v_n\}$, for all $1\leq i\leq n$, $\ell(v_i)=\sigma_i$, and, for all $1\leq i,j\leq n$, $(v_i,v_j) \in E$ if and only if $i<j$ and $(\sigma_i,\sigma_j) \notin C$. □

Example 1.2. Let $<\Sigma,C>$ be as in Example 1.1. Then G(aabced) is of the form



□

One can prove the following result (see,e.g.,[AR]).

Theorem 1.1. Let $\langle\Sigma,C\rangle$ be a concurrent alphabet and let $x,y \in \Sigma^*$. Then $x \equiv_C y$ if and only if $G(x) \simeq G(y)$.

Hence, by the above theorem, a dependency graph is really associated with (is a "signature of") a trace rather than with individual strings. This leads us to the following definition.

Definition 1.3. Let $\langle\Sigma,C\rangle$ be a concurrent alphabet and let $t \in F(\Sigma,C)$. The dependency graph of t (over $\langle\Sigma,C\rangle$), abbreviated d-graph of t (over $\langle\Sigma,C\rangle$), is the graph $G(x)$, where $x \in \Sigma^*$ is such that $t=[x]_C$; it is denoted by $G_{\langle\Sigma,C\rangle}(t)$ (or simply $G(t)$ whenever $\langle\Sigma,C\rangle$ is understood). □

For a concurrent alphabet $\langle\Sigma,C\rangle$ the set of all dependency graphs over $\langle\Sigma,C\rangle$ is denoted by $\mathcal{D}(\Sigma,C)$.

The theory of dependency graphs (over a concurrent alphabet) turns out to be quite essential within the theory of concurrent systems. It is shown in [Mz3] that several theories of concurrency are "isomorphic" to the theory of dependency graphs.

Since each trace (on a concurrent alphabet) is an equivalence class of strings, one may specify a set of traces (a trace language) by specifying a string language, each element of which defines (claims) a trace. A way of doing this is given by the following definition.

Definition 1.4. Let $\langle\Sigma,C\rangle$ be a concurrent alphabet.
(1) A trace language $T \subseteq F(\Sigma,C)$ is called regular if there exists a regular string language $K \subseteq \Sigma^*$ such that $T=\{[x]_C \mid x \in K\}$.
(2) A d-graph language (over $\langle\Sigma,C\rangle$) is a subset of $\mathcal{D}(\Sigma,C)$.
(3) A d-graph language $D \subseteq \mathcal{D}(\Sigma,C)$ is called regular if there exists a regular trace language $T \subseteq F(\Sigma,C)$ such that $D=\{G(t) \mid t \in T\}$. □

2. DNLC GRAMMARS

In this section we recall the definition of a class of graph grammars which is especially suitable for generating d-graph languages.

Since each trace (over a concurrent alphabet) has a uniquely associated dependency graph, a way to specify a d-graph language is to specify a trace language, each trace of which defines a d-graph (see Definition 1.4) - this is an indirect method. One can also try a direct method, that is to specify a d-graph language D by giving a graph grammar generating all, and only, d-graphs of D.

It turns out that DNLC grammars (see, e.g., [JR]) are particulary suitalbe for generating languages of d-graphs. In the rest of this paper we try to justify this contention.

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

Definition 2.1. (1) A directed node-label controlled graph grammar, abbreviated DNLC grammar, is a system $G=(\Gamma,\Delta,P,C_{in},C_{out},Z)$, where:
(i) Γ is an alphabet, called the total alphabet of G,
(ii) $\Delta \subseteq \Gamma$ is called the terminal alphabet of G,
(iii) $P \subseteq (\Gamma-\Delta) \times G_\Gamma$ is called the set of productions of G,
(iv) $C_{in} \subseteq \Gamma\times\Gamma$ is called the in-connection relation of G and $C_{out} \subseteq \Gamma\times\Gamma$ is called the out-connection relation of G, and
(v) Z, called the axiom of G, is a graph over Γ, consisting of one node labeled by an element of $\Gamma-\Delta$. □

Informally speaking, a DNLC grammar $G=(\Gamma,\Delta,P,C_{in},C_{out},Z)$ generates a set of graphs as follows.

Given a graph H to be rewritten and a production of the form (a,F), where a is a node-label and F is a graph, one chooses a node v of H labeled by a and replaces it by (a graph isomorphic to) F. Then, in order to embed F in "the rest of H" (the graph resulting from H by removing v) one

uses relations C_{in} and C_{out} as follows. For every pair $(b,c) \in C_{in}$ one establishes an (incoming) edge from each direct neighbour node of v labeled c to each node of F labeled b. Analogously, for every pair $(b,c) \in C_{out}$ one establishes an (outgoing) edge from each node labeled b in F to each direct neighbour node of v labeled c.

Every graph H' isomorphic to the resulting graph is said to be directly derived from H in G. This is written $H \overset{}{\underset{G}{\Rightarrow}} H'$. Iterating the direct derivation step, starting with the axiom graph Z of G, and choosing only derived graphs such that all their nodes are labeled by the labels from the terminal alphabet Δ, one gets the (graph) language L(G) of G.

These notions are defined formally in [JR].

Definition 2.2. Let $G=(\Gamma,\Delta,P,C_{in},C_{out},Z)$ be a DNLC grammar. G is called a regular DNLC grammar if every production of G is

either of the form $(X, \overset{\sigma}{\bullet} \!\!\rightarrow\!\!\!-\!\!\bullet\, Y)$,

or of the form $(X, \overset{\sigma}{\bullet})$,

with $\sigma \in \Delta$ and $Y \in \Gamma-\Delta$. □

Definition 2.3. Let $<\Sigma,C>$ be a concurrent alphabet and let $G=(\Gamma,\Delta,P,C_{in},C_{out},Z)$ be a DNLC grammar. G is called $<\Sigma,C>$ - consistent if:

(i) $\Delta = \Sigma$,

(ii) $C_{in}=(\Gamma\times\Gamma)-C$, and

(iii) $C_{out}= \emptyset$. □

3. DNLC GRAMMARS AND D-GRAPH LANGUAGES

In this section we demonstrate the use of DNLC grammars for the generation of regular d-graph languages.

It turns out that, for a concurrent alphabet $<\Sigma,C>$, any regular subset of $\mathcal{D}(\Sigma,C)$ can be generated by a regular $<\Sigma,C>$ - consistent DNLC grammar and moreover regular $<\Sigma,C>$ - consistent DNLC grammars generate only regular subsets of $\mathcal{D}(\Sigma,C)$.

Theorem 3.1. Let $<\Sigma,C>$ be a concurrent alphabet and let $H \subseteq \mathcal{D}(\Sigma,C)$. H is regular if and only if H can be generated by a regular $<\Sigma,C>$ - consistent DNLC grammar.

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

(i) Let $H \subseteq \mathcal{D}(\Sigma,C)$ be a regular d-graph language. Hence there exists a regular (string) grammar $G_1= (\Gamma,\Sigma,P_1,S)$ such that $H= \{G([x]_C) \mid x \in L(G_1)\}$.

Consider the regular $<\Sigma,C>$ - consistent DNLC grammar $G_2=(\Gamma,\Sigma,P_2,C_{in},C_{out},Z)$, where:

$P_2= \{(X, \overset{\sigma}{\bullet} \!\!\rightarrow\!\!\!-\!\!\bullet\, Y) \mid (X,\sigma Y) \in P_1$ for some $X,Y \in \Gamma-\Sigma$ and $\sigma \in \Sigma\}$

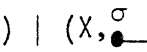
$\cup\{(X, \overset{\sigma}{\bullet}) \mid (X,\sigma) \in P_1$ for some $X \in \Gamma-\Sigma$ and $\sigma \in \Sigma\}$, and

the label of the node of Z is S.

Now it is clear that $H=L(G_2)$.

(ii) Let $G_1=(\Gamma,\Sigma,P_1,C_{in},C_{out},Z)$ be a regular $<\Sigma,C>$ - consistent DNLC grammar.

Consider the regular (string) grammar $G_2=(\Gamma,\Sigma,P_2,S)$, where:

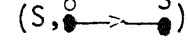
$P_2=$ $\{(X,\sigma Y) \mid (X,$  $) \in P_1$ for some $X,Y \in \Gamma-\Sigma$ and $\sigma \in \Sigma\}$

$\cup \{(X,\sigma) \mid (X,$  $) \in P_1$ for some $X \in \Gamma-\Sigma$ and $\sigma \in \Sigma\}$, and

S is the label of the node of Z.

From the construction of a d-graph it easily follows that $L(G_2)$ is a regular d-graph language over $<\Sigma,C>$.

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

As we have indicated already, the set of all d-graphs over a given concurrent alphabet forms an important object within the theory of concurrent systems (see,e.g., [Mz2] and [Mz3]). As a corollary of the above result we know that $\mathcal{D}(\Sigma,C)$ (for a given concurrent alphabet $<\Sigma,C>$) can be generated by a regular ($<\Sigma,C>$ - consistent) DNLC grammar. Actually it is worth noticing that (for a given concurrent alphabet $<\Sigma,C>$) $\mathcal{D}(\Sigma,C)$ (up to Λ) is generated by the following (very simple) regular $<\Sigma,C>$ - consistent DNLC grammar $G=(\Gamma,\Sigma,P,C_{in},C_{out},Z)$, where $\Gamma-\Sigma=\{S\}$ and P consists of the productions $(S,$  $)$ and $(S,$  $)$ for all $\sigma \in \Sigma$.

4. DISCUSSION

In this paper we have demonstrated how to use graph grammars for generating regular d-graph languages. Clearly, this paper is only the beginning of the investigation of the use of graph grammars (in particular DNLC grammars) in the study of traces.

In particular there are several research topics that, in our opinion, should be investigated now.

(i) We have considered here only one specific way of defining regular trace languages (and consequently regular d-graph languages) - this approach is referred to (in [AR]) as an "existential" approach. The usage of d-graph languages, where the notion of "regular" is established by using other than existential ways of claiming,is a natural next step of a systematic investigation.

(ii) Two transformations of (the languages of) d-graphs seem to be quite obvious from the point of view of the theory of partial order (-like) relations: consider only "clean Hasse diagrams" (as, e.g., in [R]) of d-graphs or consider only "transitive closures" of d-graphs. The usage of graph grammars to generate these "versions" of (the languages of) d-graphs should be investigated.

iii) In this paper we have pointed out a very natural connection between regular d-graph languages and regular DNLC grammars. It can be shown, that if one wants to generate more general classes of d-graph languages (e.g., context-free), then the concept of a DNLC grammar must be modified. The investigation of the ways to generate context-free d-graph languages using graph grammars (in particular modified versions of DNLC grammars) seems to be quite important.

We believe that a systematic study of the interrelations between the theory of graph grammars and the theory of traces would benefit both, (i) the theory of concurrent systems - by providing a new method of specifying non-sequential behaviour, and (ii) the theory of graph grammars by indicating well-motivated classes of graph grammars. We hope to report on our investigation concerning this relationship, in particular concerning the research topics listed above, in the near future.

REFERENCES

[AR] Aalbersberg, IJ.J. and Rozenberg, G., Trace theory - a survey, Techn.Rep., Inst. of Appl.Math. and Comp.Sc., Leiden, 1984.

[AW] Aalbersberg, IJ.J. and Welzl, E., Trace languages defined by regular string languages, Techn. Rep. 84-17, Inst. of Appl. Math. and Comp. Sc., Leiden, 1984.

[BtBbMrSb] Bertoni, A., Brambilla, M.,Mauri, G. and Sabadini, N., An application of the theory of free partially commutative monoids: asymptotic densities of trace languages, Lecture Notes in Comp. Science, Vol.118, pp.205-215, Springer, Berlin, 1981.

[JR] Janssens, D. and Rozenberg, G., A characterization of context-free string languages by directed node-label controlled graph grammars, Acta Informatica, Vol.16, pp 63-85, Springer, Berlin,1981.

[L] Lallement, G., Semigroups and combinatorial applications, J. Wiley and Sons, New York, 1979.

[Mz1] Mazurkiewicz, A., Concurrent program schemes and their interpretations, DAIMI Rep. PB-78, Aarhus University, 1977.

[Mz2] Mazurkiewicz, A., Semantics of concurrent systems: a modular fixed-point trace approach, Techn. Rep. 84-19, Inst. of Appl. Math. and Comp.Sc., Leiden, 1984.

[Mz3] Mazurkiewicz, A., Traces, histories, graphs: instances of a process monoid, Lecture Notes in Comp. Science, Vol.176, pp.115-133, Springer, Berlin, 1984.

[R] Reisig, W., Petri netze, eine Einführung, Springer Verlag, Berlin-Heidelberg, 1982.

[Sl1] Salomaa, A., Theory of automata, Pergamon Press, Oxford-New York, 1969.

[Sl2] Salomaa, A., Formal languages, Academic Press, New York, 1973.