# ACTOR GRAMMARS

D. Janssens and G. Rozenberg

CU-CS-407-88 August 1988

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

# ACTOR GRAMMARS

D. Janssens

Dept. WNIF, University of Limburg, L.U.C.,

Universitaire Campus, B-3610 Diepenbeek, Belgium

G. Rozenberg

Institute for Applied Mathematics and Computer Science

Rijksuniversiteit Leiden,

Niels Bohrweg 1, PB 9512, 2300 RA Leiden, The Netherlands

ABSTRACT Actor systems are a model of massively parallel systems based on asynchronous message passing. This paper presents a formalism for actor systems in the framework of graph grammars. To this aim actor grammars are introduced, motivated, and illustrated by examples. Some of the basic properties pertinent to graph transformations in actor grammars are discussed.

Key words : massive parallelism, actor systems, handle rewriting, actor grammars.

CONTENTS

Introduction

1. Preliminaries
2. Actor systems
3. Actor vocabularies and configuration graphs
4. Structured transformations and their composition
5. Actor grammars
6. Relationship to the approach from [JR87]
7. Examples of structured transformations
8. Discussion

## INTRODUCTION

The term "actor system" refers to a model of concurrent computation. It is based on asynchronous message passing and it has been introduced in [H77] and [HB77]. Most of the papers on actor systems present them in a rather informal way (see, e.g., [H77], [HB77], [L81] and [T83]), although there are also papers providing a formal framework (see, e.g., [C81] and [A86]).

The aim of this paper is to present a formal framework for (a version of) actor systems, which is based on the use of graph grammars. One of the basic decisions to be made in setting up such a framework is the choice of a suitable representation of a configuration (a "snapshot") of an actor system. It seems rather natural to represent a configuration by a graph. Hence to each configuration corresponds a graph and to a computation in an actor system corresponds a graph transformation. Since graph grammars provide an elegant way to describe such transformations (see e.g., [ENRR87], [E87] and [R87]), we use them to model the dynamic behaviour of actor systems. In particular, we introduce a new kind of graph grammars, called actor grammars, that are tailored for the modeling of actor systems. The central notion discussed in this paper is the notion of a structured transformation in an actor grammar, which formalizes a computation in an actor system. Throughout the paper we motivate our choice of various components of actor grammars and we investigate the fundamental properties of structured transformations.

The paper is organized as follows.

The basic mathematical terminology and notation is given in Section 1.

In Section 2 we provide a short introduction to actor systems.

In Section 3 we introduce the notion of an actor vocabulary and of a configuration graph. The former is needed to provide the supply of identifiers and values needed in the actor system considered, and the latter formalizes a "snapshot"

of the system.

In Section 4 we introduce a methodology for the formal treatment of the graph grammars we need in this paper, define the notion of a structured transformation, and investigate basic properties of structured transformations.

In Section 5 we introduce the notion of a primitive event transformation, which is a structured transformation that corresponds to an event in an actor system. Then we define the notion of an actor grammar, and we prove that structured transformations in actor grammars lead from configuration graphs to configuration graphs.

In Section 6 we explain the relationship between the formalism given in this paper and the one given in [JR87].

Section 7 supplies examples of structured transformations and the operations defined on them.

We conclude the paper by a short discussion, in Section 8.

## 1. PRELIMINARIES

In this section we recall some basic notions and terminology concerning graphs and sets. This allows us to set up a notation suitable for this paper.

### Sets and relations

(1) For a set X, $Id_X$ denotes the identity relation on X; for sets X,Y, X-Y denotes the difference of X and Y. The number of elements of a finite set X is denoted by $\# X$.

(2) Let A, B be sets and let $R \subseteq A \times B$. Then $R^{-1}$ denotes the inverse of R, and, for each subset C of A, $R(C) = \{y \in B \mid \text{there exists an } x \in C \text{ such that } (x,y) \in R\}$. If $C = \{x\}$, then we write R(x) instead of R(C). The domain and the range of R are denoted by *Dom*(R) and *Ran*(R), respectively. Hence $Dom(R) = R^{-1}(B)$ and $Ran(R) = R(A)$.

(3) Let A, B and C be sets, let $R \subseteq A \times B$ and let $S \subseteq B \times C$. Then $S \circ R$ denotes the composition of R and S (first R, then S).

(4) A relation $R \subseteq A \times B$ is *injective* if, for each $b \in B$, $\# R^{-1}(b) \leq 1$; R is *surjective* if $Ran(R) = B$.

(5) If $R \subseteq A \times B$ and $R' \subseteq A' \times B'$, where $A \cap A' = \emptyset$, then $R \oplus R' = R \cup R'$.

(6) A relation $R \subseteq A \times B$ is called a *partial function* if $R^{-1}$ is injective. It is called a *function* if, moreover, $Dom(R) = A$.

### Graphs

Let $\Sigma$ and $\Delta$ be finite sets.

(1) For a finite set X, the *set of* $\Delta$*-labeled edges over* X, denoted by $Edges(X,\Delta)$, is the set $X \times \Delta \times X$.

(2) A $(\Sigma,\Delta)$-*labeled graph* is a system $g = (V,E,\phi)$ where $V$ is a finite set (called the *set of nodes of* g), $E \subseteq Edges(V,\Delta)$ (called the *set of edges of* g), and $\phi$ is a function from $V$ into $\Sigma$ (called the *node-labeling function of* g). For a $(\Sigma,\Delta)$-labeled graph g, its set of nodes, its set of edges and its node-labeling function are denoted by $Nd(g)$, $Ed(g)$ and $\phi_g$, respectively.

(3) Let g and h be $(\Sigma,\Delta)$-labeled graphs.

(3.a) h is a *subgraph of* g if $Nd(h) \subseteq Nd(g)$, $Ed(h) \subseteq Ed(g)$ and $\phi_h$ is the restriction of $\phi_g$ to $Nd(h)$. For a subgraph h of g, g-h denotes the graph $(Nd(g)-Nd(h),\ Ed(g) \cap Edges(Nd(g)-Nd(h),\ \Delta),\ \phi')$, where $\phi'$ is the restriction of $\phi_g$ to $Nd(g) - Nd(h)$.

(3.b) g is an *augmentation* of h if h is a subgraph of g such that $Nd(h) = Nd(g)$ (i.e. g may have more edges than h). The class of all augmentations of h is denoted by $Aug(h)$.

(3.c) The graphs g and h are disjoint if $Nd(g) \cap Nd(h) = \emptyset$.

(3.d) Let g and h be disjoint. Then $g \oplus h$ is the $(\Sigma,\Delta)$-labeled graph $(Nd(g) \cup Nd(h),\ Ed(g) \cup Ed(h),\ \phi_g \oplus \phi_h)$.

(4) Let g be a $(\Sigma,\Delta)$-labeled graph.

(4.a) The *source function of* g is the function $s_g$ from $Ed(g)$ into $Nd(g)$ defined by $s_g(a,\delta,b) = a$, for each $(a,\delta,b) \in Ed(g)$. The *target function of* g is the function $t_g$ from $Ed(g)$ into $Nd(g)$ defined by $t_g(a,\delta,b) = b$, for each $(a,\delta,b) \in Ed(g)$. The *edge-labeling function of* g is the function $\psi_g$ from $Ed(g)$ into $\Delta$, defined by $\psi_g(a,\delta,b) = \delta$, for each $(a,\delta,b) \in Ed(g)$.

(4.b) The *edge-universe of* g, denoted by $Eun(g)$, is the set $Edges(Nd(g),\Delta)$. Note that $Eun(g)$ depends on the choice of $\Delta$. This should not lead to confusion : whenever talking about graphs throughout this paper, we explicitly indicate the alfabets $\Sigma$ and $\Delta$ involved.

(4.c) The graph g is *discrete* if $Ed(g) = \emptyset$.

(4.d) Let $\delta \in \Delta$. A *$\delta$-handle of* g is a subgraph h of g such that $Nd(h) = \{x,y\}$ and $Ed(h) = \{(x,\delta,y)\}$, for some $x,y \in Nd(g)$ where $x \neq y$ (i.e., a $\delta$-handle is a subgraph consisting of two distinct nodes and a $\delta$-edge connecting them).

(4.e) Let $v \in Nd(g)$ and $\delta \in \Delta$. The *$\delta$-span of* v *in* g, denoted by $\delta_g(v)$, is the set $\{e \in Ed(g) \mid s_g(e) = v \text{ and } \psi_g(e) = \delta\}$.

(5) Let g and h be $(\Sigma,\Delta)$-labeled graphs, and let $\xi : Nd(g) \to Nd(h)$ be a function.

(5.a) $\xi_{ed}$ denotes the function from $Eun(g)$ into $Eun(h)$ defined by $\xi_{ed}(v,\delta,w) = (\xi(v), \delta, \xi(w))$, for each $v, w \in Nd(g)$ and each $\delta \in \Delta$.

(5.b) $\xi_\Delta$ denotes the function from $Nd(g) \times \Delta$ into $Nd(h) \times \Delta$ defined by $\xi_\Delta(v,\delta) = (\xi(v),\delta)$, for each $v \in Nd(g)$ and each $\delta \in \Delta$.

(5.c) $\xi$ is a *node-label preserving* function if, for each $v \in Nd(g)$, $\phi_g(v) = \phi_h(\xi(v))$.

## 2. ACTOR SYSTEMS

*Actor systems* have been introduced in [H77] and [HB77] as a model of concurrent computation suited for describing the execution of programs on massively parallel computers; a more formal treatment of actor systems is presented in [C81], [T83] and [A86]. In this paper we present a formal framework for actor systems based on graph rewriting. We start by discussing, in this section, the notions (and intuitions behind them) of actor systems which determine our formal model.

The basic elements of an actor system are *actors* and *messages*. Actors are active objects which process messages, where processing a message may result in creating new actors and/or sending messages to other actors. Each actor has, at each moment, a specific *behaviour*, which determines how the actor reacts to messages arriving at this moment. As a consequence of processing a message an actor may change its behaviour. Hence the behaviour of an actor at a given moment may be viewed as a "local memory" of that actor; this local memory is unaccessible to all other actors. On the other hand, the way an actor reacts to a particular message depends on the information contained in the message. The behaviour of an actor (the contents of its local memory) will be represented by its *state* and the information contained in a message will be represented by its *value*.

All communications between actors are realized by messages. These communications are asynchronous : when an actor sends a message to another actor, it does not have to wait for an answer, but it can immediately proceed to process another message. One does not make explicit assumptions about the order in which messages arrive at their destination. The communication pattern between actors is restricted by the assumption that, at each moment, a given actor can send messages to a limited number of other actors only. If, as it is often done, one compares an actor system to a mailing system, then an actor can, at a given moment, send messages only to those actors of which it contains the mailing address. It is assumed that these mailing addresses may be transmitted between actors by including them in messages (thus dynamically changing the communication pattern). Hence both,

actors and messages, contain mailing addresses of actors. The actors to which a given actor or message contains a reference, are called the *acquaintances* of that actor or message.

In our formalism, the acquaintance-relationship will be given explicitly, separately from information about actor states and message values. Consequently, our description of both an actor and a message will contain two kinds of information : on the one hand, the actor state (in the case of an actor) or the message value (in the case of a message), and, on the other hand, the acquaintances. In the case of a message, our description may also contain a (unique) *destination* actor : the actor to which the given message has been sent.

The processing of a message by an actor is considered to be an "atomic action" in the system; it is called an *event*. It is assumed that several actors may process their messages concurrently; however, each actor processes only one message at a time. The event consisting of the processing of a message M by an actor A is denoted by $[M \rightarrow A]$. The processing of M by A, based on the behaviour of A, is sometimes referred to in the literature as the execution of a *script*. Hence the script specifies the actions to be taken when an actor A processes a message M, taking into account the actor state of A and the message value of M. One or more of the following actions may take place : (1) the actor state of A may change, (2) the set of acquaintances of A may change, (3) new actors may be created, and (4) new messages may be sent.

If more than one of these actions take place, then their order is arbitrary. The script describes how A changes (giving its updated actor state and acquaintances), which new actors are created (giving, for each of them, its actor state and its acquaintances) and which messages are to be sent (giving, for each of them, its message value, its destination and its acquaintances). In the actor paradigm it is assumed that, in computing the effect of the event $[M \rightarrow A]$, the script uses only information that is "locally available" in M or A; more precisely, the only "mailing addresses" that may be used in specifying the acquaintances and

destinations of newly created actors and messages, and in updating the acquaintances of the original actor A, are the addresses of (i) the actor A, (ii) the acquaintances of A, (iii) the acquaintances of M and (iv) the newly created actors (the actors of (i) through (iv) are called the *participants of the event* [M → A]).

To distinguish between the various acquaintances of A and M, the script uses *acquaintance names*. We will assume that the set of acquaintance names is partitioned into two disjoint subsets : the set of *actor acquaintance names* and the set of *message aquaintance names.* Actor acquaintance names are used to distinguish between acquaintances of actors, and message acquaintance names are used to distinguish between acquaintances of messages (message acquaintance names were called "communication names" in [JR87]). Informally speaking (using the theater-like terminology initiated already by the usage of terms like "actor" and "script"), acquaintance names distinguish between the different "roles" assigned to (to be played by) the various acquaintances in the script. It is assumed that roles are uniquely assigned, and hence no two acquaintances of an actor (or a message) have the same actor acquaintance name (message acquaintance name, respectively). However, an actor A can be an acquaintance of several actors or messages $X_1$, $X_2, \ldots, X_n$. In this case, each of the $X_i$'s assigns its own acquaintance name to A. Hence the assignment of names happens locally in each of the $X_i$'s. If X is an actor (or a message), B is one of its acquaintances and B corresponds, with respect to X, to the acquaintance name $\delta$, then B is called the *$\delta$-acquaintance of* X - this means that B plays the "role" $\delta$ in the script describing the behaviour of the actor X (or the behaviour of the actor processing X, if X is a message).

We can now express the basic *finiteness condition* about actor systems that we assume in this paper : it is assumed that there are fixed finite sets of actor states, message values, actor acquaintance names and message acquaintance names for the whole system.

The following example illustrates the way an actor system operates.

*Example 1.*

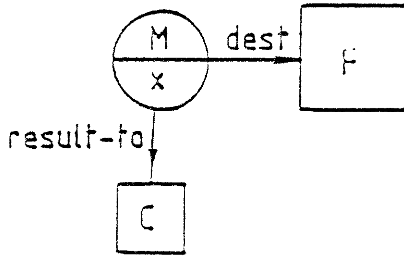
Let $\mathbb{R}$ be a finite subset of the real numbers and let $g_1, g_2 : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}^2 \to \mathbb{R}$ be functions. Let $f : \mathbb{R} \to \mathbb{R}$ be defined by $f(x) = h(g_1(x), g_2(x))$. Fig. 1 illustrates how the computation of the value $f(x)$ may be organized in an actor system - the computation is carried out in four stages.

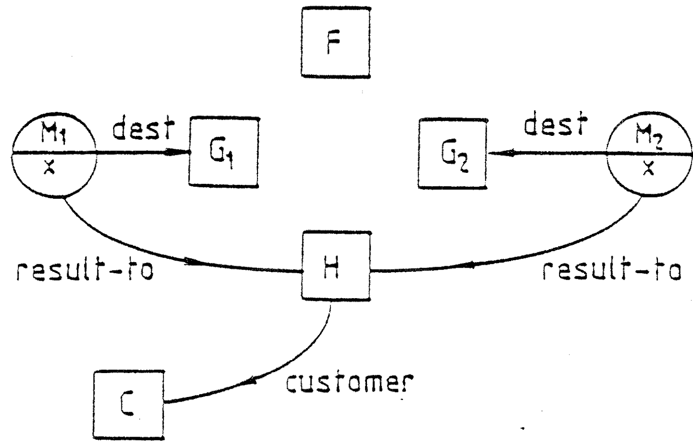
(1) The computation is initiated by sending a message M to an actor F, which is "specialized" in computing f. M contains the argument, x, and a reference to the actor C to which the result of the computation has to be sent. In F's script this reference is designated by the message acquaintance name "result-to" - hence C is the "result-to" - acquaintance of M.

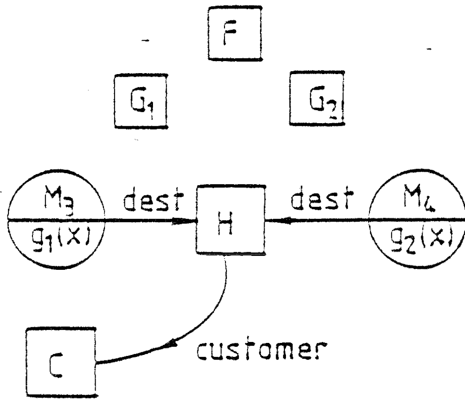
(2) As a result of processing M, F creates three new actors : $G_1$, $G_2$ and H. These are specialized in computing $g_1$, $g_2$ and h, respectively. Moreover, messages $M_1$ and $M_2$ are sent to $G_1$ and $G_2$ respectively. Both $M_1$ and $M_2$ contain x and the "result-to" - acquaintance of both $M_1$ and $M_2$ is H. C becomes the "customer" - acquaintance of H (later H will send its result to its customer-acquaintance).

(3) $G_1$ and $G_2$ process $M_1$ and $M_2$, respectively, and send messages $M_3$ and $M_4$, containing the values $g_1(x)$ and $g_2(x)$, respectively, to H.

(4) H processes $M_3$ and $M_4$, computes $h(g_1(x), g_2(x))$, and sends the result to its "customer"-acquaintance, i.e., to C, in a message $M_5$.



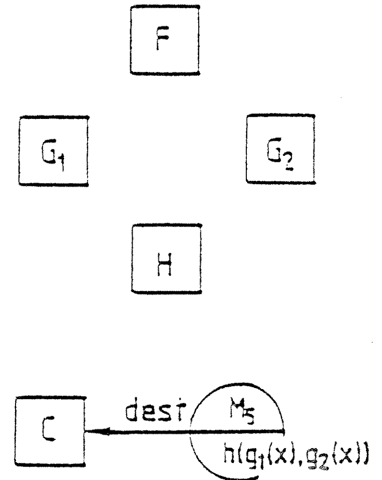




Figure 1.

Above we gave an informal description of an actor system and the way it operates. In order to get a *formal* description we will proceed as follows.

(1) First, we will define the notion of a *vocabulary* for an actor system - this will simply formalize the global supply of all actor states, all message values, all actor acquaintance names, and all message acquaintance names in the system.

(2) Then we define the notion of a *configuration graph* (for the given vocabulary) - this formalizes the notion of a possible "snapshot" of the system.

(3) Finally, we formalize the way that configuration graphs are transformed into each other. In order to do so, we will use a (specific kind of) graph grammar, called *actor grammar*.

## 3. ACTOR VOCABULARIES AND CONFIGURATION GRAPHS

In this section we define the notions of an actor vocabulary and a configuration graph. An actor vocabulary formalizes the global supply of all actor states, all message values and all acquaintance names in an actor system. On the other hand, a *configuration graph* is a (node and edge labeled, directed) graph that formalizes an instantaneous description (a snapshot) of an actor system. Before we give the formal definitions, we list the various elements of such an instantaneous description and we indicate how each of them is represented in the corresponding configuration graph.

(1) The basic elements of an actor system are actors and messages. Hence an instantaneous description should specify the sets of actors and messages present at a given moment. In a configuration graph, actors and messages are represented by *nodes* : to each actor and to each message corresponds a node.

(2) At each moment, each actor is in a certain actor state and each message contains a certain message value. In a configuration graph, actor states and message values are represented by *node labels*. We assume that the sets of node labels corresponding to actor states and message values, respectively, are disjoint. Hence one can distinguish nodes corresponding to actors from nodes corresponding to messages by their labels.

(3) At each moment, each actor (or message) X contains a set of mailing addresses of actors (called the acquaintances of X). The fact that an actor A is an acquaintance of X is represented in a configuration graph by a *directed edge from the node corresponding to X to the node corresponding to A.*

(4) At each moment, each acquaintance of an actor (or a message) X corresponds to a certain acquaintance name. Acquaintance names are represented by *edge labels* in a configuration graph : if A is the $\delta$-acquaintance of X, then there is a $\delta$-labeled edge from the node corresponding to X to the node corresponding to A.

(5) At each moment, each message has at most one destination actor. Hence we allow messages that have no destination - such a situation arises when an actor tries to send a message to its $\delta$-acquaintance at a moment when it has no $\delta$-acquaintance. Note that messages without a destination were not allowed in [JR87]. We refer to Section 6 for a discussion of this point. In a configuration graph, destinations are indicated by *edges labeled by the "special symbol"* $\emptyset$ : if A is the destination of M, then there is a $\emptyset$-labeled edge from the node corresponding to M to the node corresponding to A.

The notion of an actor vocabulary is defined as follows.

*Definition 1.* An *actor vocabulary* is a 4-tuple (A,M,AQ,MQ), where A, M, AQ and MQ are finite sets such that $A \cap M = \emptyset$, $AQ \cap MQ = \emptyset$ and $\emptyset \in MQ$. □

For an actor vocabulary S=(A,M,AQ,MQ), the sets A and M are called the *set of actor states of* S and the *set of message values of* S, respectively. The sets AQ and MQ are called the *set of actor acquaintance names of* S and the *set of message acquaintance names of* S, respectively. Also, A, M, AQ, and MQ are denoted by $A_S$, $M_S$, $AQ_S$, and $MQ_S$, respectively. Furthermore, the *set of node labels of* S, denoted by $\Sigma_S$, is the set $A_S \cup M_S$, and the *set of edge labels of* S, denoted by $\Delta_S$, is the set $AQ_S \cup MQ_S$. An *S-graph* is a $(\Sigma_S,\Delta_S)$-labeled graph. For an S-graph g, *the set of actor nodes of* g, denoted by $Act(g)$, is the set $\{v \in Nd(g) \mid \phi_g(v) \in A_S\}$ and the *set of message nodes of* g, denoted by $Msg(g)$, is the set $\{v \in Nd(g) \mid \phi_g(v) \in M_S\}$.

The notion of a configuration graph is defined as follows.

*Definition 2.* Let S be an actor vocabulary. An *S-configuration graph* is an S-graph g such that

(1) $Ed(g) \subseteq (Msg(g) \times MQ_S \times Act(g)) \cup (Act(g) \times AQ_S \times Act(g))$, and

(2) for each $v \in Nd(g)$ and each $\delta \in \Delta_S$, $\# \delta_g(v) \leq 1$. □

*Example 2.*

Let S be the actor vocabulary (A, M, AQ, MQ), where

A = $\{a_1, a_2, a_3\}$, M = $\{m_1, m_2, m_3\}$, AQ = $\{\alpha, \beta, \gamma\}$ and MQ = $\{\sigma, \tau, \emptyset\}$. Then the graph of Fig. 2 is an S-configuration graph.

Throughout the paper we use the following conventions for the pictorial representation of S-graphs.

(1) Actor nodes are drawn as boxes and message nodes are drawn as circles.

(2) The identity of a node is denoted inside the corresponding box or circle.

(3) The label of a node is denoted outside of the correponding box or circle.

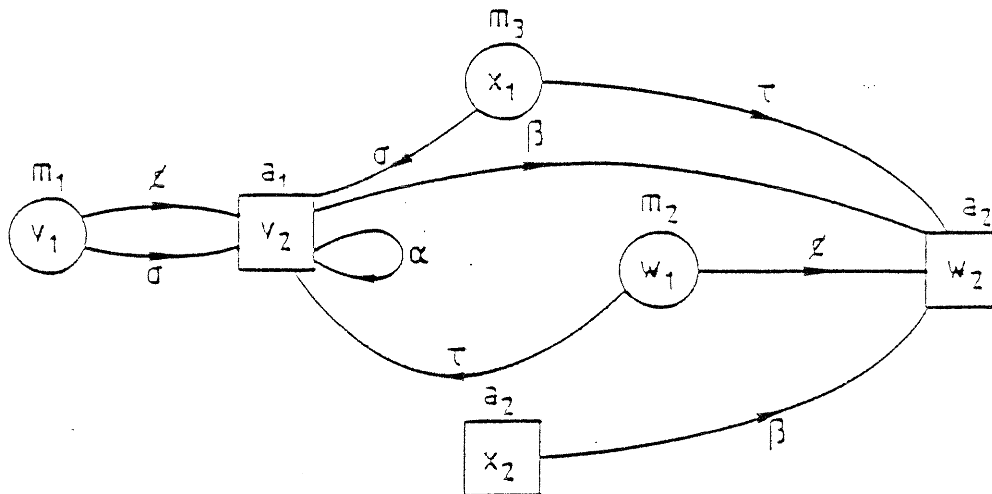



Figure 2.

The class of all S-configuration graphs is denoted by $Conf(S)$. The following technical result follows directly from the definition.

*Lemma 1.*

(1) Let g, g′ ∈ $Conf(S)$ be disjoint. Then $g \oplus g' \in Conf(S)$.

(2) Let $g \in Conf(S)$ and let h be a subgraph of g. Then $h \in Conf(S)$.

(3) Let $g \in Conf(S)$ and let $(x, \emptyset, y) \in Ed(g)$. Then, for each $\delta \in \Delta_S$, $\#(\delta_g(x) \cup \delta_g(y)) \leq 1$. □

Observe that, unlike in [JR87], we do not require that, in a configuration graph, each message node has an outgoing $\emptyset$-edge - see Section 6 for a discussion of this issue.

In this section we have described how an instantaneous description of an actor system can be represented by a graph, called a configuration graph. The dynamic evolution of an actor system will be described in terms of transformations of configuration graphs (considered in the next section). Graph grammars provide an elegant way to formalize these transformations. A particular type of graph grammars, called *actor grammars*, will be introduced (in Section 5) for this purpose.

## 4. STRUCTURED TRANSFORMATIONS AND THEIR COMPOSITIONS

In this section we introduce a methodology for handling graph transformation in the particular type of graph grammars used in this paper. Since this methodology may be useful in a more general study of graph grammars, we will introduce it without specific references to actor systems. In order to explain its main lines, we first recall some basic notions from the standard approach for handling graph grammars.

Within the standard approach (see, e.g., [ENRR87]), a graph grammar consists basically of two components : a finite set of *productions* and an *embedding mechanism*. Roughly speaking, a graph grammar production is a pair $\pi = (\alpha,\beta)$ of graphs; $\alpha$ is called the *left-hand side of* $\pi$ and $\beta$ is called the *right-hand side of* $\pi$. The production $(\alpha,\beta)$ is "applied" to a graph g by replacing an occurrence of $\alpha$ in g (i.e., a subgraph of g isomorphic to $\alpha$) by an isomorphic copy, $\beta'$, of $\beta$. However, an obvious problem arises in doing so : how should $\beta'$ be embedded into the remaining part of g; i.e., which edges should be established between nodes of $\beta'$ and nodes of $g$-$\alpha$ ? For a given graph grammar these "embedding edges" are determined by the embedding mechanism - usually, one may see these edges as being "inherited" from g. The situation is illustrated by Fig. 3, where the embedding edges are given by dotted lines.

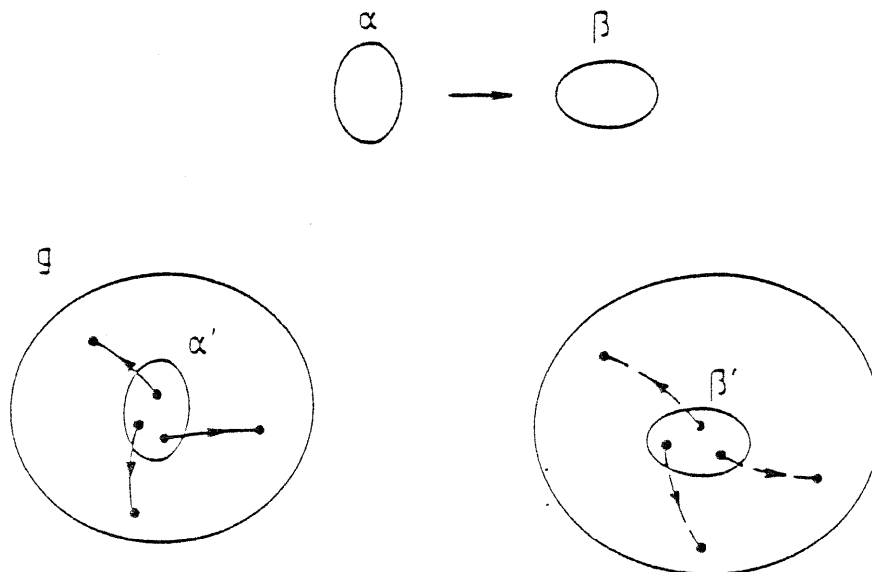



Figure 3.

The methodology used in this paper, however, is somewhat different. The idea is that the productions of a grammar specify primitive transformations, which are considered to be "valid", and that one defines a number of operations which, when applied to valid transformations, yield valid transformations. For each transformation, its initial graph and its result graph are given explicitly, together with a detailed description of the relationship between the nodes of the initial graph and the result graph - therefore such a transformation will be called a *structured transformation* throughout this paper. One may view a structured transformation as a description of the effect of a rewriting process which, when applied to its initial graph, yields its result graph. To rewrite a given graph g, one has to use a structured transformation the initial graph of which equals g - such a structured transformation will be called *applicable to* g.

In this new methodology the embedding mechanism is incorporated in the notion of a structured transformation. The operations on structured transformations are defined in such a way that, when a new structured transformation d is constructed from existing ones, the embedding mechanism of d is constructed from the

embedding mechanisms of the structured transformations d is built from. At the end of this section we explain how the "application to a graph" of a structured transformation is described in our formalism, and we briefly discuss the notion of a derivation in this context.

For a pair of finite alphabets $(\Sigma,\Delta)$, a $(\Sigma,\Delta)$-structured transformation consists of an initial graph and a result graph, together with an "identification function" and an "embedding relation". The latter two describe a relationship between nodes of the initial graph and nodes of the result graph. More specifically, the identification function shows which nodes of the result graph are identified with nodes of the initial graph - informally speaking, these are "old" nodes, in contrast with "new" nodes which have been "created" in the transformation process. Moreover, if a structured transformation d is "applied" to a graph (of which the initial graph of d is a subgraph), then the identification function and the embedding relation determine the set of embedding edges in the result graph. The identification function, on the one hand, shows how incoming edges are transferred from nodes of the initial graph of d to nodes of the result graph of d. The embedding relation, on the other hand, specifies how outgoing edges of nodes of the initial graph are transferred to nodes of the result graph. The notion is formally defined as follows.

*Definition 3.* Let $\Sigma$, $\Delta$ be finite sets. A $(\Sigma,\Delta)$-*structured transformation* is a 4-tuple $d = (g,h,iden,Emb)$, where g and h are $(\Sigma,\Delta)$-labeled graphs, *iden* is an injective partial function from $Nd(g)$ into $Nd(h)$, $Emb \subseteq (Nd(g) \times \Delta) \times (Nd(h) \times \Delta)$, and the following condition holds.
For each $x \in Nd(g)$, $v \in Nd(h)$, $\mu$, $\delta \in \Delta$ and $y \in Dom(iden)$ such that $(x,\mu,y) \in Ed(g)$ and $(v,\delta) \in Emb((x,\mu))$, $(v,\delta,iden(y)) \in Ed(h)$. □

For a $(\Sigma,\Delta)$-structured transformation $d = (g,h,iden,Emb)$, g is called the *initial graph of* d, h is called the *result graph of* d, *iden* is called the *identification function of* d and *Emb* is called the *embedding relation of* d. Also, g, h, *iden* and *Emb* are denoted by $in(d)$, $res(d)$, $iden_d$, and $Emb_d$, respectively. In order to simplify our notation we will write $Emb(x,\mu)$ rather than $Emb((x,\mu))$, for $x \in Nd(g)$ and $\mu \in \Delta$. The class of all $(\Sigma,\Delta)$-structured transformations is denoted by $Tr(\Sigma,\Delta)$. Structured transformations d and d′ are *disjoint* if $in(d)$ and $in(d')$ are disjoint, and $res(d)$ and $res(d')$ are disjoint.

The notion of a $(\Sigma,\Delta)$-structured transformation is illustrated by Example 3 - all examples of structured transformations are given in Section 7.

*Remark 1.* It is important to notice that, for a $(\Sigma,\Delta)$-structured transformation $d = (g,h,iden,Emb)$, its embedding relation $Emb_d$ may contain specifications for transferring edges that are not present in g, meaning that there may exist $x \in Nd(g)$, $v \in Nd(h)$ and $\mu,\delta \in \Delta$ such that $(v,\delta) \in Emb(x,\mu)$, but for no $y \in Nd(g)$, $(x,\mu,y) \in Ed(g)$. On the other hand, there may be edges in g that are not being transferred by $Emb_d$, meaning that there may exist $(x,\mu,y) \in Ed(g)$ such that, for no pair $v \in Nd(h)$, $\delta \in \Delta$, $(v,\delta) \in Emb(x,\mu)$. Moreover, edges with targets not in the domain of $iden_d$ are not transferred. $\square$

Note that, for a $(\Sigma,\Delta)$ structured transformation d, $iden_d$ together with $Emb_d$ define a relation between the edge-universe of $in(d)$ and the edge-universe of $res(d)$, which shows how "potential" edges of $in(d)$ are inherited by (or "transferred to") $res(d)$. Using this "edge-inheritance relation", one can see how the actual edges of $in(d)$ are inherited by $res(d)$. This is formally defined as follows.

*Definition 4.* Let $\Sigma$, $\Delta$ be finite sets and let $d \in Tr(\Sigma,\Delta)$. Then the *edge-inheritance relation of d* is the relation $Inh \subseteq Eun(in(d)) \times Eun(res(d))$ defined by $Inh = \{((x,\mu,y),(v,\delta,w)) \mid (v,\delta) \in Emb_d(x,\mu) \text{ and } w = iden(y)\}$. $\square$

For a $(\Sigma,\Delta)$-structured transformation d, its edge-inheritance relation is denoted by $Inh_d$.

*Remark 2.* Let $\Sigma$ and $\Delta$ be finite sets and let $d \in Tr(\Sigma,\Delta)$. Using the edge-inheritance relation of d, the condition from Definition 3 can be equivalently stated as : $Inh_d(Ed(g)) \subseteq Ed(h)$. Note however that $iden_d$, together with $Emb_d$ contain more information than $Inh_d$ in the sense that, in general, they cannot be recovered from $Inh_d$. □

We move now to define operations which, when applied to $(\Sigma,\Delta)$-structured transformations, yield $(\Sigma,\Delta)$-structured transformations.

Our first operation can be informally explained as follows. Let k be a graph and let d be a structured transformation such that $in(d) = k$. For an augmentation k′ of k, there exists a minimal (with respect to the set of edges of the result graph) structured transformation d′ such that $iden_{d'} = iden_d$, $Emb_{d'} = Emb_d$, $res(d') \in Aug(res(d))$, and $in(d') = k'$. Such a structured transformation d′ is formally defined as follows.

*Definition 5.* Let $\Sigma$ and $\Delta$ be finite sets, let $d \in Tr(\Sigma,\Delta)$, and let $g \in Aug(in(d))$. Then the *g-augmentation of* d, denoted by $Aug(g,d)$, equals $(g,h,iden_d,Emb_d)$, where $h \in Aug(res(d))$ is such that $Ed(h) = Ed(res(d)) \cup Inh_d(Ed(g))$. □

It is easily seen that the following lemma holds.

*Lemma 2.* Let $\Sigma$ and $\Delta$ be finite sets, let $d \in Tr(\Sigma,\Delta)$, and let $g \in Aug(in(d))$. Then $Aug(g,d) \in Tr(\Sigma,\Delta)$. □

A $(\Sigma,\Delta)$-structured transformation d′ is an *augmentation of* d if $in(d') \in Aug(in(d))$ and $d' = Aug(in(d'),d)$. For a $(\Sigma,\Delta)$-structured transformation

d, the set of all augmentations of d is denoted by $Aug(d)$. Observe that $Aug$ may be considered as an operation on $(\Sigma,\Delta)$-structured transformations. The notion of augmentation is illustrated by Example 4.

Our second operation can be informally explained as follows. Let g be a graph, and let $g_1$, $g_2$ be graphs such that $g = g_1 \oplus g_2$. Let $d_1$, $d_2$ be structured transformations applicable to $g_1$, $g_2$ respectively. Then the concurrent composition of $d_1$ and $d_2$ is the (unique) structured transformation applicable to g such that, when restricted to $g_1$, it yields $d_1$, and, when restricted to $g_2$, it yields $d_2$. Thus, intuitively speaking, the concurrent composition of $d_1$ and $d_2$ is the structured transformation obtained by putting together $d_1$ and $d_2$ in such a way that they do not "interfere" with each other. Formally, it is defined as follows.

*Definition 6.* Let $\Sigma$, $\Delta$ be finite sets and let $d_1, d_2 \in Tr(\Sigma,\Delta)$ be disjoint. Then the concurrent composition of $d_1$ and $d_2$, denoted by $d_1 \underline{cc}\, d_2$, equals $(in(d_1) \oplus in(d_2),\ res(d_1) \oplus res(d_2),\ iden_{d_1} \oplus iden_{d_2},\ Emb_{d_1} \oplus Emb_{d_2})$. □

The notion of concurrent composition is illustrated by Example 5.

The following result shows that the concurrent composition of two disjoint $(\Sigma,\Delta)$-structured transformations is a $(\Sigma,\Delta)$-structured transformation.

*Lemma 3.* Let $\Sigma$, $\Delta$ be finite sets and let $d_1$, $d_2 \in Tr(\Sigma,\Delta)$ be disjoint. Then $d_1 \underline{cc}\, d_2 \in Tr(\Sigma,\Delta)$.

*Proof.* Let $f = d_1 \underline{cc}\, d_2$. Then $Inh_f = (Inh_{d_1} \oplus Inh_{d_2}) \cup K_1 \cup K_2$, where

$K_1 = \{((x,\mu,y),(v,\delta,w)) \mid x \in Nd(in(d_1)),\ y \in Nd(in(d_2)),\ v \in Nd(res(d_1)),$
$w \in Nd(res(d_2)),\ \mu,\delta \in \Delta_S,\ (v,\delta) \in Emb_{d_1}(x,\mu) \text{ and } w = iden_{d_2}(y)\}$, and

$K_2 = \{((x,\mu,y),(v,\delta,w)) \mid x \in Nd(in(d_2)),\ y \in Nd(in(d_1)),\ v \in Nd(res(d_2)),$
$w \in Nd(res(d_1)),\ \mu,\delta \in \Delta_S,\ (v,\delta) \in Emb_{d_2}(x,\mu) \text{ and } w = iden_{d_1}(y)\}$.

Obviously, $K_1(Ed(in(f))) = K_2(Ed(in(f))) = \emptyset$. Hence from $Inh_{d_1}(Ed(in(d_1)) \subseteq Ed(res(d_1))$ and $Inh_{d_2}(Ed(in(d_2))) \subseteq Ed(res(d_2))$ it follows that $Inh_f(Ed(in(f))) \subseteq Ed(res(f))$. This proves the statement of the lemma. $\square$

The following properties are easily verified.

*Lemma 4.* Let $\Sigma$, $\Delta$ be finite sets and let $d_1, d_2, f_1, f_2 \in Tr(\Sigma,\Delta)$ be such that $d_1$ and $d_2$ are disjoint, and $f_1$ and $f_2$ are disjoint.

(1) If $f_1 \in Aug(d_1)$ and $f_2 \in Aug(d_2)$, then $f_1 \underline{cc} f_2 \in Aug(d_1 \underline{cc} d_2)$.

(2) $d_1 \underline{cc} d_2 = d_2 \underline{cc} d_1$.

(3) Let $d_3 \in Tr(\Sigma,\Delta)$ be such that $d_1$, $d_2$ and $d_3$ are pairwise disjoint. Then $(d_1 \underline{cc} d_2) \underline{cc} d_3 = d_1 \underline{cc} (d_2 \underline{cc} d_3)$. $\square$

Our last operation is that of sequential composition. Its intuitive meaning is simple : if $d_1$ is a structured transformation applicable to g and yielding h, and $d_2$ is a structured transformation applicable to h and yielding k, then the sequential composition of $d_1$ and $d_2$ results by extending $d_1$ by $d_2$ (first $d_1$, then $d_2$): it is applicable to g and yields k. It is formally defined as follows.

*Definition 7.* Let $\Sigma$, $\Delta$ be finite sets and let $d_1, d_2 \in Tr(\Sigma,\Delta)$ be such that $res(d_1) = in(d_2)$. The *sequential composition of* $d_1$ *and* $d_2$, denoted by $d_1 \underline{sq} d_2$, equals $(in(d_1), res(d_2), iden_{d_2} \circ iden_{d_1}, Emb_{d_2} \circ Emb_{d_1})$. $\square$

The notion of sequential composition is illustrated by Example 6.

The following result shows that the sequential composition of $(\Sigma,\Delta)$-structured transformations is a $(\Sigma,\Delta)$-structured transformation.

*Lemma 5.* Let $\Sigma$, $\Delta$ be finite sets and let $d_1, d_2 \in Tr(\Sigma,\Delta)$ be such that $in(d_2) = res(d_1)$. Then $d_1 \ \underline{sq}\ d_2 \in Tr(\Sigma,\Delta)$.

*Proof.* Let $f = d_1 \ \underline{sq}\ d_2$. It is easily verified that

$$Inh_{d_2} \circ Inh_{d_1} = \{((x,\mu,y),(v,\delta,w)) \mid (v,\delta) \in Emb_{d_2}(Emb_{d_1}(x,\mu)) \text{ and } w = iden_{d_2}(iden_{d_1}(y))\},$$

and hence, $Inh_f = Inh_{d_2} \circ Inh_{d_1}$. Hence, it follows from Remark 2 that the statement of the lemma holds. □

It is easily seen that the following lemma holds.

*Lemma 6.* Let $\Sigma$, $\Delta$ be finite sets, and let $d_1, d_2, f_1, f_2 \in Tr(\Sigma,\Delta)$ be such that $in(d_2) = res(d_1)$ and $in(f_2) = res(f_1)$.

(1) If $f_1 \in Aug(d_1)$ and $f_2 \in Aug(d_2)$, then $f_1 \ \underline{sq}\ f_2 \in Aug(d_1 \ \underline{sq}\ d_2)$.

(2) Let $d_3 \in Tr(\Sigma,\Delta)$ be such that $in(d_3) = res(d_2)$. Then
$(d_1 \ \underline{sq}\ d_2) \ \underline{sq}\ d_3 = d_1 \ \underline{sq}\ (d_2 \ \underline{sq}\ d_3)$.

(3) If $d_1$ and $f_1$ are disjoint and $d_2$ and $f_2$ are disjoint, then
$(d_1 \ \underline{sq}\ d_2) \ \underline{cc}\ (f_1 \ \underline{sq}\ f_2) = (d_1 \ \underline{cc}\ f_1) \ \underline{sq}\ (d_2 \ \underline{cc}\ f_2)$. □

The properties from Lemma 6 are illustrated by Examples 7, 8 and 9, respectively.

It is easily verified that the operations of augmentation, concurrent composition and sequential composition "behave well" with respect to isomorphism. The notions of a homomorphism and an isomorphism are defined as follows for labeled graphs.

*Definition 8.* Let $\Sigma$, $\Delta$ be finite sets and let g and h be $(\Sigma,\Delta)$-labeled graphs. A *homomorphism from* g *into* h is a node-label preserving function $\xi : Nd(g) \to Nd(h)$ such that $\xi_{ed}(Ed(g)) \subseteq Ed(h)$. A homomorphism $\xi$ from g into h is an *isomorphism from* g *onto* h if $\xi^{-1}$ is a homomorphism from h into g. □

If there exists an isomorphism from g onto h, then g and h *are isomorphic*. It is easily seen that a homomorphism $\xi$ from g onto h is an isomorphism from g onto h if and only if $\xi$ is bijective and $\xi_{ed}(Ed(g)) = Ed(h)$.

The notion of an isomorphism of structured transformations is defined as follows - it expresses the fact that two structured transformations differ only by the choice of nodes for their initial and result graphs.

*Definition 9.* Let $\Sigma$, $\Delta$ be finite sets and let d, d' $\in Tr(\Sigma,\Delta)$. A pair $(\eta,\xi)$ of isomorphisms (of $(\Sigma,\Delta)$-labeled graphs), $\eta : in(d) \to in(d')$ and $\xi : res(d) \to res(d')$, is an *isomorphism* (of $(\Sigma,\Delta)$-structured transformations) *from* d *onto* d' if $iden_{d'} = \xi \circ iden_d \circ \eta^{-1}$ and $Emb_{d'} = \xi_\Delta \circ Emb_d \circ (\eta_\Delta)^{-1}$. □

If d, d' $\in Tr(\Sigma,\Delta)$ are such that there exists an isomorphism from d onto d', then we say that d and d' *are isomorphic*. It is easily seen that, if $(\eta,\xi) : d \to d'$ is an isomorphism, then $Inh_{d'} = \{(\eta_{ed}(e),\xi_{ed}(u)) \mid (e,u) \in Inh_d\}$.

The properties of the following lemma obviously hold.

*Lemma 7.* Let $\Sigma$, $\Delta$ be finite sets.

(1) Let d, d', f, f' $\in Tr(\Sigma,\Delta)$ be such that there exist functions $\eta : Nd(in(d)) \to Nd(in(f))$ and $\xi : Nd(res(d)) \to Nd(res(f))$ such that $(\eta,\xi)$ is both an isomorphism from d onto f and an isomorphism from d' onto f'. Then d' $\in Aug(d)$ if and only if f' $\in Aug(f)$.

(2) Let $d_1$, $d_2$, $f_1$, $f_2 \in Tr(\Sigma,\Delta)$ be such that $d_1$, $d_2$ are disjoint and $f_1$, $f_2$ are disjoint. Let $(\eta_1, \xi_1) : d_1 \to f_1$ and $(\eta_2, \xi_2) : d_2 \to f_2$ be isomorphisms. Then $(\eta_1 \oplus \eta_2, \xi_1 \oplus \xi_2)$ is an isomorphism from $d_1 \underline{cc}\, d_2$ onto $f_1 \underline{cc}\, f_2$.

(3) Let $d_1$, $d_2$, $f_1$, $f_2 \in Tr(\Sigma,\Delta)$ be such that $res(d_1) = in(d_2)$ and $res(f_1) = in(f_2)$. Let $(\eta,\gamma) : d_1 \to f_1$ and $(\gamma,\xi) : d_2 \to f_2$ be isomorphisms. Then $(\eta,\xi)$ is an isomorphism from $d_1 \underline{sq}\, d_2$ onto $f_1 \underline{sq}\, f_2$. □

We will define now the "most elementary" structured transformations, called *node replacements*, which correspond to "rewriting processes" that are trivial in the sense that no rewriting takes place. Moreover, the initial and result graphs of a node replacement are discrete one-node graphs. The notion is formally defined as follows.

*Definition 10.* Let $\Sigma$, $\Delta$ be finite sets and let $d \in Tr(\Sigma,\Delta)$.

d is a *$(\Sigma,\Delta)$-node replacement* if both $in(d)$ and $res(d)$ are discrete one-node graphs such that the following holds. Let $Nd(in(d)) = \{v\}$ and $Nd(res(d)) = \{w\}$. Then

(i) $\phi_{in(d)}(v) = \phi_{res(d)}(w)$,

(ii) $iden_d = \{(v,w)\}$, and,

(iii) $Emb_d = \{((v,\delta),(w,\delta)) \mid \delta \in \Delta\}$. □

The $(\Sigma,\Delta)$-node replacement d such that $Nd(in(d)) = \{v\}$, $Nd(res(d)) = \{w\}$ and $\phi_{in(d)}(v) = \ell$ is denoted by $Nrep_{(v,w,\ell)}$. The class of all $(\Sigma,\Delta)$-node replacements is denoted by $Nrep(\Sigma,\Delta)$.

*Remark* 3. Let $\Sigma$, $\Delta$ be finite sets.

(1) Let g and h be $(\Sigma,\Delta)$-labeled graphs and let $\gamma : g \to h$ be an isomorphism. Let $Nd(g) = \{v_1, v_2, \ldots, v_k\}$. For each $i \in \{1,\ldots,k\}$, let $\phi_g(v_i) = \ell_i$. Then $(g, h, \gamma, \gamma_\Delta) \in Tr(\Sigma,\Delta)$ and, moreover,

$(g, h, \gamma, \gamma_\Delta) \in Aug(Nrep_{(v_1,\gamma(v_1),\ell_1)} \underline{cc}\ Nrep_{(v_2,\gamma(v_2),\ell_2)} \underline{cc} \cdots$

$\cdots \underline{cc}\ Nrep_{(v_k,\gamma(v_k),\ell_k)})$.

Notice that it follows from (1) and (2) of Lemma 3 that the order in which the concurrent composition is applied to the node replacements $Nrep_{(v_i,\gamma(v_i),\ell_i)}$ is irrelevant; therefore we may write the expression

$Nrep_{(v_1,\gamma(v_1),\ell_1)}\ \underline{cc}\ Nrep_{(v_2,\gamma(v_2),\ell_2)}\ \underline{cc} \cdots \underline{cc}\ Nrep_{(v_k,\gamma(v_k),\ell_k)}$ without parentheses.

(2) For each $(\Sigma,\Delta)$-labeled graph g, the $(\Sigma,\Delta)$-structured transformation

$(g, g, Id_{Nd(g)}, Id_{Nd(g)\times\Delta})$ is denoted by $Itr_g$.

(3) For each $d \in Tr(\Sigma,\Delta)$, $Itr_{in(d)}\ \underline{sq}\ d = d\ \underline{sq}\ Itr_{res(d)} = d$. □

The class of $(\Sigma,\Delta)$-structured transformations generated by (built from) a given set of "primitive" $(\Sigma,\Delta)$-structured transformations is defined as follows.

*Definition 11.* Let $\Sigma$, $\Delta$ be finite sets and let P be a set of $(\Sigma,\Delta)$-structured transformations. The *class of* $(\Sigma,\Delta,P)$*-structured transformations*, denoted by $Tr(\Sigma,\Delta,P)$, is the smallest subclass, X, of $Tr(\Sigma,\Delta)$ such that

(1) $Nrep(\Sigma,\Delta) \subseteq X$,

(2) $P \subseteq X$,

(3) for each $d \in X$, $Aug(d) \subseteq X$,

(4) for each d, d' $\in$ X such that d and d' are disjoint, d $\underline{cc}$ d' $\in$ X, and

(5) for each d, d' $\in$ X such that *in*(d') = *res*(d), d $\underline{sq}$ d' $\in$ X. □

*Example 10.*

Let $\Sigma$ and $\Delta$ be the alphabets given in Section 7 and let $d_1$, $d_4$ be the $(\Sigma,\Delta)$-structured transformations from Example 3. Then the $(\Sigma,\Delta)$-structured transformation $d_{14}$ given in Example 10 of Section 7 is a $(\Sigma,\Delta,P)$-structured transformation, where $P = \{d_1, d_4\}$. (Observe that $d_6 \in Aug(Nrep_{(v_5,v_8,a_3)}\ \underline{cc}\ d_4))$.

The following result shows that the definition of $Tr(\Sigma,\Delta,P)$ is compatible with our notion of an isomorphism of structured transformations.

*Lemma 8.* Let $\Sigma$, $\Delta$ be finite sets and let $d \in Tr(\Sigma,\Delta,P)$. For each $f \in Tr(\Sigma,\Delta)$ isomorphic with d, $f \in Tr(\Sigma,\Delta,P)$.

*Proof.* The statement follows from (1) of Remark 3 and (5) of Definition 11. □

The particular type of graph grammars we use in this paper is defined as follows.

*Definition 12.* Let $\Sigma$, $\Delta$ be finite sets. A $(\Sigma,\Delta)$-*grammar* is a system (P, *Init*), where P is a finite set of $(\Sigma,\Delta)$-structured transformations and *Init* is a finite set of $(\Sigma,\Delta)$-labeled graphs. □

The class of $(\Sigma,\Delta)$-structured transformations corresponding to a given $(\Sigma,\Delta)$-grammar is defined as follows.

*Definition 13.* Let $\Sigma$, $\Delta$ be finite sets and let $G = (P, Init)$ be a $(\Sigma,\Delta)$-grammar. A *G-structured transformation* is a $(\Sigma,\Delta,P)$-structured transformation d such that $in(d) \in Init$. □

The class of all G-structured transformations is denoted by $Tr(G)$.

We conclude this section by (1) relating the traditional notion of "applying a production to a graph" to our notion of a structured transformation, and (2) discussing the notion of a derivation.

(1) The notion of a structured transformation d describes the effect of a transformation process. An essential property of a transformation process transforming *in*(d) into *res*(d) is that it may be "applied" to a graph g of which *in*(d) is a subgraph. Hence it is natural to ask how a structured transformation d can be applied to a graph g of which *in*(d) is a subgraph. In our framework such an application would be described by a structured transformation in which *in*(d) is transformed into *res*(d), and in which g - *in*(d) remains unchanged. Formally, this would be the structured transformation $Aug(g,(d \underline{cc} Itr_{g - in(d)}))$. This situation is illustrated in Fig. 4.

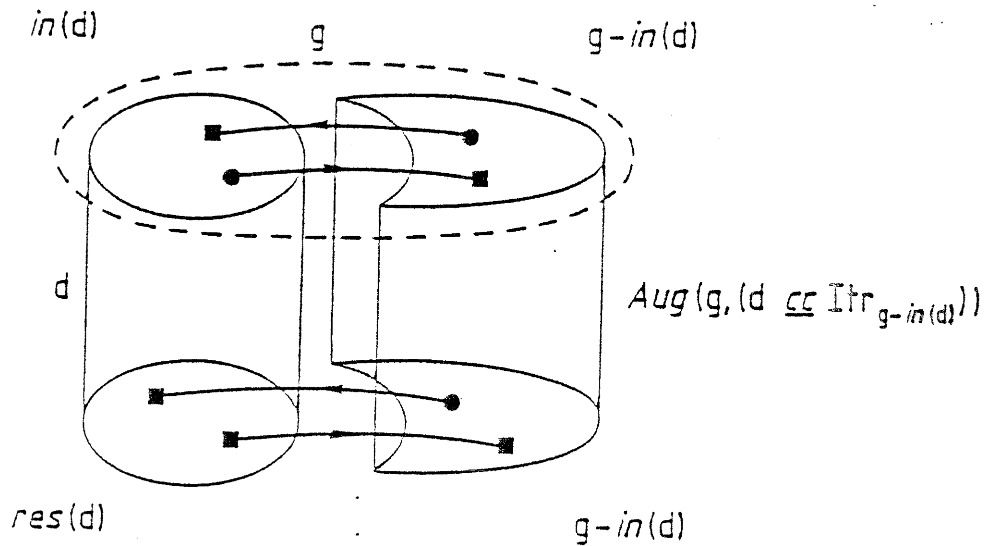



Figure 4.

Once we will have related the notion of a production to the notion of a primitive event transformation (see Section 5), the above explains how the application of a production to a graph can be described in our framework. As a matter of fact, a primitive event transformation is a special case of a structured transformation (see Definition 15).

(2) In the standard approach to graph grammars, the notion of a derivation formalizes (in quite a detail) how a rewriting process proceeds step-by-step in transforming its initial graph into its result graph. The notion of a structured transformation, introduced in this section, is weaker (contains less information) than a derivation : it specifies the initial and the result graphs of a rewriting process, together with the relationships between nodes of these graphs needed for the composition of structured transformations, but it does not specify how the rewriting process is realized.

If desired, the notion of a derivation can be defined in the framework presented here as follows.

*Definition 14.* Let $\Sigma,\Delta$ be finite sets and let $P \subseteq Tr(\Sigma,\Delta)$.
A $(\Sigma,\Delta,P)$-*derivation* is a sequence $d_1,\ldots,d_n$, $n \geq 1$, of structured transformations such that

(1) for each $i \in \{1,\ldots,n\}$, $d_i \in Tr(\Sigma,\Delta,P)$ and $d_i \in Aug(f_i)$, where $f_i \in Tr(\Sigma,\Delta,P)$ is obtained by composing elements of $P \cup Nrep(\Sigma,\Delta)$ using only the operation $\underline{cc}$,

(2) for each $i \in \{1,\ldots,n-1\}$, $res(d_i) = in(d_{i+1})$. □

Observe that the elements $d_1,\ldots,d_n$ of a $(\Sigma,\Delta,P)$-derivation as above correspond to *steps* of a derivation in a classical sense. One can view a $(\Sigma,\Delta,P)$-derivation as a realization of the $(\Sigma,\Delta)$-structured transformation $d = d_1\ \underline{sq}\ \ldots\ \underline{sq}\ d_n$. Clearly, in general, the same d may be realized by other $(\Sigma,\Delta,P)$-derivations. As a matter of fact, it can be shown that the following holds.

*Theorem 1.* Let $\Sigma$, $\Delta$ be finite sets, let $P \subseteq Tr(\Sigma,\Delta)$ and let $d \in Tr(\Sigma,\Delta)$. Then $d \in Tr(\Sigma,\Delta,P)$ if and only if there exists a $(\Sigma,\Delta,P)$-derivation $d_1,\ldots,d_n$ such that $d = d_1\ \underline{sq}\ d_2\ \ldots\ \underline{sq}\ d_n$. □

Since we relate, in Section 5, the notion of a production to an element of P, we get in this way an interpretation of the usual notion of a derivation in our framework.

The way (1) of Definition 14 was formulated allows concurrent composition of several elements of P within one step of the derivation. If one wants to get an analog of the classical *sequential* notion of a derivation (where *one* production is applied in *one* derivation step), then (1) would be replaced by :

(1′) for each $i \in \{1,\ldots,n\}$, $d_i \in Tr(\Sigma,\Delta,P)$ and $d_i \in Aug(f_i)$, where $f_i \in Tr(\Sigma,\Delta,P)$ is obtained by composing *one* element of P with elements of $Nrep(\Sigma,\Delta)$, using only the operation $\underline{cc}$.

## 5. ACTOR GRAMMARS

In this section we use the methodology of the previous section to describe the dynamic behaviour of actor systems. We define the notion of an actor grammar, and consider the set of structured transformations associated with it. We start by defining "primitive" structured transformations, called primitive event transformations, which formalize the notion of an event (hence, the execution of a script in an actor system). Primitive event transformations are the basic building blocks for constructing structured transformations using the three basic operations introduced in Section 4.

Before we formally introduce primitive event transformations, we informally describe their various components with the underlying interpretation in the framework of actor systems. Let d be a primitive event transformation describing an event $\epsilon = [M \rightarrow A]$ in an actor system (i.e., the processing of a message M by an actor A). We do not assume that M is the only message or that A is the only actor in the system, hence we consider d as a building block of a "larger" structured transformation.

### (1) The initial graph

All actors and messages different from M and A are not affected by the event $\epsilon$. Hence, the natural choice for the initial graph of d is a $\emptyset$-handle consisting of a node x corresponding to M, a node y corresponding to A, and a $\emptyset$-edge from x to y. Since a $\emptyset$-handle is determined (up to an isomorphism) by the labels of its nodes, *in*(d) is determined (up to an isomorphism) by the pair (m,a), where m is the message value of M and a is the actor state of A. This reflects the fact that,

in processing a message by an actor, the actions to be taken are decided "locally", only on the basis of the actor state and the message value.

(2) The result graph

The result graph of d describes those actors and messages which are either affected or created as the result of the event $\epsilon$, as well as the relationships between them. Hence the nodes of *res*(d) correspond to (i) the actor A, (ii) the newly created actors, and (iii) the newly created messages. The corresponding node labels represent the updated actor state of A, the initial actor states of the newly created actors, and the message values of the newly created messages. Moreover, the edges of *res*(d) describe which of these actors and messages are acquaintances or destinations of each other. In our terminology this means that *res*(d) is a configuration graph. One of its nodes will be the updated representation of A.

(3) The identification function

The identification function describes which nodes of *res*(d) may be identified with nodes from *in*(d). Since the message M "disappears" as a result of the event $\epsilon$, $iden_d$ is defined only on the unique actor node y of *in*(d). Moreover, since incoming edges of y represent references to A, all incoming edges of y must be transferred to the node $iden_d(y)$ of *res*(d).

(4) The embedding relation

The embedding relation of d describes how the script uses acquaintance names to transfer acquaintances from A and M to actors and messages corresponding to nodes of *res*(d). More specifically, let $z \in Nd(in(d))$, $v \in Nd(res(d))$; let $\mu$ and $\delta$ be

acquaintance names. Let Z be the actor or message (i.e., A or M) corresponding to z and let V be the actor or message corresponding to v. Then $((z,\mu),(v,\delta)) \in Emb_d$ if and only if the script says that the $\delta$-acquaintance of V (or the destination of V, if V is a message and $\delta = \emptyset$) is the $\mu$-acquaintance of Z.

Consequently, the assumptions about actor systems imply that $Emb_d(z,\emptyset) = \emptyset$ for each $z \in Nd(in(d))$, and that the set of participants of $[M \to A]$ corresponds to the union of (i) the set of nodes of $res(d)$, and (ii) the "outgoing neighbourhood" of $in(d)$, i.e., the set of targets of edges that have their sources in $in(d)$.

We now define formally the notion of a primitive event transformation. For an actor vocabulary S and a set P of $(\Sigma_S,\Delta_S)$-structured transformations, we write $Tr(S)$ instead of $Tr(\Sigma_S,\Delta_S)$ and $Tr(S,P)$ instead of $Tr(\Sigma_S,\Delta_S,P)$.

*Definition 15.* Let $S = (A, M, AQ, MQ)$ be an actor vocabulary and let $d \in Tr(S)$. d is a *primitive S-event transformation* if $in(d)$ is a $\emptyset$-handle and the following conditions hold. Let $Nd(in(d)) = \{x,y\}$ and $Ed(in(d)) = \{(x,\emptyset,y)\}$.

(1) $x \in Msg(in(d))$ and $y \in Act(in(d))$,

(2) $res(d) \in Conf(S)$,

(3) $Dom(iden_d) = \{y\}$ and $iden_d(y) \in Act(res(d))$,

(4) $Emb_d$ is injective,

(5) for each $((u,\mu),(v,\delta)) \in Emb_d$,

(5.i) $\mu \neq \emptyset$,

(5.ii) $(u,\mu) \in (\{x\} \times MQ) \cup (\{y\} \times AQ)$,

(5.iii) $(v,\delta) \in (Msg(res(d)) \times MQ) \cup (Act(res(d)) \times AQ)$, and

(5.iv) $\delta_{res(d)}(v) = \emptyset$. □

*Example 11.*

Let S be the actor vocabulary from Example 2. Then the structured transformations $d_1$ and $d_4$ from Example 3 are primitive S-event transformations.

It is easily verified that the following technical result holds.

*Lemma 9.* Let S be an actor vocabulary and let P be a set of primitive S-event transformations. Let $d \in Tr(S,P)$. Then

(1) $Dom(iden_d) \supseteq Acc(in(d))$.

(2) $Iden_d \subseteq (Acc(in(d)) \times Acc(res(d))) \cup (Msg(in(d)) \times Msg(res(d)))$.

(3) $Emb_d$ is injective. □

We are ready now to define the notion of an actor grammar.

*Definition 16.* Let S be an actor vocabulary. A $(\Sigma_S, \Delta_S)$-grammar $G = (P, Init)$ is an S-*actor grammar* if each $p \in P$ is a primitive S-event transformation. □

*Example 12.*

Let S be the actor vocabulary from Example 2, let $d_1$, $d_4$, $d_{14}$ be the structured transformations from Example 3 and Example 10 and let g be the S-configuration graph from Example 2. Then $(\{d_1, d_4\}, \{g\})$ is an S-actor grammar and $d_{14} \in Tr(G)$.

In our formal framework, structured transformations of actor grammars model computations of actor systems. More precisely, they describe the way such a computation changes the configuration of the system, by transforming the configuration graph that describes this configuration. Obviously, the result of such a transformation should again be a configuration graph. The next result shows that this is indeed the case.

*Theorem 2.* Let S be an actor vocabulary and let $G = (P, Init)$ be an S-actor grammar. Then, for each $d \in Tr(G)$, $res(d) \in Conf(S)$.

*Proof.* Let $S = (A,M,AQ,MQ)$. For an S-graph g, let $Allow_g$ denote the set $(Msg(g) \times MQ) \cup (Acc(g) \times AQ)$. A structured transformation $d \in Tr(S)$ will be called a *well-behaved* structured transformation if

(i) $Emb_d(Allow_{in(d)}) \subseteq Allow_{res(d)}$, and

(ii) for each $((x,\mu),(v,\delta)) \in Emb_d$ such that $\mu_{in(d)}(x) = \emptyset$, $\delta_{res(d)}(v) = \emptyset$.

We show that the following two propositions hold for each $d \in Tr(S,P)$.

P1 : d is a well-behaved structured transformation.

P2 : if $in(d) \in Conf(S)$, then $res(d) \in Conf(S)$.

It is easily seen that the statement of the theorem holds if P2 holds for each $d \in Tr(S,P)$, because $Init \subseteq Conf(S)$ and, for each $d \in Tr(G)$, $in(d) \in Init$.

Let $d \in Tr(S,P)$.

(a) To show that P1 holds for d, consider the following cases.

(1) $d \in Nrep(S)$. Then, obviously, d is well-behaved.

(2) $d \in P$. It follows easily from Definition 15 that d is well-behaved : the conditions (i) and (ii) correspond to (5.iii) and (5.iv) of Definition 15, respectively.

(3) $d \in Aug(d')$, for some $d' \in Tr(S,P)$ such that d' is well-behaved.

Then $Emb_d(Allow_{in(d)}) \subseteq Allow_{res(d)}$, because $Emb_d = Emb_{d'}$.

On the other hand, let $((x,\mu),(v,\delta)) \in Emb_d$ be such that $\mu_{in(d)}(x) = \emptyset$.

Then $(v,\delta) \in Emb_{d'}(x,\mu)$. It follows from $\mu_{in(d)}(x) = \emptyset$ and $in(d) \in Aug(in(d'))$ that $\mu_{in(d')} = \emptyset$ and, since d' is well-behaved, $\delta_{res(d')}(v) = \emptyset$. It follows from the injectivity of $Emb_d$ (see (3) of Lemma 9) and $\mu_{in(d)}(x) = \emptyset$ that $\delta_{res(d)}(v) = \emptyset$ and hence, that d is well-behaved.

(4) $d = d_1 \underline{cc}\ d_2$, for some $d_1, d_2 \in Tr(S,P)$ such that $d_1$ and $d_2$ are disjoint and well-behaved. It easily follows from Definition 6 that d is well-behaved.

(5) $d = d_1 \underline{sq}\ d_2$, for some $d_1, d_2 \in Tr(S,P)$ such that $in(d_2) = res(d_1)$, and $d_1$ and $d_2$ are well-behaved. It easily follows from Definition 7 that d is well-behaved.

It follows from (1) through (5) that P1 holds for each $d \in Tr(S,P)$.

## 6. RELATIONSHIP TO THE APPROACH FROM [JR87].

Initially, this paper was intended as the "full version" of [JR87]. However, during the work on the current paper we have changed the notion of a configuration graph and, quite considerably, the methodology of defining derivations and structured transformations in actor grammars.

(1) *The notion of a configuration graph.*

In [JR87], each message node of a configuration graph was required to have an outgoing $\emptyset$-edge. Hence we did not consider actor systems in which it is possible to send messages which do not have a destination actor. It turns out that changing the notion of a configuration graph in the way we did it in this paper can be considered as merely a technical convenience. One can prove that, for each actor grammar G, there exists an actor grammar $\overline{G}$ such that the class of all $\overline{G}$-structured transformations equals (under a suitable coding) the class of all G-structured transformations such that their initial and result graphs are configuration graphs where each message has a destination. The proof is somewhat involved and for this reason we present it in a separate, forthcoming paper (see [JR 88]).

(2) *The methodology for defining derivations and structured transformations.*

In [JR87] we used the "standard approach" towards graph grammars and the derivations in them : first we defined the (concurrent) application of productions to a number of disjoint occurrences of their left-hand sides in a graph, yielding a direct derivation step. Then we defined a derivation as an iteration of direct derivation steps. In defining a direct derivation step we distinguished clearly between a replacement (of the left-hand side by the right-hand side) and an embedding (of the right-hand side in the rest of the graph).

(b) To show that P2 holds for d, assume that $in(d) \in Conf(S)$. Consider the following cases.

(1) $d \in Nrep(S)$. It is easily seen that $res(d) \in Conf(S)$.

(2) $d \in P$. Since d is a primitive S-event transformation, $res(d) \in Conf(S)$.

(3) $d \in Aug(d')$, for some $d' \in Tr(S,P)$ such that P2 holds for d'. It follows from (2) of Lemma 1 that $in(d') \in Conf(S)$, and, since P2 holds for d', $res(d') \in Conf(S)$. Now let $e = (v,\delta,w) \in Ed(res(d))\text{-}Ed(res(d'))$. Then $e \in Inh_d(Ed(in(d)))$. Since, for each $(x,\mu,y) \in Ed(in(d))$, $(x,\mu) \in Allow_{in(d)}$, and d is well-behaved, $(v,\delta) \in Allow_{res(d)}$. It follows from (2) of Lemma 9 that (1) of Definition 2 holds for $res(d)$.

To show that (2) of Definition 2 holds for $res(d)$, let $v \in Nd(res(d))$ and $\delta \in \Delta_S$. If $\delta_{res(d)}(v) \subseteq Ed(res(d'))$, then the condition holds because $res(d') \in Conf(S)$. If, on the other hand, there exists an edge $e \in \delta_{res(d)}(v) - Ed(res(d'))$, where $e = (v,\delta,w)$, then there exists an edge $u = (x,\mu,y) \in Ed(in(d))$ such that $e \in Inh_d(u)$. Moreover, it follows from the injectivity of $Emb_d$ and $in(d) \in Conf(S)$ that there exists at most one edge $u \in Ed(in(d))$ such that $\delta_{res(d)}(v) \cap Inh_d(u) \neq \emptyset$. Hence, $Inh_d(Ed(in(d)) \cap \delta_{res(d)}(v) = \{e\}$. Since $Inh_d = Inh_{d'}$ and $Inh_{d'}(Ed(in(d'))) \subseteq Ed(res(d'))$, it follows from $e \notin Ed(res(d'))$ that $u \notin Ed(in(d'))$, and hence, $\mu_{in(d')}(x) = \emptyset$. Since d' is well-behaved, $\delta_{res(d')}(v) = \emptyset$. Hence $\delta_{res(d)}(v) = \{e\}$. We conclude that (2) of Definition 2 holds for $res(d)$, and hence, that P2 holds for d.

(4) $d = d_1 \underline{cc}\, d_2$, for some $d_1$, $d_2 \in Tr(S,P)$ such that P2 holds for $d_1$ and $d_2$. Since $in(d) \in Conf(S)$, it follows from (2) of Lemma 1 that $in(d_1)$, $in(d_2) \in Conf(S)$. Hence $res(d_1)$, $res(d_2) \in Conf(S)$ and, by (1) of Lemma 1, $res(d) \in Conf(S)$. Hence P2 holds for d.

(5) $d = d_1 \underline{sq}\, d_2$, for some $d_1$, $d_2 \in Tr(S,P)$ such that P2 holds for $d_1$ and $d_2$. Since $res(d_1) = in(d_2)$, P2 holds for d.

It follows from (1) through (5) that P2 holds for each $d \in Tr(S,P)$. This completes the proof. □

In the present paper, however, we investigate actor grammars via the notion of a structured transformation. In this approach the "productions" of a grammar are a priori given, "primitive" structured transformations, which play the role of basic building blocks. The class of all structured transformations corresponding to a given grammar is built from these basic building blocks and from node replacements by the operations of augmentation, concurrent composition and sequential composition.

It is easily verified that the notion of an actor grammar as presented in [JR87] is equivalent to the notion of an actor grammar as presented in this paper : for a given actor grammar G, each G-structured transformation may be realized as a G-derivation (as defined in Definition 14) using the properties from Lemma 4 and Lemma 6. It is easily seen that a G-derivation corresponds to a sequence of direct derivation steps in [JR87].

In [JR87] it is shown (Property 2) that the effect of applying productions to disjoint subgraphs of a given configuration graph does not depend on the particular order in which one applies these productions. It is instructive to notice that this property is expressed in the current paper by (3) of Lemma 6.

## 7. EXAMPLES OF STRUCTURED TRANSFORMATIONS

In all examples in this section we use fixed alphabets $\Sigma$ and $\Delta$, where $\Sigma = A \cup M$, $\Delta = AQ \cup MQ$, $A = \{a_1, a_2, a_3\}$, $M = \{m_1, m_2, m_3\}$, $AQ = \{\alpha, \beta, \gamma\}$ and $MQ = \{\sigma, \tau, \emptyset\}$.

For the pictorial representation of a $(\Sigma,\Delta)$-structured transformation we use the following conventions :

(1) The upper half of the figure represents the initial graph and the lower half represents the result graph.

(2) The identification function is represented by dotted lines.

(3) If, in the representation of a $(\Sigma,\Delta)$-structured transformation d, $Emb_d(x,\mu)$ is not explicitly specified, for some $x \in Nd(in(d))$ and some $\mu \in \Delta$, then it is assumed that $Emb_d(x,\mu) = \emptyset$.

(4) For each of the $(\Sigma,\Delta)$-structured transformations $d_1, d_2, \ldots, d_{14}$ given in this section we write $Emb_i$ instead of $Emb_{d_i}$.

*Example 3.*

$d_1, d_2, d_3, d_4$ as below are $(\Sigma,\Delta)$-structured transformations.

$d_1$ :

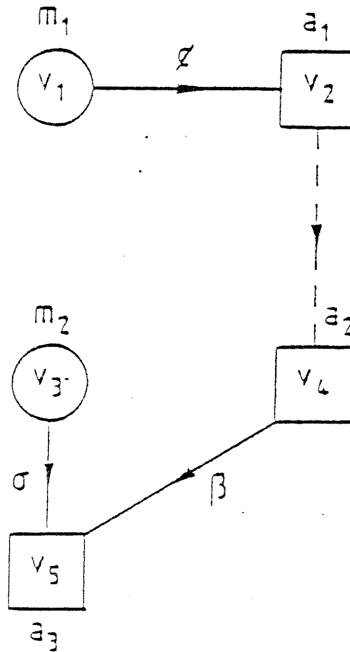



Figure 5.

$Emb_1(v_1,\sigma) = \{(v_5,\gamma),(v_3,\tau)\}$

$Emb_1(v_2,\alpha) = (v_3,\emptyset)$

$Emb_1(v_2,\beta) = (v_4,\alpha)$

$d_2$ :

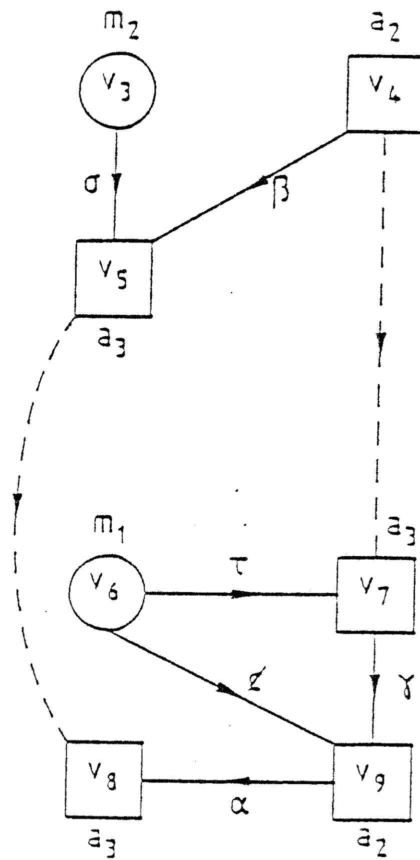



Figure 6.

$Emb_2(v_3, \sigma) = (v_9, \alpha)$

$Emb_2(v_3, \tau) = \{(v_6, \sigma), (v_9, \beta)\}$

$Emb_2(v_4, \alpha) = (v_9, \gamma)$

$Emb_2(v_5, \eta) = (v_8, \eta)$, for each $\eta \in \Delta$

$d_3$ :

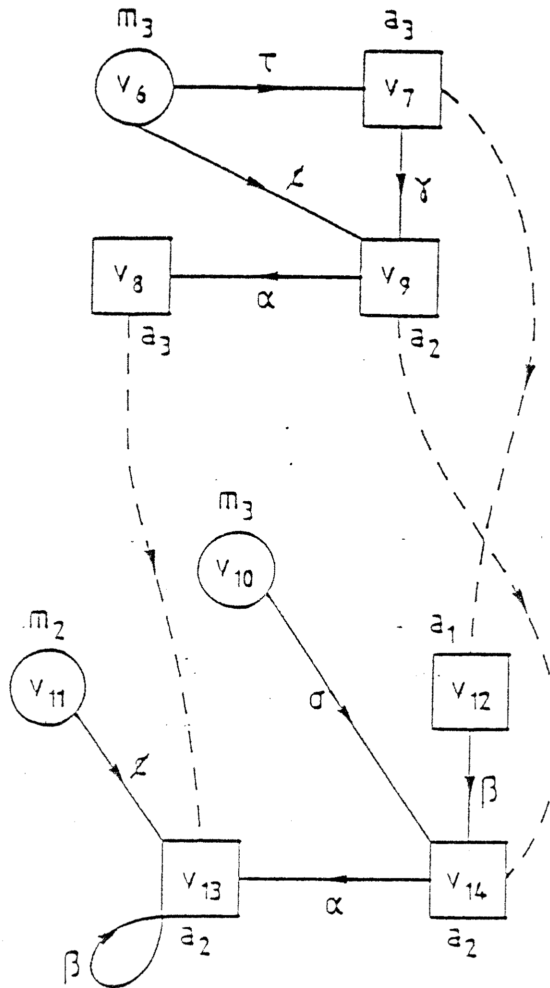



Figure 7.

$Emb_3(v_7, \gamma) = (v_{10}, \sigma)$

$Emb_3(v_8, \gamma) = (v_{12}, \alpha)$

$Emb_3(v_9, \alpha) = (v_{13}, \beta)$

$Emb_3(v_9, \beta) = \{(v_{11}, \tau), (v_{14}, \gamma)\}$

$d_4$ :

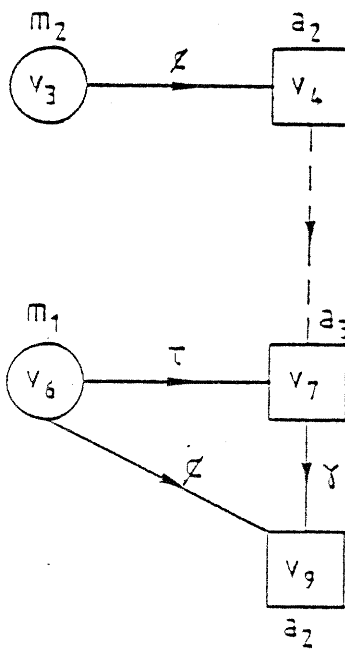



Figure 8.

$Emb_4(v_3, \sigma) = (v_9, \alpha)$

$Emb_4(v_3, \tau) = \{(v_6, \sigma), (v_9, \beta)\}$

$Emb_4(v_4, \alpha) = (v_9, \gamma)$

*Example 4.*

Let $d_5, d_6, d_7$ be as below. Then $d_5 \in Aug(d_1)$, $d_6 \in Aug(d_2)$ and $d_7 \in Aug(d_3)$.

$d_5$ :

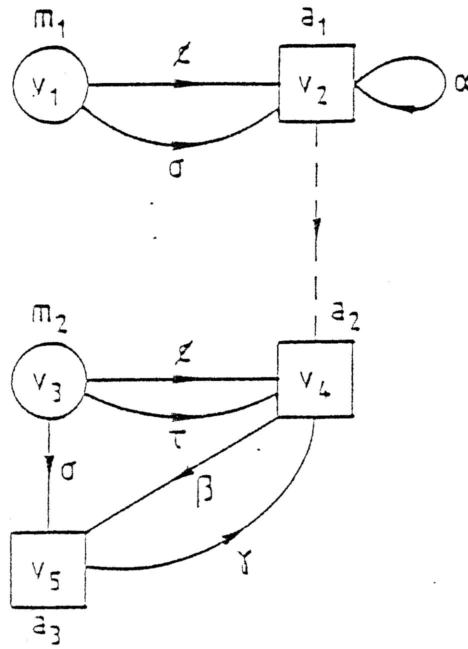



Figure 9.

$Emb_5 = Emb_1$

$d_6$ :

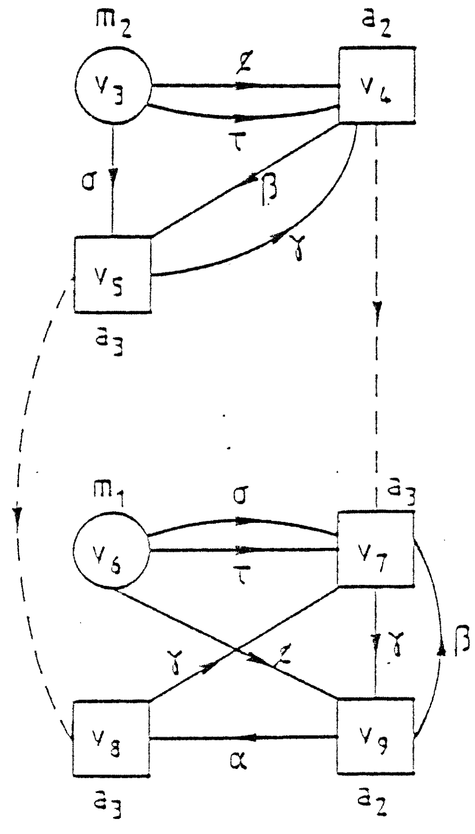



Figure 10.

$Emb_6 = Emb_2$

$d_7$ :

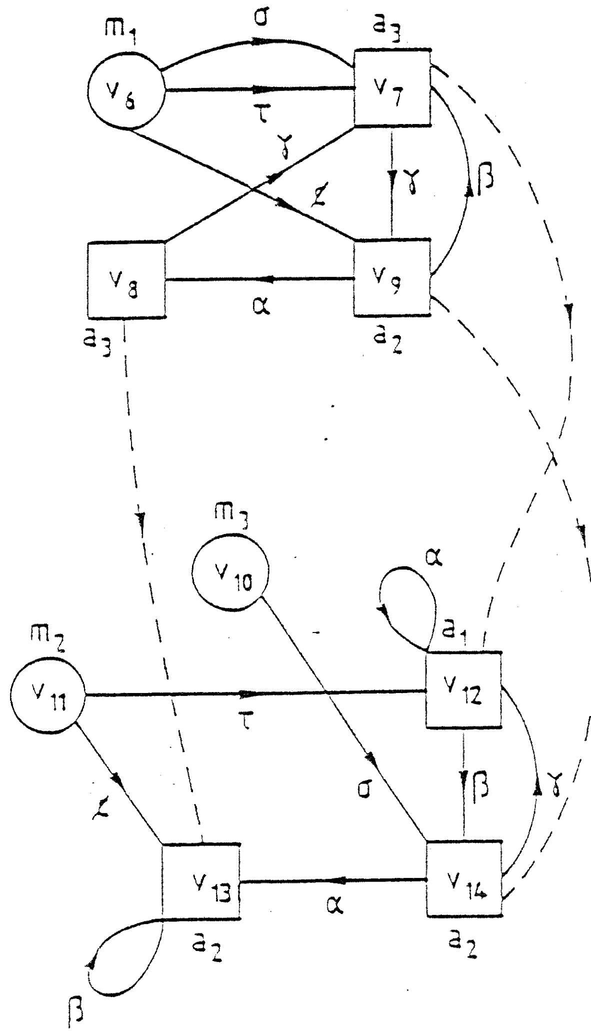



Figure 11.

$Emb_7 = Emb_3$

*Example 5.*

For $d_8$ as below, $d_8 = d_1$ <u>cc</u> $d_6$.

$d_8$ :

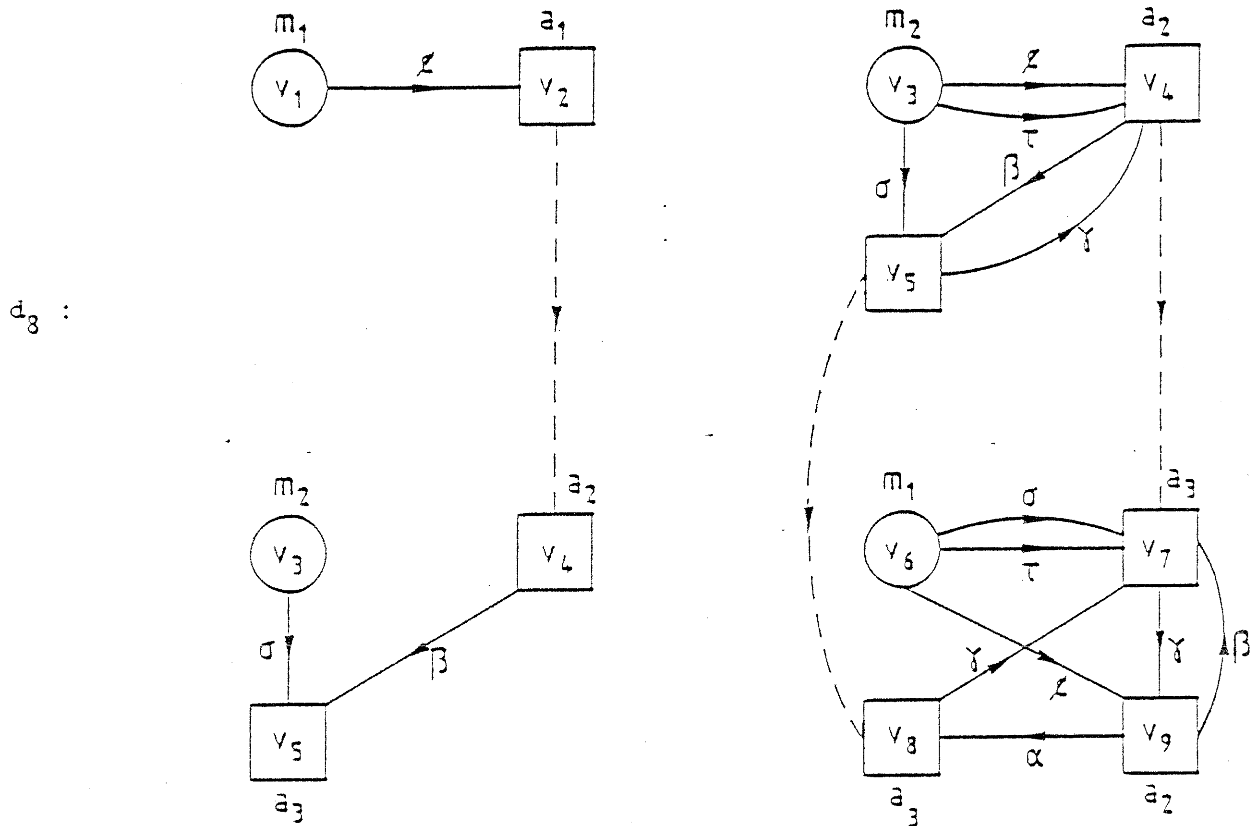



Figure 12.

$Emb_8(v_1,\sigma) = \{(v_5,\gamma),(v_3,\tau)\}$

$Emb_8(v_2,\alpha) = (v_3,\emptyset)$

$Emb_8(v_2,\beta) = (v_4,\alpha)$

$Emb_8(v_3,\sigma) = (v_9,\alpha)$

$Emb_8(v_3,\tau) = \{(v_6,\sigma),(v_9,\beta)\}$

$Emb_8(v_4,\alpha) = (v_9,\gamma)$

$Emb_8(v_5,\eta) = (v_8,\eta)$, for each $\eta \in \Delta$

*Example 6.*

For $d_9$ and $d_{10}$ as below, $d_9 = d_1 \ \underline{sa} \ d_2$ and $d_{10} = d_2 \ \underline{sa} \ d_3$.

$d_9$ :

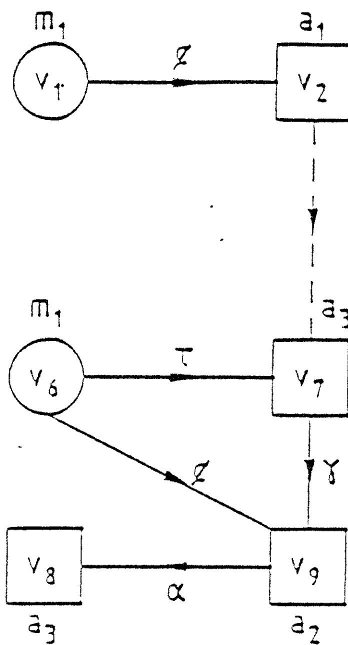



Figure 13.

$Emb_9(v_1,\sigma) = \{(v_8,\gamma),(v_6,\sigma),(v_9,\beta)\}$

$Emb_9(v_2,\beta) = (v_9,\gamma)$

$d_{10}$ :

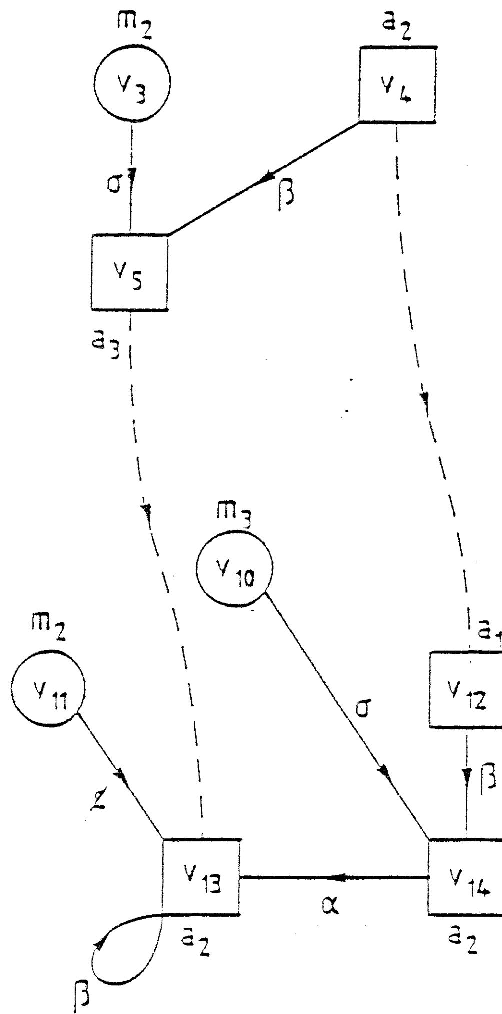



Figure 14.

$$Emb_{10}(v_3,\sigma) = (v_{13},\beta)$$

$$Emb_{10}(v_3,\tau) = \{(v_{11},\tau),(v_{14},\gamma)\}$$

$$Emb_{10}(v_5,\gamma) = (v_{12},\alpha)$$

*Example 7.*

Let $d_{11}$ be as below. Then $d_{11} = d_6 \; \underline{sq} \; d_7$ and $d_{11} \in Aug(d_{10})$. Since $d_6 \in Aug(d_2)$, $d_7 \in Aug(d_3)$ and $d_{10} = d_2 \; \underline{sq} \; d_3$, this illustrates (1) of Lemma 5.

$d_{11}$ :

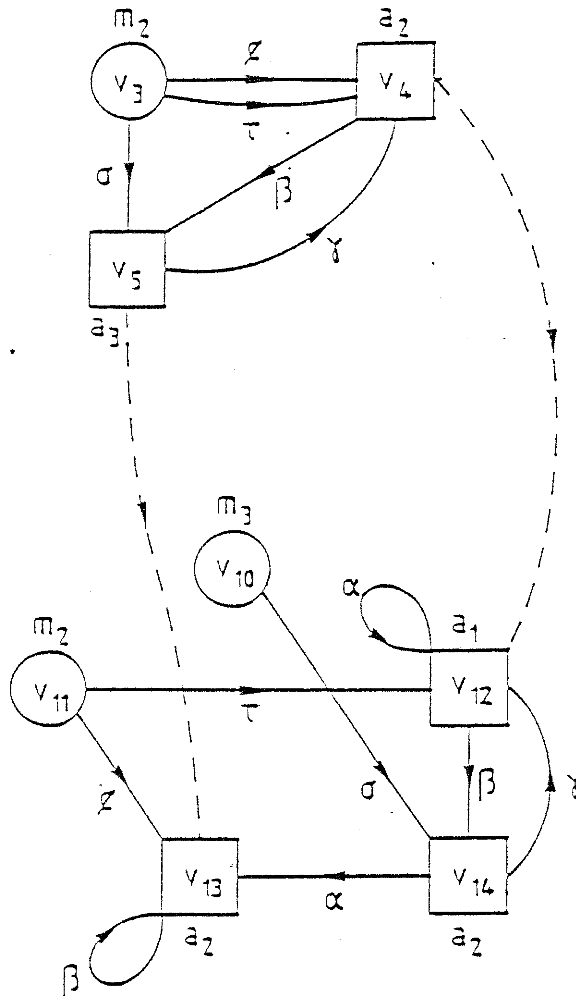



Figure 15.

$Emb_{11} = Emb_{10}$

*Example 8.*

Let $d_{12}$ be as below. Then $d_{12} = d_9 \underline{sa}\ d_3 = d_1 \underline{sa}\ d_{10}$. Since $d_9 = d_1 \underline{sa}\ d_2$ and $d_{10} = d_2 \underline{sa}\ d_3$, this illustrates (2) of Lemma 5.

$d_{12}$ :

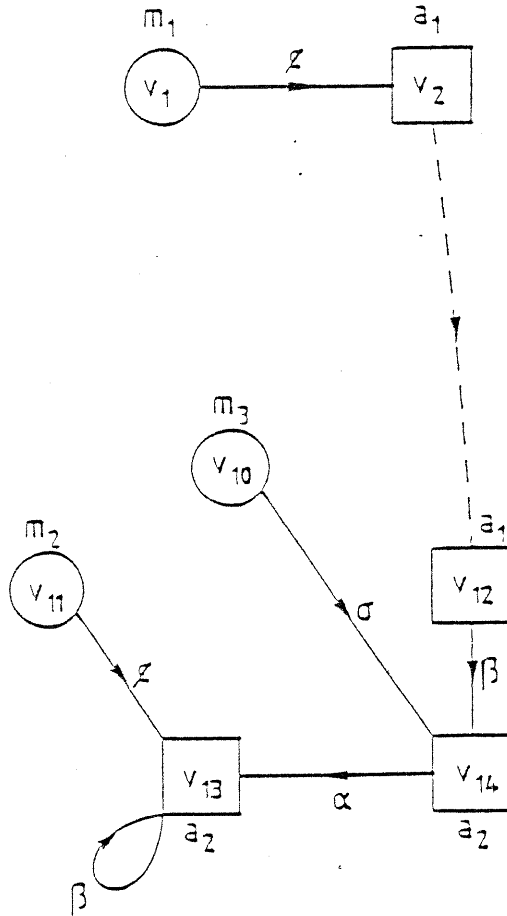



Figure 16.

$Emb_{12}(v_1, \sigma) = \{(v_{12}, \alpha), (v_{11}, \tau), (v_{14}, \gamma)\}$

*Example 9.*

Let $d_{13}$ be as below.

Then $d_{13} = d_9$ <u>cc</u> $d_{11}$ = ($d_1$ <u>sa</u> $d_2$) <u>cc</u> ($d_6$ <u>sa</u> $d_7$) = ($d_1$ <u>cc</u> $d_6$) <u>sa</u> ($d_2$ <u>cc</u> $d_7$).

This illustrates (3) of Lemma 5.

$d_{13}$ :

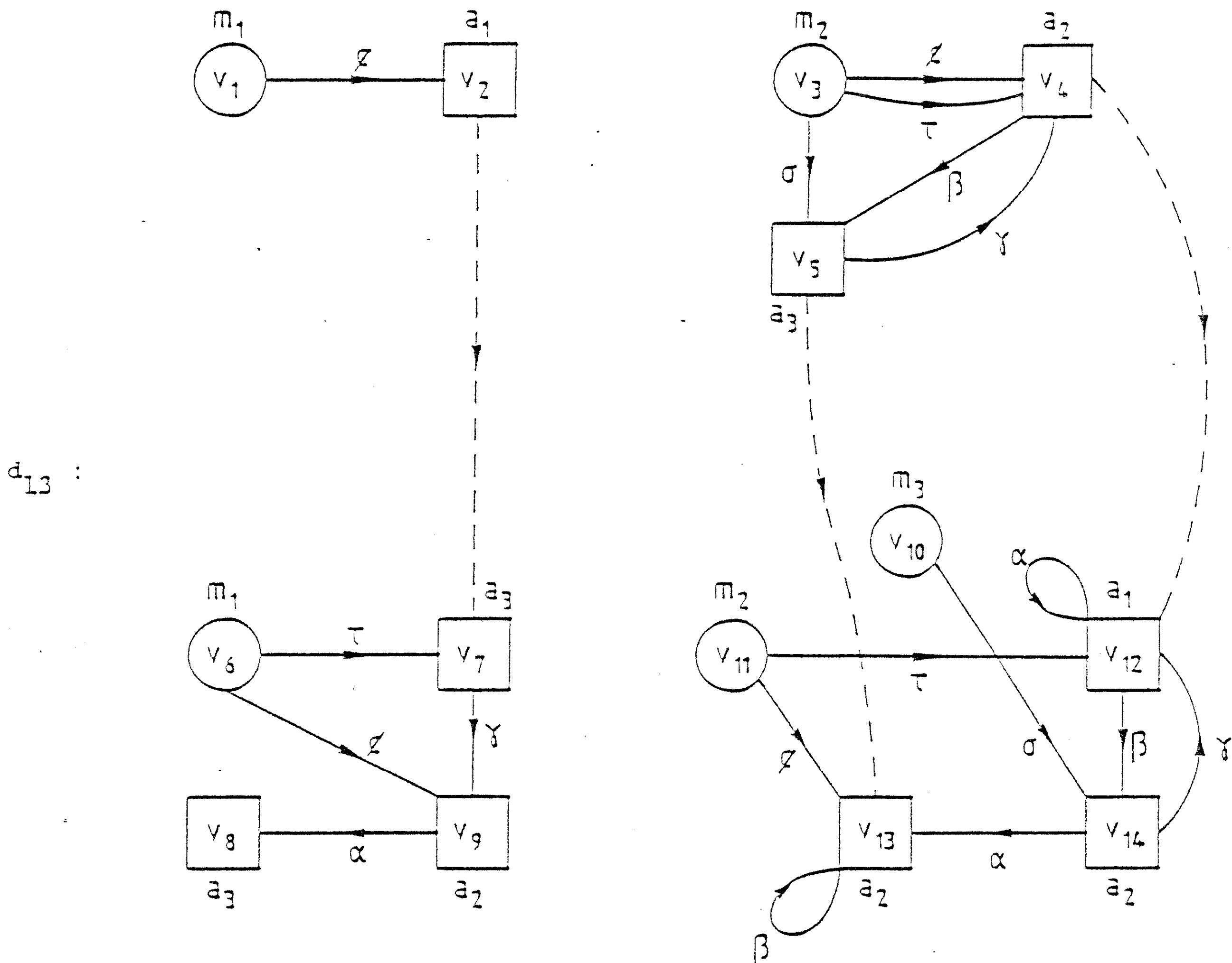



Figure 17.

$Emb_{13}(v_1,\sigma) = \{(v_8,\gamma),(v_6,\sigma),(v_9,\beta)\}$

$Emb_{13}(v_2,\beta) = (v_9,\gamma)$

$Emb_{13}(v_3,\sigma) = (v_{13},\beta)$

$Emb_{13}(v_3,\tau) = \{(v_{11},\tau),(v_{14},\gamma)\}$

$Emb_{13}(v_5,\gamma) = (v_{12},\alpha)$

*Example 10.*

Let $d_{14}$ be as below.

Then $d_{14} \in Aug((d_5 \underline{sq}\ d_6)\ \underline{cc}\ d_4')\ \underline{cc}\ Nrep_{(x_1,x_3,m_3)}\ \underline{cc}\ Nrep_{(x_2,x_4,a_2)}$, where $d_4'$ is isomorphic with $d_4$.

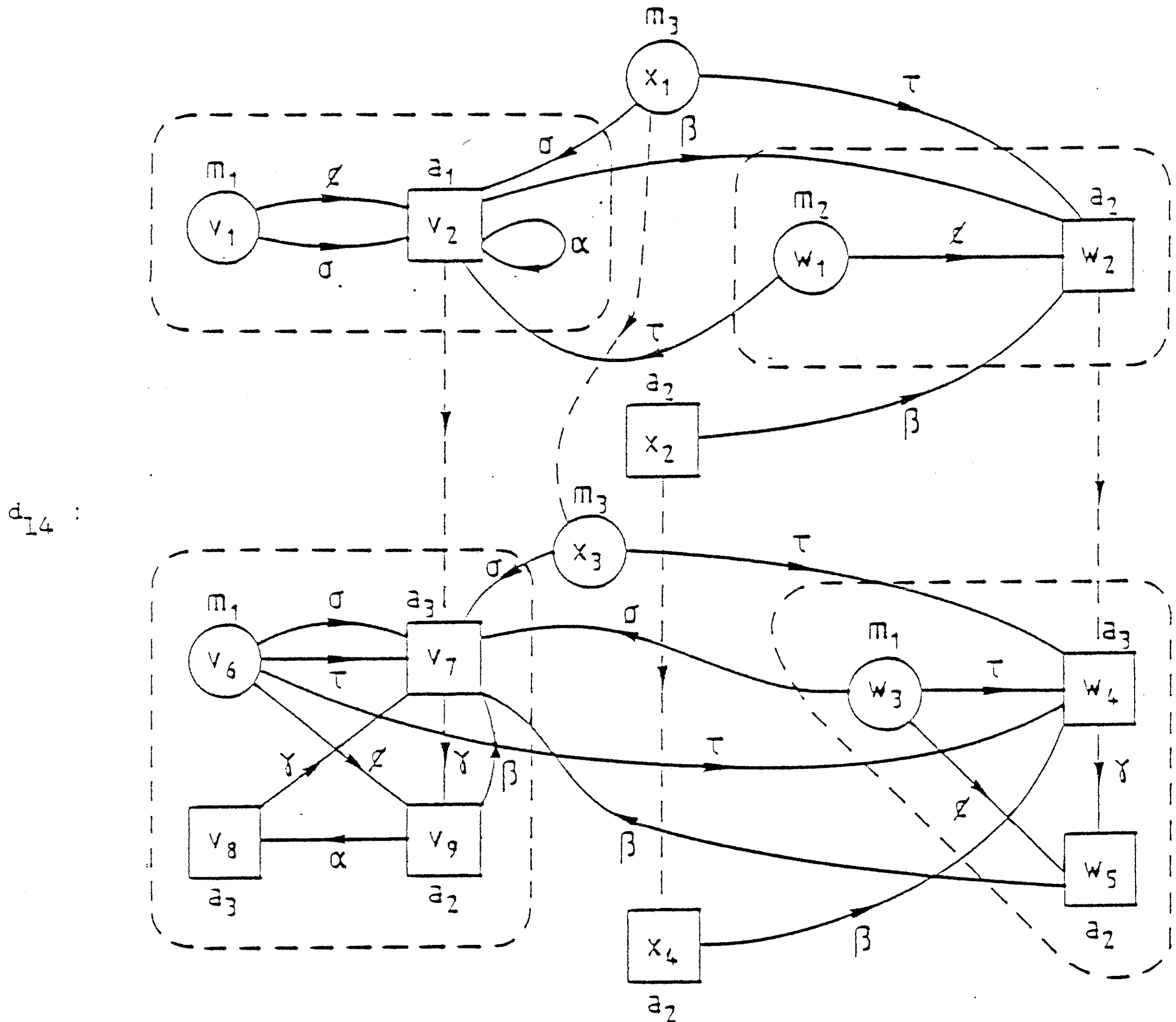



Figure 18.

$Emb_{14}(v_1,\sigma) = \{(v_8,\gamma),(v_6,\sigma),(v_9,\beta)\}$

$Emb_{14}(v_2,\beta) = (v_9,\gamma)$

$Emb_{14}(w_1,\sigma) = (w_5,\alpha)$

$Emb_{14}(w_1,\tau) = \{(w_3,\sigma),(w_5,\beta)\}$

$Emb_{14}(w_2,\alpha) = (w_5,\gamma)$

$Emb_{14}(x_1,\eta) = (x_3,\eta)$, for each $\eta \in \Delta$

$Emb_{14}(x_2,\eta) = (x_4,\eta)$, for each $\eta \in \Delta$

## 8. DISCUSSION

The aim of this paper was to introduce actor grammars as a formalization of actor systems. Since configurations in actor systems may be seen as graphs, and the dynamic evolution of actor systems may be seen as a transformation process of graphs, graph grammars seem to be quite suitable to formalize actor systems. We hope that this point was well-illustrated in our paper, where basic notions of actor grammars were motivated by basic notions of actor systems. In this way, some of the basic properties of structured transformations we have investigated express basic properties of the dynamic evolution of configurations in actor systems.

Initially, this paper was intended as the "full version" of [JR87]. However, during the work on the current paper we have changed the notion of a configuration graph and, quite considerably, the methodology of defining derivations and structured transformations in actor grammars. The relationship between the two papers was discussed in Section 6.

The approach to graph grammars introduced in this section is similar to the algebraic approach to derivations of type-0 phrase structure grammars based on X-categories (see, e.g., [H66],[B74]). Viewed as morphisms in an X-category, derivations of a type-0 grammar can be composed sequentially (by the usual composition of morphisms) and concurrently (by the X-operation on morphisms). The main law of an X-category is the one corresponding to (3) of Lemma 6. Clearly, before one can use the X-category formalism for graph grammars, an additional operation is needed : augmentation. It remains to be seen whether the methods used in X-category theory can be generalized to graph grammars.

Clearly, a lot of work remains to be done in order to understand the relationship between actor grammars and actor systems and, in particular, in order

to understand the usefulness of actor grammars in providing a workable formalism for dealing with actor systems. Here are some natural lines of research to be pursued to this aim.

(1) Technical properties of structured transformations should be investigated ; also, the notion of a derivation as a stepwise realization of a structural transformation should be better understood.

(2) The relationship of actor grammars to other models of concurrency, such as, e.g., Petri nets (see e.g.,[BRG87]), should be investigated.

(3) The notion of a process in an actor system should be defined in the framework of actor grammars. Here, the existing knowledge on processes in graph grammars (see, e.g., [K87]) can be quite useful.

(4) As discussed before, in this paper we have introduced a novel methodology for formalizing rewriting processes in graph grammars. It is certainly worth to pursue this line of research and investigate the usefulness of this methodology in the framework of graph grammars.

Acknowledgement The authors are indebted to J. Engelfriet and G. Leih for comments on the previous version of this paper.

## REFERENCES


[A86] G.A. Agha, *Actors : A Model of Concurrent Computation in Distributed Systems*, M.I.T. Press (1986).

[B74] D.B. Benson, *Semantic Preserving Translations*, Math. Syst. Theory 8, 105-126 (1974).

[BRG87] W. Brauer, W. Reisig and G. Rozenberg, eds., *Advances in Petri Nets 1986*, I and II, Lecture Notes in Computer Science 254 and 255, Springer Verlag (1987).

[C81] W.D. Clinger, *Foundations of Actor Semantics*, Ph.D. Thesis, Massachusetts Institute of Technology (1981). Available as technical report 633, M.I.T. AI Lab.

[E87] H. Ehrig, *Tutorial Introduction to the Algebraic Approach of Graph Grammars*, in [ENRR87], 3-14.

[ENRR87] H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld, eds., *Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 291, Springer Verlag (1987).

[H66] G. Hotz, *Eindeutigkeit und Mehrdeutigkeit Formaler Sprachen*, EIK 2, 235-246 (1966).

[H77] C. Hewitt, *Viewing Control Structures as Patterns of Passing Messages*, Journal of Artificial Intelligence 8, (1977), 323-364.

[HB77] C. Hewitt and H. Baker, *Laws for Communicating Parallel Processes*, IFIP-77, Toronto (1977), 987-992.

[JR87] D. Janssens and G. Rozenberg, *Basic Notions of Actor Grammars : A Graph Grammar Model for Actor Computation*, in [ENRR87], 280-298.

[JR88] D. Janssens and G. Rozenberg, *The Destination - Normal Form for Actor Grammars*, in preparation.

[K87] H.J. Kreowski, *Parallelism and Concurrency in Graph Grammars*, Bulletin of the EATCS, 25, 63-79 (1987).

[L81] H. Lieberman, *A Preview of Act 1*, technical report 625, M.I.T. AI lab (1981).

[R87] G. Rozenberg, *An Introduction to the NLC Way of Rewriting Graphs*, in [ENRR87], 55-66.

[T83] D. Theriault, *Issues in the Design and Implementation of ACT 2*, technical report 728, M.I.T. AI lab (1983).