December 1986 LIDS-P-1627

CONTROL OF LARGE DISCRETE EVENT SYSTEMS: CONSTRUCTIVE ALGORITHMS

by
Gilead Tadmor[†]
and
Oded Maimon[*†]

Abstract

Extending recent developments in the theory of controlled discrete event systems, constructive algorithms are derived for some basic elements in large system integration, analysis and control systhesis. Recognizing the fact that in large, combined systems (such as intricate Flexible Manufacturing Systems) different users have different views, and interact with only portions of the overall systems, a class of procedures for integration of those individual interactions is an acutely missing link both in the literature and in practice. The constructions follow, thereby, a hierarchical approach. Starting with the level of the small subunit, and local, incomplete information, global products are obtained via automated integration processes. The proposed schemes provide orderly and rapid substitutes for many current ad-hoc developments, which are mostly done with great difficulties, high expenditures, prolonged development time, and compromised results. The ideas presented here will be further utilized in prospected continuations of this work: both in system analysis, and in forming a man-machine interface with detailed constraint-understanding mechanisms.

[†]Laboratory for Information and Decision Systems, MIT, Cambridge, MA 02139.

[*]Digital Equipment Co., Maynard, MA 01754.

1. INTRODUCTION

In a recent series of papers [6-8, 10-12], a basis has been laid for a new approach to control of discrete event systems. By that approach, concepts of automata and formal languages are utilized for the modeling, analysis, and systhesis of systems, their operational constraints, and supervisory mechanisms. The world of automata is to assume, thereby, the role played by time-driven dynamical systems (differential and difference equations, etc.) in classical control theory.

The new theory is versatile in its applications. It is of particular importance in areas where paradigms for efficient and orderly system analysis and synthesis (versus lengthy, costly, and all-too-often disorderly, ad-hoc work) are much needed. The authors' motivation come mainly from problems of operational control in Flexible Manufacturing Systems (FMSs), but the reader can easily think of other areas, where large integrated, hierarchical, operational and communication systems are the subject matter.

In the most part, the articles cited above are dedicated to presentation of new objects into the theory, and to their structural analysis. In real-life, implementation of the theory will often be in very large, combined systems, and give rise to untractable problems' size. Our aim here is, therefore, to show how can the theory's fundamental elements be actually (automatically) constructed in such systems. We present a scheme whose input consists of graph-models for various system subunits ([6] and [8] explain how to construct such models), and of a set of operational constraints, each involving one or several of this subunits, which the user

wishes to impose. The output comprises of both analytic tools (e.g. graphs of maximal legal- and maximal controllable legal-sublanguages) and feedback supervisors, which impose the prescribed constraints. These are to be constructed first in the subunit level, and then, integrated to yield their counterparts in the combined system, global level.

This scheme complies with the viewpoint (taken by the authors) which stresses the hierarchical structure of large systems: the algorithms prompt handling first, smaller subunits (with partial and local information pieces), and then, yield the global picture (and whatever global products) as a result of automated system-integration. This scheme of work has many advantages. The following are among the most obvious ones.

First, partial pieces of information, and local operational constraints, are mostly all one may hope to have as primitive input. Starting, rather, with global-level system description, and statement of constraints, would anyway require preparatory compilation steps. Yet these would make the developed techniques less accessible to the non-expert. In point of fact, even constraints of the fairly simple type, considered in this work (i.e. forbidden strings of transitions), would mostly be the result of non-trivial preliminary compilations of simpler descriptions of the illegal behaviour, in smaller frameworks and higher-level (i.e. closer to "natural") language. The latter are discussed in a separate article [8].

A second advantage of the hierarchical-modular approach is its structural-flexibility. The ability to decompose the system arbitrarily into smaller subsystems, to define constraints and make preliminary analysis

and synthesis in the subsystem's framework, on the one hand, and to easily integrate local products into a global picture, on the other, enables fast and frequent changes in the system's operational rules, inner structure, units configurations, etc. Taking FMSs, for example, it is exactly this freedom which makes the FMS the efficient and versatile instrument it can be. (Not surprisingly, failures and inefficient usage of FMSs in this country are often attributed to heavy software implementation problems which markedly reduce the availability of their potential flexibility.)

Third, the structural flexibility, with the motivations mentioned above, carries over to the supervisory level: the hierarchical-modular approach allows easy and rapid moves between centralized and decentralized control mechanisms. At different times, and according to needs of different system-configurations, it thus enables local supervisors in subunits which are more susceptible to changes, and higher-level, hence more efficient supervisors, in other subsystems. This flexibility also allows more efficient allocation of available processors for the changing supervisory tasks.

Finally, as is painfully well known, problems which involve automata tend to grow exponentially-hard with system's size. The difficulties in handling large systems, much too often transform into untractable implementation problems. It is therefore a practical and useful policy to perform as much as possible of the required work in smaller frameworks.

As for the technical aspect of our scheme, it is based on two simple and natural ideas. Treating each construction problem, we regard both any graph's transition rules, and the imposed operational constraints, as a set

of predicates which should be satisfied by the resulting system. (cf. [11].) Using multiple labels for vertices in the produced graphs, we can dedicate a label-component to the task of maintaining each of the predicates true, through all possible system transitions. In some cases, though, that multiple-labeling approach results in redundant storing of information and yields unnessarily rich graph-structures. Then we invoke a vertex-lumping technique [6,7], which provides more economic models.

2. PRELIMINARIES

Let us start with a brief survey of notation and terminology[1]: A controlled discrete event system is modelled by a connected direct-graph

$$G = (Q, \Sigma, \Sigma_c, \delta, q_0).$$

Here Q is the finite set of states (vertices). Letters from the finite alphabet Σ label transitions (arcs). $\delta: Q \times \Sigma \to Q$ is the transition function: if σ labels an arc from p to q ($p,q \in Q$, $\sigma \in \Sigma$) then $\delta(p,\sigma) = q$, if an arc σ does not leave p then $\delta(p,\sigma) = ud$ (= undefined). (In some cases, δ may be a multivalued function $\delta: Q \times \Sigma \to 2^Q$. Then G be a nondeterministic graph, and the notation "ud" be substituted by the empty-set ϕ.) As in previous work, transitions are assumed to occur spontaneously and asynchronously.

A state $q_0 \in Q$ is fixed as the initial state in G. We shall mostly denote the initial state simply by 0. Let Σ^* be the set of all finite strings of letters from Σ. The function δ extends inductively to the domain $Q \times \Sigma^*$: $\delta(q,\sigma_1 \ldots \sigma_n \sigma_{n+1}) := \delta(\delta(q,\sigma_1 \ldots \sigma_n), \sigma_{n+1})$ and $\delta(ud,\sigma) := ud$. Now we can define the language of G -

$$L(G) := \{s \in \Sigma^* : \delta(0,s) \neq ud\}.$$

The language labels the set of pathes from the origin in G.

---

[1] An extensive discussion of finite automata, formal languages and direct graphs can be found in [3, chs. 1,2]. The detailed introduction to the theory of controlled discrete event systems is given in Ramadge and Wonham's paper [10]. Since our notations differ at some points from the above, the reader may also consult our previous article [6], a paper we shall refer to later as well.

The alphabet Σ decomposes into the two disjoints subsets: Σ_c, the set of labels of controlled transitions and $\Sigma_u = \Sigma \backslash \Sigma_c$, the uncontrolled alphabet set. Controlled transition can be disabled or enabled (but not enforced), at will, at different times.

A supervisor $\underline{S} = (S,\phi)$ consists of a direct graph $S = (X,\Sigma,\Sigma_c,\xi,0)$, and of a set valued function $\phi: X \to 2^{\Sigma_c}$. Events carrying the same labels should occur simultaneously in G and in S. The function ϕ determines the control policy: having arrived at the vertex x in S, transitions labeled by letters in the set $\phi(x)$ are disabled in G. (In actual implementation, while the graph G models a real plant, $\underline{S}$ would mostly stand for a computer program, driven by the output of G.)

The coupled system, denoted $\underline{S}/G$ is constrained by the two transition functions, δ and ξ, and by the control function ϕ: A transition σ may leave the pair of states $(q,x) \in Q \times X$ only if both $\delta(q,\sigma)$ and $\xi(x,\sigma)$ are defined and $\sigma \in \phi(x)$. The language of the coupled system, thus constrained, is denoted by $L(\underline{S}/G)$. (Notice that since it is physically impossible to disable transitions from Σ_u, if the coupled system arrives at (q,x) and $\sigma \in \Sigma_u$, then $\delta(q,\sigma) \neq$ ud implies $\xi(q,\sigma) \neq$ ud and $\sigma \notin \phi(x)$.)

For notational convenience we define, given a supervisor $\underline{S}$, a second set valued function, $\varphi : X \to 2^{\Sigma}$ as follows

$$\varphi(x) := \{\sigma \in \Sigma : \xi(x,\sigma) \neq \text{ud}\}.$$

Using this notation we state the following three hypothesis -

(H1) If $s \in L(\underline{S}/G)$, $\sigma \in \Sigma$, $s\sigma \in L(G)$, and $x := \xi(0,s)$ then either $\sigma \in \phi(x)$ or $\sigma \in \varphi(x)$. (Supervisors with this property are called complete in [10]. It is

a technical assumption which means that a transition cannot be disabled just by the structure of S, it must be positively forbidden via the control function ϕ, or be allowed.)

(H2) Every vertex in S can be reached via a word in $L(\underline{S}/G)$. (Trim is the term used in [10].) Also, each arc in S is part of a path labeled by a word in $L(\underline{S}/G)$.

(H3) The set $\phi(x)$ is minimal, i.e. a letter σ may belong to $\phi(x)$ only if there exists $s\varepsilon L(\underline{S}/G)$ such that $x = \xi(0,s)$ and $s\sigma\varepsilon L(G)$.

These hypothesis enforce an economical and simple structure on supervisors, and will be assumed throughout. We shall use the term well-defined, as an attribute, when wishing to stress that indeed, a specific supervisor complies with H1-3. It turns out that H1-3 are not restrictive:

Observation 2.1. (I) Given a supervisor $\underline{S}$ for G, there exists another supervisor $\tilde{\underline{S}}$, which complies with H1-3, and such that $L(\tilde{\underline{S}}/G) = L(\underline{S}/G)$. (i.e. H1-3 cause no reduction in generality.)

(II) Hypotheses H1-3 imply the following equalities -

(i) $L(\underline{S}/G) = L(S) \cap L(G)$

(ii) $\psi(x) = \{\sigma: \exists s\varepsilon L(S)\cap L(G)$ such that $x = \xi(0,s)$ and $s\sigma \varepsilon L(S)\cap L(G)\}$

(iii) $\phi(x) = \{\sigma: \exists s \varepsilon L(S)\cap L(G)$ such that $x = \xi(0,s)$ and $s\sigma \varepsilon L(G)\backslash L(S)\}$

(iv) $\phi(x)\cap\psi(x) = 0$.

(III) Conversely, (by connectedness of S), properties (ii) and (iii) imply hypotheses H1-3.

The proof can be found in [6]. Notice that in well defined supervisors

the function θ is totally determined by the graphs S and G.

A supervisor is built in order to impose a given set of operational constraints. Here we consider constraints that are put each in terms of a forbidden chain of events[2], say $t=\tau_1\tau_2\ldots\tau_7$, $\tau_i\varepsilon\Sigma$, which should not be completed. Words in L(G) that satisfy the imposed constraints (i.e., which do not contain forbidden strings) are termed legal, and the collection of all legal words is the maximal legal sublanguage of L(G), denoted $L_L(G)$. It follows from our assumptions that $L_L(G)$ is a regular language, i.e. it has a direct graph model. We denote by G_L any graph for which $L(G_L) = L_L(G)$. (There might be more than one such graph.)

The very maximal legal behavior, $L_L(G)$, may not be ever obtained by use of a supervisor: this happens if some words in $L_L(G)$ can be concatenated by strings from Σ_u^* (i.e., such which cannot be disabled), to form illegal words in L(G). A sublanguage K L(G) is called controllable when that cannot happen, namely if $s\varepsilon K$ and $\sigma\varepsilon\Sigma_u$ then $s\sigma\varepsilon K$.

Lemma 2.2 [10, Proposition 5.1]. A supervisor $\underline{S}$ such that $L(\underline{S}/G) = K$ exists, if and only if K L(G) is a controllable sublanguage. If indeed K is controllable and H is a graph model for K (i.e., L(H)=K), then one can construct $\underline{S}$ with S=H.

Thereby, given operational constraints on G, hence the sublanguage $L_L(G)$, we are interested in finding the maximal controllable sublanguage of the latter; that is, the maximal controllable legal sublanguage of L(G),

---

[2]A more general framework may include forbidden pathes (rather than strings). A path may include closed loops. Thus it may stand for countably many different strings. The tools developed here easily extend to that level of generality, and it is for the sake of clarity that we choose the simpler case.

denoted $L_C(G)$. We denote by G_C any graph for which $L(G_C) = L_C(G)$.

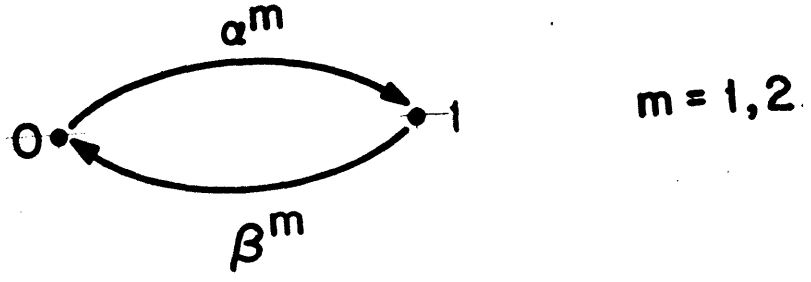
Both G_L and G_C are important tools for the analysis of the legal behavior in a given system, an analysis which we shall further pursue in a forthcoming paper. As stated in the Lemma 2.2, G_C can also serves as a graph of a supervisor which imposes the given constraints [6,10]: just set $S=G_C$ and define the control function $\phi(\cdot)$ by formula (iii) in Observation 2.1.

A notion which comes naturally into the theory is that of a direct-product of many controlled discrete event systems: Let $G_1,\ldots,G_M$ be the models of M controlled discrete event systems, with $G_m = (Q_m,\Sigma_m,\Sigma_{mc},\delta_m,0_m)$. The direct product of these systems, denoted $\bigotimes_{m=1}^{M} G_m$, is the graph of the system formed by coupling G_m, $m=1,\ldots,M$: A state in $G := \bigotimes_{m=1}^{M} G_m$ is the M-tuple of states in all the M systems. The set of transitions (i.e., the alphabet) of G is the union $\Sigma := \bigcup_{m=1}^{M} \Sigma_m$; if $\sigma_m \in \Sigma_m$ and $\delta_m(p_m,\sigma_m) = q_m$ we define $\delta((p_1,\ldots,p_m,\ldots,p_n),\sigma_m) := (p_1,\ldots,p_{m-1},q_m,p_{m+1},\ldots,p_n)$. The language of G is therefore the one termed in Ramadge and Wonham [10] - the shuffled language of $G_1,\ldots,G_M$.

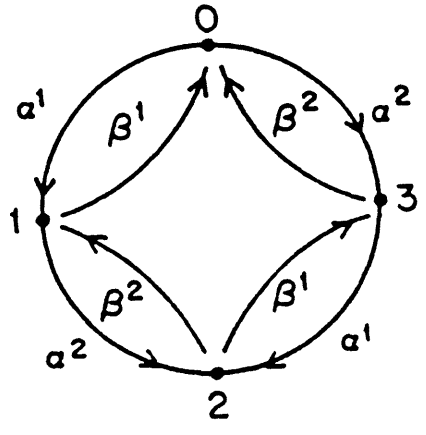
3. CONSTRUCTION OF G_L

We consider a system modeled by G and a set of N forbidden strings, $t_n = \tau_1^n \tau_2^n \ldots \tau_{f(n)}^n$, which should not be completed as chains of transitions. Here is an example:

Example 3.1. (An FMS which motivates this example is described in [6].) Consider two simple two-states systems



and let G be their direct product. G is modeled by the following graph



Suppose we want to impose a FIFO-type behavior, with respect to system #2, as follows: if α^2 occurred before α^1 then β^2 will follow α^2 before β^1 follows α^1. This amounts, as one can easily check, to two forbidden strings: $\alpha^2\alpha^1\beta^1$ and $\alpha^2\beta^1\alpha^1\beta^1$.

Our task in this section is to build G_L. (Recall that G_L is a graph such that $L(G_L)$ is the maximal sublanguage of L(G), whose words do not

contain any of the forbidden strings.) Our method is to track processes which follow prefixes of forbidden strings. To that effect we first build N string-recognizors, $R_1,\ldots,R_N$, one for each constraint.

String recognizors are abound and are implemented widely (e.g., in text editors). The ones we construct now are tailored for our particular needs. We ignore, henceforth, the trivial case of a forbidden string of length f=1, and assume $f\geq 2$ throughout.

Algorithm 3.2 (a string-recognizor construction).

Input - An alphabet $\Sigma = \{\sigma_1,\ldots,\sigma_L\}$ and a string $t=\tau_1\ldots\tau_f$, with $\tau_i\varepsilon\Sigma$, i=1,...,f, $f\geq 2$.

Output - A non-deterministic directed graph $R = (P,\Sigma,\Gamma,0)$. (We supress the irrelevant notations of Σ_c in R. Note that Γ is a set valued transition function.)

Variables - i,j - labels of vertices in R

ℓ - a label of a letter in Σ.

Definitions of the simple terminology we use in the algorithms henceforth are in the Appendix. Construction steps will be explained in comments (between asterisks), and in more detail, in following propositions and their proofs.

```
begin(0)
   P := {0,1,...,f-1};
* all letters will label arcs to the origin *
   do(1) i=0, f-1
      do(2) ℓ=1, L
         Γ(i,σ_ℓ) := {0};
```

```
      end(1);
   end(2);
* second construction loop *
   do(3) i=1, f-1
      do(4) j=1, f-i
* if τ1τ2...τj = τiτi+1...τi+j-1 then τj+1 will also lead from i+j-1 to j+1,
j≥0 *
         Γ(i+j-2, τj) := Γ(i+j-2, τj) ∪{j};
         if τj≠τi+j-1 then exit(4);
      end(4);
   end(3);
* in compliance with the constraint, τ1...τf-1τf must not be completed *
   Γ(f-1, τf) := ∅;
end (0).
```

Considering this algorithm, we have

Proposition 3.3. A word $t \in L(R)$ labels a path to the vertex f-1 (i.e., $f-1 \in \Gamma(0,t)$) if and only if t terminates with the string $\tau_1 \ldots \tau_{f-1}$.

Proof. Necessity. We establish a stronger claim: for each $j=1,\ldots,f-1$ and $t \in L(R)$, we have $j \in \Gamma(0,t)$ only if t ends with $\tau_1 \ldots \tau_j$.

The proof goes by (finite) induction on k, $1 \leq k \leq f-1$. For each k we show that if $j \geq k$ and $j \in \Gamma(0,t)$ then t ends with $\tau_{j-k+1} \ldots \tau_j$. For k=1 the claim follows since j was added to $\Gamma(x,\sigma)$ only if $\sigma=\tau_j$. This completes the proof if f=2.

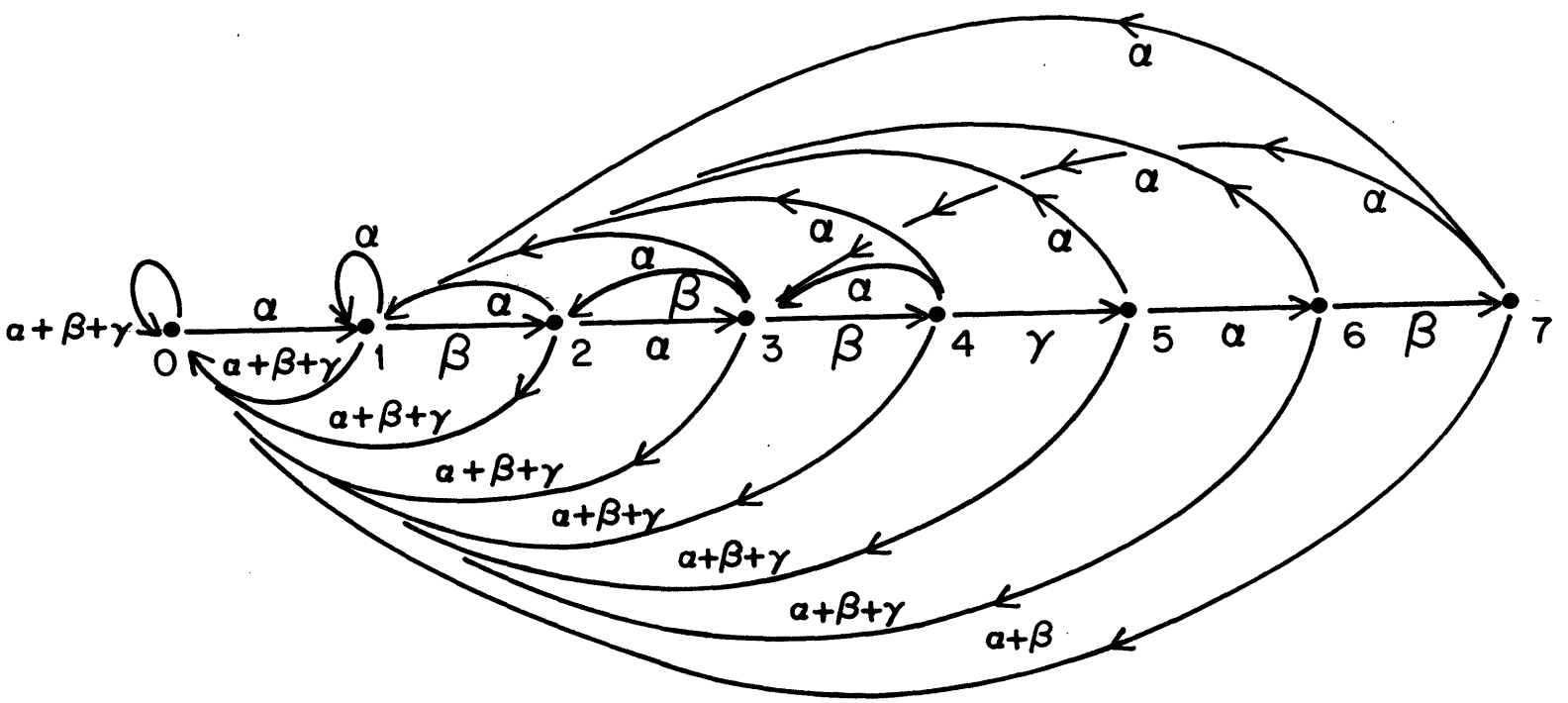
If f>2, suppose the claim holds for some k, $1 \leq k \leq f-2$. It is of interest to check the vertices $j=k+1,\ldots,f-1$. Fix one such vertex j: it is reached

via τ_j from j-1 (take i=1 in the second construction loop), and possibly, from some vertex x with $x \geq j$ (take $i \geq 2$ and x=i+j-2). The latter case occurs only if $\tau_y = \tau_{y+x-j+1}$ for y=1,...,j-1. In particular the two strings of length k, $\tau_{j-k}\cdots\tau_{j-1}$ and $\tau_{x-k+1}\cdots\tau_x$, are identical.

Now we invoke the induction hypothesis: in view of the established equaltiy, all words which lead to x (and to j-1) must end with the string $\tau_{j-k}\cdots\tau_{j-1}$. Hence all words which lead to j must terminate with $\tau_{j-k}\cdots\tau_j$, as claimed.

Sufficiency. Taking j=1 in the second loop, we see that $1 \varepsilon \Gamma(x,\tau_1)$ for all x=0,1,...,f-1. Also (as we have already shown), $j+1 \varepsilon \Gamma(j,\tau_1)$ for j=1,...,f-2 (when $f \geq 2$). Hence for all x, the set $\Gamma(x,\tau_1\cdots\tau_{f-1})$ contains f-1, as claimed. □

Example 3.4. Set $\Sigma = \{\alpha,\beta,\gamma\}$, and suppose the (forbidden) string at hand, is $\alpha\beta\alpha\beta\gamma\alpha\beta\gamma$ (f=8). Then the following is the corresponding string recognizing graph, R -



We see that transitions to the origin (from each state) are possible under any label, that $\alpha(=\tau_1)$ always lead to 1, but sometimes (following $\alpha\beta$, as from 4 and 7) also to 3; and that following α, an arc β leads to 2 not only from 1, but from 3 as well.

In fact some of the arcs generated by the algorithm in R (precisely, any arc labeled τ_j from j-1, other than that leading to j; in the example: α, from 2 and 5 to 1, and β from 3 to 2) are redundant for the purpose of the construction of G_L. (This can be seen in Algorithm 3.5, hereafter.) Yet both the current construction, and its use in justifying Algorithm 3.5 seem simpler than their counterparts in a more economic structure. Since the needed corrections can be easily figured out by the interested reader,

we prefer at this stage, clarity to the little extra efficiency.

Algorithm 3.5 (Construction of G_L).

Input - A graph $G = (Q,\Sigma,\Sigma_c,\delta,0)$ with $\Sigma = \{\sigma_1,\ldots,\sigma_L\}$, and N string-recognizing graphs, $R_1,\ldots,R_N$ ($R_n=(P_n,\Sigma,\Gamma_n,0)$), each of which is the product of Algorithm 3.2, for some forbidden string.

Output - A graph $G_L = (Q_L,\Sigma,\Sigma_C,\delta_L,0)$.

Variables - $\bar{p},\bar{q}$ - N+1 tuples; p_n, $q_n \varepsilon P_n$ for n=1,...,N; $p_{N+1},q_{N+1}\varepsilon Q$. $\bar{p}$ and $\bar{q}$ label vertices in Q_L.

u,v - enumerate vertices in Q_L; thus $u(\bar{p})$ and $\bar{p}(u)$ are 1:1 functions.

w - the incumbent maximal enumeration of a vertex in Q_L.

ℓ - a letter enumeration.

n - a string-recognizor enumeration.

j,k - vertices in string-recognizing graphs.

r - a vertex in G.

* The algorithm builds G_L recursively: starting at the origin, new arcs are added if they are both legal and feasible in G.*

```
begin(0)
*define the origin as 0̄ = (0,...,0) *
   p̄ := 0̄, Q_L := {p̄}, u{p̄} := 0, w := 0;
   do(1) v=0, w
      p̄ := p̄(v);
      do(2)  ℓ = 1, L
* test of feasibility in G of the transition σ_ℓ from the state p̄ *
         if δ(p_{N+1},σ_ℓ) = ud then δ_L(p̄,σ_ℓ) := ud, end(2);
```

* henceforth $\bar{q}$ is the candidate value of $\delta_L(\bar{p},\sigma_\ell)$ *

$q_{N+1} := \delta(p_{N+1},\sigma_\ell)$;

do(3) n=1, N

* test of legality of σ_ℓ from $\bar{p}$, according to the n-th constraint *

if $\Gamma_n(p_n,\sigma_\ell) = \emptyset$ then $\delta_L(\bar{p},\sigma_\ell)$:= ud, exit(3), end(2);

* $\Gamma_n(p_n,\sigma_\ell) = \emptyset$ only if $p_n = f(n) - 1$ and $\sigma_\ell = \tau^n_{f(n)}$; the previous proposition shows that then σ_ℓ is an illegal move; if not, in order to choose $q_n \in \Gamma_n(p_n,\sigma_\ell)$ (the R_n component) check whether the n-th illegal string can be completed from q_{N+1}, in G, within less than f(n) transitions *

$\Gamma := \Gamma_n(p_n,\sigma_\ell)$;

begin(4)

k := max {j:j$\in\Gamma$};

* recall that vertices in R_n are labeled by integers 0,1,...,f(n)-1; following the proof of Proposition 3.3, $\sigma_\ell = \tau_k$ and p_n can be reached only via $\tau_1 \ldots \tau_k$ *

if k=0 then exit(4);

$r := q_{N+1}$;

* test of feasibility in G of the string $\tau_{k+1} \ldots \tau_f$ from q_{N+1} *

do(5) j = k+1, f(n)

if $\delta(r,\tau_j)$ = ud then $\Gamma := \Gamma \backslash k$, exit(5), return(4);

$r := \delta(r,\tau_j)$;

end(5);

* arrive at this point if it is possible to complete an illegal string from q_{N+1} with the string $\tau_{k+1} \ldots \tau_{f(n)}$ *

end(4);

$q_n := k$;

end(3);

* it should be $\delta_L(\bar{p},\sigma_\ell) = \bar{q}$; if there is not vertex labeled $\bar{q}$, add it *

if $\bar{q} \notin Q_L$ then $Q_L := Q_L \cup \{\bar{q}\}$, $w := w+1$, $u(\bar{q}) := w$;

$\delta_L(\bar{p}, \sigma_\ell) := \bar{q}$;

end(2);

end(1);

end(0).

Proposition 3.6. Let G_L be the product of Algorithm 3.5. Then $L(G_L) = L_L(G)$ = the maximal legal sublanguage of G.

Prior to the rigorous proof, let us review the construction's main lines: multiple state labels keep track of the process in G (and thus maintain feasibility of each transition), and in each R_n (and thus ascertain that no illegal path is pursued all the way through). The meaning of having a non zero value, say k, for the n-th label-component, is that an illegal chain of events can be completed within (f(n)-k)transitions from the current state. Wishing not to maintain redundant information, we set the n-th component as zero otherwise. In point of fact, a simpler algorithm in which R_n are substituted by deterministic analogs can be easily set, but in that version prefixes of $\tau_1 \ldots \tau_f$ will be tracked even if they cannot be feasibly completed into illegal words. The resulting G_L will then have more vertices and arcs.

Proof. Termination of the construction in finite time follows from the fact that the collection of different N+1-tupples $\bar{p}$, with $p_n \varepsilon f_n$, n=1,...,N, and $p_{N+1} \varepsilon Q$, is finite.

The inclusion $L(G_L) \subset L(G)$ follows because chains of transitions were allowed in G_L only if they were feasible in G.

For simplicity (without loss of generality) we deal henceforth with the case N=1, and set f=f(1).

Legality of $L(G_L)$ is equivalent to this next assertion: For each $\bar{p} \in Q_L$ there holds $\delta_L(\bar{p},\tau_1 \ldots \tau_f) = ud$. Indeed, fix $\bar{p} \in Q_L$ and assume the converse holds, i.e. that for $\bar{q}_0 := \bar{p}$, all $\bar{q}_i := \delta_L(\bar{q}_{i-1}, \tau_i)$, i=1,...,f, are well defined vertices in G_L. In particular, denoting $\bar{q}_i := (q_1^i, q_2^i)$, we have $\delta(q_2^{i-1},\tau_i) = q_2^i \neq ud$ for i=1,..., f.

Our aim is to show that $q_1^{f-1} = f-1$, which is the desired contradiction (for $\Gamma(f-1,\tau_f) = \emptyset$, hence $\delta_L(\bar{q}_{f-1},\tau_f) = ud$). Since $q_1^{f-1} \leq f-1$, we can complete this part of the proof with the following (based on recursion (4) and loop (5)).

Claim. The inequalities $q_1^i \geq i$ hold for i = 0,1,...,f-1.

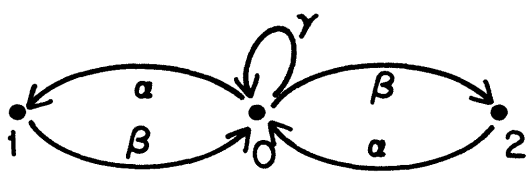
Proof. The claim is trivial for i=0. Assume it holds for i=0,...,j, for some j. We recall (cf. proof of necessity in Proposition 3.3) that since q_1^j can be reached by some word which ends with the string $\tau_1 \ldots \tau_j$ and since $q_1^j \geq j$, all the words which enter q_1^j end with that string. Checking the second construction loop in Algorithm 3.2, we see that then, j+1 must belong to $\Gamma(q_1^j,\tau_{j+1})$.

Now, for any letter σ, the apriori candidate for the R-component (i.e., the first component) of $\delta_L(\bar{q}_j,\sigma)$ is the maximal value, say k, of $\Gamma(q_1^j,\sigma)$ (in which case $\sigma=\tau_k$). This value is replaced by the second (third, etc.) - maximum, if no path labeled by $\tau_k\tau_{k+1}\ldots\tau_f$ leaves q_2^j in G. Since (by the contradiction assumption) for $\sigma=\tau_{j+1}$ this feasibility criterion is met by

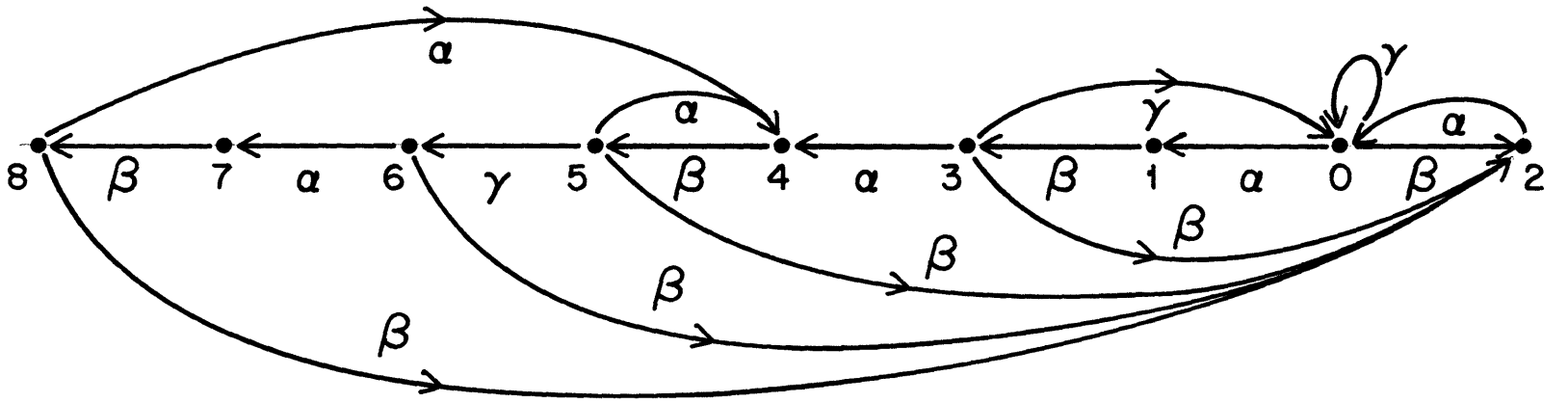
the candidate value $j+1 \varepsilon \Gamma(q_1^j, \tau_j)$, the chosen value of q_1^{j+1} will not be smaller than j+1, which completes the proof of the claim, hence of the legality assertion.

Maximality of $L(G_L)$ among all legal sublanguages. Let t be any word in $L(G) \backslash L(G_L)$. Then t can be written as $t=t_1 t_2$, where t_1 is a word of maximal length in $L(G_L)$ for such a factorization. (t_1 might be the empty string.) Let τ be the first letter in the string t_2. Then $t_1\tau \varepsilon L(G) \backslash L(G_L)$. Set $\bar{p} = \delta_L(\bar{0}, t_1)$. Since $\delta(p_2, \tau) \neq$ ud but $\delta_L(\bar{p}, \tau) =$ ud, there must hold $\Gamma(p_1, \tau) = \not\sigma$, i.e. $p_1 = f-1$ and $\tau = \tau_f$. By proposition 3.3 this means that $\bar{p}$ can be reached only via $\tau_1 \ldots \tau_{f-1}$, hence that $t_1\tau$ (hence t) is illegal. We have thus proved that $L(G_L)$ contains all legal words in L(G). [ ]

Example 3.7. We continue with the case of Example 3.4. Suppose the following is the graph G –



Then (with the graph R given in 3.4) Algorithm 3.6 yields this next graph for G_L



Here vertices are marked by their enumerations. The following table relate enumerations two pair-labels $\bar{p} = (p_1, p_2)$ (p_1 in R and p_2 in G):

enumeration	0	1	2	3	4	5	6	7	8
pair-label	(0,0)	(1,1)	(0,2)	(2,0)	(3,1)	(4,0)	(5,0)	(6,1)	(7,0)

4. CONSTRUCTION OF G_C.

The system remains as in §3, and our task here is to build a graph for $L_C(G)$ = the maximal controllable legal sublanguage of L(G).

The idea is to start with G_L, and to recursively erase vertices from which G-feasible illegal uncontrollable transitions leave. G_C will be the connected component of the origin in the resulting graph. This plan would have been easy if G_L were a subgraph of G, but it is mostly not (see e.g. Example 3.7, right above). So a useful preparatory step is this next construction of a new graph, H, for the language L(G) (i.e., such that L(H) = L(G)), with the property that G_L is a subgraph of H.

Algorithm 4.1 (Construction of H).

Input - the graphs G and R_n, n=1,...,N, as in Algorithm 3.5.

Output - a graph $H = (Q_H, \Sigma, \Sigma_c, \delta_H, 0)$.

```
begin(0)
   do(1) n=1, N
* instead of Γn(f(n)-1, τf(n)-1) = ∅ we set *
      Γn(f(n)-1, τf(n)-1) := {0};
* now Γn(p,σ) ≠ ∅ for all p and σ *
   end(1);
   call Algorithm 3.5 with G, R1,...,RN and denote its product H;
end(0).
```

Proposition 4.2. G_L is a subgraph of H and L(H) = L(G).

Proof. With its modified definitions, $\Gamma_n(p,\sigma) \neq \emptyset$ for all p and σ. Hence every G-feasible chain of transitions is feasible in H. The converse also holds: only G-feasible transitions are prompt in Algorithm 3.5. Hence

$L(H) = L(G)$. Since the modified graphs R_n contain those used in the construction of G_L as subgraphs, it follows that $G_L \subset H$. □

Algorithm 4.3 (construction of G_C).

Input - the graphs G_L and H, with $\Sigma = \{\sigma_1,\ldots,\sigma_L\}$ and

$$\Sigma_u = \{\rho_1,\ldots,\rho_K\} \qquad (K<L).$$

Output - a graph $G_C = (Q_C,\Sigma,\bar{\Sigma}_C,\delta_C,0)$.

Variables

- u - enumerates vertices in G_C, G_L and H. (Common vertices to two or three of these graphs carry the same enumeration. Vertices are labeled only by their enumerations and not by N+1-tuple labels, as before.)
- v_L - the maximal enumeration of a vertex in G_L (a given constant).
- v_C - counts the vertices in Q_C (updated within the algorithm); $u : \{0,1,\ldots,v_C\} \to Q_C$ is a 1:1 function.
- P - a preliminary version of Q_C.
- v,w - counting variables.

```
begin(0)
   P := Q_L, v := v_L, w := v_L;
* start eliminating vertices from which uncontrollable illegal chain of
transitions leave *
   begin(1)
      do(2) u=0, v_L
         if uεP then do(3) k=1, K
```

if $\delta_H(u,\rho_k) \neq$ ud and ($\delta_L(u,\rho_k)$ = ud or $\delta_L(u,\rho_k) \notin P$) then $P=P\setminus\{u\}$,

$w := v-1$, exit(3);

end(3);

end(2);

* if w<v then some vertex was eliminated and the recursion (1) must repeat *

if w<v then v := w, return (1);

* if no vertex was eliminated, the recursion ends *

end(1);

* at this point the set P is invariant under all H-feasible uncontrollable transitions; reconstruction of Q_C as the connected component of the origin in P starts *

$Q_C := \{0\}$, $u(0) := 0$, $v_c := 0$;

do(4) v=0, v_C

do(5) $\ell = 0$, L

if $\delta_L(u(v),\sigma_\ell) \notin P$ then $\delta_C(u(v),\sigma_\ell)$:= ud, end(5);

if $\delta_L(u(v), \sigma_\ell) \notin Q_C$ then $v_C := v_C+1$, $u(v_C) := \delta_L(u(v),\sigma_\ell)$,

$Q_C := Q_C \cup \{u(v_C)\}$;

$\delta_C(u(v),\sigma_\ell) := u(v_C)$;

end(5);

end(4);

end(0).

Proposition 4.4. $L(G_C) = L_C(G)$.

Proof. Controllability of G_C means invariance of the set of vertices, Q_C under H-feasible, uncontrollable transitions. Now, by construction, the set P is invariant under such transitions. The set Q_C is the maximal

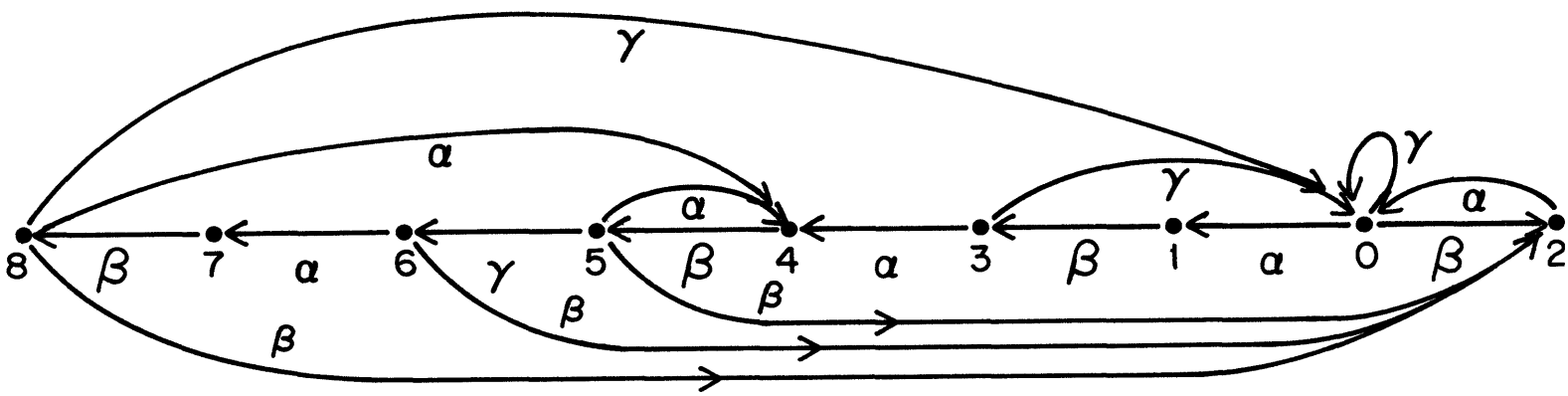
connected component of the origin in the graph formed by the restriction of δ_H to P, hence it has the invariance property as well, and G_C is controllable.

The transition function δ_C is the restriction of δ_L to $Q_C x \Sigma$, hence G_C is a subgraph of G_L and $L(G_C)$ is a legal sublanguage of L(G).

In order to see that $L(G_C)$ is the maximal controllable legal sublanguage, consider the set $L(G_L)\backslash L(G_C) = L_L(G)\backslash L(G_C)$: During the k-th elimination iteration (in recursion (1)) those vertices in Q_L from which uncontrollable strings of length k lead to $Q_H\backslash Q_L$, were erased. But the definition of G_L (as the graph of the maximal legal sublanguage) these strings are illegal. Hence the words which lead to the erased vertices cannot belong to any legal controllable sublanguage, in particular, not to $L_C(G)$. □

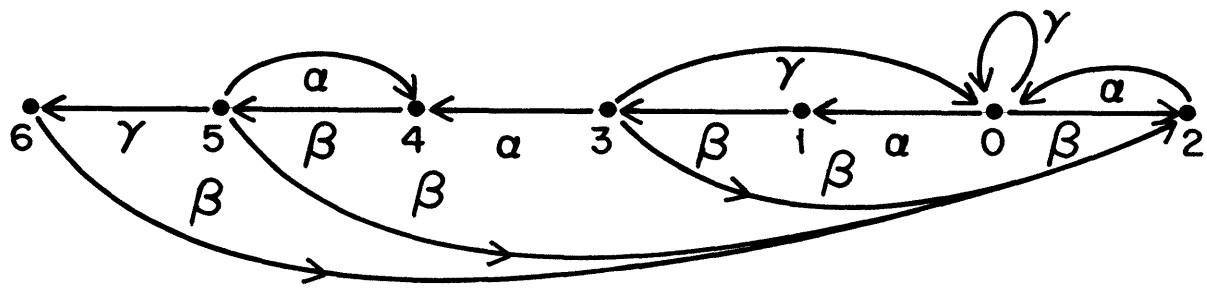
Comment 4.5. Algorithm 4.3 requires that vertices in G_L and in H be enumerated the same way. (To start with - this might not be the case.) Such coherent enumeration can be obtained simply by giving same numbers to vertices labeled by same N+1-tuples.

Example 4.6. We pursue the case study of Examples 3.4 and 3.7. In this case, the graph H (the product of Algorithm 4.1) will be the following



The difference between H and G_L (drawn in Ex. 3.7) is the additiona (illegal) arc γ from 8 to 0.

Suppose now that $\sum_c = \{\alpha\}$ and $\sum_u = \{\beta,\gamma\}$. Then the graph G_C (the product of Algorithm 4.3) will take this next form



Vertex 8 will be erased from $P = Q_L = \{0,1,\ldots,8\}$ in the first iteration, since γ is uncontrollable, $\delta_L(8,\gamma)$ = ud (because γ is illegal at 8), and $\delta_H(8,\gamma) \neq$ ud. At the second iteration, 7 will be erased, because $\beta\varepsilon\sum_u$, $\delta_H(7,\beta) \neq$ ud, and at that stage, $\delta_L(7,\beta) = 8 \not\in P$. Indeed, the string $\beta\gamma$ is H-feasible, uncontrollable, and illegal from 7. End of example.

To complete this section we present an algorithm for the construction of a supervisor $\underline{S}$, with $S=G_C$ and $L(\underline{S}/G) = L_C(G) = L(G_C)$. The procedure follows the recipe given in [10, Proposition 5.1], or following the equivalent statement, Lemma 2.2, above.

Algorithm 4.7 (Construction of a Supervisor)

Input - the graphs G_C and H, with $\Sigma = \{\sigma_1,\ldots,\sigma_L\}$.

Output - a supervisor $\underline{S} = (S=G_C, \phi)$.

Variables - u,v,v_C - as in Algorithm 4.3

```
* we need only the definition of the function φ *
begin(0)
   do(1) v=0, v_C
      u := u(v), φ(u) := ∅;
      do(2)  ℓ=1, L
         if δ_C(u,σ_ℓ) = ud and δ_H(u,σ_ℓ) ≠ ud then φ(u) = φ(u) ∪ {σ_ℓ};
      end(2);
   end(1);
end(0).
```

Proposition 4.8. $\underline{S}$ is a well defined supervisor and $L(\underline{S}/G) = L(G_C)$.

Proof. It is easy to see that (iii) and (iv) of Observation 2.1, above, are satisfied by $\underline{S}$, hence it is a well defined supervisor. Since $G_C \subset H$ we have $L(G_C) \subset L(H) = L(G)$, and by Observation 2.1 (i), the equality $L(\underline{S}/G) = L(S=G_C) \cap L(G) = L(G_C)$, follows. □

Remark 4.8. The supervisor we have just described is mostly very wasteful in its graph-size, and more efficient ones for which $L(\underline{S}/G) = L(G_C)$, can be built. Techniques for deriving more efficient supervisors from an existing one (e.g., from our $\underline{S}$) are discussed in [6].

5. THE GRAPHS G_L AND G_C IN A MULTI-SYSTEM

We consider a framework of M subunits (which we call henceforth, "machines"), modeled by $G_1,\ldots,G_M$. ($G_m = (Q_m, \Sigma_m, \Sigma_{mc}, \delta_m, 0)$.) The machines are organized in N groups: Each machine belongs to several (one or more) of these groups. There are K(n) machines in group #n. Within each group, machines are recognized not by their global enumeration, but rather, according to an independent intra-group enumeration. (We discuss the reason for that later on.) Relations between intra-group and global enumerations are stored as a function $k:\{1,\ldots,M\}X\{1,\ldots,N\} \rightarrow \{0,1,2,\ldots.\}$, as follows

$$k(m,n) = \begin{cases} \text{the place of machine \#m in group \#n} & \text{if machine \#m belongs to group \#n} \\ 0 & \text{otherwise} \end{cases}$$

(In particular, in the sequel it will be convenient to use "k(m,n) = 0" as a shorthand for "machine #m does not belong to group #n".)

We denote by G the combined system, namely, the direct product

$$G = \bigotimes_{m=1}^{M} G_m .$$

By F_n we denote the combined subsystem formed by the machines in the n-th group, namely

$$F_n = \bigotimes \{G_m : k(m,n) \neq 0\} .$$

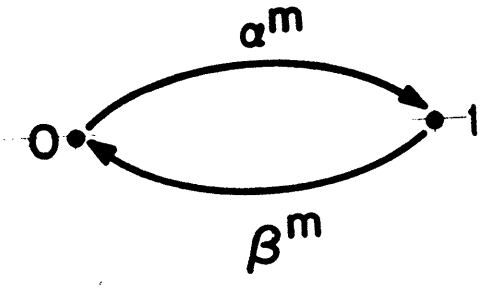
The alphabet in that group is

$$T_n = U\{\sum_m : k(m,n) \neq 0\}.$$

Constraints are imposed <u>independently</u>[3] on each group. Subject to the imposed constraints on F_n, the groups F_{nL} and F_{nC} can be built as described in the previous two sections.

The collection of all group-constraints defines a set of global constraints on G: A chain of transitions will be legal in G if it violates none of the group-constraints. With this definition we look for G_L and G_C.

<u>Example 5.1</u>. Recall the two-state machines of Example 3.1



and assume G_1, G_2 and G_3 are each of that form. Further, assume we wish to maintain the FIFO condition discussed in Example 3.1, between any two of these machines; namely, we forbid the completion of the strings

[3]In fact, the reader may regard a machine-group not merely as an ordered sub-collection of machines, but rather, as a set of constraints imposed on these machines. Thus, different sets of constraints imposed on the same set of machines can be taken for distinct groups.

$$\begin{cases} \alpha^j\alpha^i\beta^i \\ \alpha^j\beta^i\alpha^i\beta^i \end{cases} \qquad i<j \quad .$$

This case suits our framework with M=3 and $N=\binom{3}{2}=3$, since the constraints are imposed independently on each pair of machines. In this example intra-group enumeration is induced by the global enumeration as follows: if group #n ($n\varepsilon\{1,2,3\}$) consists of the pair (i,j) with i<j, then k is given by

$$k(m,n) = \begin{cases} 1 & m=i \\ 2 & m=j \\ 0 & m\neq i,j \end{cases} \quad .$$

Algorithm 5.2 (Construction of G_L and G_C in the Combined Systems).

Input - Since we treat F_{nL} and F_{nC} similarly we represent both by input graphs $A_n = (P_n,T_n,\xi_n,0)$ (we omit the unneeded notation of the alphabet T_{nc}). Also given, is the function k(m,n). Recall that $T_n = U\{\sum_m : k(m,n)\neq 0\}$. We denote $\sum_m = \{\sigma_1^m,\ldots,\sigma_{L(m)}^m\}$.

Output - a graph $A = (P,\sum,\xi,0)$. ("A" will stand for either G_L or G_C) with $L(A)\subset L(G)$.

Variables - $\bar{p}$ - N-tuple; $p_n \varepsilon P_n$, n=1,...,N; $\bar{p}$ labels a vertex in A.

u,v - enumerates vertices in A; $u(\bar{p})$ and $\bar{p}(u)$ are 1:1 functions.

w - the incumbent maximal value of the function u.

k - intra-group machine enumeration.

ℓ - letter enumeration.

m - machine enumeration.

n - group enumeration.

* the following does an inductive construction, subject to the constraints put by the structures of $A_1,\ldots,A_N$ *

being(0)

* set the origin *

$\bar{P} := \bar{0} = (0,..,0)$, $P := \{\bar{p}\}$, $u(\bar{p}) := 0$, $w := 0$;

* start the construction of A *

do(1) v=0, w

$\bar{p} := \bar{p}(v)$;

do(2) m=1, M

do(3) ℓ=1, L(m)

do(4) n=1, N

* test feasibility of σ_ℓ^m from p_n in A_n *

$k := k(m,n)$;

if k=0 then $q_n := p_n$, end(4);

* $\bar{q}$ is a candidate for $\xi(\bar{p},\sigma_\ell^m)$; no move is done in irrelevant groups *

if $\xi_n(p_n,\sigma_\ell^k) = ud$ then $\xi(\bar{p},\sigma_\ell^m) := ud$, exit(4), end(3);

* recall that σ_ℓ^m is recognized as $\sigma_\ell^{k(m,n)}$ in group #n, if it is not feasible in An, so will it be in A, otherwise *

$q_n := \xi_n(p_n,\sigma_\ell^k)$;

end(4);

* $\sigma_\ell^m:\bar{p} \to \bar{q}$ is feasible; if $\bar{q} \notin P$, it is added, enumerated and w is updated *

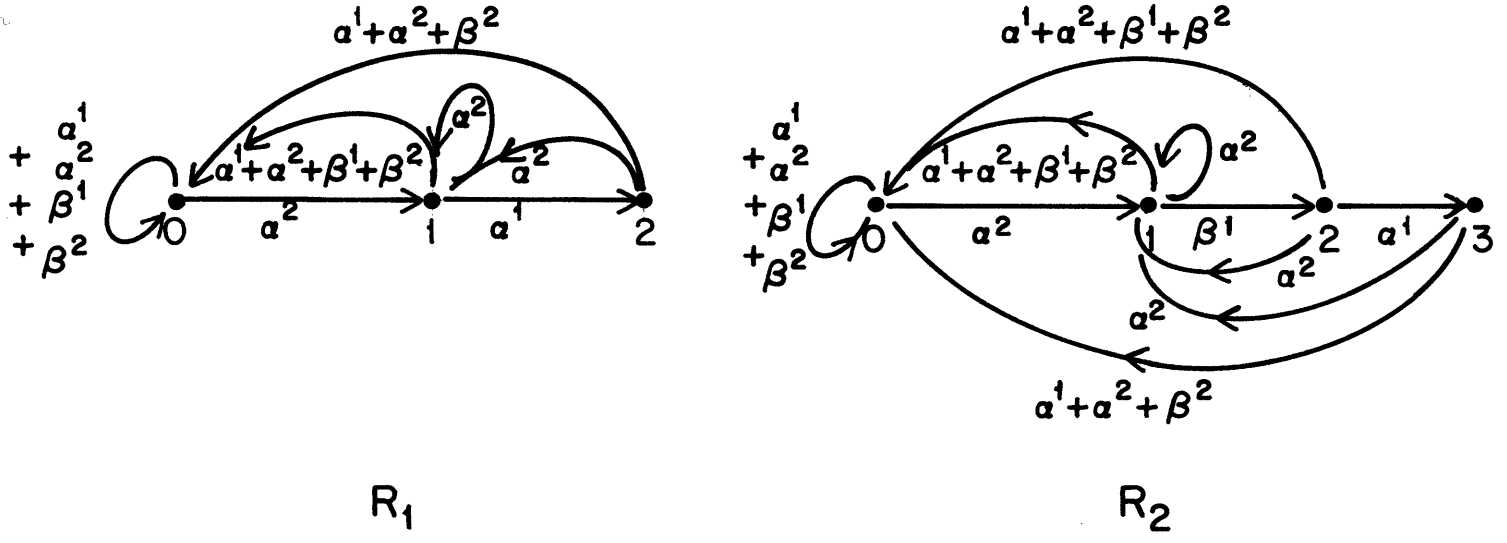
```
            if q̄∉P then P := P∪{q̄}, w := w+1, u(q̄) := w;
            ξ(p̄) := q̄;
          end(3);
        end(2);
    end(1);
end(0).
```

Proposition 5.3. If $A_n = F_{nL}$ (respectively, $= F_{nC}$) then $A = G_L$ (respectively, $= G_C$).

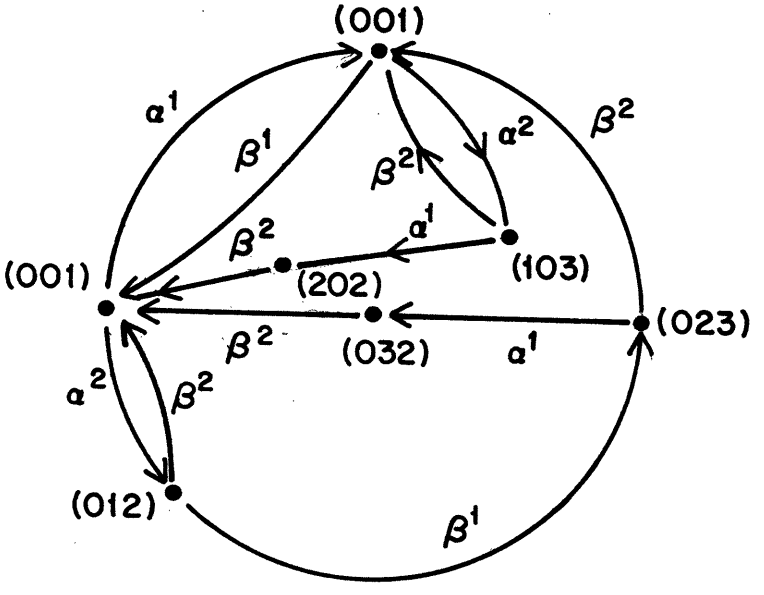
Proof. By construction, a chain of transitions forms a word in L(A) if and only if the parts of this chain, which are relevant to each of the groups (i.e., in group #n, the subchain that one obtains by erasing letters which are not from the alphabet T_n) form words in the corresponding languages $L(A_n)$. Since $L(A_n) \subset L(F_n)$ and $L(F_n) \subset L(G)$, it follows that L(A) is a sublanguage of L(G), and the maximal sublanguage, subject to the constraints placed by the structures of all A_ns. □

Example 5.4. Let us continue with the case of Examples 3.1 and 5.1, above.

Applying Algorithm 3.2 to the two forbidden strings $\alpha^2\alpha^1\beta^1$ and $\alpha^2\beta^1\alpha^1\beta^1$, one gets these next graphs R_1 and R_2 - (in a two-machine system) -



Algorithm 3.5, in turn, yields the following graph as G_L, in the two machine system –



In the three machine system, we use the following group enumeration and intra-group machine enerations: Group #1 is formed by the pair (G_1,G_2), Group #2 by (G_1,G_3), and #3, by (G_2, G_3). As in Example 5.1, the function k, and the intra-group enumeration, are set consistent with the global machine enumeration. Precisely -

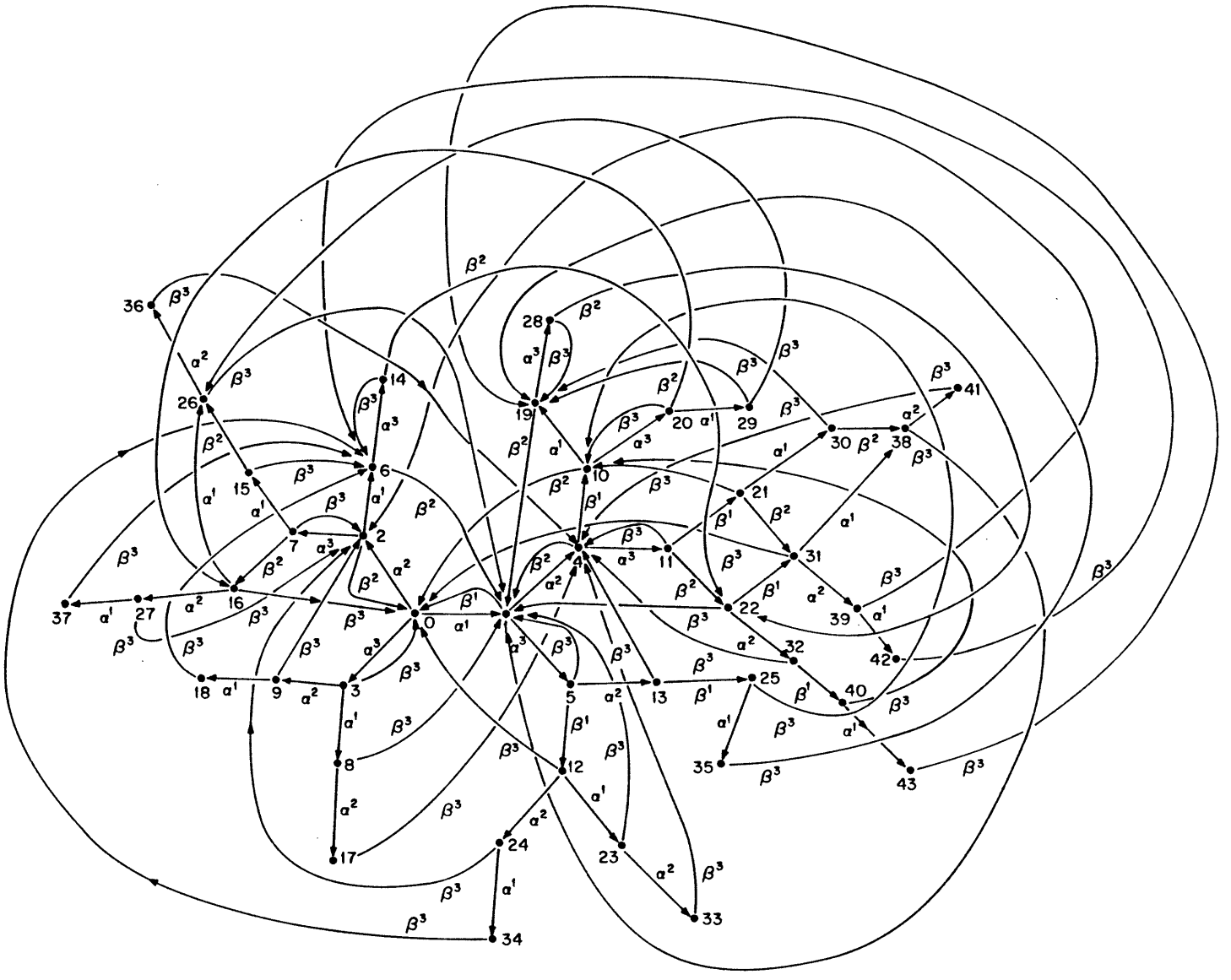
$k(1,1) = k(1,2) = k(2,3) = 1$

$k(2,1) = k(3,2) = k(3,3) = 2$

$k(1,3) = k(2,2) = k(3,1) = 0.$

With these definitions in the three-machines systems, the graph G_L above can serve as both F_{1L}, F_{2L} and F_{3L}. Here we see the advantage of the independent intra-group enumerations: When distinct groups have isomorphic intra-structrue and have to obey the same rules (subject to the isomorphism transformation) then it suffices to compute only one of the F_{nL} (likewise F_{nC}, and in the following section, $\underline{S}_n$), and extrapolate that information by use of the function k (which actually defines the isomorphism).

Invoking Algorithm 5.2 we get this next intricate graph as G_L, in the three-machine system.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

6. A SUPERVISOR FOR THE COMBINED SYSTEM.

We continue the discussion of the combined system. It is assumed now that each group is associated with a (model of a) supervisor that imposes a prescribed set of constraints. (The supervisor in group #n is $\underline{S}_n = (S_n, \phi_n)$, $S_n = (X_n, T_n, T_{nc}, \xi_n, 0)$.) Our goal is to construct a supervisor $\underline{S} = (S, \phi)$ for the combined system, which will enforce all group-level constraints on G.

The current problem is similar to that of the previous section, inasmuch as we have to accumulate the structural constraints of all the graphs S_n, in S (as we did e.g. with those of F_{nL}, when constructing G_L). However now, $L(S_n)$ will mostly not be a sublanguage of $L(F_n)$, hence of L(G) (unless $S_n = F_{nC}$ as explained in §2; cf. [6]). So in order that hypothesis H_2 of §2 be maintained, it is necessary to track the process not only in all group-supervisors, but also in all the participating machines. (H_2 requires that each arc and vertex in S be used and reached by a word in $L(\underline{S}/G)$, we shall illustrate this point in Example 6.3.) This explains the first part in the following algorithm.

After doing the job of preventing never-to-be-used elements from being added to S, tracking processes in individual machines (and not only in group supervisors) might cause considerable structural redundance in the created supervisor. The algorithm's second part is a lumping procedure which simplifies and economize the final version of $\underline{S}$.

Algorithm 6.1 (construction of $\underline{S}$ in the combined system).

Input – the graphs G_m, m=1,...,M, the supervisors $\underline{S}_n$, n=1,...,N and the function k(m,n).

Output - a supervisor $\underline{S} = (S,\phi)$, $S = (X,\Sigma,\Sigma_C,\xi,0)$, on $G=\textcircled{x}G_m$.

Variables - $\bar{x},\bar{y},\bar{z}$ - N+M-tuples; $x_n,y_n,z_n \in X_n$, n=1,...,N; x_{N+m}, Y_{N+m}, $z_{N+m} \in Q_m$, m=1,...,M; $\bar{x},\bar{y}$ and $\bar{z}$ will label vertices in S,

u,v - enumerate vertices in S; $u=u(\bar{x})$ and $\bar{x}=\bar{x}(u)$ being 1:1 functions;

w - incumbent maximal enumeration of a vertex in S.

ℓ,m,n - as in Algorithm 5.2.

```
begin(0);
* set the origin in S *
   x̄ := 0, X := {x̄}, u(x̄) := 0, w := 0;
   do(1) v=0, w
      x̄ := x̄(v), φ(x̄) := ∅;
      do(2) m=1, M
         do(3) ℓ=1, L(m)
* test feasibility of σ_ℓ^m from x_{N+m} in G_m*
            if δ_m(x_{N+m}, σ_ℓ^m) = ud then ξ(x̄,σ_ℓ^m) := ud, end(3);
            y_{N+m} := δ_m(x_{N+m},σ_ℓ^m);
* ȳ is the candidate value of ξ(x̄,σ_ℓ^m) *
            do(4) n=1, N
* test of feasibility of σ_ℓ^m in group #n *
               k := k(m,n);
               if k=0 than y_n := x_n, end(4);
               if σ_ℓ^k ε φ_n(x_n) then φ(x̄) := φ(x̄) ∪ {σ_ℓ^m }, ξ(x̄,σ_ℓ^m) := ud,
               exit(4), end(3);
```

$y_n := \xi_n(x_n, \sigma_\ell^k)$;

end(4);

* arrived at this point if σ_ℓ^m is feasible in G_m and legal according to all group-supervisors; y_{N+i} is yet undefined for $i \neq m$, but in these entries there is no change when σ_ℓ^m occurs *

do(5) $1 \leq i \leq M$, $i \neq m$

$y_{N+i} := x_{N+i}$;

end(5);

if $\bar{y} \notin X$ then $X := X \cup \{\bar{y}\}$, $w := w+1$, $u(\bar{y}) := w$;

$\xi(\bar{x}, \sigma_\ell^m) := \bar{y}$;

end(3);

end(2);

end(1);

* here ends the first part of the construction; now starts the lumping of vertices with same N-tuple prefix in their labels *

do(6) u=0, w-1

$\bar{x} := x(u)$;

do(7) v = u+1, w

$\bar{y} := \bar{y}(v)$;

do(8) n=1, N

if $y_n \neq x_n$ then exit(8), end(7);

end(8);

* arriving here $\bar{x}$ and $\bar{y}$ have the same N-tuple prefix, and will be lumped into a new $\bar{x}$ *

$\phi(\bar{x}) := \phi(\bar{x}) \cup \phi(\bar{y})$;

```
         do(9) m=1, M
            do(10) ℓ=1, L(m)
* divert all arcs which enter ȳ to enter x̄, instead *
               do(11) i=1, w
                  if ξ(z̄(i),σ_ℓ^m) = ȳ then ξ(z̄(i), σ_ℓ^m) := x̄;
               end(11);
* also, arcs which originally leave ȳ, will leave x̄ *
               if ξ(x̄,σ_ℓ^m) = ud then ξ(x̄,σ_ℓ^m) := ξ(ȳ,σ_ℓ^m);
            end(10);
         end(9);
         X := X\{ȳ};
         if v=w then w := w-1, exit(7), end(6);
         do(12) i=v+1, w
            z̄(i-1) := z̄(i);
         end(12);
         w := w-1;
      end(7);
   end(6);
end(0).
```

Proposition 6.2. $\underline{S}$ is a well defined supervisor and $L(\underline{S}/G)$ is the maximal sublanguage of L(G), subject to the constraints placed by $\underline{S}_n$, n=1,...,N.

Proof. It is a straightforward generalization of Proposition 5.3, which shows that the proposition holds for the product of the first part of the algorithm. It remains to show that the second part, the lumping

procedure, does not ruin the desired properties.

To that effect we shall be using some of the theory developed in [6], concerning transformations on supervisors. Lumping is the operation of substituting several distinct vertices in S by a single new vertex, letting all arcs which entered (respectively, left) any of the lumped vertices, enter (respectively, leave) the new vertex. In our case, all vertices with same N-tuple prefix in their labels are lumped into a new vertex, the meaningful part in whose label is its N-tuple prefix. A condition for this operation to bear no effect on $L(\underline{S}/G)$ is control compatability of the lumped vertices [6, Proposition 4.1]: we say that two vertices x and y in a supervisor's graph are control compatible, if the following three conditions hold

(i) $\phi(x) \cap \varphi(y) = \phi(y) \cap \varphi(x) = \emptyset$,

(ii) $\forall s \in \Sigma^*$, if $x_1 := \xi(x,s) \neq$ ud and $y_1 := \xi(y,s) \neq$ ud, then x_1 and y_1 satisfy (i).

(iii) if $x_1 = x$ in (ii) then (i) holds when y_1 substitutes for x.

Let us check now that, indeed, if $\bar{x}$ and $\bar{y}$ share the same N-tuple prefix then these two vertices are control compatible: By construction, $\phi(\bar{x})$ and $\phi(\bar{y})$ are both subsets of

$$\bigcup_{n=1}^{N} \phi_n(x_n) = \bigcup_{n=1}^{N} \phi_n(Y_n)$$

(since $x_n = y_n$, $n=1,\ldots,N$). Also, an arc may leave $\bar{x}$ or $\bar{y}$, i.e. - belong to $\varphi(\bar{x})$ or $\varphi(\bar{y})$, only if it is feasible in G, and allowed by all $\underline{S}_n$, i.e. - if it belongs to $(\bigcup_{n=1}^{N} \phi_n(x_n))^c$. Hence

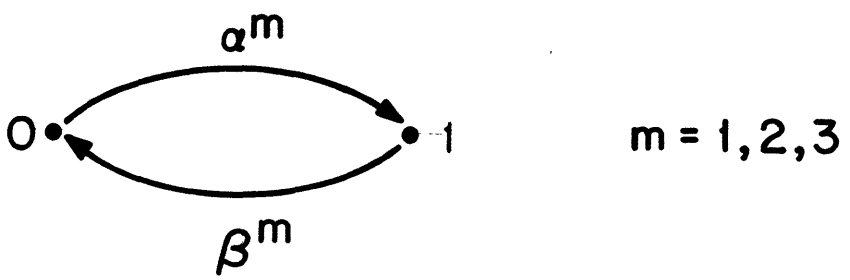
$$\phi(\bar{x}) \cap \psi(\bar{y}),\ \phi(\bar{y}) \cap \psi(\bar{x}) \subset (\cup \phi_n(x_n)) \cap (\cup \phi_n(x_n))^c = \emptyset.$$

This is condition (i).

If $\bar{x}_1 := \xi(\bar{x},s) \neq ud$ and $\bar{y}_1 := \xi(\bar{y},s) \neq ud$, then $(\bar{x}_1)_n := \xi_n(x_n,s)$ and $(\bar{y}_1)_n := \xi_n(y_n,s)$. So if $\bar{x}$ and $\bar{y}$ share the same N-tuple prefix, so do $\bar{x}_1$ and $\bar{y}_1$, and condition (ii) holds. The same is true for $\bar{y}_1$ and $\bar{y}$ in the case described in condition (iii). □

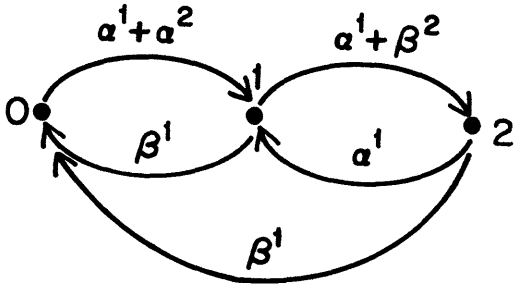
Example 6.8. Here we illustrate a difference between the task of constructing $\underline{S}$ from the $\underline{S}_n$s, and that of building G_L from the F_{nL}s, or G_C from the F_{nC}s. As in Algorithm 5.3, the effective part of a vertex label in the final product of Algorithm 6.1, is an N-tuple of labels (here - of vertices in $S_1,\ldots,S_N$). Yet, not as in the previous case, here we do need the intermediate part of tracking transitions in the various G_ms. Otherwise, the resulting S might include vertices and arcs which will never be used in $\underline{S}/G$ (in contradiction to hypothesis H2).

Consider once more three identical two-states machines, each of the type described in Examples 3.1, 5.1, and 5.3, above; namely, with the graph-models



Let now the constraints on $G = \bigotimes_{m=1}^{3} G_m$ be given in terms of this next list of

forbidden strings: for i<j, these are $\alpha^i\alpha^j$, $\beta^i\beta^j$, $\beta^j\alpha^j$ and $\alpha^j\alpha^i\beta^j$. Suppose further that all the transitions involved are controllable ($\Sigma_c=\Sigma$). Organizing the machines in three pairs and defining the function k(m,n) as in Example 5.4, above, we may choose $\underline{S}_1 = \underline{S}_2 = \underline{S}_3$, all built according to the following plan: The graph of S_n will be,

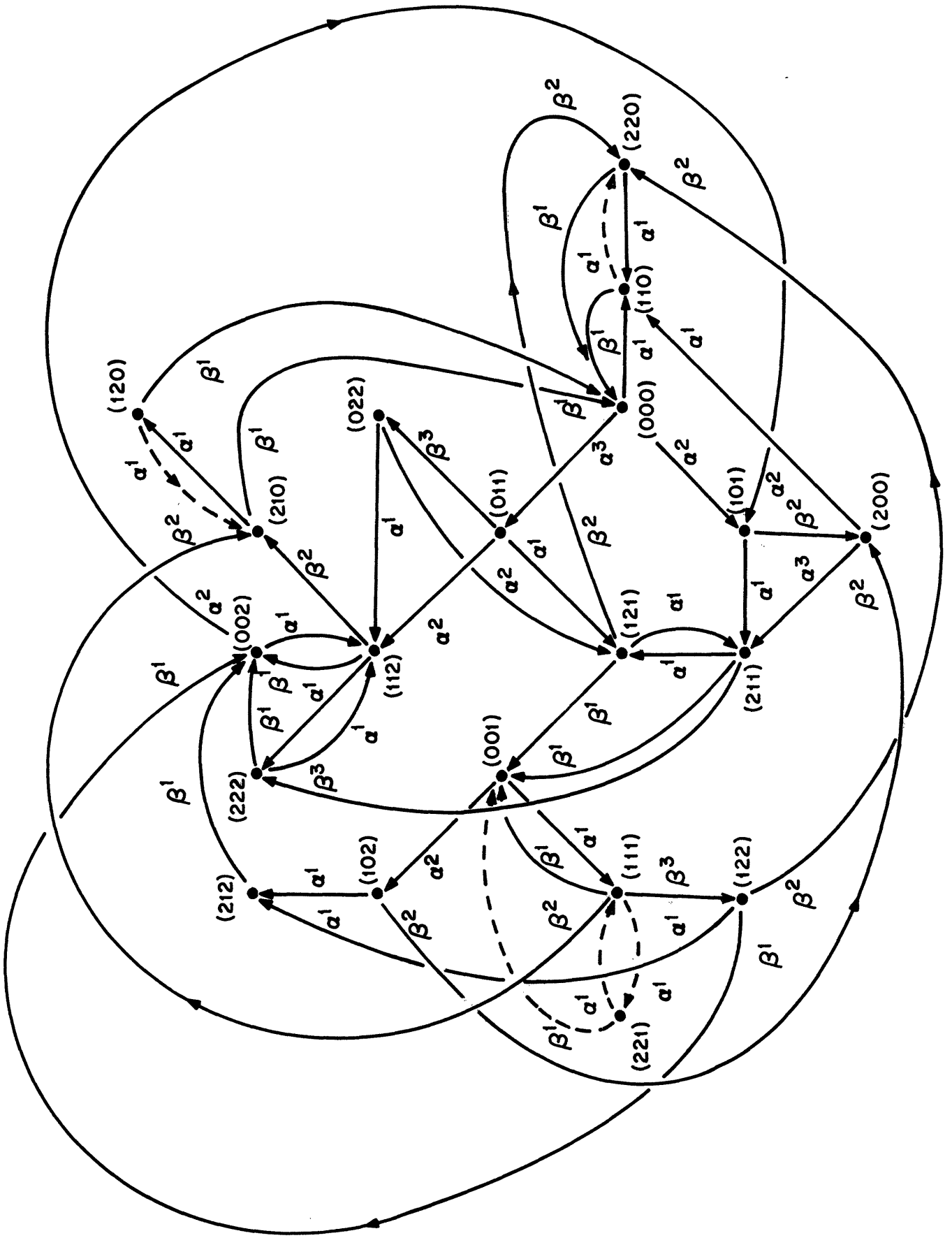


associated with the control function $\phi_n(0) = \{\beta^2\}$, $\phi_n(1) = \{\alpha^2\}$, $\phi_n(2) = \{\alpha^2,\beta^2\}$.

(The interested reader can build $\underline{S}_n$ in steps: First building F_{nL} for a two-machine group, using Algorithm 3.5. Since $\Sigma_c = \Sigma$, it follows that $F_{nC} = F_{nL}$; Algorithm 4.7 provides a plan for a machine-pair-supervisor. The latter can be improved via some lumpings of control-compatible vertices, as explained in the proof of Proposition 6.2.)

The following graph is the product of Algorithm 5.2, taking $A_n = S_n$ as input. Several of its arcs (those drawn in dashed lines) will never be used during controlled processes (i.e., by words in $L(\underline{S}/G)$, if the above is to be a graph of $\underline{S}$). The vertex (221) will never be visited by a controlled

process. The reason for these phenomena is that the string $\alpha^1\alpha^1$, which is impossible as a chain of transition in G, can be realized in $L(S_n)$, n=1,2. By erasing the dashed arcs and the redundant vertex (221), one obtains the same model for S, which Algorithm 6.1 yields.


(220)
(110)
(000)
(120)
(210)
(022)
(011)
(101)
(200)
(002)
(112)
(121)
(211)
(001)
(222)
(102)
(111)
(122)
(212)
(221)

7. CONCLUSIONS AND FUTURE WORK.

This work is an extention of recent developments in the theory of controlled discrete event systems. We show how to use this theory for real-life systems, which are large and complex, both in their model dimension (e.g., their numbers of states and transitions), and in the amount of logical and physical constraints (and their implication on the forbidden strings), which these systems are subject to.

The methods provided here enable simple and orderly handling of large systems with flexible, rapidly changing functional structures. In particular, they contribute to improvements in the following two aspects: (1) They allow for distributed control systems (which, with the advent of cheap computers and communication standards - e.g. Baseway [1] and MAP [2] - is the current trend in factory automation); and (2) They result with easier human interface and understanding of the system. For instance, it suffices that users know only about parts of the overall system, or of its operational constraints. This is a very important feature that agrees with industrial organizations, where e.g., some people are in charge of maintenance, others deal with inventories, with process sequences, etc.. Now each department supervisor can include his constraints and requirements, which, by the proposed method, be integrated, to come up with the overall supervisor.

Some individual constraints might be in conflict, and their coupled implementation may reduce system performance. This calls for further development of tools for system analysis, which is one of the topics of our future research. Another key step towards successful application, will be

to extend the theory developed so far, in deriving algorithms which translate simply expressed expert rules into forbidden strings, from which the control system can be derived.

REFERENCES

[1] Digital Equipment Co., Baseway and Shop Floor Gateway, SPD 14.85.02, January 1986.

[2] General Motors Co., Manufacturing Automation Protocol, MAP Specifications, V.2.2., GM Technical Center, 1986.

[3] J.H. Hopcroft and J.O. Ullman, Introduction to Automata Theory, Languages and Computation, Addison Wesley, 1979.

[4] B.H. Krogh and C.L. Beck, "Synthesis of place/transition nets for simulation and control of manufacutring systems", Proceedings of the 4th IFAC/IFORS symposium, Zurich, 1986.

[5] B.R. Krogh and G. Ekberg, "Automated programming of controllers for discrete manufacturing processes", (to appear).

[6] O. Maimon and G. Tadmor, "Efficient low-level control of flexible manufacturing systems", Technical report LIDS-P-1571, MIT, 1986.

[7] O. Maimon and G. Tadmor, "Efficient supervisors in discrete event systems", Proceedings of the 1986 International Conference on Systems, Man and Cybernetics, Atlanta, GA.

[8] O. Maimon and G. Tadmor, "First step in constraint understanding in low-level discrete event system control", (in preparation).

[9] C.M. Mitchell, T. Govindaraj, O. Dunkler, S.P. Krosner, and J.C. Ammons, "Real-time scheduling in FMS: A supervisory control model of cell oerator function", Proceedings of the 1986 International conference in Systems, Man and Cybernetics, Atlanta, GA.

[10] P.J. Ramadge and W.M. Wonham, "Supervisory control of a class of discrete event processes", Systems Control Group Report #8515

(1985), University of Toronto, 1985.

[11] P.J. Ramadge and W.M. Wonham, "Modular feedback logic", Systems Control Group Report #48 (1985), University of Toronto, 1985.

[12] A.F. Vaz and W.M. Wonham, "On supervisor reduction in discrete event systems", Int. J. Control 44 (1986), pp. 475-491.

Appendix: Convention for algorithm presentation

begin(i)	-	begins the i-th process or recursion.
do(i)	-	starts the i-th loop. (iterative process)
return(i)	-	returns to begin(i).
end(i)	-	continues with the next iteration of the do-loop(i) (until the loop's parameter reaches its upper value), or ends the process which started with begin(i).
";"	-	ends a command.
","	-	separates between parts of same command.
"."	-	ends the algorithm.
exit(i);	-	continues after end(i), (i.e. leaves iteration(i), or process (i)).
exit(i),...,...;	-	leaves do(i) or begin(i), continues with the statement after "," (until ";"). The statement following exit(i) does not use local parameters of do(i) or begin(i).