Petri Net Control for Grammar Systems

Maurice ter Beek and Jetty Kleijn*

LIACS, Leiden University, P.O.Box 9512, 2300 RA Leiden, The Netherlands
kleijn@liacs.nl

Abstract. It is demonstrated how Petri nets may be used to control the derivations in systems of cooperating grammars. This allows to define grammar systems with a concurrent rewriting protocol. Some basic properties are established.

1 Introduction

A grammar system consists of grammars and a protocol prescribing their cooperation. The first grammar systems appearing as such in the literature are the cooperating grammar systems defined in 1978 in [MR78] and [MRV78] by G. Rozenberg and his two Ph.D. students R. Meersman and D. Vermeir. These systems derived from a study of two-level substitution mechanisms by focussing on the collaboration and communication between multiple rewriting systems. Some ten years later, the link between grammar systems and the blackboard model for problem solving ([Nii89]) was established and a grammatical theory of cooperation protocols was developed (see [CDKP94] and [DPR97]). Two basic classes of grammar systems are distinguished: cooperating distributed (CD) grammar systems (introduced in [CK89] and [CD90]) and parallel communicating (PC) grammar systems ([PS89]). Like the cooperating grammar systems of [MR78], CD grammar systems are sequential in nature. The grammars work on a common sentential form and at each moment during the rewriting process only one of them is active. On the other hand in a PC grammar system the grammars work simultaneously, each on its own sentential form. The operations are synchronized (by means of a global clock): in each time unit the grammars either all rewrite their current string or they communicate through queries which leads to the sending of sentential forms from one component to another. Since their introduction many variations of CD and PC grammar systems have been formulated not just by varying the type of the participating grammars, but rather by defining and investigating various protocols modelling aspects of distributed computations (derivations) in grammar systems. For the sequential CD grammar systems, e.g., one distinguishes so-called modes of derivations to control the de-activation of the active grammar in a derivation of the system, whereas for the simultaneously operating grammars in a PC grammar system, protocols for the transfer of sentential forms have been a main focus of research.

* corresponding author

W. Brauer et al. (Eds.): Formal and Natural Computing, LNCS 2300, pp. 220–243, 2002.

This paper demonstrates how Petri nets may be used to control the derivations in grammar systems, thus creating an explicit possibility for the study of concurrent rewriting protocols. We propose to use a control mechanism based on Petri nets that has been developed within the framework of Vector Controlled Concurrent Systems. These systems were introduced in a two part paper ([KKR90] and [KKR91]) by G. Rozenberg together with H.C.M. Kleijn and N.W. Keesmaat as the beginning of the latter's Ph.D. research ([Kee96]). A Vector Controlled Concurrent System (VCCS) consists of concurrently operating sequential components together with a mechanism which synchronizes and controls their behaviour. The specific control mechanism we have in mind is the Generalized Individual Token Net Controller (GITNC), investigated in detail in [KK97] and apparently well suited for implementation in grammar systems. A GITNC is a (finite) labelled Petri net with individual tokens — one for each component — which monitor the progress of the components. It controls the operation of the system as a whole by allowing or disallowing certain synchronizations depending on the current marking (the distribution of the individual tokens over the places). Therefore the transitions of the GITNC are labelled by vectors which describe a synchronous execution of components' actions. Such a vector label has one entry for each component: a non-empty entry represents an action to be executed by the corresponding component, while an empty entry indicates that that component is not involved (remains idle) in the synchronization. Moreover, this vector labelling is consistent in the sense that a transition uses the token associated to a certain component if and only if that component is involved in the synchronization described by its vector label. The occurrence of a transition of a GITNC implies a combined action of those components whose tokens are used by the transition. These components then execute synchronously the actions in the vector which labels the transition. A GITNC is a finite state device and may be seen as a finite automaton with vectors as its action labels. However in contrast with a finite automaton, a GITNC has a distributed state space and allows concurrent execution of transitions which do not use a common token. This implies that synchronizations which involve different components can occur independently of each other (and hence also in any order). Using simply regular control languages (over an alphabet of vectors) rather than GITNC languages would destroy the potential concurrency in the computations of the controlled system (see [KK97]).

Thus we introduce Petri net (PN) grammar systems as consisting of a number of grammars and a GITNC which describes a concurrent protocol for rewriting in the participating grammars. (Actually, some initial ideas concerning this subject were presented at the MFCS Workshop on Grammar Systems in Brno in 1998 and the Seminar Recent Trends in Language Theoretic Models of Multi-Agent Systems in Budapest in 1999.) Within a PN grammar system, the grammars are viewed as separate entities. Hence, each has its own sentential form to work on just like in a PC grammar system. The grammars collaborate by synchronizing their actions (rewriting steps) subject to the control exercised by the GITNC. A system derivation in a PN grammar system starts from the axioms of the

grammars with the controller in an initial marking. At each moment during the derivation rewritings are applied synchronously according to the labels of the transitions that occur. When all components have derived a terminal string and the GITNC is in one of its final markings, the derivation has been successful.
As follows from the above, the grammars in a PN grammar system cooperate subject to a regular control language. However because of its implementation as a GITNC, this protocol is explicitly concurrent and PN grammar systems are essentially concurrent grammar systems. Moreover like CD and PC grammar systems, also PN grammar systems are related to the blackboard approach of problem solving. The grammars correspond to knowledge sources, which each possesses some knowledge to solve the problem, and their sentential forms represent the blackboard with the current state of the problem's solution. Rewriting corresponds to changing the state of the blackboard. The problem solving strategy is reflected in the cooperation protocol and the solutions are represented by the thus generated terminal strings. In a PN grammar system, the grammars work on separate regions of a common blackboard and a separately defined control mechanism allows to concurrently pursue multiple lines of reasoning. So the model proposed here relates to the blackboard paradigm as discussed in [Cor91] and [CL92].

After having defined Generalized Individual Token Net Controllers, we introduce PN grammar systems consisting of context-free grammars and a GITNC enforcing synchronized applications of productions. Some initial observations are made relating to concurrency in the derivations of a PN grammar system and the vector languages defined by PN grammar systems are discussed. In particular the vector languages of PN grammar systems with regular component grammars are characterized. In a concluding section we point out some topics which may warrant further investigations.

2 Preliminaries

We assume some familiarity with basic notions from the theory of formal languages ([RS97]), regulated rewriting ([DPS97]), and Petri nets ([RR98]). The paper is however to a large extent self-contained and knowledge of grammar systems or GITNCs is not a prerequisite.
We use $\subset$ to denote strict set inclusion. For a set V, its powerset is denoted by $\mathcal{P}(V)$. All functions considered are total. As ususal, if $f : V \to W$ is a function and U is a subset of V, then $f(U) = \{f(u) \mid u \in U\}$. Let $n \in \mathbb{N}$, the set of non-negative integers. Then $[n] = \{1, \ldots, n\}$ with $[0] = \emptyset$. Let $V_1, \ldots, V_n$ be sets. We refer to the elements of the cartesian product $V_1 \times \cdots \times V_n$ as (n-dimensional) vectors. Given an n-dimensional vector $v = (v_1, \ldots, v_n)$ and $i \in [n]$, the i-th entry of v is obtained by applying the projection function $proj_i(v) = v_i$.

An alphabet is a finite, possibly empty, set of symbols. Let Σ be an alphabet. A word (over Σ) is a finite sequence (concatenation) of symbols (from Σ). The empty word is denoted by λ. A language (over Σ) is a set of words (over Σ). Σ^* is the set of all words over Σ. Recall that Σ^* is a free monoid generated by Σ with λ as its unit element.

Let $n \in \mathbb{N}$ and let $\Sigma_1, \ldots, \Sigma_n$ be alphabets. An (n-dimensional) vector letter (over $\Sigma_1, \ldots, \Sigma_n$) is a tuple $(\sigma_1, \ldots, \sigma_n)$ with $\sigma_i \in \Sigma_i \cup \{\lambda\}$ for all $i \in [n]$ and $(\sigma_1, \ldots, \sigma_n) \neq (\lambda, \ldots, \lambda)$. By $tot(\Sigma_1, \ldots, \Sigma_n)$ we denote the set of all vector letters over $\Sigma_1, \ldots, \Sigma_n$. An n-dimensional vector alphabet (over $\Sigma_1, \ldots, \Sigma_n$) is a finite, possibly empty, subset of $tot(\Sigma_1, \ldots, \Sigma_n)$. Since vector alphabets are alphabets, all terminology and notation for alphabets, words and languages can be carried over. Next to normal concatenation, also component-wise concatenation can be considered in the context of vectors. An (n-dimensional) word vector (over $\Sigma_1, \ldots, \Sigma_n$) is a tuple $(v_1, \ldots, v_n)$ with $v_i \in \Sigma_i^*$ for all $i \in [n]$. The component-wise concatenation of two word vectors $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$ is defined by $v \circ w = (v_1 w_1, \ldots, v_n w_n)$. Thus $v \circ w$ is again a word vector. The support of an n-dimensional word vector v tells which entries of v are not λ: $support(v) = \{i \in [n] : v(i) \neq \lambda\}$.
Let $\Theta \subseteq tot(\Sigma_1, \ldots, \Sigma_n)$ be a vector alphabet. A word vector over Θ is a finite component-wise concatenation of vector letters from Θ. An (n-dimensional) vector language (over Θ) is a set of (n-dimensional) word vectors (over Θ). $\Theta^{\otimes}$ is the set of all word vectors over Θ. Thus $(tot(\Sigma_1, \ldots, \Sigma_n))^{\otimes} = \Sigma_1^* \times \cdots \times \Sigma_n^*$ and $\Theta^{\otimes}$ with component-wise concatenation is a monoid generated by Θ with the n-dimensional word vector $(\lambda, \ldots, \lambda)$ as its unit element. In general $\Theta^{\otimes}$ is not a free monoid. We have, e.g., $(\lambda, b, d) \circ (a, c, \lambda) = (a, b, d) \circ (\lambda, c, \lambda) = (a, bc, d)$. To *collapse* a sequence of vector letters into a word vector, we define the (monoid) homomorphism $coll_\Theta$ from Θ^* to $\Theta^{\otimes}$ by $coll_\Theta(\sigma) = \sigma$ for all $\sigma \in \Theta$. Hence $coll_\Theta(\sigma\tau) = \sigma \circ \tau$ for all vector letters σ and τ in Θ. The subscript Θ is usually omitted. Thus, e.g., $coll((\lambda, b, d)(a, c, \lambda)) = (\lambda, b, d) \circ (a, c, \lambda) = (a, bc, d)$.

A (context-free) grammar is a construct $G = (\Sigma, \Delta, S, \Pi)$, where Σ and Δ are two disjoint alphabets, $S \in \Sigma$, and Π is a finite set of pairs $(A, x) \in \Sigma \times (\Sigma \cup \Delta)^*$. We call Σ the non-terminal alphabet, Δ the terminal alphabet, S the axiom, and the elements of Π the productions of G. Usually, a production (A, x) is written as $A \to x$. Productions of the form $A \to \lambda$ are called λ-productions. If G does not have λ-productions, then it is called λ-free. If all productions in Π are of the form $A \to aB$, $A \to a$, or $A \to \lambda$ with $A, B \in \Sigma$ and $a \in \Delta$, then G is called a regular grammar.
A string x directly derives a string y in G by applying $\pi = A \to x \in \Pi$, notation $x \Longrightarrow_G^{\pi} y$, if $x = w_1 A w_2$ and $y = w_1 x w_2$ for some $w_1, w_2 \in (\Sigma \cup \Delta)^*$. The subscript G may be omitted if G is clear from the context. We write $x \Longrightarrow_G y$ or $x \Longrightarrow y$ if $x \Longrightarrow_G^{\pi} y$ for some production π of G. The transitive and reflexive closure of $\Longrightarrow$ and $\Longrightarrow_G$ are denoted by $\Longrightarrow^*$ and $\Longrightarrow_G^*$, respectively. $L(G)$, the language generated by G, is defined by $L(G) = \{w \in \Delta^* \mid S \Longrightarrow^* w\}$. A language is called context-free (regular) if it is generated by a context-free (regular) grammar.

3 Generalized Individual Token Net Controllers

In this section we introduce the Generalized Individual Token Net Controllers from [KK97]. A GITNC has an underlying structure which is a special type of

Petri net suitable to control and record the progress of individual sequential components. Such Petri nets are called Individual Token Nets and like ordinary Petri nets, they consist of places (drawn as circles), transitions (rectangles) and arcs (arrows) between them. The difference lies in the specification of a set of individual tokens and the special arc inscriptions. The individual tokens are to be related on a one-to-one basis to the sequential components, while the arc inscriptions describe the flow of these tokens through the net.

Definition 1. *An* Individual Token Net *(ITN) is a construct* $N = (P, T, F, I)$ *where* P*,* T*, and* I *are finite mutually disjoint sets, and* F *is a function from* $(P \times T) \cup (T \times P)$ *to* $\mathcal{P}(I)$*.*
P *is the set of* places *of* N*;* T *is the set of* transitions *of* N*;* I *is the set of* individual tokens *of* N*; and* F *is the* flow function *of* N*, assigning to each pair* $(x, y) \in (P \times T) \cup (T \times P)$ *a subset of* I *such that, for all* $t \in T$*,*
(i) for all distinct $p, p' \in P$*:* $F(p,t) \cap F(p',t) = \emptyset$ *and* $F(t,p) \cap F(t,p') = \emptyset$*;*
(ii) $\bigcup_{p \in P} F(p,t) = \bigcup_{p \in P} F(t,p)$*.* □

Let N be an ITN as specified in the above definition. The flow function F describes for each transition t, its input places (those p such that $F(p,t) \neq \emptyset$) and its input tokens ($\bigcup_{p \in P} F(p,t)$); its output places (those p such that $F(t,p) \neq \emptyset$) and its output tokens ($\bigcup_{p \in P} F(t,p)$). The conditions on F guarantee that, for any given transition, no individual token appears more than once as an input token or more than once as an output token. Moreover, for each transition, its set of input tokens is the same as its set of output tokens. Consequently, a transition either does not use a token at all or it uses it exactly once as input and once as output, a property which in [KKR90] is called 1-throughput.

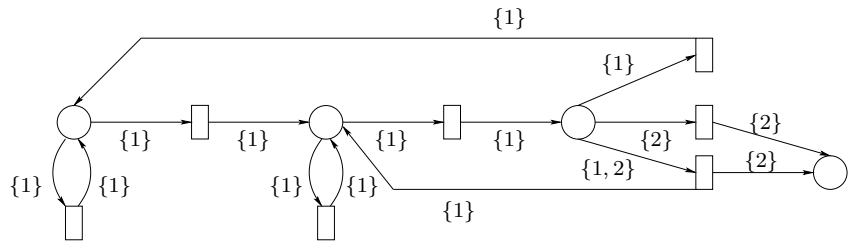

Fig. 1. An ITN with individual tokens $\{1, 2\}$.

This is the basis of the firing rule for ITNs. A transition t can fire (occur) if each of its input places p contains the individual tokens specified in $F(p,t)$. When t occurs, its input tokens are consumed and in each ouput place q of t new tokens as described by $F(t,q)$ are produced. This defines the dynamics of an ITN once we have determined its state space. The states of an ITN, or in Petri net terminology its markings, consist of a distribution of the individual tokens over the places. According to the intuition that the individual tokens are used to

follow individual sequential components, we require that in every marking each token appears exactly once. Firing a transition preserves this property, because the transitions are 1-throughput.

Definition 2. *Let $N = (P, T, F, I)$ be an ITN.*

(1) A marking with individual tokens *of N (it-marking of N), is a function $M : P \to \mathcal{P}(I)$ such that for all distinct $p, p' \in P$: $M(p) \cap M(p') = \emptyset$ and $\bigcup_{p \in P} M(p) = I$. $\mathcal{M}_N$ is the set of all it-markings of N.*

(2) Let $t \in T$ and let $M \in \mathcal{M}_N$ be an it-marking of N. Then t has concession *in M, notation $M[t\rangle_N$, if $F(p,t) \subseteq M(p)$ for all $p \in P$.*

(3) Let $t \in T$ and let $M, M' \in \mathcal{M}_N$ be two it-markings of N. Then t fires from *M to M', notation $M[t\rangle_N M'$, if $M[t\rangle_N$ and $M'(p) = (M(p) - F(p,t)) \cup F(t,p)$ for all $p \in P$.*

(4) Let $m \geq 0$, let $t_1, \ldots, t_m \in T$, and let $M, M' \in \mathcal{M}_N$. Then $t_1 \cdots t_m$ is a firing sequence of N leading from M_0 to M_m, *notation $M_0[t_1 \cdots t_m\rangle_N M_m$, if there exist it-markings $M_0, \ldots, M_m$ of N such that $M_{j-1}[t_j\rangle_N M_j$ for all $1 \leq j \leq m$.* □

We have $M[\lambda\rangle_N M$ for all $M \in \mathcal{M}_N$. This follows from (4) above when $m = 0$. Sometimes it is convenient to consider it-markings from a dual viewpoint and to know for each individual token, the place where it resides rather than specifying per place the individual tokens it contains. If $M : P \to \mathcal{P}(I)$ is an it-marking of the ITN $N = (P, T, F, I)$, then we associate with M the function $\overline{M} : I \to P$ defined by $\overline{M}(i) = p$ if $i \in M(p)$. Note that M can be recovered from $\overline{M}$, because $M(p) = \{i \mid \overline{M}(i) = p\}$ for all places p.

When used to control the behaviour of the components of a system, the transitions of a GITNC model synchronous occurrences of components' actions. They are labelled with vectors indicating which actions from which components participate in the synchronization. To facilitate the reasoning about tokens corresponding to components which in turn correspond to the entries in the vectors, we assume that the individual tokens of an ITN form the set $[n]$, where $n \geq 1$ is the number of components of the system. From now on n is fixed.
The vector labelling a transition has to be consistent with the components synchronizing through that transition: the i-th entry of the label is empty if and only if the synchronization does not involve the i-th component, that is, the i-th token is neither input nor output to that transition. Thus we arrive at the intermediate notion of an ITN $N = (P, T, F, [n])$ with vector labels.
By $use(t) = \bigcup_{p \in P} F(p,t) = \bigcup_{p \in P} F(t,p)$ we denote the set of tokens used by transition t in N.

Definition 3. *A* (n-dimensional) Vector Labelled Individual Token Net, *(n-) VLITN, is a construct $N = (P, T, F, [n], \Theta, l)$, where $(P, T, F, [n])$ is an ITN, Θ is an n-dimensional vector alphabet, and $l : T \to \Theta$ is a labelling of the transitions such that $support(l(t)) = use(t)$ for all $t \in T$.* □

The terminology and notation introduced for ITNs is carried over to VLITNs in the standard way through their underlying ITNs.

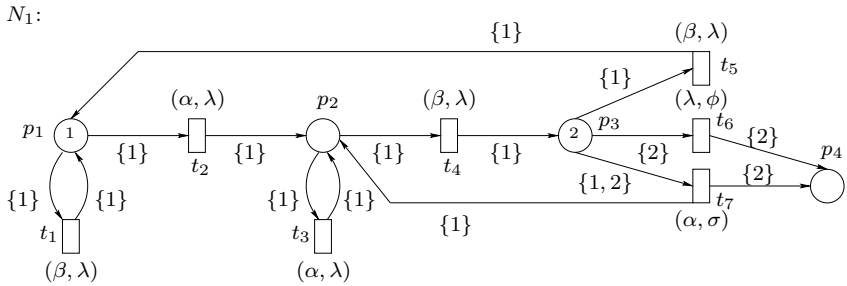

Fig. 2. A 2-VLITN with an it-marking.

Distinguishing initial and final it-markings leads to the notion of a Generalized Individual Token Net Controller.

Definition 4. *A* (n-dimensional) Generalized Individual Token Net Controller*, (n-)GITNC, is a construct* $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$ *where* N *is an n-VLITN,* $\mathcal{M}_{in} \subseteq \mathcal{M}_N$ *is* the set of initial it-markings of C *and* $\mathcal{M}_{fin} \subseteq \mathcal{M}_N$ *is* the set of final it-markings of C. □

In a GITNC individual tokens follow the progress of individual sequential components. Since the transitions are 1-throughput, each token uniquely determines a sequential subnet (a state machine). Thus a GITNC can be viewed as being composed of finite state machines each monitoring the behaviour of a component. The transitions of the GITNC represent synchronized combinations of state-action transitions from the state machines. The vector label of a transition combines the action labels of the state machine transitions involved and has an empty component for a state machine if and only if that state machine is not involved in the synchronization.

A GITNC exercises its control through its firing rule. The vector labels of the transitions in a firing sequence leading from an initial it-marking to a final one describe a history of synchronizations taking place between the components. Thus the labelled firing sequences form a control language which describes in which order the synchronizations may take place. Component-wise concatenating (collapsing) the vector labels of a firing sequence yields a word vector. These word vectors, collected in the vector language of the system, describe which combinations of individual behaviours of the components are allowed by the GITNC without referring to the underlying decomposition in vector labels.

Definition 5. *Let* $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$ *be a GITNC with* $N = (P, T, F, [n], \Theta, l)$.

(1) $FS(C) = \{u \in T^* \mid M[u\rangle_N M', M \in \mathcal{M}_{in}$ *and* $M' \in \mathcal{M}_{fin}\}$ *is the set of* firing sequences of C.
(2) $L(C) = l(FS(C))$ *is the* language of C.
(3) $V(C) = coll(l(FS(C)))$ *is the* vector language of C. □

The label function l is extended in the standard way to a homomorphism for sequences of transitions by $l(\lambda) = \lambda$ and $l(t_1 \cdots t_k) = l(t_1) \cdots l(t_k)$.

Example 1. Consider the 2-GITNC C_1 defined by the 2-VLITN N_1 of Fig. 2, with the it-marking M defined by $\overline{M}(1) = p_1$ and $\overline{M}(2) = p_3$ depicted in that figure as its single initial it-marking, and all it-markings of N_1 as its final it-markings. Then, e.g., $t_1t_2t_4t_5t_6 \in FS(C)$ and hence $(\beta, \lambda)(\alpha, \lambda)(\beta, \lambda)(\beta, \lambda)(\lambda, \phi) \in L(C_1)$. Also, $t_1t_2t_4t_6t_5$ and $t_6t_1t_2t_4t_5$ are firing sequences of C_1 and thus $(\beta, \lambda)(\alpha, \lambda)$ $(\beta, \lambda)(\lambda, \phi)(\beta, \lambda)$ and $(\lambda, \phi)(\beta, \lambda)(\alpha, \lambda)(\beta, \lambda)(\beta, \lambda) \in L(C_1)$. Each when collapsed yields the word vector $(\beta\alpha\beta\beta, \phi) \in V(C_1)$. Yet another firing sequence of C_1 is $t_2t_4t_7t_4$ and thus $(\alpha, \lambda)(\beta, \lambda)(\alpha, \sigma)(\beta, \lambda) \in L(C_1)$ and $(\alpha\beta\alpha\beta, \sigma) \in V(C_1)$. It is easily seen that $V(C) = \{(w, \sigma) \mid w \in \{\alpha, \beta\}^*, \alpha\beta\alpha$ occurs exactly once in $w\}$ $\cup \{(w, x) \mid w \in \{\alpha, \beta\}^*, x = \lambda$ or $x = \phi$, and $\alpha\beta\alpha$ does not occur in $w\}$. □

By $\mathcal{L}_n(GITNC)$ we denote the family of n-GITNC languages and the family of n-GITNC vector languages is denoted by by $\mathcal{V}_n(GITNC)$.

GITNCs are finite state devices and consequently define regular languages. In general, regular languages over vector letters can be defined as the languages accepted by vector labelled finite automata. These automata are however sequential in nature, whereas a GITNC has a distributed state space implying that the occurrences of transitions are not necessarily ordered. In particular, whenever two transitions which do not use a common token occur consecutively in a firing sequence, they can occur in either order. Due to the consistency of the vector labelling this implies that whenever two vector labels which have disjoint supports occur consecutively in a labelled firing sequence of the GITNC, also the labelled firing sequence with these two occurrences interchanged will belong to the language of the GITNC (see also Example 1). Consequently, the regular languages (over vector letters) defined by GITNCs have a particular structure and not every regular language over vector letters (of the same dimension) can be implemented as the language of a GITNC. In fact, as discussed in [KK97], GITNC languages are related to regular trace languages.
However, when vector languages are considered, the situation changes. The vector language of a vector labelled finite automaton is defined, just as for a GITNC, by applying component-wise concatenation to (collapsing) the words in its language. The vector languages thus obtained are exactly the rational relations (see, e.g., [Ber79]). It was shown in [KK97] how changing the underlying vector alphabet makes it possible to transform a vector labelled finite automaton into a GITNC with the same vector language, but not necessarily the same language. Let now $\mathcal{L}_n(REG)$ denote the family of regular languages over n-dimensional vector alphabets and let $n\mathcal{RAT}$ be the family of n-dimensional rational relations or equivalently, the collapses of the languages in $\mathcal{L}_n(REG)$. Then we have

Theorem 1. *([KK97])*

(1) $\mathcal{L}_1(GITNC) = \mathcal{L}_1(REG)$.
(2) $\mathcal{L}_n(GITNC) \subset \mathcal{L}_n(REG)$ *if* $n \geq 2$.
(3) $\mathcal{V}_n(GITNC) = n\mathcal{RAT}$. □

4 PN Grammar Systems

We are now ready to define PN grammar systems in which context-free grammars cooperate by synchronizing the application of their productions under the control of a GITNC. The productions of the grammars are given names and the vectors labelling the transitions of the GITNC are vector letters over these alphabets of names.

Definition 6. *An* (n-dimensional) PN grammar system*, (n-)PNGS, is a construct* $\Gamma = ((G_1, \Theta_1, \varphi_1), \ldots, (G_n, \Theta_n, \varphi_n); C)$ *where, for each* $i \in [n]$, $G_i = (\Sigma_i, \Delta_i, S_i, \Pi_i)$ *is a grammar, the* i-th component grammar of Γ, Θ_i *is an alphabet (of labels),* $\varphi_i : \Pi_i \to \Theta_i$ *is a function assigning labels to the productions of* G_i*, and* $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$ *with VLITN* $N = (P, T, F, [n], \Theta, l)$ *is an n-GITNC,* the controller of Γ*, with* $\Theta \subseteq tot(\Theta_1, \ldots, \Theta_n)$.
Γ *is called* regular *if all of its components are regular grammars.* □

For the rest of this section Γ is an arbitrary n-dimensional PN grammar system as specified in the above definition.

The component grammars of a PN grammar system each have their own sentential form and so the states of the system are described by the current sentential forms of the grammars and the current it-marking of the controller.

Definition 7. *A* configuration *of* Γ *is a tuple* $(v_1, \ldots, v_n; M)$ *with* $v_i \in (\Sigma_i \cup \Delta_i)^*$ *for all* $i \in [n]$*, and* $M \in \mathcal{M}_N$ *an it-marking of* C.
Let $\gamma = (v_1, \ldots, v_n; M)$ *be a configuration of* Γ*. Then*
(1) γ *is an* initial configuration of Γ *if* $v_i = S_i$ *for all* $i \in [n]$ *and* $M \in \mathcal{M}_{in}$.
(2) γ *is a* final configuration of Γ *if* $v_i \in \Delta_i^*$ *for all* $i \in [n]$ *and* $M \in \mathcal{M}_{fin}$. □

A configuration changes when a transition fires and this entails a synchronized application of productions to which its vector label refers. Thus a production can only be applied by a component grammar to its current sentential form if this can happen synchronously with the application of productions by other grammars in accordance with the vector label of a transition with concession. Dually, a transition with concession can only occur if the productions to which it refers can actually be applied in the sentential forms.

Definition 8. *(1) Let* $t \in T$ *and let* $\gamma = (v_1, \ldots, v_n; M)$ *be a configuration of* Γ*. Then* t can occur (in Γ) at γ*, notation* $\gamma \vdash_\Gamma^t$*, if* $M[t\rangle_N$ *and if, for all* $i \in [n]$*,* $proj_i(l(t)) \neq \lambda$ *implies that there is a production* $\pi = A \to w \in \Pi_i$ *such that* $\varphi_i(\pi) = proj_i(l(t))$ *and* A *appears in* v_i.
(2) Let $t \in T$ *and let* $\gamma = (v_1, \ldots, v_n; M)$ *and* $\gamma' = (w_1, \ldots, w_n; M')$ *be two configurations of* Γ*. Then* γ is transformed into γ' by the occurrence of t (in Γ)*, notation* $\gamma \vdash_\Gamma^t \gamma'$*, if* $M[t\rangle M'$ *and for all* $i \in [n]$*,* $v_i = w_i$ *whenever* $proj_i(l(t)) = \lambda$*, otherwise* $v_i \Longrightarrow_{G_i}^{\pi} w_i$ *where* $\pi \in \Pi_i$ *is such that* $\varphi_i(\pi) = proj_i(l(t))$. □

We write $\gamma \vdash_\Gamma \delta$ if there exists a $t \in T$ such that $\gamma \vdash_\Gamma^t \delta$. The subscript Γ may be omitted if there is no danger of confusion with which system we are dealing.

A successful run of a PN grammar system starts in an initial configuration and leads by to a final one a sequence of successive transformations. Thus the system operates by synchronizing the derivation steps of its component grammars according to the control exercised by its controller through firing sequences. The resulting combinations of terminal strings form the vector language of the system.

Definition 9. *The* vector language generated by Γ, *denoted by* $V(\Gamma)$, *is defined by* $V(\Gamma) = \{(v_1, \ldots, v_n) \in \Delta_1^* \times \cdots \times \Delta_n^* \mid (S_1, \ldots, S_n; M) \vdash^* (v_1, \ldots, v_n; M')$ *for some* $M \in \mathcal{M}_{in}, M' \in \mathcal{M}_{fin}\}$. □

By $\mathcal{V}_n(PNGS)$ we denote the family of vector languages generated by n-PNGSs and the family of vector languages generated by n-dimensional regular PN grammar systems is denoted by $\mathcal{V}_n(regPNGS)$.

Example 2. Let $\Gamma = ((G_1, \Theta_1, \varphi_1), (G_2, \Theta_2, \varphi_2); C_1)$ be the regular 2-PNGS with C_1 as in Example 1, $G_1 = (\{S\}, \{a, b\}, S, \Pi_1)$ and $G_2 = (\{S\}, \{s, f\}, S, \Pi_2)$ where $\Pi_1 = \{S \to aS,\ S \to a,\ S \to bS,\ S \to b\}$ and $\Pi_2 = \{S \to s,\ S \to f\}$.
The productions are labelled by $\varphi_1(S \to aS) = \varphi_1(S \to a) = \alpha$, $\varphi_1(S \to bS) = \varphi_1(S \to b) = \beta$, and $\varphi_2(S \to s) = \sigma$ and $\varphi_2(S \to f) = \phi$.
The initial configuration of the system is $(S, S; M)$. At $(S, S; M)$ the three transitions t_1, t_2, and t_6 can occur. If t_1 occurs, then the configuration is changed in either $(bS, S; M)$ or $(b, S; M)$, depending on which production labelled by β is chosen to be applied. If t_6 occurs, then the configuration is changed in $(S, f; M_1)$ with $\overline{M_1}(1) = p_1$ and $\overline{M_1}(2) = p_4$.
Consider the firing sequence $t_1t_2t_4t_5t_6$ labelled by $(\beta, \lambda)(\alpha, \lambda)(\beta, \lambda)(\beta, \lambda)(\lambda, \phi)$. On basis of this sequence we obtain the transformation sequence $(S, S; M) \vdash (bS, S; M) \vdash (baS, S; M_2) \vdash (babS, S; M_3) \vdash (babb, S; M) \vdash (babb, f; M_1)$ with $\overline{M_2}(1) = p_2$ and $\overline{M_2}(2) = p_3$, $\overline{M_3}(1) = \overline{M_3}(2) = p_3$. Since every it-marking is final and both grammars have derived a terminal word we have $(babb, f) \in V(\Gamma)$.
If for the last occurrence of β we would have chosen to apply $S \to bS$ rather than $S \to b$, then the non-final configuration $(babbS, f; M_1)$ would have resulted. If at the occurrence of t_2, we would have chosen to apply $S \to a$ instead of $S \to aS$, then we would have $(S, S; M) \vdash (bS, S; M) \vdash (ba, S; M_2)$ and t_4 can no longer occur, since the first grammar has produced a terminal word. The only transition that can still occur is t_6 entailing a rewriting by the second grammar.
Observe that the firing sequences $t_1t_2t_4t_6t_5$ and $t_6t_1t_2t_4t_5$ also give rise to the generation of $(babb, f)$, since both define the same vector word $(\beta\alpha\beta\beta, \phi)$ which enforces the successive application of productions labelled by β, α, β, and β respectively by the first grammar and the application of $S \to f$ in the second grammar.
A successful run of Γ controlled by the firing sequence $t_2t_4t_7t_4$ labelled by $(\alpha, \lambda)(\beta, \lambda)(\alpha, \sigma)(\beta, \lambda)$, leads to $(abab, s; M_4)$ or $(ababS, s; M_4)$ with $\overline{M_4}(1) = p_3$ and $\overline{M_4}(2) = p_4$. The application of $S \to s$ by G_2 is synchronized with the application of a production labelled by α in order to signal the introduction by G_1 of a subword aba. As long as this has not happened, transition t_6 can occur which implies the application of the production $S \to f$ by G_2 and prohibits the introduc-

tion of *aba* by G_1. Thus we have $V(\Gamma) = \{(w, s) \mid w \in \{a, b\}^*$, *aba* occurs exactly once in $w\} \cup \{(w, f) \mid w \in \{a, b\}^*, w \neq \lambda$, and *aba* does not occur in $w\}$. A special property of this PN grammar system is the one-to-one correspondence between the labels of the productions (α, β, σ, and ϕ) and the terminal symbols (a, b, s, and f) they introduce. As will become clear in Section 5.2, this induces a strong similarity between the vector language of a regular PNGS and the vector languages of its GITNC.
Finally, note that without the control of C_1, the grammar G_1 can generate all non-empty words over $\{a, b\}$. □

As an immediate consequence of the definitions we have that the vector language of a PN grammar system consists of those combinations of terminal words for which there exists an approved (by the controller) combination of sequences of productions applied in the component grammars.

Corollary 1. $V(\Gamma) = \{(v_1, \ldots, v_n) \in \Delta_1^* \times \cdots \times \Delta_n^* \mid$ *there exists an* $x \in V(C)$ *such that, for all* $i \in [n]$, $S_i \Longrightarrow_{G_i}^* v_i$ *by applying the sequence of productions* $proj_i(x)\}$. □

In the PN grammar system Γ, the control exercised by the GITNC C concerns the *application* of productions. *Listing* the productions as abstract entities rather than applying them describes the effect of the cooperation protocol enforced by the GITNC on the rewriting steps within the grammars. The result is a vector language of the form $(Sz_1 \times \cdots \times Sz_n) \cap V(C)$ where Sz_i is the Szilárd language (consisting of labelled sequences of productions that can be successfully applied) of the i-th component grammar. This vector language describes which sequences of productions in the component grammars can be successfully combined in a system derivation. It represents the concurrency in the derivations of a PN grammar system resulting from its rewriting protocol. This corresponds to the set-up of the model of Vector Controlled Concurrent Systems where the concurrent behaviours of a VCCS are given in the form of word vectors representing successful synchronized combinations of sequential behaviours of the components.
The potential concurrency in the derivations in a PN grammar systems is due to the fact that without the control exercised by the GITNC the grammars work independently, that is, concurrently, each on their own sentential form. The protocol represented by the GITNC enforces certain synchronized applications of productions. Since however, as observed before, the GITNC is itself a concurrent device with a distributed state space, synchronizations involving disjoint groups of grammars can still be executed concurrently. As we explain next, this allows to "speed up" the allowed derivation sequences represented in the language of the controller without affecting the protocol itself as represented in its vector language.
The grammars of Γ are sequential rewriting systems. Productions are applied one after the other. In C we have one individual token for each grammar and the occurrences of transitions which use a common individual token are always ordered by their use of this token. Let us say that two transitions t and t' are independent if they do not use a common individual token, i.e. $use(t) \cap use(t') = \emptyset$.

A non-empty subset U of T is called independent if every pair of distinct transitions in U is independent. Independent transitions can be fired concurrently which we formalize in a step semantics as is usually done for Petri nets. Thus, for an it-marking $M \in \mathcal{M}_N$ and an independent set $U \subseteq T$, we say that U is a step at M, notation $M[U\rangle_N$, if $M[t\rangle_N$ for all $t \in U$. The effect of a step U occurring at an it-marking M is the accumulated effect of the transitions forming the step. Thus firing U leads from M to M', notation $M[U\rangle_N M'$, with M' defined by $M'(p) = (M(p) - \bigcup_{t\in U} F(p,t)) \cup \bigcup_{t\in U} F(t,p)$ for all $p \in P$. Note that M' is again an it-marking. Since the transitions forming a step at a given it-marking all use their own subset of the individual tokens, they can fire in any order from that it-marking. More precisely, $M[U\rangle_N M'$ if and only if U is an independent set and $M[t_1 \cdots t_k\rangle_N M'$ for every permutation $t_1 \cdots t_k$ of the elements of U.
Steps may be fired consecutively. We write $M[U_1 \cdots U_k\rangle_N M'$ if for all $1 \leq j \leq k$, U_j is an independent set and there exist it-markings $M_0 = M, M_1, \ldots, M_k = M'$ such that $M_{j-1}[U_j\rangle_N M_j$ for all $1 \leq j \leq k$.
Because of the consistency of the vector labelling, independency of transitions is equivalent with their vector labels having disjoint supports. Thus the componentwise concatenation of the vector labels of independent transitions forms a vector letter. Formally, if $U \subseteq T$ is an independent set, then $proj_i(l(t_1) \cdots l(t_k)) \in proj_i(\Theta) \cup \{\lambda\}$ for every $i \in [n]$ and for every permutation $t_1 \cdots t_k$ of the elements of U. Now we can unambiguously associate to U a vector label $l_{step}(U)$ defined by $l_{step}(U) = coll(l(t_1) \cdots l(t_k))$ where $t_1 \cdots t_k$ is any permutation of the elements of U. It is immediate that allowing steps with vector label function l_{step} does not add to the vector language of a GITNC $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$. Moreover, $l_{step}(\{t\}) = l(t)$ for all $t \in T$ and every firing sequence of C can be considered as a step sequence.

Lemma 1. *$V(C) = \{coll(l_{step}(u)) \mid u = U_1 \cdots U_k$ with $U_j \subseteq T$ for all $1 \leq j \leq k$ and $M[u\rangle_N M'$ for some $M \in \mathcal{M}_{in}$ and $M' \in \mathcal{M}_{fin}\}$.* □

We extend the definition of transformations to transformations by occurrences of steps as follows. Let $U \subseteq T$ be an independent set and let $\gamma = (v_1, \ldots, v_n; M)$ be a configuration of Γ. Then U can occur at γ, if $M[U\rangle_N$ and if, for all $i \in [n]$, $proj_i(l_{step}(U)) \neq \lambda$ implies that there is a production $\pi = A \to w \in \Pi_i$ such that $\varphi_i(\pi) = proj_i(l_{step}(U))$ and A has an occurrence in v_i. If $U \subseteq T$ occurs at $\gamma = (v_1, \ldots, v_n; M)$, then γ is transformed into $\gamma' = (w_1, \ldots, w_n; M')$ by the occurrence of U, notation $\gamma \vdash^U_{step} \gamma'$, if $M[U\rangle_N M'$ and for all $i \in [n]$, $v_i = w_i$ whenever $proj_i(l_{step}(U)) = \lambda$, otherwise $v_i \Longrightarrow^{\pi}_{G_i} w_i$ where $\pi \in \Pi_i$ is such that $\varphi_i(\pi) = proj_i(l_{step}(U))$. We write $\gamma \vdash_{step} \gamma'$ if $\gamma \vdash^U_{step} \gamma'$ for some step U.
Thus, by Corollary 1 and Lemma 1, the derivations in Γ can be based on the step semantics of C.

Theorem 2. *$V(\Gamma) = \{(v_1, \ldots, v_n) \in \Delta_1^* \times \cdots \times \Delta_n^* \mid (S_1, \ldots, S_n; M) \vdash^*_{step} (v_1, \ldots, v_n; M')$ for some $M \in \mathcal{M}_{in}, M' \in \mathcal{M}_{fin}\}$.* □

The step semantics offers the possibility to "speed up" the (sequential) derivation processes in a PN grammar system since independent rewritings by the component grammars of a PN grammar system may be executed simultaneously. The

vector labels assigned to steps show that the step semantics translates concurrency into synchronization. These two phenomema are however not the same. Concurrency is based on independence and implies that independent derivation steps in the component grammars may be executed in any order and also synchronously. Synchronization of productions on the other hand is a tool that can be used to guarantee that different components will simultaneously reach a given state from a given state and hence, unlike concurrency, implies a dependence between the components.

5 Vector Languages of PN Grammar Systems

Definition 9 describes the result of the work of a PN grammar system Γ in terms of vectors consisting of those combinations of terminal words for which the component grammars have a cooperation strategy approved by the controller. There are of course more options to define the result of a successful run of a PN grammar system. As for PC grammar systems, one might, e.g., decide to focus on the result of one (central) grammar and thus define a language rather than a vector language. Or one could consider the *concatenated language of* Γ, obtained by concatenating the strings of the generated word vectors into one terminal string: $L_\bullet(\Gamma) = \{v_1 \cdots v_n \mid (v_1, \ldots, v_n) \in V(\Gamma)\}$. Yet another option is to define the *agreement language of* Γ by considering only those terminal strings on which the components agree: $L_\cap(\Gamma) = \{w \mid$ there exists a $(v_1, \ldots, v_n) \in V(\Gamma)$ such that for all $i \in [n]$, $v_i = w$ or $v_i = \lambda\}$. In terms of the blackboard model, both the vector language and the concatenated language present a view of the blackboard as a whole once all knowledge sources (the grammars) have come to a conclusion. In case of agreement however, a solution is only accepted if all knowledge sources offering an answer have come to the same conclusion.
In this introduction of PN grammar systems we opt for the vector language as the description of the result of the collaboration of the grammars, since it appears as the more general definition which forms a basis for other definitions.

5.1 A Normal Form

In order to control the derivations of the grammars in a PN grammar system, productions are assigned labels and these are combined into vectors expressing the allowed synchronizations. Different productions from the same grammar may have the same label. This makes it possible to efficiently represent the intended synchronizations in a succinct GITNC (see Example 2). A disadvantage may be the ambiguity of the vector labels which in this set-up can represent different combinations of productions. As we show next, this ambiguity can be avoided. In fact, every PNGS can effectively be transformed into a PNGS which defines the same vector language and in which the vector labels have only entries referring to unique productions. Let Γ be a PNGS specified as in Definition 6.

Definition 10. Γ *is* in normal form *if, for all distinct* $i, j \in [n]$, $\Sigma_i \cap \Sigma_j = \emptyset$, φ_i *is a bijection, and* $\Theta \subseteq tot(\varphi_1(\Pi_1), \ldots, \varphi_n(\Pi_n))$. □

Thus a PNGS is in normal form if the productions of the grammars and the entries of the vector labels of the controller are in one-to-one correspondence, and moreover all participating grammars have their own sets of non-terminals. It is not difficult to show, that this is indeed a normal form. Only when productions share a label, one has to be a bit careful.

Theorem 3. *For every PN grammar system, a PN grammar system in normal form and generating the same vector language can be constructed.*

Proof. Consider Γ. It is clear that the requirement of disjoint sets of non-terminals can easily be satisfied without affecting the vector language of Γ by renaming the non-terminal symbols with the obvious modifications in the set of productions and the definition of the label functions. Hence $\Sigma_i \cap \Sigma_j = \emptyset$, for all distinct $i, j \in [n]$ is considered to already hold for Γ.
Assume that for some $i \in [n]$, there are two distinct productions $\pi, \pi' \in \Pi_i$ such that $\varphi_i(\pi) = \varphi_i(\pi')$. Let us fix such i, π, and π'. For π' we add a new label θ to Θ_i and define a new label function φ_i' by $\varphi_i'(\pi') = \theta$ and $\varphi_i'(\rho) = \varphi_i(\rho)$ for all $\rho \in \Pi_i - \{\pi'\}$. For each transition t of C such that $proj_i(l(t)) = \varphi_i(\pi)$, we add a new transition t' with the same neighbourhood relations as t has. Each new transition t' is labelled by the same vector label as t except that the i-th entry is replaced by θ. The construction is illustrated in Fig. 3.

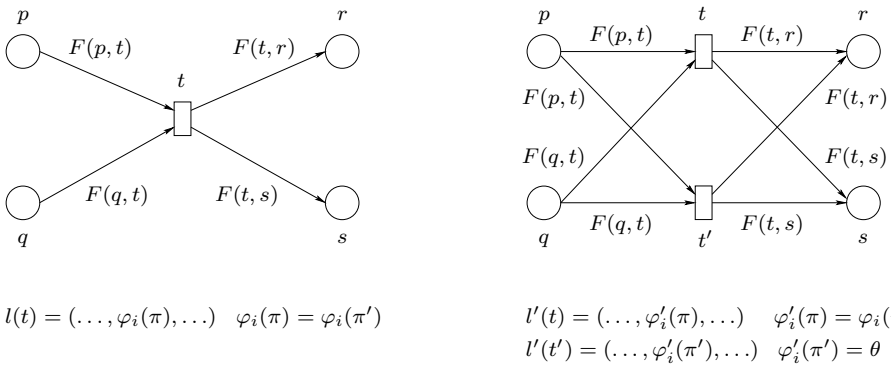

Fig. 3. A transition t with an ambiguous i-th entry in its vector label is split. The implicit choice between the productions π or π' is modelled explicitly using a (Petri net) conflict between t and a new transition t'.

Each new transition is 1-throughput and consistently labelled and so we obtain a new PN grammar system Γ'. That Γ' generates the same vector language as Γ is easy to prove. We repeat this procedure until no component grammar is left with different productions which have the same label associated to them. As the last step we remove for each component grammar all labels that are not assigned to productions and we obtain the desired bijection between labels and productions.

Finally, transitions which have a vector label with a non-empty entry that does not occur as the label of a production in the corresponding grammar can never be used to transform a configuration of the PN grammar system. Hence we can delete all such transitions without affecting the generated vector language. The remaining transitions are all labelled by vector letters over the labelling alphabets of the productions, as required. □

5.2 Expressive Power

In a 1-dimensional PN grammar system there is only one grammar which works under the control of its GITNC. For a derivation to be successful, the order in which the productions are applied has to form a sequence belonging to the language of the controller. Thus, by Theorem 1, these grammar systems can be considered to be context-free grammars with a regular control language (see, e.g., [DPS97]) and so $\mathcal{V}_1(PNGS)$ can be identified with $\mathcal{L}(RC)$, the family of regularly controlled context-free languages (without appearance checking). However, since we deal with vectors (in this case 1-dimensional), the formal relationship is as follows.

Theorem 4. $proj_1(\mathcal{V}_1(PNGS)) = \mathcal{L}(RC)$. □

This family is incomparable with the family of context-sensitive languages, and strictly included in the family of recursively enumerable languages. When λ-productions are not allowed, the resulting family is strictly inbetween the families of context-free and context-sensitive languages.

To describe the vector languages of more dimensional PN grammar systems appears to be more complicated and we give only a first impression. Again one could use grammars with some form of regulated rewriting (see [DPS97]) and describe, e.g., using matrix grammars the concatenated versions of the vector languages. In matrix grammars, rewriting is based on finite sequences of productions which have to be applied consecutively. In a PN grammar system in normal form, each component has its own set of non-terminals and without the risk of confusing grammars we can apply the productions in a vector one after the other. Thus by concatenating the components' sentential forms and applying the synchronized rules in a vector sequentially as if it were a matrix, the concatenated version of the vector language is derived. That the sequence of productions to be used is determined by the (regular) control of the GITNC poses no problems, since regular control can be included in matrix grammars. Hence the concatenated vector languages of PN grammar systems can be generated by context-free matrix grammars (without appearance checking). Since the context-free matrix grammars and the regularly controlled context-free grammars define the same family of languages, this shows that as far as the concatenated vector languages of PN grammar systems are concerned, one grammar is as good as many.

Example 3. Let $\Gamma = ((G_1, \Theta_1, \varphi_1), (G_2, \Theta_2, \varphi_2), (G_3, \Theta_3, \varphi_3); C)$ be the 3-PNGS with $G_1 = (\{S\}, \{a, b\}, S, \Pi_1)$, $G_2 = (\{A\}, \{a\}, A, \Pi_2)$, $G_3 = (\{B\}, \{b\}, B, \Pi_3)$, with the productions and labelling of G_1 given by:

$\varphi_1(S \to \lambda) = \phi$, $\varphi_1(S \to LR) = \omega$, $\varphi_1(L \to aL) = \varphi_1(L \to a) = \alpha_L$ and $\varphi_1(R \to aR) = \varphi_1(R \to a) = \alpha_R$, $\varphi_1(L \to bL) = \varphi_1(L \to b) = \beta_L$ and $\varphi_1(R \to bR) = \varphi_1(R \to b) = \beta_R$;
the productions and labelling of G_2 are given by:
$\varphi_2(A \to \lambda) = \phi$, $\varphi_2(A \to A) = \varphi_2(A \to \lambda) = \omega$, $\varphi_2(A \to aA) = \varphi_2(A \to a) = \alpha$;
and those of G_3 by:
$\varphi_3(B \to \lambda) = \phi$, $\varphi_3(B \to B) = \varphi_3(B \to \lambda) = \omega$, $\varphi_3(B \to bB) = \varphi_3(B \to b) = \beta$.
The GITNC C consists of the VLITN N depicted in Fig. 4 with one and the same initial and final it-marking M which assigns each of the three tokens 1,2, and 3 to the place in the middle.

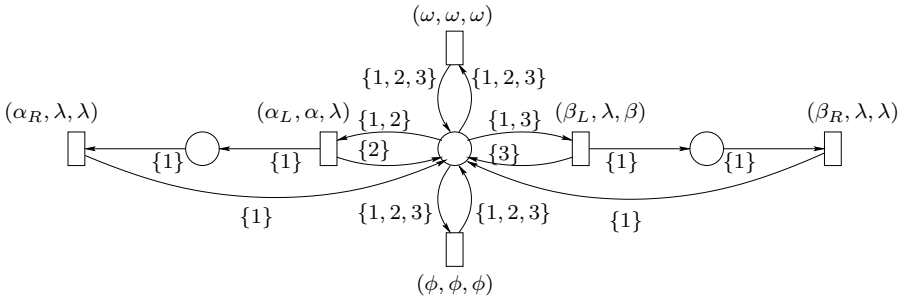

Fig. 4. The VLITN N underlying C of Example 3.

Since every transition of N uses token 1, grammar G_1 is involved in every transformation of a configuration of Γ and the control exercised by C is sequential. Initially only two transitions can occur. The transition labelled by (ϕ, ϕ, ϕ) transforms the initial configuration $(S, A, B; M)$ into $(\lambda, \lambda, \lambda; M)$ and the rewriting stops. If the transition labelled by (ω, ω, ω) starts the rewriting process, then the initial configuration is transformed into one of the four configurations $(LR, X, Y; M)$ with $X \in \{A, \lambda\}$ and $Y \in \{B, \lambda\}$. To rewrite LR the control either switches to the left part of N or to its right part. The consecutive occurrences of the two left-most transitions labelled by $(\alpha_L, \alpha, \lambda)$ and $(\alpha_R, \lambda, \lambda)$ force G_1 to generate an a using either $L \to aL$ or $L \to a$ followed by $R \to aR$ or $R \to a$. In that case, the second grammar G_2 has to synchronize by rewriting A and producing an a (hence A still has to be present in the second sentential form). When the right-most transitions occur, G_1 is forced to generate a b using either $L \to bL$ or $L \to b$ followed by $R \to bR$ or $R \to b$. Then the third grammar G_3 has to synchronize by rewriting B and adding a b to its sentential form. This process is repeated. Thus, on the one hand, the GITNC forces the first grammar (by a regular control mechanism) to generate the non-context-free language $\{ww \mid w \in \{a, b\}^*\}$. On the other hand G_2 and G_3 are forced to keep track, respectively, of $\#_a w$ the number of occurrences of a in the word w generated from L, and of $\#_b w$ the number of occurrences of b in the word w generated from L. They may only terminate when no more a's or b's, respectively, will be

introduced by G_1, or conversely, as soon as they terminate, grammar G_1 can no longer introduce a's or b's by rewriting L. Hence Γ generates the vector language $\{(ww, a^n, b^m) \mid w \in \{a,b\}^*, n = \#_a w, \text{ and } m = \#_b w\}$. □

The components of the vector language generated by a PN grammar system cannot simply be viewed as generated by a component grammar under the control of the corresponding subnet of the GITNC, since the control not only depends on the subnet but also on the behaviour of the other components. Both the synchronization constraints within the GITNC and the requirement that it must be possible to execute the productions associated to transitions synchronously have to be taken into account. The vector language of the system is in general strictly contained in the cartesian product of the (controlled) languages of its component grammars (see Example 3). Hence, simply using the projection function will not lead to a proper characterization of the structure of more-dimensional vector languages of PN grammar systems. For regular PN grammar systems we do have such a description. We will next demonstrate that the vector languages of these systems are precisely the rational relations, which form a class strictly larger than the cartesian products of regular languages.

Regular PN Grammar Systems From Theorem 1 we know that every rational relation can be defined as the vector language of a GITNC. Thus it is fairly easy to prove — by adding "dummy" grammars — that every rational relation can be generated by a regular PN grammar system.

Lemma 2. $n\mathcal{RAT} \subseteq \mathcal{V}_n(regPNGS)$.

Proof. Let $R \in n\mathcal{RAT}$ and let C be an n-GITNC such that $V(C) = R$. By Theorem 1 such C exists. Let $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$ with $N = (P, T, F, [n], \Theta, l)$. We add regular grammars $G_1, \ldots, G_n$ to C and change the vector labels of the transitions of N by replacing each non-empty entry with (a label referring to) a regular production that generates the original entry as a terminal symbol. For each $i \in [n]$, we let $G_i = (\{S_i\}, proj_i(\Theta), S_i, \Pi_i)$, with $\Pi_i = \{S_i \to \theta S_i \mid \theta \in proj_i(\Theta)\} \cup \{S_i \to \lambda\}$. The VLITN N' is N extended with new transitions t_M where $M \in \mathcal{M}_{fin}$ (note that $\mathcal{M}_{fin}$ is a finite set). Thus $N' = (P, T \cup \{t_M \mid M \in \mathcal{M}_{fin}\}, F', [n], \bigcup_{i\in[n]} \Pi_i, l')$ where for all $t \in T$ and $p \in P$, $F'(p,t) = F(p,t)$ and $F'(t,p) = F(t,p)$; furthermore, for all $M \in \mathcal{M}_{fin}$ and $p \in P$, $i \in F'(p, t_M) = F'(t_M, p)$ if and only if $\overline{M}(i) = p$. We label the productions of the G_i by identity. The vector labelling l' of the transitions of N' is defined by $proj_i(l'(t)) = S_i \to \theta S_i$ if $t \in T$ and $proj_i(l(t)) = \theta$, and $proj_i(l'(t_M)) = S_i \to \lambda$ for all $M \in \mathcal{M}_{fin}$ and $i \in [n]$. Clearly, $C' = (N'; \mathcal{M}_{in}, \mathcal{M}_{fin})$ is a GITNC.
Let $\Gamma = ((G_1, \Pi_1, \varphi_1), \ldots, (G_n, \Pi_n, \varphi_n); C')$ where φ_i is the identity on Π_i for each $i \in [n]$. Thus Γ is a regular PNGS. The firing of a transition t in N corresponds to the transformation of a configuration by the occurrence of t in Γ: $M[t\rangle_N M'$ if and only if $(x_1 S_1, \ldots, x_n S_n; M) \vdash^t_\Gamma (y_1 S_1, \ldots, y_n S_n; M')$ with for all $i \in [n]$, whenever $proj_i(l(t)) = \lambda$, then $y_i = x_i$; otherwise $y_i = x_i \theta S_i$ where $\theta = proj_i(l(t))$. The transitions t_M are used to check that the final it-marking

M has been reached and when such a transition occurs, the non-terminals S_i are erased: $\gamma \vdash_{\Gamma}^{t_M} \gamma'$ for some $M \in \mathcal{M}_{fin}$ if and only if γ' is a final configuration $(x_1, \ldots, x_n; M)$ of Γ and $\gamma = (x_1 S_1, \ldots, x_n S_n; M)$. From these observations it follows that $V(C) = V(\Gamma)$ and so $R = V(C) \in \mathcal{V}_n(regPNGS)$. □

To prove that the regular PN grammar systems generate exactly the rational relations, we provide a construction which incorporates the grammars of a regular PNGS into its GITNC in such a way that the resulting GITNC defines the same vector language as the original system. Then, again by Theorem 1, we may conclude that $\mathcal{V}_n(regPNGS)$ is included in $n\mathcal{RAT}$.

Let Γ be a fixed regular n-PNGS in normal form. Hence the productions of its component grammars are in one-to-one correspondence with their label. Thus the vector labels of the transitions can have productions as entries and there is no need to define label functions for the productions. Thus we specify Γ by $\Gamma = (G_1, \ldots, G_n; C)$ where, for all $i \in [n]$, $G_i = (\Sigma_i, \Delta_i, S_i, \Pi_i)$ is a regular grammar and $C = (N; \mathcal{M}_{in}, \mathcal{M}_{fin})$ is an n-GITNC with VLITN $N = (P, T, F, [n], \Theta)$ and $\Theta \subseteq tot(\Pi_1, \ldots, \Pi_n)$. In the construction we assume that each G_i is λ-free.
The idea behind the incorporation construction (see also Fig. 5) is to replace in the vectors labelling the transitions, every production by the terminal symbol it introduces. Thus a vector label $(\ldots, A \to aB, \ldots)$ is replaced by the vector label $(\ldots, a, \ldots)$. In order to faithfully implement the synchronizations of the productions, we have to keep track of the non-terminals. Therefore for each place p we will have copies (p, A), where A is a non-terminal symbol. Token i in place (p, A) in the new net represents the situation that token i resides in place p in the original net while A is the current non-terminal in the sentential form of the i-th component. (Note that we can speak of *the* non-terminal since G_i is regular.) The original places will still be present in the new net and token i in place p in the new net corresponds with token i in place p in the original net while the sentential form of the i-th component is a terminal string. A transition t labelled with a vector with $A \to aB$ as its i-th component, which in the original net consumes token i from place p and produces token i in place q, in the new net will take token i from place (p, A) and put it in place (q, B). If the i-th component of the label of t is $A \to a$, then in the new net t will still take token i from place (p, A) but now put it in place q rather than in (q, B). The original places p will not be input places of transitions. The initial it-markings of the new net are copies of the original initial it-markings in the sense that if the original marking had token i in place p, then the copy has token i in place (p, S_i). The final it-markings are not changed.
In the construction we use the following notation and terminology.
Let $\pi \in \bigcup_{i \in [n]} \Pi_i$ be a production of Γ and let G_j be the component grammar it belongs to. Note that since Γ is in normal form, there is only one index j such that $\pi \in \Pi_j$. Since G_j is a λ-free regular grammar, either $\pi = A \to aB$ or $\pi = A \to a$ for some $A, B \in \Sigma_j$ and $a \in \Delta_j$. In both cases, we refer to A as the left-hand side of π and write $lhs(\pi) = A$.
If $\pi = A \to aB$, then B is called the right-hand side of π, denoted by $rhs(\pi) = B$.
If $\pi = A \to a$ with $A \in \Sigma_j$ and $a \in \Delta_j$, then π is said to be terminating.

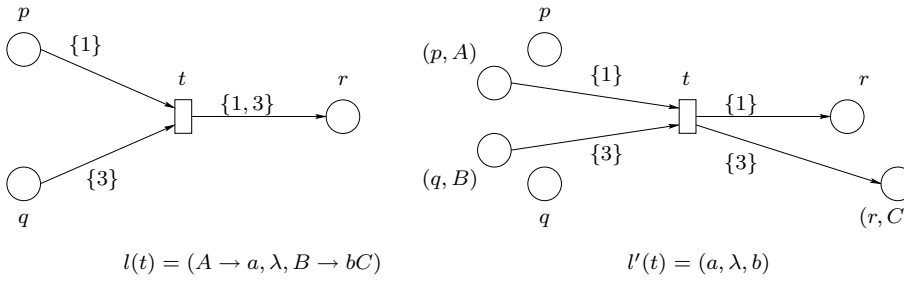

Fig. 5. The incorporation construction illustrated.

a is the symbol introduced by π, denoted by $sym_j(\pi)$. This leads, for each $i \in [n]$, to the definition of a homomorphism $sym_i : \Pi_i^* \to \Delta_i^*$ defined by $sym_i(\rho) = b$ if $\rho = X \to bY$ with $b \in \Delta_i$. The homomorphism $lsym : \Theta^* \to \Delta^{\otimes}$ is defined by combining these homomorphisms and applying them to vectors. Thus $lsym(\rho_1, \ldots, \rho_n) = (sym_1(\rho_1), \ldots, sym_n(\rho_n))$. Observe that $sym_i(\rho_i) = \lambda$ if and only if $\rho_i = \lambda$.

Definition 11. The controller incorporating $G_1, \ldots, G_n$ *into* C *is the construct* $inc(\Gamma) = (N'; \mathcal{M}'_{in}, \mathcal{M}_{fin})$ *with* $N' = (P', T, F', [n], \Delta, l')$ *where,*
$P' = P \cup (P \times \bigcup_{i \in [n]} \Sigma_i)$,
F' *is defined as follows:*
  for all $t \in T$, *for all* $p \in P$
$F'(p,t) = \emptyset$,
$F'(t,p) = \{i \mid i \in F(t,p)$ *and* $proj_i(l(t))$ *is terminating*$\}$,
  for all $t \in T$, *for all* $(p, A) \in P \times \bigcup_{i \in [n]} \Sigma_i$
$F'((p,A),t) = \{i \mid i \in F(p,t)$ *and* $lhs(proj_i(l(t))) = A\}$,
$F'(t,(p,A)) = \{i \mid i \in F(t,p)$ *and* $rhs(proj_i(l(t))) = A\}$,
$\Delta = tot(\Delta_1, \ldots, \Delta_n)$,
$l' : T \to \Delta$ *is defined by* $l'(t) = lsym(l(t))$ *and*
$\mathcal{M}'_{in} = \{M' \mid \exists M \in \mathcal{M}_{in}$ *such that for all* $i \in [n] : \overline{M'}(i) = (\overline{M}(i), S_i)\}$. □

This construction resembles the one used in [KKR91] to prove that regular languages can be incorporated into GITNCs. Now however, the situation is more complicated because here we incorporate grammars.

Before comparing the vector languages of $inc(\Gamma)$ and Γ, we observe that the incorporation construction indeed yields a GITNC.

Lemma 3. $inc(\Gamma)$ *is a GITNC.*

Proof. To prove the statement we have to show that the transitions of N' are 1-throughput (conditions 1 and 2 in Definition 1), that the vector labelling of the transitions is consistent (Definition 3), and that $\mathcal{M}'_{in}$ and $\mathcal{M}_{fin}$ consist of it-markings of N' (Definition 4).
The last requirement is clearly fulfilled, since $\mathcal{M}_{fin}$ is not changed and each element of $\mathcal{M}'_{in}$ is defined by specifying per token the place where it resides

and hence is an it-marking. The vector labelling is consistent, because in N the transitions are consistently labelled and all productions introduce a terminal symbol (there are no λ-productions). Finally, N' has the same transitions as N has and the flow function F' is derived from F. They only differ where the flow of a token i from (to) a place p to (from) a transition t according to F is redirected by F' from (to) a place (p, A) to (from) t where A is uniquely determined by the production in the i-th entry of $l(t)$. Hence, like N, also N' is 1-throughput. □

Now we are ready to prove that $inc(\Gamma)$ defines the same vector language as Γ.

Lemma 4. $V(inc(\Gamma)) = V(\Gamma)$.

Proof. To prove that $V(\Gamma) \subseteq V(inc(\Gamma))$ we relate the configurations of Γ and their transformations to it-markings and the firing of transitions in N'. With each configuration $\gamma = (v_1, \ldots, v_n; M')$ of Γ we associate an it-marking M_γ of N' defined as follows. $\overline{M_\gamma}(i) = \overline{M'}(i)$ for all $i \in [n]$ such that v_i is a terminal string, that is, in this case M_γ has token i in the same place as M'. For all $i \in [n]$, such that $v_i = xA$ for some $x \in \Delta_i^*$ and $A \in \Sigma_i$, token i is assigned to the A-copy of the place where M' puts it, thus $\overline{M_\gamma}(i) = (\overline{M'}(i), A)$. As a consequence, M_γ is an initial it-marking of $inc(\Gamma)$ whenever γ is an initial configuration of Γ. Furthermore, M_γ is a final it-marking of $inc(\Gamma)$ if and only if γ is a final configuration of Γ.

We first prove that a transformation $\gamma \vdash^t \delta$ implies that $M_\gamma[t\rangle M_\delta$ in N'. Let $\gamma = (v_1, \ldots, v_n; M_1)$ and $\delta = (w_1, \ldots, w_n; M_2)$ and let $t \in T$ be such that $\gamma \vdash^t \delta$. Let i be an input token of t. Thus $i \in F'(p', t)$ for some $p' \in P'$. Since in N' places from P are never input to any transition, this implies that $p' = (p, A)$ for some $p \in P$ and $A \in \Sigma_i$. From the definition of F' it then follows that $i \in F(p, t)$ and $lhs(proj_i(l(t))) = A$. Since t can occur at γ in Γ, we deduce that $\overline{M_1}(i) = p$ and that the last symbol of v_i is an A. Hence $\overline{M_\gamma}(i) = (p, A) = p'$. We conclude that $F'(p', t) \subseteq M_\gamma(p')$ for all $p' \in P'$ and so t has concession in N' in the it-marking M_γ. Now, let M be the it-marking such that $M_\gamma[t\rangle_{N'} M$. We prove that $M = M_\delta$. Let i be a token used by t in N'. Thus $i \in F'(p', t)$ for some $p' \in P'$. Hence, as above, there is $p \in P$ such that $i \in F(p, t)$, and there is $A \in \Sigma_i$ such that $p' = (p, A)$, $lhs(proj_i(l(t))) = A$, and $v_i = xA$ for some $x \in \Delta_i^*$. Let $q \in P$ be the output place of t in N for token i, that is $\overline{M_2}(i) = q$. The occurrence of t at γ implies that v_i is rewritten using the production $proj_i(l(t))$. This production is either $A \to aB$ or $A \to a$ for some $a \in \Delta_i$ and $B \in \Sigma_i$. If $proj_i(l(t)) = A \to aB$, then $w_i = xaB$ and the definition of F'' implies that $\overline{M}(i) = (q, B) = \overline{M_\delta}(i)$. If $proj_i(l(t)) = A \to a$, then $w_i = xa$ and the definition of F'' implies that $\overline{M}(i) = q = \overline{M_\delta}(i)$. A token i which is not used by t in N' is also not used by t in N and so v_i is not rewritten if t occurs. Consequently, $\overline{M}(i) = \overline{M_\gamma}(i) = \overline{M_\delta}(i)$. We conclude that $M = M_\delta$.

Consider a word vector $(w_1, \ldots, w_n) \in V(\Gamma)$. Hence there is a successful run $\gamma_0 \vdash^{t_1} \cdots \vdash^{t_k} \gamma_k$ of Γ such that γ_0 is an initial configuration of Γ and γ_k is a final configuration of Γ. From the above we know that $M_{\gamma_0}[t_1\rangle \cdots [t_k\rangle M_{\gamma_k}$ in N', M_{γ_0} is an initial it-marking of $inc(\Gamma)$, and M_{γ_k} is a final it-marking of $inc(\Gamma)$. Thus

$t_1 \cdots t_k \in FS(inc(\Gamma))$. Let $i \in [n]$. In each transformation step $\gamma_{j-1} \vdash^{t_j} \gamma_j$, either the i-th string is not rewritten, in which case $proj_i(l(t_j)) = proj_i(l'(t_j)) = \lambda$, or it is rewritten according to the production $proj_i(l(t_j))$ which adds the terminal symbol $sym(proj_i(l(t_j))) = proj_i(l'(t))$ to the right of the terminals in the i-th string. So, after the successful sequence of transformations $\gamma_0 = (S_1, \ldots, S_n; M'_0) \vdash^{t_1} \cdots \vdash^{t_k} (w_1, \ldots, w_n; M'_k) = \gamma_k$, we have $w_i = proj_i(l'(t_1)) \cdots proj_i(l'(t_k))$ and hence $(w_1, \ldots, w_n) = coll(l'(t_1) \cdots l'(t_k)) \in V(inc(\Gamma))$.

To prove that $V(inc(\Gamma)) \subseteq V(\Gamma)$ we consider once more the relation between the configurations of Γ and the it-markings of N'. When associating to a configuration $\gamma = (v_1, \ldots, v_n; M')$ of Γ the it-marking M_γ of N', the terminal parts of the v_i are not considered. Hence each it-marking M of N' is related to infinitely many configurations of Γ. However, as we show next, every firing sequence of $inc(\Gamma)$, determines a unique sequence of transformations in Γ in such a way that the it-markings encountered in the firing sequence are associated to the respective configurations in the transformation sequence.
Let $k \geq 0$, let $M_0 \in \mathcal{M}'_{in}$, and let $M_0[t_1\rangle \cdots [t_k\rangle M_k$ in N' for some transitions $t_1, \ldots, t_k$ and it-markings $M_1, \ldots, M_k$. Set $\gamma_0 = (S_1, \ldots, S_n; M'_0)$, with for all $i \in [n]$, $\overline{M'_0}(i) = p$ whenever $\overline{M_0}(i) = (p, S_i)$. Since M_0 is an initial it-marking of $inc(\Gamma)$, this is well defined and γ_0 is an initial configuration with M_0 as its associated it-marking. Let $1 \leq j \leq k$ and assume that $\gamma_{j-1} = (v_1, \ldots, v_n; M'_{j-1})$ has already been defined and is such that M_{j-1} is the it-marking of N' associated to γ_{j-1}. We first show that t_j can occur at γ_{j-1} in Γ. Let i be an input token for t_j in N. Hence $i \in F(p, t_j)$ for some $p \in P$. The definition of F' implies that i is also an input token for t_j in N'. In particular, there is a non-terminal $A \in \Sigma_i$ such that $i \in F'((p, A), t_j)$. Since t_j has concession in M_{j-1}, it follows that $\overline{M_{j-1}}(i) = (p, A)$. Combining this with the fact that M_{j-1} is the it-marking associated to γ_{j-1}, we obtain that $\overline{M'_{j-1}}(i) = p$ and $v_i = xA$ for some $x \in \Delta_i^*$. Since t_j uses i, the i-th entry of its vector label in N is not empty and we have $proj_i(l(t_j)) \in \Pi_i$. From the definition of F' it follows that $lhs(proj_i(l(t_j))) = A$ and hence v_i can be rewritten using this production. Consequently, t_j has concession in M_{j-1} and the productions in its vector label can be applied to the corresponding v_i and so t_j can occur at γ_{j-1}. We define γ_j to be the configuration obtained by the transformation of γ_{j-1} by the occurrence of t_j. Thus $\gamma_{j-1} \vdash^{t_j} \gamma_j$. We show that M_j is the it-marking of N' associated to γ_j. Let i be an input token of t_j in N. Then $v_i = xA$ for some $x \in \Delta_i^*$ and $A \in \Sigma_i$ is the left-hand side of the production $\pi = proj_i(l(t_j)) \in \Pi_i$. If π is a terminating production $A \to a$ with $a \in \Delta_i$, then v_i is rewritten into the terminal string xa. By the construction, $i \in F'(t_j, q)$, where q is the output place of t_j in N such that $i \in F(t_j, q)$. Thus, $\overline{M_j}(i) = \overline{M'_j}(i)$ as required. If $\pi = A \to aB$ with $a \in \Delta_i$ and $B \in \Sigma_i$, then v_i is rewritten into xaB. By the construction, $i \in F'(t_j, (q, B))$. Thus, $\overline{M_j}(i) = (\overline{M'_j}(i), B)$ as required. If i is not an input token of t_j in C, then the occurrence of t_j does not affect i and it entails no rewriting of v_i. Hence nothing is changed with respect to the i-th component: $\overline{M'_j}(i) = \overline{M'_{j-1}}(i)$ and v_i is not changed. In N' token i is also not used by t_j and $\overline{M_j}(i) = \overline{M_{j-1}}(i)$. Consequently, also in this case $\overline{M'_j}(i)$ satisfies the requirements.

Now consider a word vector $(w_1, \ldots, w_n) \in V(inc(\Gamma))$. Hence there is a firing sequence $t_1 \cdots t_k$ with $t_j \in T$ for all $1 \leq j \leq k$, and there are it-markings $M_0, \ldots, M_k$ of N' where M_0 is initial and M_k is final, such that $M_0[t_1\rangle \cdots [t_k\rangle M_k$ and $(w_1, \ldots, w_n) = coll(l'(t_1) \cdots l'(t_k))$. From the above we deduce that in Γ, there are configurations $\gamma_0, \ldots, \gamma_k$ such that $\gamma_0 \vdash^{t_1} \cdots \vdash^{t_k} \gamma_k$ and M_j is the associated it-marking of γ_j for each $0 \leq j \leq k$. Thus γ_0 is an initial configuration of Γ and $\gamma_k = (v_1, \ldots, v_n; M'_k)$ is a final configuration of Γ. Hence $(v_1, \ldots, v_n) \in V(\Gamma)$. Since $(v_1, \ldots, v_n) = coll(l'(t_1) \cdots l'(t_k)) = (w_1, \ldots, w_n)$ it follows that $(w_1, \ldots, w_n) \in V(\Gamma)$. □

Using this result we can prove

Lemma 5. $\mathcal{V}_n(regPNGS) \subseteq \mathcal{V}_n(GITNC)$.

Proof. For the λ-free regular PN grammar systems, the result follows from the incorporation construction and Lemma 4. In case Γ is a regular n-PNGS with λ-productions, each production $A \to \lambda$ is replaced by a production $A \to \$$ where $\$$ is a new symbol. Then the incorporation construction is applied. The resulting GITNC defines a regular language L over a vector alphabet in which some vectors have entries $\$$ instead of λ. By Lemma 4, the vector language $coll(L)$ coincides with $V(\Gamma)$ once all occurrences of $\$$ have been replaced by λ. Regular languages are closed under (arbitrary) homomorphisms and so we can define a homomorphism h which maps every vector letter to the corresponding vector letter which has λ instead of $\$$ and the resulting language $h(L)$ will also be regular. Thus $V(\Gamma) = coll(h(L)) \in n\mathcal{RAT} = \mathcal{V}_n(GITNC)$ by Theorem 1. □

Combining Lemmas 2 and 5 with Theorem 1 shows that the vector languages of regular n-PNGSs coincide with the n-dimensional rational relations.

Theorem 5. $\mathcal{V}_n(regPNGS) = \mathcal{V}_n(GITNC) = n\mathcal{RAT}$. □

From this result we can deduce that $proj_1(\mathcal{V}_1(regPNGS)) = \mathcal{L}_1(REG)$ (cf. Theorem 4). This is consistent with the well-known fact that adding regular control to regular grammars does not lead to an increase of the generative power, and contrasts with the effect of adding regular control to context-free grammars.

6 Conclusion

The theory of grammar systems is concerned with the study and development of grammatical models for distributed computations and cooperation. By viewing the grammars in a grammar system as concurrently operating partners subject to certain synchronization constraints, a link has been established with the theory of Vector Controlled Concurrent Systems. This has made it possible to use the Petri net based vector control mechanism of GITNCs to model concurrent rewriting protocols. This paper is clearly no more than an introduction proposing a definition and establishing some basic facts. A lot of work is still to be done before concurrent rewriting protocols for grammar systems and in particular the possibilities of PN grammar systems are fully understood.

A topic worth investigating is the optimalization of derivations on basis of both the structure of the controller and the form of the productions. Different criteria can be used here, like minimizing the number of steps in a derivation sequence, avoiding delays for component grammars, minimizing the number of synchronization partners in the vector labels, and reducing the size of the controller.
Also the connections between PN grammar systems and other models for regulated rewriting deserve more study. In a GITNC a transition can occur if the non-terminals forming the left-hand sides of the productions in its vector label occur in the sentential forms to be rewritten. This makes it difficult if not impossible to implement in the current PN grammar systems the appearance checking often used in regulated rewriting. For Petri nets, inhibitor arcs are an extension well suited to model situations in which a certain condition has to be tested (see, e.g., [Pet81]). Perhaps PN grammar systems with GITNCs extended with inhibitor arcs have sufficient modelling power to simulate appearance checking.
PN grammar systems are appealingly close to PC grammar systems. A feature of PC grammar systems which the PN grammar systems lack, is the possibility to communicate through queries, but also bounded communication and non-communicating variants have been studied ([CDKP94], [IP94]). The rewriting in a PC grammar system is synchronized in the sense that at each step of the derivation process, all grammars with a not yet terminal sentential form have to apply a production. There are variants of the model in which the synchronizations are subject to additional constraints like rules synchronization which only allows fixed combinations of productions in rewriting steps (see, e.g., [CV01]). In [Mit00], PC finite automata systems are considered with synchronizations between state-action transitions resembling the synchronizations in regular PN grammar systems. Furthermore PC grammar systems with components not necessarily regular or context-free have been investigated. Hence insight in properties of PN grammar systems may come from results obtained for variants of the PC grammar systems model. On the other hand dynamic (state-dependent) and concurrent control on derivations as exercised by a GITNC is not part of PC grammar systems. In this sense, the introduction of PN grammar systems may be perceived as broadening the spectrum of the study of grammatical models for cooperation.

References

[Ber79] J. Berstel, *Transductions and Context-Free Languages*, Teubner, Stuttgart, 1979.

[Cor91] D.D. Corkill, Blackboard Systems. *AI Expert* 6, 9 (1991), 40-47.

[CD90] E. Csuhaj-Varjú and J. Dassow, On Cooperating Distributed Grammar Systems. *J. Inf. Process. Cybern. EIK* 26 (1990), 49-63.

[CDKP94] E. Csuhaj-Varjú, J. Dassow, J. Kelemen and Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[CK89] E. Csuhaj-Varjú and J. Kelemen, Cooperating Grammar Systems: A Syntactical Framework for the Blackboard Model of Problem Solving. In *Proc. AI*

and Information-Control Systems of Robots '89 (I. Plander, Ed.), North-Holland Publ. Co., 1989, 121-127.

[CV01] E. Csuhaj-Varjú and Gy. Vaszil, On Context-Free Parallel Communicating Grammar Systems: Synchronization, Communication, and Normal Forms. *Theoretical Computer Science* 255 (2001), 511-538.

[CL92] N. Carver and V. Lesser, The Evolution of Blackboard Control Architectures. In *Expert Systems with Applications, Special Issue on The Blackboard Paradigm and Its Applications* 7, 1 (1994), 1-30.

[DPR97] J. Dassow, Gh. Păun and G. Rozenberg, Grammar Systems. In [RS97] vol. 2 (1997), 155-213.

[DPS97] J. Dassow, Gh. Păun and A. Salomaa, Grammars with Controlled Derivations. In [RS97] vol. 2 (1997), 101-154.

[IP94] C.-M. Ionescu and O. Procopiuc, Bounded Communication in Parallel Communicating Grammar Systems. *J. Inf. Process. Cybern. EIK* 30 (1994), 97-110.

[Kee96] N.W. Keesmaat, *Vector Controlled Concurrent Systems*. Ph.D. thesis, Leiden University, 1996.

[KK97] N.W. Keesmaat and H.C.M. Kleijn, Net-Based Control versus Rational Control: The Relation between ITNC Vector Languages and Rational Relations. *Acta Informatica* 34 (1997), 23-57.

[KKR90] N.W. Keesmaat, H.C.M. Kleijn and G. Rozenberg, Vector Controlled Concurrent Systems, Part I: Basic Classes. *Fundamenta Informaticae* 13 (1990), 275-316.

[KKR91] N.W. Keesmaat, H.C.M. Kleijn and G. Rozenberg, Vector Controlled Concurrent Systems, Part II: Comparisons. *Fundamenta Informaticae* 14 (1991), 1-38.

[MR78] R. Meersman and G. Rozenberg, Cooperating Grammar Systems. *Lect. Notes in Comp. Sci.* 64 (1978), Springer-Verlag, 364-374.

[MRV78] R. Meersman, G. Rozenberg and D. Vermeir, Cooperating Grammar Systems. *Techn. Report 78-12*, Univ. Antwerp, Dept. Math., 1978.

[Mit00] V. Mitrana, On the Degree of Communication in Parallel Communicating Finite Automata Systems. *J. Automata, Languages and Combinatorics* 5 (2000), 301-314.

[Nii89] P.H. Nii, Blackboard Systems. In *The Handbook of Artificial Intelligence* vol. 4 (A. Barr, P.R. Cohen and E.A. Feigenbaum, Eds.), Addison-Wesley, 1989.

[Pet81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981.

[PS89] Gh. Păun and L. Santean, Parallel Communicating Grammar Systems: The Regular Case. *A. Univ. buc., Ser. Matem.-Inform.* 38 (1989), 55-63.

[RR98] *Lectures on Petri Nets I: Basic Models* (W. Reisig and G. Rozenberg, Eds.) Lect. Notes in Comp. Sci. 1491, Springer-Verlag, 1998.

[RS97] *Handbook of Formal Languages* (G. Rozenberg and A. Salomaa, Eds.), Springer-Verlag, 1997.