Theoretical Computer Science 44 (1986) 275-305
North-Holland

NET PROCESSES CORRESPOND TO DERIVATION PROCESSES IN GRAPH GRAMMARS*

Hans-Jörg KREOWSKI and Anne WILHARM

Department of Mathematics and Computer Science, Universität Bremen, D-2800 Bremen 33, Fed. Rep. Germany

Communicated by E. Engeler
Received April 1985
Revised August 1985

Abstract. The aim of this paper is to compare the running behaviour of Petri nets, given by firing sequences and processes, with derivations and derivation processes in graph grammars. In a first step, Petri nets are simulated by graph grammars so that each firing in a net corresponds exactly to a direct derivation in the simulating graph grammar. In a second step the non-sequential behaviour of nets described by net processes is related to the non-sequential behaviour of graph grammars given by derivation processes. A one-to-one correspondence can be established between the processes on a Petri net and the complete conflict-free processes in the graph grammar simulating the net. This adds a new piece of evidence substantiating the close relationship between net and graph grammar theory.

Introduction

In computer science one deals with systems and their sequential and non-sequential behaviour, frequently described as processes. This requires to study various questions: Which manipulations can be performed in parallel! Which actions influence or exclude each other and must be synchronized? Which activities can be organized without deadlocks and inconsistencies? etc.

Because such problems are important and occur in several different areas, there are many approaches claiming to describe and to solve some of these problems. Among them you can find Petri nets and graph grammars which both use graphical notations to investigate systems and their behaviour.

Some work has already been done in comparing both theories (cf., e.g., [2, 4, 8, 11, 12]). In particular, in [4] a graph grammar is constructed from a given place/transition net so that direct derivations in the graph grammar correspond to firings of transitions in the net. Generalizing this simulation to arc-weighted nets with capacity, the present paper establishes a one-to-one correspondence between

* A short version of this paper was presented at the Workshop on Graph-theoretic Concepts in Computer Science, 1984 (see [6]).

processes on a Petri net and complete, conflict-free processes in the graph grammar simulating the net. Intuitively speaking, the notion of a process in both approaches relates actions in a concurrent or distributed system by a partial order in contrast to the assumption that changes happen one after the other as in a sequential system.

This paper consists of two parts. The first part concerns the sequential behaviour of systems described by Petri nets and graph grammars. For this purpose we shall recall the basic notions of Petri nets (Section 1) and of graph grammars (Section 2). Then a net will be simulated by a graph grammar (Section 3) in the way that graphs and a graph grammar rules are obtained from a given arc-weighted net with capacity. It will be shown that there is a one-to-one correspondence between the sequential behaviour of a Petri net and the sequential behaviour of the simulating graph grammar, i.e., between firing sequences and derivations.

In the second part of this paper the non-sequential behaviour of systems described by nets and graph grammars is investigated. For that reason the concepts of net processes, specialized to condition/event systems (Section 4), and of derivation processes in graph grammars (Section 5) will be recalled. The main result (Section 6) establishes a one-to-one correspondence between processes on a net and complete, conflict-free processes in the graph grammar simulating the net. Section 7 will present some concluding remarks. The concepts of graph theory used in this paper are piled up in Appendix A.

1. Petri nets

In this section we recall some basic notions of net theory (cf., e.g., [9]).

The underlying invariant structure is a net, which is a graph where the set of nodes is bi-partitioned into places and transitions.

Each place of the net may store tokens, the number of which is unbounded or bounded by a given capacity.

Transitions represent actions that can take place in the net. They are connected with places by the flow relation. For each edge of the net, i.e., an element of the flow relation, the width denotes the number of tokens flowing through this edge if its transition fires.

Definition 1.1. A *net* $N = (S, T, F, w, k)$ consists of
- a set S of *places*,
- a set T of *transitions* with $S \cap T = \emptyset$,
- a *flow* relation $F \subseteq S \times T \cup T \times S$,
- a *width* function $w : F \to \mathbb{N}$, and
- a *capacity* function $k : S \to \mathbb{N} \cup \{\infty\}$.

Remark. (1) For technical reasons we extend w to the *extended-width* function

$\bar{w}: S\times T \cup T\times S \to \mathbb{N}$ in the following way:

$$\bar{w}(a, b) := \begin{cases} w(a, b) & \text{for all } (a, b) \in F, \\ 0 & \text{otherwise.} \end{cases}$$

(2) The *input places* of a transition $t \in T$ are given by the set $^*t = \{s \in S \mid (s, t) \in F\}$, and the *output places* of a transition $t \in T$ are given by the set $t^* = \{s \in S \mid (t, s) \in F\}$.

(3) We say that $s \in S$ is a place with *bounded capacity* if $k(s) \neq \infty$. Otherwise, s is a place with *unbounded capacity.*

All components of a net are static, i.e., they cannot be changed. To obtain a dynamic system a net is equipped with a marking assigning a number of tokens to each place of the net up to its capacity.

The behaviour of the system will be desribed by the transformation of markings.

Definition 1.2. Let $N = (S, T, F, w, k)$ be a net. A function $m: S \to \mathbb{N}$ is called *marking* of N if $m(s) \leq k(s)$ for all $s \in S$ with $k(s) \neq \infty$.

Remark. Let $N = (S, T, F, w, k)$ be a net $m: S \to \mathbb{N}$ be a marking. A *slot* function $\mathrm{sl}: S \to \mathbb{N}$, indicating the free capacity of a place with bounded capacity, is defined in the following way:

$$\mathrm{sl}(s) := \begin{cases} k(s) - m(s) & \text{for all } s \in S \text{ with } k(s) \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1.3. Let N be a net, and let m be a marking of N. Then the pair (N, m) is called a *marked net.*

Example 1.4. Let example $= (S, T, F, w, k)$ be the net with $S = \{1, 2, 3, 4, 5\}$, $T = \{a, b, c, d, e\}$, $F = \{(1, a), (a, 2), (4, b), (b, 5), (2, c), (c, 3), (3, d), (5, d), (d, 1), (d, 4), (1, e), (4, e), (e, 3), (e, 5)\}$, $w(e) = 1$ for all $e \in F$ and $k(s) = 1$ for all $s \in S$ and let $\mathrm{init}: S \to \mathbb{N}$ be a marking with $\mathrm{init}(1) = \mathrm{init}(4) = 1$ and $\mathrm{init}(2) = \mathrm{init}(3) = \mathrm{init}(5) = 0$. Then EXAMPLE = (example, init) is a marked net and can be drawn as usual in net theory (see Fig. 1) (confer, e.g., [9]).

The marking of a net can be transformed into a follower marking by firing enabled transitions. Transitions are enabled if all their input places carry at least as many tokens as required by the width of the connecting edges and if analogously, all their output places with bounded capacity have enough slots. Transitions transform a marking into a follower marking by decreasing the number of tokens of each input place and by increasing the number of tokens of each output place both according to the width. We do not only allow that different transitions are fired simultaneously, but also that each transition may be fired several times within one transformation. To make this precise, a frequency is assigned to each transition so that a bag of transitions rather than a set of transitions is involved in a transformation.

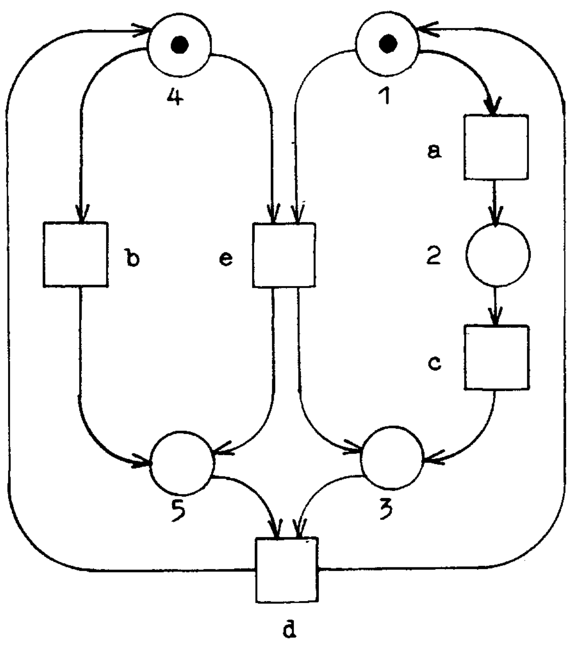

Fig. 1. EXAMPLE.

Definition 1.5. Let $N = (S, T, F, w, k)$ be a net and m be a marking of N. Let $U : T \to \mathbb{N}$ be a function.

(1) U is *enabled* under m if

$$m(s) \geq \sum_{t \in T} U(t) \times \bar{w}(s, t) \qquad \text{for all } s \in S, \text{ and}$$

$$\text{sl}(s) = k(s) - m(s) \geq \sum_{t \in T} U(t) \times \bar{w}(t, s) \quad \text{for all } s \in S \text{ with } k(s) \neq \infty.$$

(2) In this case U *transforms* m to m' defined by

$$m'(s) = m(s) + \sum_{t \in T} U(t) \times [\bar{w}(t, s) - \bar{w}(s, t)] \quad \text{for all } s \in S.$$

Remark. (1) This is called a *step* from marking m to m' by firing U and denoted by $m[U\rangle m'$, and m' is called *follower marking* of m (under U).

(2) A function $U : t \to \mathbb{N}$ is called a *bag of transitions*. It should be noted that some authors call such a function a multiset. For a transition t, $U(t)$ is the frequency t is fired with in a step.

(3) The case that a single transition t_0 is fired is included in our notion by means of the following bag U_{t_0} defined by $U_{t_0}(t_0) = 1$ and $U_{t_0}(t) = 0$ otherwise. In this case we write $m[t_0\rangle m'$ instead of $m[U_{t_0}\rangle m'$.

(4) The transitions involved in a step are said to be *concurrent*.

(5) $[*\rangle$ denotes the reflexive, transitive closure of $[\ \rangle$. Let m and m' be markings of a net N; m' is called *reachable* from m if $m[*\rangle m'$.

Example 1.6. Let EXAMPLE be the marked net from Example 1.4. Let $U : T \to \mathbb{N}$ be the bag of transitions with $U(a) = U(b) = 1$ and $U(c) = U(d) = U(e) = 0$.

U is enabled under init and transforms init to the follower marking follow where follow is defined by follow(2) = follow(5) = 1 and follow(1) = follow(3) = follow(4) = 0. The transitions a and b are concurrent in the step init[U⟩follow.

Figure 2 illustrates the effect of the step init[U⟩follow.

Later on we shall make use of the well-known fact that transitions which can be fired simultaneously can also be fired one after the other with the same resulting marking.

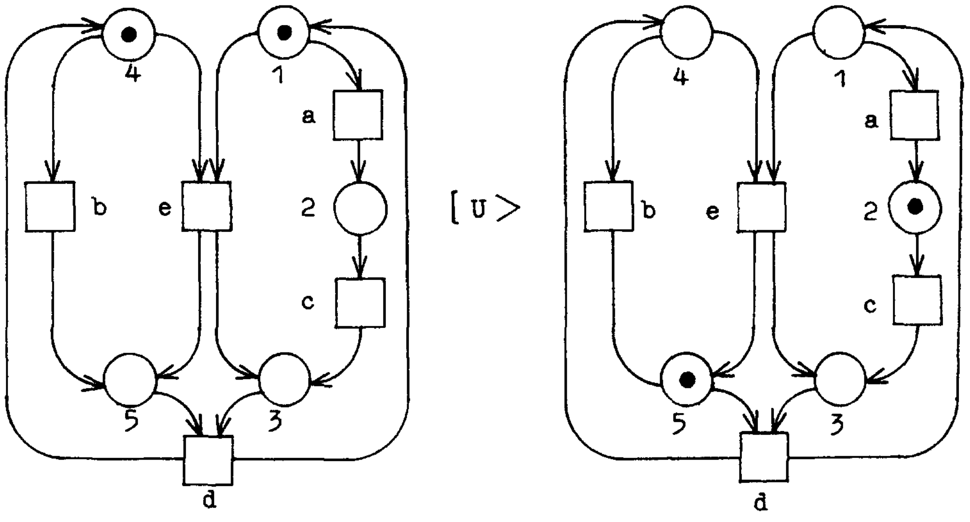

Fig. 2. init[u⟩follow.

Theorem 1.7. *Let N be a net. Let U, U_1, and U_2 be bags of transitions with $U(t) = U_1(t) + U_2(t)$ for all $t \in T$. Let m and m'' be markings with $m[U\rangle m''$. Then there is a marking m' such that $m[U_1\rangle m'[U_2\rangle m''$.*

Remark. $m[U_1\rangle m'[U_2\rangle m''$ will be called a *decomposition* of $m[U\rangle m''$.

Example 1.8. For the step init[U⟩follow in Example 1.6, there are two proper decompositions init[$a\rangle m'[b\rangle$follow as well as init[$b\rangle m''[a\rangle$follow, where m' is given by $m'(2) = m'(4) = 1$ and $m'(1) = m'(3) = m'(5) = 0$, and m'' is given by $m''(1) = m''(5) = 1$ and $m''(2) = m''(3) = m''(4) = 0$.

2. Graph grammars

In this section we specialize and adapt the basic notions of graph grammar rules and derivations (cf., e.g., [3, 1]) to our purposes. The procedure of deriving graphs from graphs is based on a specific kind of graph removing and gluing. Note that all notions, constructions, and notational conventions concerning graphs are summarized in the appendix.

An alternative approach to graph rewriting can be found, e.g., in Nagl's book [7].

Definition 2.1. A *rule* $r = (L, R, K)$ consists of three graphs L, R, and K with $K \subseteq L$ and $K \subseteq R$.

Remark. (1) L is called the *left-hand side* of r, R the *right-hand side* of r, and K the *gluing graph* of r.

(2) $r^{-1} = (R, L, K)$ is called the *inverse rule* of r.

To apply a rule $r = (L, R, K)$ to a graph M, four steps must be performed:

(1) choose an occurrence of L in M,

(2) check the gluing condition,

(3) remove the non-gluing part of the occurrence,

(4) add to the remainder the non-gluing part of R by gluing together corresponding gluing parts.

Definition 2.2. Let $r = (L, R, K)$ be a rule. Let M and N be graphs. Let $g: L \to M$ and $h: R \to N$ be graph morphisms.

(1) The image $g(L)$ is called a *(left-)occurrence* of r in M. The image $h(R)$ is called a *right-occurrence* of r in N.

(2) A graph morphism g is a *valid occurrence map* if the following conditions are satisfied:

(a) $s_M(e) \in g_V(V_L)$ implies $s_M(e) \in g_V(V_K)$, and $t_M(e) \in g_V(V_L)$ implies $t_M(e) \in g_V(V_K)$ for all $e \in E_M - g_E(E_L)$;

(b) $x \neq y$, but $g(x) = g(y)$ implies $x, y \in K$ for all $x, y \in L$.

Remark. (1) The graph morphisms g and h are also called *occurrence maps*.

(2) The images $g(K)$ and $h(K)$ are called the *gluing parts*, $g(L) - g(K)$ and $h(R) - h(K)$ the *non-gluing parts* of the occurrences.

(3) Part (a) of Definition 2.2(2) is called *contact condition*, because it requires that edges from outside of the occurrence contact the gluing part at most.

(4) Part (b) is called *identification condition*, because it makes sure that the non-gluing part of the occurrence is not deformed.

(5) Contact condition and identification condition together are called *gluing condition*.

Lemma 2.3. *Let $r = (L, R, K)$ be a rule and let M be a graph. Let $g: L \to M$ be a graph morphism.*

(1) *$M - (g(L) - g(K))$ induces a subgraph of M, denoted by* REM, *if and only if g satisfies the contact condition.*

In addition, let g satisfy the contact condition.

(2) *Then, $d: K \to$ REM given by $d(x) = g(x)$ for all $x \in K$ is a graph morphism.*

(3) *Moreover, M is isomorphic to the d-gluing of L and* REM *along K provided that g satisfies the identification condition, too.*

Remark. REM is called the *remainder* (of M according to g and r).

Definition 2.4. Let $r=(L, R, K)$ be a rule. Let M, N be graphs, and let $g:L\rightarrow M$ specify a valid occurrence map. Let REM be the remainder of M according to g and r with the corresponding graph morphism $d:K\rightarrow \text{REM}$ (see Lemma 2.3). Let GLUE be the d-gluing of R and REM along k. Then *M directly derives N by applying r (according to g)* if N is isomorphic to the graph GLUE.

Remark. (1) We write $M\Rightarrow N$ through r (and g) or $M\Rightarrow_r N$ (and call this a *direct derivation*) if M directly derives N by applying r according to g.

(2) This defines a relation $\Rightarrow$ on graphs, the reflexive and transitive closure of which is denoted by $\overset{*}{\Rightarrow}$. A graph N is said to be *derivable* from M if $M\overset{*}{\Rightarrow}N$.

(3) A sequence of direct derivations $M_0\Rightarrow_{r_1}M_1\Rightarrow_{r_2}\cdots\Rightarrow_{r_n}M_n$, where $M_{i-1}\Rightarrow_{r_1}M_i$ is a direct derivation for $i=1,\ldots,n$, is called a *derivation* from M_0 to M_n. Obviously, M_n is derivable from M_0 in this case.

(4) Let $i:\text{GLUE}\rightarrow N$ be the assumed isomorphism. Let $j:R\rightarrow \text{GLUE}$ be the graph morphism according to Appendix A(9). Then the composition $i\circ j$ defines an occurrence map $h:R\rightarrow N$. Due to the identification condition the following is true: $M\Rightarrow N$ through r and g if and only if $N\Rightarrow M$ through r^{-1} and h.

Figure 3 sketches the relation of the graphs involved in a direct derivation $M\Rightarrow N$ through $r=(L, R, K)$ and $g:L\rightarrow M$.

Now we allow to derive a graph from a given graph not only by applying one rule but by applying a parallel rule which is a combination of given rules.

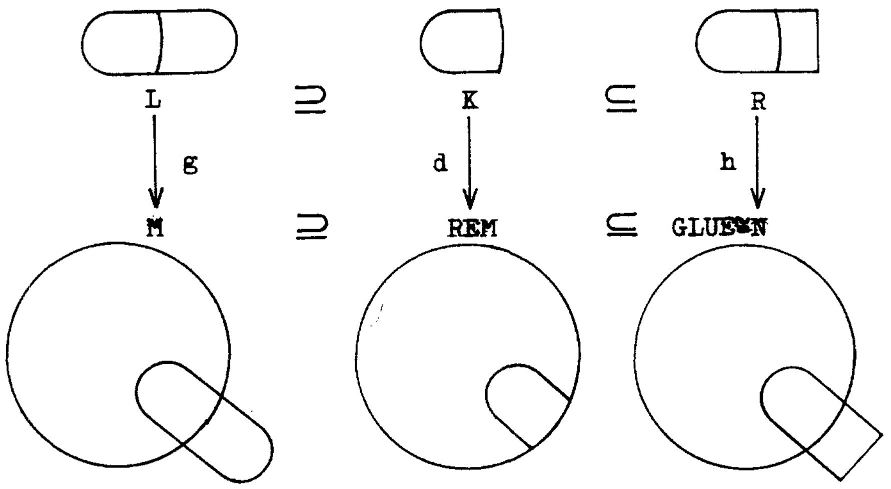

Fig. 3. Scheme of a direct derivation.

Definition 2.5. Let $r_i=(L_i, R_i. K_i)$ for $i=1,\ldots,n$ $(n\geq 1)$ be rules, and let P be a set of rules.

(1) The *parallel rule* $r=r_1+\cdots+r_n$ is given by the disjoint union of the components of r_i, i.e., $r=(L_1+\cdots+L_n, R_1+\cdots+R_n, K_1+\cdots+K_n)$.

(2) r is said to be a *parallel rule over P* if $r_i\in P$ for $i=1,\ldots,n$.

P^+ denotes the set of parallel rules over P.

Remark. (1) For $r = r_1 + \cdots + r_n$ we shall write $r = \sum_{i=1}^{n} r_i$, too. Instead of $r + \cdots + r$ (n-times) we shall write $n \times r$ for short.

(2) The disjoint union of rules defining parallel rules is commutative and associative (up to isomorphism). Hence, P^+ may be given by the following recursion: $P \in P^+$ and $r_1 + r_2 \in P^+$ provided that $r_1, r_2 \in P^+$.

(3) Parallel rules are ordinary rules, so that they can be applied to graphs in the way defined above. A direct derivation $M \Rightarrow N$ through a parallel rule $r_1 + \cdots + r_n$ is called *direct parallel derivation.*

Example 2.6. Figure 4 explicitly shows a direct parallel derivation from a graph called GRAPH(init) to a graph called GRAPH(follow) through a parallel rule $r_U = r_a + r_b$.

The following theorem shows that parallel derivations can always be sequentialized and states conditions under which direct derivations can be parallellized. Hence, the use of parallel rules may reduce the length of a derivation sequence, but cannot

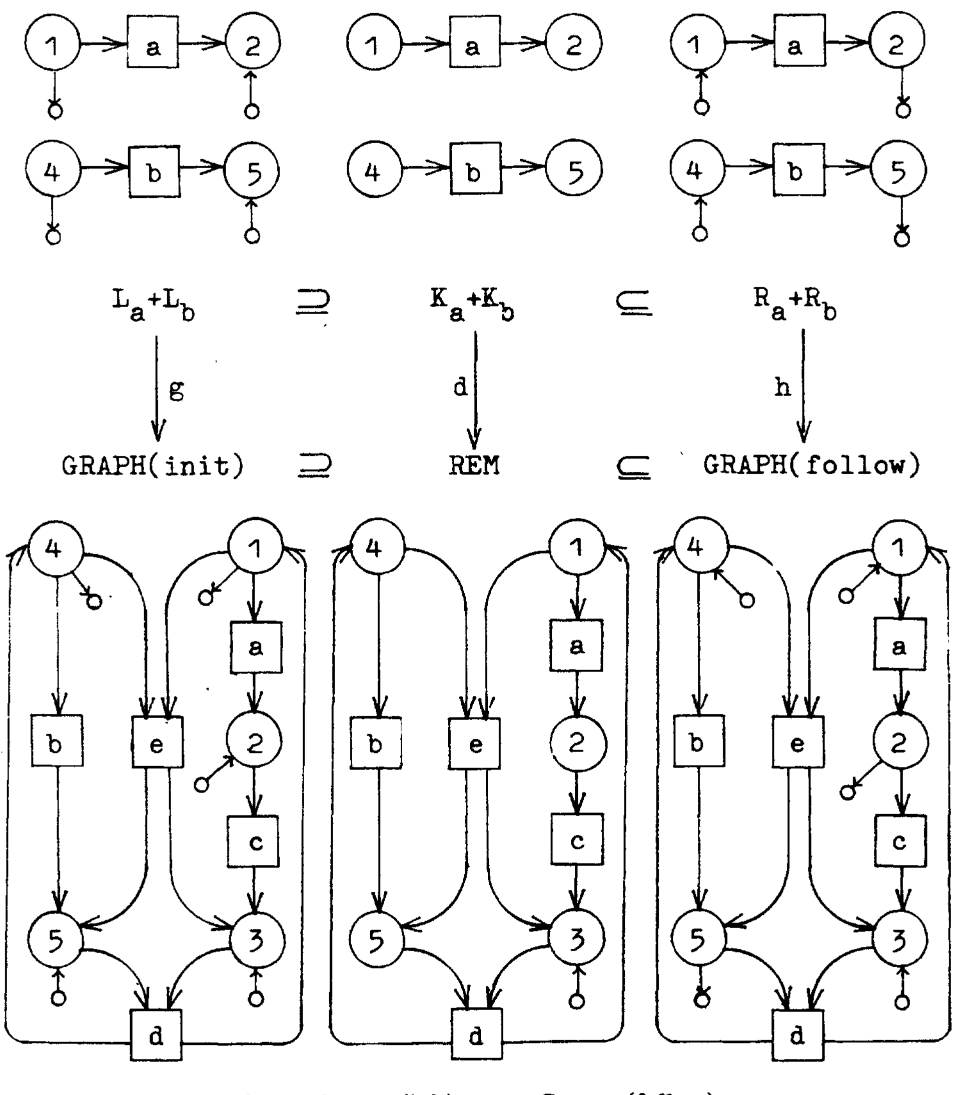

Fig. 4. GRAPH(init)$\Rightarrow_{r_a+r_b}$ GRAPH(follow).

increase the generative power. For the formulation of this result we introduce the notion of independency.

Definition 2.7. (1) Let $M \Rightarrow_{r_1} N \Rightarrow_{r_2} X$ be a derivation sequence. Let $h_1 : R_1 \to N$ be a right-occurrence map of the first derivation and let $g_2 : L_2 \to N$ be a left-occurrence map of the second derivation. Then the given direct derivations are (*sequential-*) *independent* if they satisfy the following condition:

$$h_1(R_1) \cap g_2(L_2) \subseteq h_1(K_1) \cap g_2(K_2).$$

(2) Let $M \Rightarrow_{r_i} N$ be direct derivations according to $g_i : L_i \to M$ for $i = 1, 2$. Then the given derivations are (*left-parallel-*) *independent* if they satisfy the following condition:

$$g_1(L_1) \cap g_2(L_2) \subseteq g_1(K_1) \cap g_2(K_2).$$

(3) Let $N_i \Rightarrow_{r_i} X$ be direct derivations according to $g_i : L_i \to N_i$ for $i = 1, 2$. Then the given derivations are (*right-parallel*) *independent* if the direct derivations $X \Rightarrow N_i$ through r_i^{-1} are left-parallel-independent.

Remark. (1) We say independent for short if it is clear which sort of independency we mean.

(2) Independency means that the occurrences, which are assumed to be independent, may overlap, but share common gluing parts only.

Theorem 2.8. (1) *Let $r = r_1 + r_2$ be a parallel rule. Let $M \Rightarrow_r X$ be a direct derivation. Then there exist two graphs N_1 and N_2 and two derivation sequences $M \Rightarrow_{r_1} N_1 \Rightarrow_{r_2} X$ and $M \Rightarrow_{r_2} N_2 \Rightarrow_{r_1} X$.*

(2) *Let $M \Rightarrow_{r_1} N_1 \Rightarrow_{r_2} X$ be sequential-independent derivations. Then there is a direct parallel derivation $M \Rightarrow_{r_1+r_2} X$.*

(3) *Let $M \Rightarrow_{r_1} N_1$ and $M \Rightarrow_{r_2} N_2$ be left-parallel-independent derivations. Then there is a direct parallel derivation $M \Rightarrow_{r_1+r_2} X$.*

(4) *Let $N_1 \Rightarrow_{r_2} X$ and $N_2 \Rightarrow_{r_1} X$ be right-parallel-independent derivations. Then there is a direct parallel derivation $M \Rightarrow_{r_1+r_2} X$.*

Remark. (1) The derivation sequences $M \Rightarrow N_i \Rightarrow X$ for $i = 1, 2$, constructed in Theorem 2.8(1), are called *sequentializations* of the direct derivation $M \Rightarrow X$.

(2) The direct parallel derivation, constructed in (2), (3), and (4), is called *parallelization* of the given derivations.

(3) Sequentializations ar sequential-independent.

(4) The derivations $M \Rightarrow_{r_1} N_1$ and $M \Rightarrow_{r_2} N_2$ as well as $N_1 \Rightarrow_{r_2} X$ and $N_2 \Rightarrow_{r_1} X$, constructed in Theorem 2.8(1), can be proved to be parallel-independent.

(5) Note that r_1 and r_2 are parallel rules themselves.

Example 2.9. For the direct derivation $\text{GRAPH}(\text{init}) \Rightarrow_{r_a+r_b} \text{GRAPH}(\text{follow})$, shown in Example 2.6, there exist two sequentializations $\text{GRAPH}(\text{init}) \Rightarrow_{r_a} \text{GRAPH}(m') \Rightarrow_{r_b} \text{GRAPH}(\text{follow})$ as well as $\text{GRAPH}(\text{init}) \Rightarrow r_b\, \text{GRAPH}(m'') \Rightarrow_{r_a} \text{GRAPH}(\text{follow})$.

Now we can introduce the notion of a graph grammar and a graph language.

Definition 2.10. (1) A *graph grammar* $G = (C, P, S)$ consists of
- a pair of *colour alphabets* $C = (C_V, C_E)$ for vertices and edges,
- a set of *rules* P, and
- a *startgraph* S.

(2) The *(graph-) language* generated by G is $L(G) = \{M \mid S \overset{*}{\Rightarrow}_P M\}$.

Remark. (1) Note that we do not distinguish between nonterminals and terminals, because we have no use of this distinction in our investigation so far. The reason is that we apply graph grammars to some models of Petri nets where the full reachability class of initial markings is of interest and no mechanism is employed to distinguish between 'terminal' and 'nonterminal' markings. In the terminology of formal languages this corresponds to the consideration of exhaustive languages which contain all derivable strings (or graphs) with and without nonterminals.

(2) Because parallel derivations yield nothing more than sequential derivations, it is enough to give the (finite) set of rules in the grammar, but we can also work with the (infinite) set of parallel rules.

3. Simulation of Petri nets by graph grammars

Given a net, we associate a graph grammar rule to each transition so that each transformation of a marking is simulated by a direct derivation. The basic idea is a slight modification of marked nets where tokens and slots are no longer considered as labels but as additional bundles of nodes attached to their places by edges.

To be able to represent the net by graphs and graph grammar rules we introduce the auxiliary notions of an underlying graph and of a bundle which means a place decorated by given numbers of incoming and outgoing edges.

Assumption 3.1. Let $N = (S, T, K, w, k)$ be a net. Let $C = (C_V, C_E)$ be the pair of colour alphabets given by $C_V = S \cup T \cup \{*\}$ and $C_E = \{*\}$.

Definition 3.2. (1) The graph $M = (S \cup T, F, s, t, l, m)$, where $s, t : F \to S \cup T$ and $l : S \cup T \to C_V$ and $m : F \to C_E$ are defined by

$$\begin{aligned} &s(a, b) = a \text{ and } t(a, b) = b && \text{for } (a, b) \in F, \\ &l(a) = a && \text{for } a \in S \cup T, \\ &m(e) = * && \text{for } e \in F, \end{aligned}$$

is called the *underlying graph* of N and denoted by $\mathbf{und}(N)$.

(2) For $s \in S$ and $\mu, \nu \in \mathbb{N}$ we construct a graph bundle(s, μ, ν) as follows:

- $V_{\text{bundle}(s,\mu,\nu)} = \{s\} \cup \{s_i \mid i = 1, \ldots, \mu + \nu\}$,
- $E_{\text{bundle}(s,\mu,\nu)} = \{\bar{s}_i \mid i = 1, \ldots, \mu + \nu\}$,
- $s_{\text{bundle}(s,\mu,\nu)}: E \to V$ is defined by

$$s_{\text{bundle}(s,\mu,\nu)}(\bar{s}_i) = \begin{cases} s & \text{for } 1 \leqslant i \leqslant \mu, \\ s_i & \text{for } \mu < i \leqslant \mu + \nu, \end{cases}$$

- $t_{\text{bundle}(s,\mu,\nu)}: E \to V$ is defined by

$$t_{\text{bundle}(s,\mu,\nu)}(\bar{s}_1) = \begin{cases} s_i & \text{for } 1 \leqslant i \leqslant \mu, \\ s & \text{for } \mu < i \leqslant \mu + \nu, \end{cases}$$

- $l_{\text{bundle}(s,\mu,\nu)}: V \to C_V$ is defined by

$$l_{\text{bundle}(s,\mu,\nu)}(s) = s, \qquad l_{\text{bundle}(s,\mu,\nu)}(s_i) = * \quad \text{for } 1 \leqslant i \leqslant \mu + \nu,$$

- $m_{\text{bundle}(s,\mu,\nu)}: E \to C_E$ is defined by $m_{\text{bundle}(s,\mu,\nu)}(e) = * \quad \text{for } e \in E$.

Remark. (1) If $\nu = 0$, then the bundle is called *token-bundle*, if $\mu = 0$, then the bundle is called a *slot-bundle.*

(2) The vertices of the bundle except the place s are called *satellites*, the satellites s_i for $1 \leqslant i \leqslant \mu$ are called *token-satellites*, the satellites s_i for $\mu < i \leqslant \mu + \nu$ are called *slot-satellites*, because they will represent tokens and slots respectively.

(3) Note that, by the choice of nodes and edges, bundles of different places are disjoint.

(4) Moreover, we assume that the intersection of a bundle with **und**(N) consists of the place of the bundle only.

A marked net (N, m) is represented by a graph constructed as an extension of the underlying graph by bundles: the marking of a place $s \in S$ is represented in the graph by a token-bundle with $m(s)$ satellites; for a place $s' \in S$ with bounded capacity, the slots are represented by a slot-bundle with sl(s') satellites. The width of the edges of the net will be expressed in the rules.

Construction 3.3. Let m be a marking of N. Let S^{d} be the (discrete) graph with $V_{S^{\text{d}}} = S$, $E_{S^{\text{d}}} = \emptyset$, and $l_{S^{\text{d}}}(v) = v$ for all $v \in V_{S^{\text{d}}}$. Let i be the inclusion of S^{d} to **und**(N). The *associated graph* GRAPH(m) is the i-gluing of $\sum_{s \in S}$ bundle$(s, m(s), \text{sl}(s))$ and **und**(N) along S^{d} (see Fig. 5).

Remark. (1) Note that S^{d} is subgraph of $\sum_{s \in S}$ bundle$(s, m(s), \text{sl}(ws))$. Therefore, the i-gluing above is defined according to (8) of Appendix A.

(2) GRAPH(m) is given by the following properties:

- $S \cup T \cup \bigcup_{s \in S} (V_{\text{bundle}(s,m(s),\text{sl}(s))} - \{s\})$ is the set of nodes,
- $F \cup \bigcup_{s \in S} (E_{\text{bundle}(s,m(s),\text{sl}(s))})$ is the set of edges, and
- source, target, and labels are defined in such a way that **und**(N) and, for all $s \in S$, bundle$(s, m(s), \text{sl}(s))$ are subgraphs of GRAPH(m).

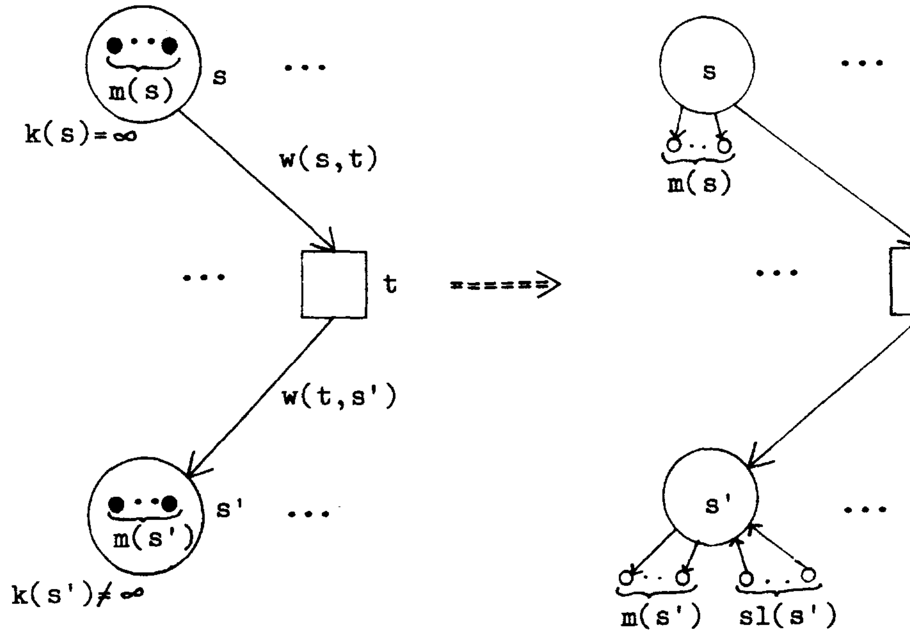

Fig. 5. Construction of an associated graph.

Example 3.4. For the marked net EXAMPLE = (example, init) in Example 1.4, the associated graph GRAPH(init) is given in Example 2.6.

To be able to simulate the behaviour of the net, we construct graph grammar rules, one for each transition. Such a rule is obtained from a transition $t \in T$ in the following way:

- The gluing graph consists of the transition t, all input and output places, and all connecting edges. All further nodes in the rules will be satellites, and all further edges will be incident to satellites. Hence, the application of such rules will never change the underlying graph (as the transformation of markings never changes the underlying net).
- The left-hand side has, in addition to the gluing graph, for each input place $i \in {}^*t$, a token-bundle with $w(i, t)$ satellites, and, for each output place $o \in t^*$ with bounded capacity, a slot-bundle with $w(t, o)$ satellites. Whenever such a rule is applied, these satellites are removed. This exactly simulates how the number of tokens and slots decreases if the corresponding transition fires.
- The right-hand side of the rule has, in addition to the gluing graph, for each output place $0 \in t^*$, a token-bundle with $w(t, o)$ satellites, and, for each input place $i \in {}^*t$ with bounded capacity, a slot-bundle with $w(i, t)$ satellites. In an application of a rule these satellites are added simulating how the number of tokens and slots increases if the corresponding transition fires.

Construction 3.5 (cf. Fig. 6). (1) Let $t \in T$. Let $({}^*t + t^*)^d$ be the (discrete) graph with $V_{({}^*t+t^*)^d} = ({}^*t + t^*)^d$, $E_{({}^*t+t^*)^d} = \emptyset$, and $l_{({}^*t+t^*)^d}(v) = v$ for all $v \in V_{({}^*t+t^*)^d}$. The *corresponding rule* $r_t = (L_t, R_t, K_t)$ is defined by

- K_t is the subgraph of **und**(N) induced by $\{t\} \cup {}^*t \cup t^*$;

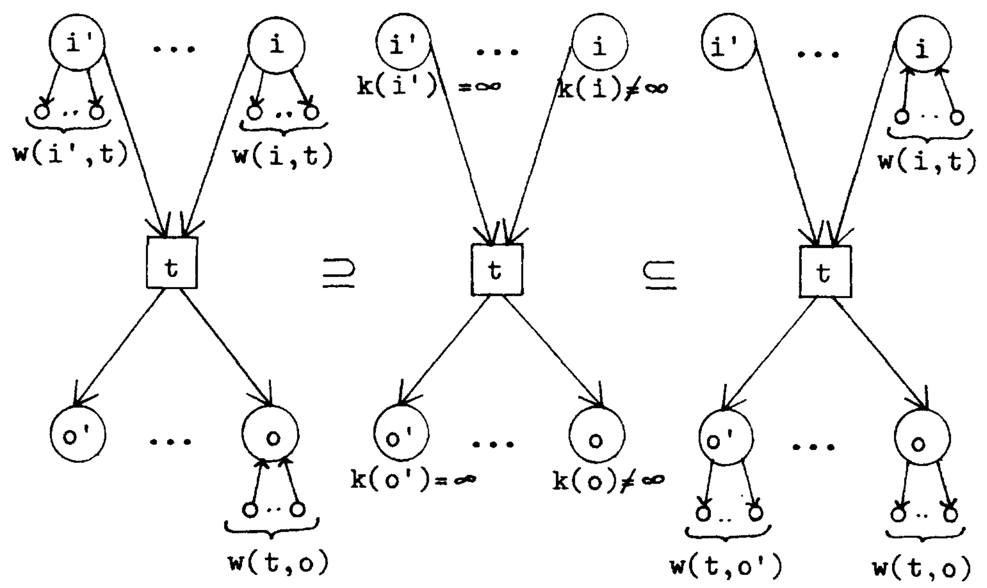

Fig. 6. Construction of a corresponding rule.

- L_t is the d-gluing of K_t and

$$\sum_{s\in{}^*t} \text{bundle}(s, w(s,t), 0) + \sum_{s\in t^*, k(s)\neq\infty} \text{bundle}(s, 0, w(t,s))$$
$$+ \sum_{s\in t^*, k(s)=\infty} \text{bundle}(s, 0, 0)$$

along $({}^*t+t^*)^d$, where $d:({}^*t+t^*)^d \to K_t$ is the graph morphism with $d_V(s)=s$ for all $s\in V_{({}^*t+t^*)^d}$;

- R_t is the d-gluing of K_t and

$$\sum_{s\in{}^*t, k(s)\neq\infty} \text{bundle}(s, 0, w(s,t)) + \sum_{s\in{}^*t, k(s)=\infty} \text{bundle}(s, 0, 0)$$
$$+ \sum_{s\in t^*} \text{bundle}(s, w(t,s), 0)$$

along $({}^*t+t^*)^d$.

(2) Let $U: T\to \mathbb{N}$ be a bag of transitions. The *corresponding parallel rule* r_U is defined as $r_U = \sum_{t\in T} U(t)\times r_t$.

Remark. (1) Note that $({}^*t+t^*)^d$ is subgraph of the two disjoint unions of bundles so that the both d-gluings above are defined according to (8) of Appendix A.

(2) $r_t=(L_t, R_t, K_t)$ is defined by

- K_t is the subgraph of **und**(N) induced by $\{t\}\cup{}^*t\cup t^*$;
- L_t is characterized by the following properties:

(i) $V_{L_t} = V_{K_t} \cup \bigcup_{s\in{}^*t} (V_{\text{bundle}(s,w(s,t),0)} - \{s\}) \cup \bigcup_{s\in t^*, k(s)\neq\infty} (V_{\text{bundle}(s,0,w(t,s))} - \{s\})$,

(ii) $E_{L_t} = E_{K_t} \cup \bigcup_{s\in{}^*t} E_{\text{bundles}(s,w(s,t),0)} \cup \bigcup_{s\in t^*, k(s)\neq\infty} E_{\text{bundle}(s,0,w(t,s))}$,

(iii) source, target, and labels are defined in such a way that K_t and bundle$(s, w(s, t), 0)$ for $s \in {}^*t$ amd bundle$(s, 0, w(t, s))$ for $s \in t^*$ with $k(s) \neq \infty$ are subgraphs of L_t;

- R_t is characterized by the following properties:

(i) $V_{R_t} = V_{K_t} \cup \bigcup_{s \in {}^*t, k(s) \neq \infty} (V_{\text{bundle}(s,0,w(s,t))} - \{s\}) \cup \bigcup_{s \in t^*} (V_{\text{bundle}(s,w(t,s),0)} - \{s\})$,

(ii) $E_{R_t} = E_{K_t} \cup \bigcup_{s \in {}^*t, k(s) \neq \infty} E_{\text{bundle}(s,0,w(s,t))} \cup \bigcup_{s \in t^*} E_{\text{bundle}(s,w(t,s),0)}$,

(iii) source, target, and labels are defined in such a way that K_t and bundle$(s, 0, w(s, t))$ for $s \in {}^*t$ with $k(s) \neq \infty$ and bundle$(s, w(t, s), 0)$ for $s \in t^*$ are subgraphs of R_t.

Example 3.6. For the marked net EXAMPLE = (example, init) and the bag of transition U (see Example 1.6), the corresponding rule r_U is given in Example 2.6.

The gluing condition makes sure that the transition t is enabled in the net whenever the corresponding rule r_t can be applied. Removing the non-gluing part of the left-hand side and adding the non-gluing part of the right-hand side simulates the transformation of the marking. The same reasoning applies to r_U as the following theorem states.

Theorem 3.7. *Let U be a bag of transitions of N, and let m, m' be markings of N. Then $m[U\rangle m'$ if and only if* GRAPH$(m) \Rightarrow_{r_U}$ GRAPH(m').

Proof. If U is enabled under m, we have

$$(*) \quad m(s) \geq \sum_{t \in T} U(t) \times \bar{w}(s, t) \qquad \text{for all } s \in S, \text{ and}$$

$$(**) \quad k(s) - m(s) \geq \sum_{t \in T} U(t) \times \bar{w}(t, s) \qquad \text{for all } s \in S \text{ with } k(s) \neq \infty.$$

In the associated graph GRAPH(m) there is, for each $s \in S$, a bundle with $m(s)$ token-satellites and sl(s) slot-satellites. The left-hand side L_U of the rule r_U is given by $L_U = \sum_{t \in T} U(t) \times L_t$.

In each left-hand side L_t we have, for each $s \in {}^*t$, a bundle with $w(s, t)$ token-satellites and, for each $s \in t^*$ with $k(s) \neq \infty$, a bundle with $w(t, s)$ slot-satellites. Therefore, in the left-hand side of the rule r_U we have for each $s \in S$, $\sum_{t \in T} U(t) \times \bar{w}(s, t)$ token-satellites and, for each $s \in S$ with $k(s) \neq \infty$, $\sum_{t \in T} U(t) \times \bar{w}(t, s)$ slot-satellites. Now, $(*)$ makes sure that there are at least as many token-satellites in GRAPH(m) as in L_U for each $s \in S$, and $(**)$ makes sure that there are at least as many slot-satellites in GRAPH(m) as in L_U for each $s \in S$ with bounded capacity. Therefore, there are injective mappings for all edges and satellites of bundles of the left-hand side into corresponding bundles in GRAPH(m). These can be extended to a graph morphism, $g : L_U \to$ GRAPH(m) by mapping all gluing graphs K_t identically.

The non-gluing nodes of the occurrence are the satellites, but, by construction, each satellite is either source or target of exactly one edge which belongs to the occurrence, too. This proves the contact condition.

The identification condition is satisfied because g injectively maps satellites and adjacent edges (being the non-gluing items) to satellites and adjacent edges respectively. Parts of the underlying graph may be identified with each other, but this does not conflict with the identification because these are gluing terms.

Now the application of r_U to $\text{G}\textsc{raph}(m)$ leaves the underlying graph **und**(N) unchanged, but removes and adds edges and satellites of the token- and slot-bundles of each place.

The token-bundle of each place $s \in S$ has the following form after the application of r_U:

$$\begin{aligned}&\text{bundle}(s, m(s), 0)\\&\quad+\sum_{t\in T} U(t)\times\Big(\sum_{s\in t^*}\text{bundle}(s, w(t,s), 0)-\sum_{s\in {}^*t}\text{bundle}(s, w(s,t), 0)\Big)\\&=\text{bundle}(s, m(s), 0)\\&\quad+\sum_{t\in T} U(t)\times\Big(\sum_{s\in S}\text{bundle}(s, \bar{w}(t,s), 0)-\sum_{s\in S}\text{bundle}(s, \bar{w}(s,t), 0)\Big),\end{aligned}$$

so the number of token-satellites for each place $s \in S$ is given by $m(s)+\sum_{t\in T} U(t)\times(\bar{w}(t,s)-\bar{w}(s,t))$ which is $m'(s)$ by the definition of the transformation of m to m' by U. Hence, the new number of token-satellites for each place $s\in S$ is $m'(s)$.

The slot-bundle of each place $s \in S$ with $k(s) \neq \infty$ is given by

$$\begin{aligned}&\text{bundle}(s, 0, \text{sl}(s))\\&\quad+\sum_{t\in T} U(t)\times\Big(\sum_{s\in {}^*t}\text{bundle}(s, 0, w(s,t))-\sum_{s\in t^*}\text{bundle}(s, 0, w(t,s))\Big)\\&=\text{bundle}(s, 0, \text{sl}(s))\\&\quad+\sum_{t\in T} U(t)\times\Big(\sum_{s\in S}\text{bundle}(s, 0, \bar{w}(s,t))-\sum_{s\in S}\text{bundle}(s, 0, \bar{w}(t,s))\Big),\end{aligned}$$

so the number of slot-satellites for each place $s \in S$ with bounded capacity is $\text{sl}(s)+\sum_{t\in T} U(t)\times(\bar{w}(s,t)-\bar{w}(t,s))$ but this is exactly $\text{sl}'(s)$ which is defined as

$$\begin{aligned}\text{sl}'(s)&=k(s)-m'(s)\\&=k(s)-m(s)+\sum_{t\in T} U(t)\times(\bar{w}(s,t)-\bar{w}(t,s))\\&=\text{sl}(s)+\sum_{t\in T} U(t)\times(\bar{w}(s,t)-\bar{w}(t,s)).\end{aligned}$$

Note that for places $s \in S$ with unbounded capacity we have $\mathrm{sl}(s)=0$ and the new slot-bundles for these places are given by

$$\begin{aligned}&\mathrm{bundle}(s,0,\mathrm{sl}(s))\\ &\quad+\sum_{t\in T} U(t)\times\left(\sum_{s\in{}^*t}\mathrm{bundle}(s,0,0)-\sum_{s\in t^*}\mathrm{bundle}(s,0,0)\right)\\ &=\mathrm{bundle}(s,0,0),\end{aligned}$$

so $\mathrm{sl}'(s)=0$ for all $s\in S$ with $k(s)=\infty$ satisfies the definition of slots for places with unbounded capacity.

Therefore, the derived graph is the associated graph GRAPH(m').

Conversely, if r_U is applicable to GRAPH(m), then we have a graph morphism $g: L_U \to \text{GRAPH}(m)$ satisfying especially the identification condition. Hence, g is injective on satellites. This guarantees that U is enabled under m. And as shown above, due to the construction of r_U, the application of the rule removes and adds token- and slot-satellites according to the width and simulates the transformation of m to m' by U so that the derived graph is the graph associated to the marking m'. □

It should be mentioned that, as an immediate consequence of this result, the notion of concurrency in nets and the notion of independency in graph grammars are closely related, as is shown in the following corollary.

Corollary 3.8. *Let t and $t^\S$ be two transitions of N which are enabled under the marking m. Then t and $t^\S$ are concurrent if and only if there exist corresponding direct derivations* GRAPH(m)⇒GRAPH(m') *through r_t and* GRAPH(m)⇒GRAPH(m'') *through $r_{t^\S}$ which are independent.*

Proof. The two transitions t and $t^\S$ are concurrent if there is a bag of transitions $U: T\to\mathbb{N}$ with $U(t)\geq 1$ and $U(t^\S)\geq 1$ for $t\neq t^\S$, or $U(t)\geq 2$ for $t=t^\S$ which is enabled under m; so there is a step $m[U\rangle m''$. Then, by Theorem 3.7, there is a direct derivation GRAPH(m)⇒GRAPH(m'') through r_U. By assumption, r_U has the following form: $r_U=\cdots+r_t+r_{t^\S}+\cdots$. So there exists a sequentialization GRAPH(m)⇒GRAPH(m') through $r_{U'}=\cdots+r_t$ and GRAPH(m')⇒ GRAPH(m'') through $r_{U''}=r_{t^\S}+\cdots$ and the direct derivations of this sequentialization are independent (confer Theorem 2.8).

On the other hand, direct derivations $r_{U'}=\cdots+r_t+\cdots$ and $r_{U''}=\cdots+r_{t^\S}+\cdots$ which are independent can be parallelized to a direct derivation through $r_U=r_{U'}+r_{U''}$ and by Theorem 3.7 there is a corresponding step in the net by U, where t and $t^\S$ are both involved, so they are concurrent. □

Example 3.9. As an illustration of Theorem 3.7, Fig. 2 shows a step init[$U\rangle$follow and Fig. 4 shows the direct derivation GRAPH(init)⇒GRAPH(follow) through r_U.

According to Corollary 3.8, in this case, the transitions a and b are concurrent and there are decompositions (see Example 1.8) as well as there are sequentializations and independent derivations through r_a and r_b (see Example 2.9).

Using the notions of associated graphs and corresponding rules we can introduce the graph grammar simulating a given net.

Definition 3.10. Let (N, m_0) be a marked net. $G(N, m_0) = (C, P, \text{GRAPH}(m_0))$ with $P = \{r_t \mid t \in T\}$ is called the *simulating graph grammar.*

Now, the results of this section can be formulated in terms of reachability and derivability.

Corollary 3.11. *Let (N, m_0) be a marked net and let $G(N, m_0)$ be the simulating graph grammar. Then $m_0[*\rangle m$ if and only if $\text{GRAPH}(m_0) \overset{*}{\Rightarrow}_P \text{GRAPH}(m)$.*

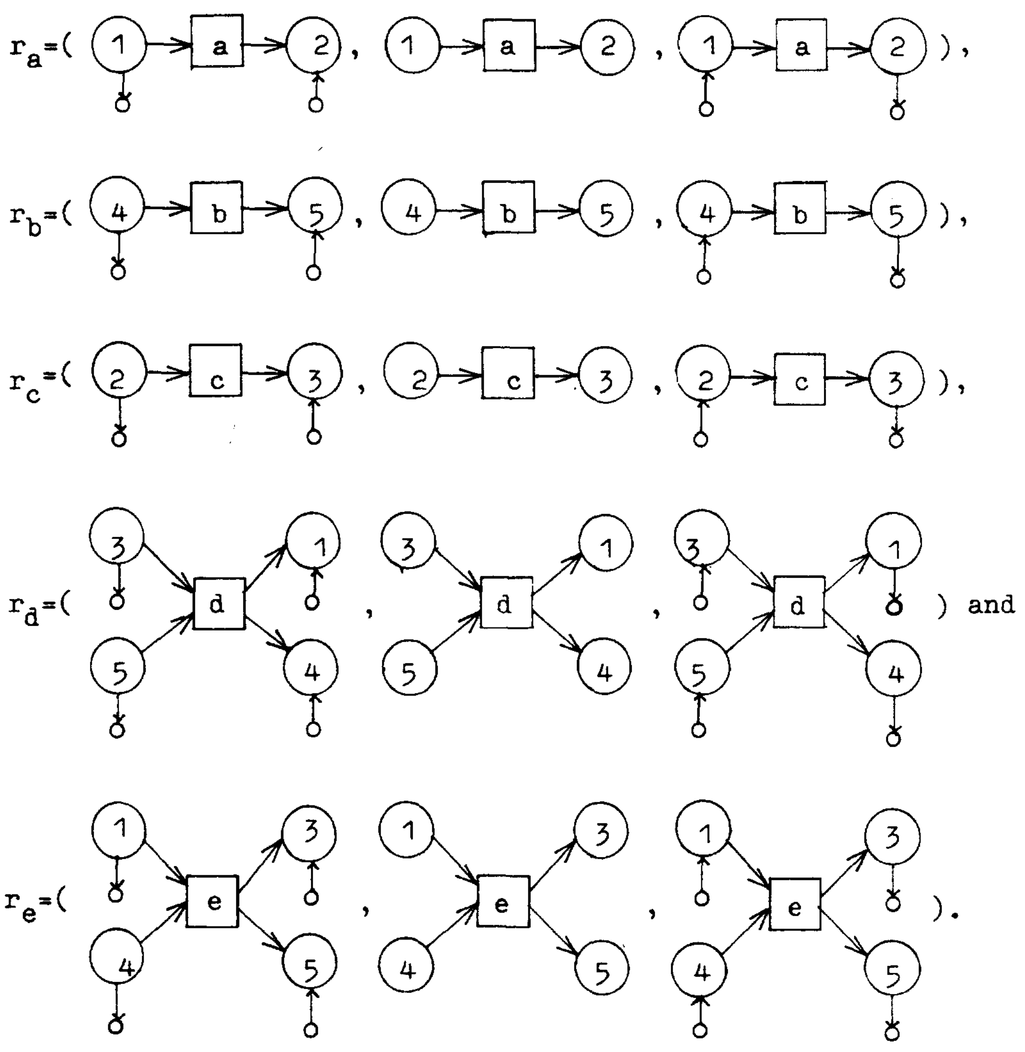

Fig. 7. Corresponding rules.

Example 3.12. The simulating graph grammar for the marked net EXAMPLE (see Example 1.4) is defined as $G(\text{EXAMPLE}) = (C, \{r_a, r_b, r_c, r_d, r_e\}, \text{GRAPH(init)})$, where C is given by $C_V = \{1, 2, 3, 4, 5\} \cup \{a, b, c, d, e\} \cup \{*\}$ and $C_E = \{*\}$; GRAPH(init) is shown in Example 2.6 and the rules are given in Fig. 7.

4. Net processes

Transformations of markings of nets describe a sequential behaviour of nets including 'synchronous' firing of transitions. In contrast to that, the intuition of concurrent systems allows 'asynchronous' activities where a strictly sequential or parallel relation of actions in time may not be known or observable. In the theory of nets such a view of concurrency is covered by the notion of processes.

Assumption 4.1. We restrict our consideration to processes on condition/event systems (see Definition 4.2), because they are well studied in the literature (cf., e.g., [9]).

Condition/event systems form a special case of the nets introduced in Section 1 with capacity 1 for all places and width 1 for all edges.

Definition 4.2. A *condition/event system* (N, m_0) consists of
- a net $N = (S, T, F, w, k)$ with $k(s) = 1$ for all $s \in S$ and $w(e) = 1$ for all $e \in F$, and
- an initial marking m_0.

Remark. For condition/event systems, markings are also called *cases*.

Example 4.3. The marked net EXAMPLE, shown in Fig. 1, is a condition/event system.

All possible transformations of cases are represented in the case graph.

Definition 4.4. Let (N, m_0) be a condition/event system with $N = (S, T, F, w, k)$. Then the *case graph* CASE of (N, m_0) is an unlabelled graph defined as follows:
- $V_{\text{CASE}} = \{m \mid m_0[*\rangle m\}$,
- $E_{\text{CASE}} = \{(m, U, m') \mid m[U\rangle m', m, m' \in V_{\text{CASE}}, U : T \to \mathbb{N}\}$,
- $s_{\text{CASE}}((m, U, m')) = m$,
- $t_{\text{CASE}}((m, U, m')) = m'$.

Example 4.5. Figure 8 shows the case graph $\text{CASE}_{\text{EXAMPLE}}$ of the condition/event system EXAMPLE, given in Example 1.4. A node of $\text{CASE}_{\text{EXAMPLE}}$, i.e., the marking m of example, is represented by the set of places $s \in S$ with $m(s) = 1$ and an edge by the sum of all transitions involved in the step.

$$\begin{tikzcd}
 & \{1,5\} \arrow[dr, "a"] & & \\
\{1,4\} \arrow[ur, "b"] \arrow[rr, "a+b"'] \arrow[dr, "a"'] \arrow[drrr, bend right=60, "e"'] & & \{2,5\} \arrow[dr, "c"] & \\
 & \{2,4\} \arrow[ur, "b"'] \arrow[rr, "b+c"'] \arrow[dr, "c"'] & & \{3,5\} \arrow[ulll, bend right=60, "d"'] \\
 & & \{3,4\} \arrow[ur, "b"'] &
\end{tikzcd}$$

Fig. 8. $\text{CASE}_{\text{EXAMPLE}}$.

Note that all paths in the case graph correspond to transformation sequences in the net. Hence, CASE describes the sequential behaviour of the system including parallel steps.

In contrast to that, the concurrent behaviour of the system is formalized by processes. The idea of processes is to relax the sequential behaviour in such a way that the order in time of concurrent transitions need not be fixed. This is obtained by the notion of an occurrence net where the sequential order is replaced by the partial order.

Definition 4.6. An *occurrence net* $K = (S, T, F, w, k)$ is a net, where

- $k(s) = 0$ for all $s \in S$,
- $w(e) = 0$ for all $e \in F$,
- $\mathbf{und}(N)$ is acyclic, and
- for each $s \in S$, there is at most one $t \in T$ with $(t, s) \in F$, and at most one $t' \in T$ with $(s, t') \in F$.

Remark. We have defined occurrence nets as a special form of nets so that we can use all notational conventions for nets but we shall never use their (somewhat strange) capacity and width.

The notion of processes relates an occurrence net to the behaviour of a given net. The formal definition of a process is based on the concept of slices which will be introduced for an occurrence net in the following definition.

Definition 4.7. Let $K = (S, T, F, w, k)$ be an occurrence net.

(1) A subset $S' \subseteq S$ is called *concurrent* if, for all $s, s' \in S'$ with $s \neq s'$, there is neither a path from s to s' nor a path from s' to s.

(2) A maximal concurrent subset of places is said to be a *slice*.

Definition 4.8. A *process* on a condition/event system (N, m_0) with $N = (S_N, T_N, F_N, w_N, k_N)$ consists of an occurrence net $K = (S_K, T_K, F_K, w_K, k_K)$ and a map $p: S_K \cup T_K \to S_N \cup T_N$ with the following properties:

- p is injective on each slice;
- the image of each slice is a node in CASE of (N, m_0), i.e., a case of N reachable from m_0;
- p preserves the input and output structure of each transition in K, i.e., $^*p(t) = p(^*t)$ and $p(t)^* = p(t^*)$ for all $t \in T_K$.

Remark. Note that processes will be represented graphically by the occurrence net coloured with the places and transitions of the condition/event system. (See, e.g., [9] and Fig. 9).

Example 4.9. Figure 9 shows a process p_{EXAMPLE} on the condition/event system EXAMPLE. The slices of p_{EXAMPLE} are $\{s1, s4\}$, $\{s2, s4\}$, $\{s3, s4\}$, $\{s1, s5\}$, $\{s2, s5\}$, $\{s3, s5\}$, and $\{s6, s7\}$.

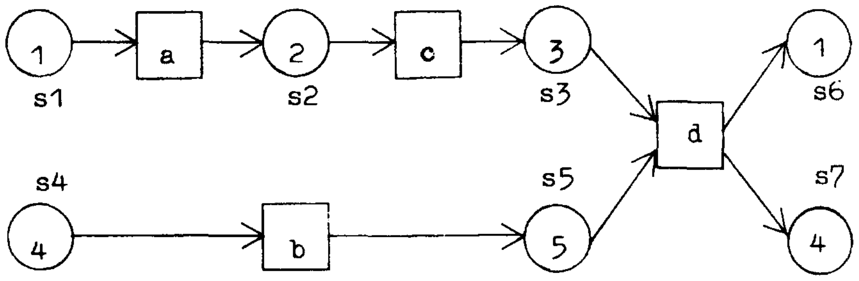

Fig. 9. Process p_{EXAMPLE}.

More details can be found in [9] where, in addition, the following correspondence between paths in the case graph of a condition/event system (N, m_0) and the processes on (N, m_0) is shown. This relationship only works in case of a so-called contact-free condition/event system. This means that each transition is already enabled in each case if all its input places are marked. To assume contact-freeness does not establish a semantical restriction because each condition/event system can be transformed in a contact-free one with the same case graph (up to isomorphism).

Definition 4.10. A condition/event system (N, m_0) is called *contact-free* if, for all markings m with $m_0[*\rangle m$ and for all $t \in T$, it holds that

$$m(s) = 1 \text{ implies } m(s') = 0 \quad \text{for all } s \in {}^*t,\ s' \in t^*.$$

Example 4.11. The condition/event system EXAMPLE is contact-free.

The relationship between processes and paths in the case graph is based on the following observation.

Definition 4.12. Two paths p, p' in CASE are called *equivalent* if there are paths $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = p'$, and p_i and p_{i+1} differ only by a decomposition of one step (in p_i or p_{i+1}).

Remark. Note that the notion of decomposition in the sense of Theorem 1.7 is meaningful in the situation above because edges in CASE are steps in the system.

Example 4.13. The following paths of the case graph $\text{path}_{\text{EXAMPLE}}i$ for $i = 1, \ldots, 5$ are equivalent:

$\text{path}_{\text{EXAMPLE}}1 = (\{1,4\}, a, \{2,4\}), (\{2,4\}, c, \{3,4\}), (\{3,4\}, b, \{3,5\}), (\{3,5\}, d, \{1,4\}),$

$\text{path}_{\text{EXAMPLE}}2 = (\{1,4\}, a, \{2,4\}), (\{2,4\}, b+c, \{3,5\}), (\{3,5\}, d, \{1,4\}),$

$\text{path}_{\text{EXAMPLE}}3 = (\{1,4\}, a, \{2,4\}), (\{2,4\}, b, \{2,5\}), (\{2,5\}, c, \{3,5\}), (\{3,5\}, d, \{1,4\}),$

$\text{path}_{\text{EXAMPLE}}4 = (\{1,4\}, a+b, \{2,5\}), (\{2,5\}, c, \{3,5\}), (\{3,5\}), d, \{1,4\}),$

$\text{path}_{\text{EXAMPLE}}5 = (\{1,4\}, b, \{1,5\}), (\{1,5\}), a, \{2,5\}), (\{2,5\}, c, \{3,5\}), (\{3,5\}), d, \{1,4\}).$

Theorem 4.14. *Let (N, m_0) be a contact-free condition/event system. Then there is a one-to-one correspondence between processes on (N, m_0) and the sets of equivalent paths in the case graph of (N, m_0).*

Remark. Note that there exists exactly one process corresponding to a given path, but on the other hand, several equivalent paths may be related to the same process.

Example 4.15. The process p_{EXAMPLE} in Fig. 9 corresponds to $\{\text{path}_{\text{EXAMPLE}}i \mid i = 1, \ldots, 5\}$ reflecting that the transition b is concurrent to a and c, whereas the transition c is not concurrent to a, and d is only enabled after c and b have fired.

5. Graph grammar processes

Similarly to net processes, a process in a graph grammar is a partial order of direct derivations. In this section we shall recall concepts and results from [5] as far as needed.

Assumption 5.1. Let $G = (C, P, S)$ be an arbitrary but fixed graph grammar.

Definition 5.2. The *derivation graph* (of G) DERIVATION $= (V, E, s, t)$ is an unlabelled graph defined as follows:

- $V = \{M \mid S \overset{*}{\Rightarrow} M\}$,
- $E = \{M \Rightarrow_r M' \mid M, M' \in V, r \in P^+\}$,
- $s(M \Rightarrow_r M') = M$,
- $t(M \Rightarrow_r M') = M'$.

$$\begin{tikzcd}
 & \mathrm{GRAPH}(\{1,5\}) \arrow[dr, "r_a"] & & \\
\mathrm{GRAPH}(\{1,4\}) \arrow[ur, "r_b"] \arrow[rr, Rightarrow, "r_a+r_b"'] \arrow[dr, "r_a"'] \arrow[drrr, bend right=60, "r_e"'] & & \mathrm{GRAPH}(\{2,5\}) \arrow[dr, "r_c"] & \\
 & \mathrm{GRAPH}(\{2,4\}) \arrow[ur, "r_b"] \arrow[rr, Rightarrow, "r_b+r_c"'] \arrow[dr, "r_c"'] & & \mathrm{GRAPH}(\{3,5\}) \arrow[ulll, bend right=60, "r_d"'] \\
 & & \mathrm{GRAPH}(\{3,4\}) \arrow[ur, "r_b"'] &
\end{tikzcd}$$

Fig. 10. DERIVATION$_{G(\text{EXAMPLE})}$.

Example 5.3. Figure 10 shows the derivation graph DERIVATION$_{G(\text{EXAMPLE})}$ of the graph grammar G(EXAMPLE) simulating the marked net EXAMPLE (see Example 3.12). For the denotation of markings we use the same conventions as in Example 4.5. The edges are drawn as usual for direct derivations.

Definition 5.4. A *derivation process* (in G) is a construct $p = (A, a : A \to \text{DERIVATION})$, where A is a connected acyclic unlabelled graph and a is a graph morphism.

Remark. In examples we shall graphically represent a derivation process by the underlying graph A, but coloured in the following way: each $v \in V_A$ gets the graph $a_V(v)$ as colour and each $e \in E_A$ gets the direct derivation $a_E(e)$ as colour. In drawings we explicitly give the colours only because they completely reflect the underlying graphical structure (see, e.g., Fig. 11).

Example 5.5. Figure 11 shows a derivation process $p_{G(\text{EXAMPLE})}$ in the graph grammar G(EXAMPLE).

There are some special situations of some importance later on.

Definition 5.6. (1) A derivation process $p = (A, a)$ is called *sequential* if the underlying graph A is a path.

$$\begin{tikzcd}
\mathrm{GRAPH}(\{1,4\}) \arrow[r, Rightarrow, "r_a+r_b"'] & \mathrm{GRAPH}(\{2,5\}) \arrow[d, "r_c"'] & \\
\mathrm{GRAPH}(\{2,4\}) \arrow[r, "r_b+r_c"'] & \mathrm{GRAPH}(\{3,5\}) \arrow[r, "r_d"'] & \mathrm{GRAPH}(\{1,4\})
\end{tikzcd}$$

Fig. 11. Process $p_{G(\text{EXAMPLE})}$.

$$\begin{tikzcd} & N \\ M \arrow[ur, Rightarrow, "r"] \arrow[dr, Rightarrow, "r"'] & \\ & N \end{tikzcd}$$
(a)

$$\begin{tikzcd} M \arrow[r, Rightarrow, bend left, "r"] \arrow[r, Rightarrow, bend right, "r"'] & N \end{tikzcd}$$
(b)

$$\begin{tikzcd} M \arrow[dr, Rightarrow, "r"] & \\ & N \\ M \arrow[ur, Rightarrow, "r"'] & \end{tikzcd}$$
(c)

Fig. 12. Doubles.

(2) Let $p = (A, a)$ be a derivation process, SUB a subgraph of A and sub the restriction of a to SUB. Then (SUB, sub) is called a *subprocess* of p.

(3) A derivation process $p = (A, a)$ is called *double-free* if there is no subprocess of p of one of the forms shown in Fig. 12.

(4) A process is said to be *locally conflict-free* if each subprocess of p where two direct derivations start from the same graph are left-parallel independent, and each subprocess of p where two direct derivations lead to the same graph are right-parallel independent.

Remark. (1) Note that sequential processes correspond to ordinary derivations.

(2) By definition, subprocesses are derivation processes.

(3) Each of the processes (a), (b), and (c) in Fig. 12 is called a *double.*

(4) There is no *local conflict* within a local conflict-free process.

Example 5.7. The derivation process $p_{G(\text{EXAMPLE})}$ in Fig. 11 is not sequential. The derivation

$$\text{GRAPH}(\{1,4\}) \underset{r_a+r_b}{\Longrightarrow} \text{GRAPH}(\{2,5\}) \underset{r_c}{\Longrightarrow} \text{GRAPH}(\{3,5\}) \underset{r_d}{\Longrightarrow} \text{GRAPH}(\{1,4\})$$

is a subprocess of $p_{G(\text{EXAMPLE})}$.

The results of Section 2 concerning independency can be summarized in terms of processes in the following way.

Corollary 5.8. *For each derivation process p which contains a subprocess p_{sub} of one of the forms shown in Fig.* 13 (*where the direct derivations are assumed to be independent*), *there is a process $\bar{p}$ which contains the subprocess p_{compl}* (*see Fig.* 14) *instead of p_{sub}, but equals p in all other parts.*

$$\begin{tikzcd} & N_1 \\ M \arrow[ur, Rightarrow, "r_1"] \arrow[dr, Rightarrow, "r_2"'] & \\ & N_2 \end{tikzcd}$$
(a)

$$\begin{tikzcd} & N_1 \arrow[dr, Rightarrow, "r_2"] & \\ M \arrow[ur, Rightarrow, "r_1"] & & X \end{tikzcd}$$
(b)

$$\begin{tikzcd} M \arrow[r, Rightarrow, "r_1+r_2"'] & X \end{tikzcd}$$
(c)

$$\begin{tikzcd} N_1 \arrow[dr, Rightarrow, "r_2"] & \\ & X \\ N_2 \arrow[ur, Rightarrow, "r_1"'] & \end{tikzcd}$$
(d)

Fig. 13. Independent situations.

$$\begin{tikzcd}
& N_1 \arrow[dr, "r_2"] & \\
M \arrow[ur, "r_1"] \arrow[rr, Rightarrow, "r_1+r_2"'] \arrow[dr, "r_2"'] & & X \\
& N_2 \arrow[ur, "r_1"'] &
\end{tikzcd}$$

Fig. 14. Complete situation.

Remark. (1) $\bar{p}$ is called a *local completion* of p.

(2) Note that the original process p is a subprocess of $\bar{p}$ such that $p \cap p_{\mathrm{compl}} = p_{\mathrm{sub}}$ and $p \cup p_{\mathrm{compl}} = \bar{p}$. In this sense, $\bar{p}$ is covered by p and p_{compl} meaning that each item of $\bar{p}$ belongs to p or to p_{compl}.

This means that independent direct derivations can be performed in parallel or in arbitrary order with the same result.

From a point of view of concurrency independent situations and their common local completion should not be distinguished, because they differ only in the order of time of occurring actions and in doubles.

Definition 5.9. Derivation processes p and p' are called *equivalent* if there are processes $p_1, \ldots, p_n$ with $p_1 = p$ and $p_n = p'$ and, for $i = 1, \ldots, n-1$, p_i is a local completion of p_{i+1} or p_{i+1} is a local completion of p_i, or p_i and p_{i+1} differ by a double (in p_i or p_{i+1}) only.

It turns out that the completion procedure does not add new information to a derivation process after some steps. This leads to a normal form result for equivalent derivation processes in the following way.

Definition 5.10. A double-free derivation process is *complete* if it differs from its local completions by doubles only.

Example 5.11. The process $\bar{p}_{G(\text{EXAMPLE})}$ (see Fig. 15) is the complete process equivalent to the process $p_{G(\text{EXAMPLE})}$ in Example 5.5.

Using this notion we can formulate the main result of this section.

$$\begin{tikzcd}
& \mathrm{GRAPH}(\{1,5\}) \arrow[dr, "r_a"] & & & \\
\mathrm{GRAPH}(\{1,4\}) \arrow[ur, "r_b"] \arrow[rr, Rightarrow, "r_a+r_b"'] \arrow[dr, "r_a"'] & & \mathrm{GRAPH}(\{2,5\}) \arrow[dr, "r_c"] & & \\
& \mathrm{GRAPH}(\{2,4\}) \arrow[ur, "r_b"'] \arrow[rr, Rightarrow, "r_b+r_c"'] \arrow[dr, "r_c"'] & & \mathrm{GRAPH}(\{3,5\}) \arrow[r, Rightarrow, "r_d"'] & \mathrm{GRAPH}(\{1,4\}) \\
& & \mathrm{GRAPH}(\{3,4\}) \arrow[ur, "r_b"'] & &
\end{tikzcd}$$

Fig. 15. Process $\bar{p}_{G(\text{EXAMPLE})}$.

Theorem 5.12. *Each equivalence class of derivation processes contains a complete process as unique normal form.*

On the base of complete processes, we can introduce the notion of (globally) conflict-free processes.

Definition 5.13. A process is called *conflict-free* if its complete process is locally conflict-free.

Example 5.14. The process $p_{G(\text{EXAMPLE})}$ (confer Fig. 11) and its completion $\bar{p}_{G(\text{EXAMPLE})}$ (confer Fig. 15) are conflict-free.

Lemma 5.15. (1) *A sequential process is conflict-free.*

(2) *A process equivalent to another conflict-free process is conflict-free, too.*

(3) *The set of equivalent sequential processes forms a subset of the class of equivalent conflict-free processes.*

(4) *Two sequential processes s and s' are equivalent if and only if there are derivation sequences $s_1, \ldots, s_n$ with $s_1 = s$ and $s_n = s'$, and s_i and s_{i+1} differ only by a sequentialization (in s_i or s_{i+1}).*

6. Relationship between processes in Petri nets and graph grammars

In this section the non-sequential behaviour of a system described by a Petri net is related to the non-sequential behaviour of the graph grammar simulating the net.

Analogously to Theorem 3.7 for the sequential behavior, we get as the main result of this paper a one-to-one correspondence between the non-sequential behaviours.

Main Theorem 6.1. *Let (N, m_0) be a contact-free condition/event system. Let $G(N, m_0)$ be the simulating graph grammar. Then there is a one-to-one correspondence between processes on (N, m_0) and the complete conflict-free processes on $G(N, m_0)$.*

Proof. A process on (N, m_0) uniquely corresponds to a set of equivalent paths in CASE, the case graph of (N, m_0), according to Theorem 4.14. Using the following Lemma 6.2 this set of paths uniquely corresponds to a set of equivalent sequential processes in $G(N, m_0)$ which is a subset of their full equivalence class. By Lemma 5.15, this class consists of conflict-free processes because it contains sequential processes. Using Theorem 5.12, this yields a unique complete conflict-free process. $\square$

Remark. Note that we need the assumption of contact-freeness for the one-to-one correspondence between processes and equivalence classes of paths in CASE. All other steps of the proof above work for arbitrary condition/event systems.

Lemma 6.2. (1) *Let* Case *be the case graph of a condition/event system* (N, m_0) *and let* Derivation *be the derivation graph of the simulating graph grammar* $G(N, m_0)$. *Then* Case *is isomorphic to* Derivation.

(2) *Paths in* Case *are equivalent if and only if the corresponding sequential processes in* Derivation *are equivalent.*

Proof. (1) By mapping a case m to $\text{Graph}(m)$ and a step $m[U\rangle m'$ to a direct derivation $\text{Graph}(m) \Rightarrow \text{Graph}(m')$ through $r_{U'}$ we get a graph morphism $i : \text{Case} \to \text{Derivation}$. In condition/event systems the bundle of each place consists of one satellite only. Hence, using Theorem 3.7, it turns out that i is an isomorphism.

(2) By Theorem 3.7, a step $m[U\rangle m''$ corresponds to a direct derivation $\text{Graph}(m) \Rightarrow \text{Graph}(m'')$ through r_U and a decomposition $m[U'\rangle m'[U''\rangle m''$ corresponds to a sequentialization $\text{Graph}(m) \Rightarrow \text{Graph}(m')$ through $r_{U'}$ and $\text{Graph}(m') \Rightarrow \text{Graph}(m'')$ through $r_{U''}$. Decomposition generates the equivalence of paths in Case and on the other hand, sequentialization generates the equivalence of sequential processes (cf. Lemma 5.15). This completes the proof. $\square$

Example 6.3. Obviously, the case graph $\text{Case}_{\text{Example}}$ in Example 4.5 and the derivation graph $\text{Derivation}_{G(\text{Example})}$ in Example 5.6 are isomorphic.

Moreover, the complete process $\bar{p}_{G(\text{Example})}$ on $G(\text{Example})$ in Fig. 15 corresponds to the process p_{Example} on Example in Fig. 9.

7. Conclusion

The diagram in Fig. 16 relates the results of this paper to each other. In the left column the results from net theory are summarized. For each condition/event system there is a case graph containing the sets of all equivalent paths. And according to Theorem 4.14 there is a one-to-one correspondence between the sets of equivalent paths and the processes, but only in case of contact-freeness of the condition/event system.

$$\begin{tikzcd}
\text{condition/event system} \arrow[rr, Rightarrow, "\text{simulation}" below] \arrow[d, mapsto] & & \text{graph grammar} \arrow[d, mapsto] \\
\text{case graph} \arrow[rr, equal, "\sim"] \arrow[d, mapsto] & & \text{derivation graph} \arrow[d, mapsto] \\
\text{equivalent paths} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow, "\text{contact-free}" left] & \text{equivalent sequential processes} \arrow[r, phantom, "\subseteq"] & \text{equivalent conflict-free processes} \arrow[d, leftrightarrow] \\
\text{process} \arrow[rr, leftrightarrow, "\text{MAIN THEOREM}" below] & & \text{conflict-free complete process}
\end{tikzcd}$$

Fig. 16. Summary.

Summarizing the results from graph grammar theory in the right column, the derivation graph of a graph grammar contains all equivalent processes for which the complete process is a unique representation by Theorem 5.12.

If we consider now especially the graph grammar simulating a contact-free condition/event system, we get as a consequence of Theorem 3.7 (as shown in Lemma 6.2) the isomorphy of the case graph and the derivation graph, i.e., this result relates the sequential behaviour of nets and graph grammars as closely as possible.

Due to Lemma 6.2, there is a one-to-one correspondence between equivalent paths and the equivalent sequential processes which form a subset of the equivalent conflict-free processes. And the bottom reflects the Main Theorem 6.1 stating a one-to-one correspondence between the processes on a contact-free condition/event system and the complete conflict-free processes in the simulating graph grammar.

To give a more complete picture of the situation, there is a result of former studies (cf. [5]) stating that in each equivalence class of conflict-free processes there is a unique *canonical derivation*; this is a sequential process where all components are executed as early as possible so that each rule of the parallel derivation is dependent of the predecessing direct derivation or belongs to the first derivation of the sequence. In this way canonical derivations uniquely represent conflict-free non-sequential processes as complete processes, but, in general, the size of canonical derivations (being sequential) is much more feasible than the usually 'baroque' form of complete processes ornamented with the various diamonds of local completion (see Lemma 5.15).

Example 7.1. The derivation sequence, given in Example 5.7, is the canonical derivation of the complete conflict-free process $\bar{p}_{G(\text{EXAMPLE})}$ in Fig. 15.

It should be noticed that in the simulation presented in this paper we allow a transition t to be concurrent to itself (cf. Remarks 1.5(2) and (4) and Corollary 3.8) which is normally forbidden in net theory. We obtain a graph grammar reflecting this restriction by modifying nothing but the gluing graph K_t of each corresponding rule r_t in such a way that we remove the transition t and all its incoming and outgoing edges from it. All results remain true under this modification. Hence, with respect to our considerations, it does not matter whether a transition may be concurrent to itself or not.

A lot of further work must be done in the line of this paper. In particular, the relationship between processes on nets and processes in graph grammars should be extended to more general marked nets than condition/event systems. On the other hand, having established a solid bridge between net and graph grammar theory, one may wonder whether the respective concepts and results can be carried over and applied mutually—with success and new insight.

Final Example 7.2. Consider the (complete) conflict-free process in Fig. 17 and note that it would look even more horrible if more than two direct derivations would be

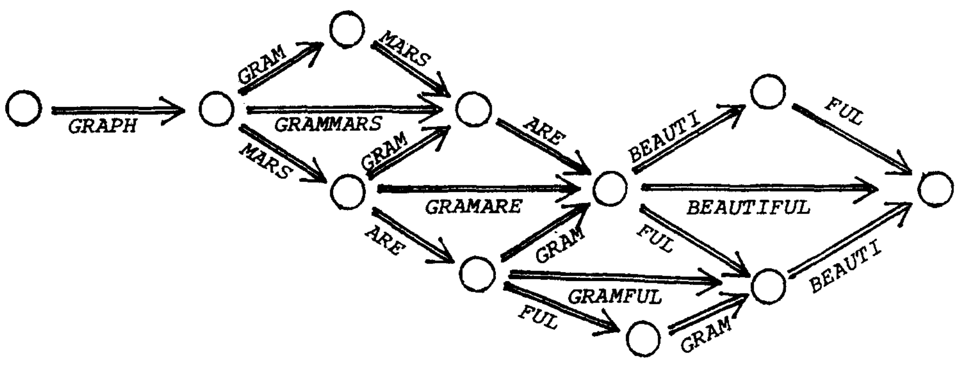

Fig. 17. Complete conflict-free process.

parallel-independent. The mess of edges tends to muddle all information. What can be done?

In some circles, a popular discussion takes place around "*Small is beautiful*" (cf. [10]). We would like to call attention to a slightly different topic which becomes apparent if you have a look at the equivalent canonical derivation of the given process in Fig. 18.

$$\begin{tikzcd}
\bigcirc \arrow[r, Rightarrow, "GRAPH"'] & \bigcirc \arrow[r, Rightarrow, "GRAMMARS"'] & \bigcirc \arrow[r, Rightarrow, "ARE"'] & \bigcirc \arrow[r, Rightarrow, "BEAUTIFUL"'] & \bigcirc
\end{tikzcd}$$

Fig. 18. Canonical derivation.

Appendix A

In this section we recall some basic notions, definitions, constructions and results concerning graphs as far as they are needed in this paper. For further details see, e.g., [3, 1].

(1) Let $C = (C_V, C_E)$ be a pair of *colour* alphabets for graphs, consisting of a colour alphabet C_V for vertices and a colour alphabet C_E for edges.

(2) A (*directed labelled*) *graph* $M = (V, E, s, t, l, m)$ consists of
- a set of *vertices* V,
- a set of *edges* E,
- two functions $s, t: E \to V$ assigning *source* and *target* to each edge, and
- two functions $l: V \to C_V$ and $m: E \to C_E$ *colouring* vertices and edges of the graph respectively.

The components of a graph M are referred to by indexing them with the name of the graph, i.e., by V_M, E_m, s_M,, t_M, l_M, m_M. The denotation $x \in M$ is an abbreviation for $x \in V_M$ and $x \in E_M$ and is used whenever nodes and edges have not to be distinguished. Given graphs M and N, we consider frequently the set-theoretic differences $V_M - V_N$ and $E_M - E_N$. For this pair of sets we write somewhat ambiguously $M-N$. A graph M is called *node-labelled* if C_E is a one-element set. A graph M is called *edge-labelled,* if C_V is a one-element set. A graph M is *unlabelled* if C_V and C_E are one-element sets. In case of unlabelled graphs we omit the colouring functions l_M and m_M.

(3) Let $M=(V, E, s, t, l, m)$ be a graph. A sequence of edges $e_1, \ldots, e_n$ is a *path* if $t(e_i)=s(e_{i+1})$ for $i=1, \ldots, n-1$. A path $e_1, \ldots, e_n$ is a *cycle* if $t(e_n)=s(e_1)$. A graph M is *acyclic* if no path in M is a cycle. A graph M is *connected* if, for each two nodes $v, v' \in V$, there is a sequence of nodes $v_1, \ldots, v_n$ such that $v=v_1$ and $v'=v_n$ and, for $i=1, \ldots, n-1$, there is an edge e_i with $s(e_i)=v_i$ and $t(e_i)=v_{i+1}$, or $t(e_i)=v_i$ and $s(e_i)=v_{i+1}$.

(4) Let U and M be graphs with $V_U \subseteq V_M$ and $E_U \subseteq E_M$. U is a *subgraph* of M, denoted by $U \subseteq M$, if $s_U(e)=s_M(e)$, $t_U(e)=t_M(e)$, $l_U(v)=l_M(v)$, and $m_U(e)=m_M(e)$ for all $e \in E_U$ and $v \in V_U$. A subgraph U of M is called *induced by* $V \subseteq V_M$ if $V_U = V$ and $E_U = \{e \in E_M \mid s_M(e), t_M(e) \in V\}$. A subgraph U of M is called *induced by* $V \subseteq V_M$ *and* $E \subseteq E_M$ if $V_U = V$ and $E_U = E$. Let M be a graph and $U_1, U_2 \subseteq M$. Then $V_{U_1} \cap V_{U_2}$ and $E_{U_1} \cap E_{U_2}$ induce a subgraph of M, called *intersection* of U_1 and U_2, and is denoted by $U_1 \cap U_2$.

(5) Let M and N be graphs. $M+N$ denotes te *disjoint union* of the graphs M and N, and is given by the disjoint union of sets for nodes and edges separately. All nodes and edges of $M+N$ keep their original labels, each edge keeps its original source and target.

(6) Let L and M be graphs. A *graph morphism* $g: L \to M$ from L to M consists of two maps $g_V: V_L \to V_M$ and $g_E: E_L \to E_M$ with $g_V(s_L(e))=s_M(g_E(e))$, $g_V(t_L(e))=t_M(g_E(e))$, $l_L(v)=l_M(g_V(v))$, and $m_L(e)=m_M(g_E(e))$ for all $e \in E_L$ and $v \in V_L$.

(7) Let $g=(g_V, g_E): M \to N$ be a graph morphism. The subsets $g_V(V_M) \subseteq V_N$ and $g_E(E_M) \subseteq E_N$ induce the subgraph $g(M)$ of N, called the *image* of M under g. The denotation $g(x)$ is an abbreviation for $g_V(x)$ or $g_E(x)$ and is used whenever nodes and edges have not to be distinguished. If g is a bijective graph morphism, then it is called a *graph isomorphism.* In this case M and N are called *isomorphic,* which is denoted by $M \cong N$. U is a subgraph of M if and only if the two inclusions $\mathrm{in}_V: V_U \to V_M$ and $\mathrm{in}_E: E_U \to E_M$ define a graph morphism $\mathrm{in}: U \to M$.

Let $h: N \to P$ be another graph morphism. The *composition* $h \circ g: M \to P$, defined componentwise, is a graph morphism. Let $U \subseteq M$ and $\mathrm{in}: U \to M$ be the corresponding inclusion morphism. Then $g \circ \mathrm{in}: U \to N$ is called the *restriction* of g to U.

Let $g': M' \to N'$ be another graph morphism. Then $g+g': M+M' \to N+N'$ denotes the *disjoint union* of the graph morphisms g and g', and is defined by $g+g'(x)=g(x)$ for $x \in M$ and $g+g'(x)=g'(x)$ for $x \in M'$.

(8) Let B, K, D be three graphs, and let $K \subseteq B$ and $d: K \to D$ be a graph morphism. Then the *d-gluing of B and D along K* is the graph M, constructed as follows:

$$V_M = V_D + (V_B - V_K);$$

$$E_M = E_D + (E_B - E_K);$$

$$s_M : E_M \to V_M \quad \text{with } s_M(e) = \begin{cases} s_D(e) & \text{for } e \in E_D, \\ s_B(e) & \text{for } e \in E_B - E_K \\ & \quad \text{with } s_B(e) \in V_B - V_K, \\ d_V(s_B(e)) & \text{for } e \in E_B - E_K \\ & \quad \text{with } s_B(e) \in V_K; \end{cases}$$

$$t_M : E_M \to V_M \quad \text{with } t_M(e) = \begin{cases} t_D(e) & \text{for } e \in E_D, \\ t_B(e) & \text{for } e \in E_B - E_K \\ & \quad \text{with } t_B(e) \in V_B - V_K, \\ d_V(t_B(e)) & \text{for } e \in E_B - E_K \\ & \quad \text{with } t_B(e) \in V_K; \end{cases}$$

$$l_M : V_M \to C_V \quad \text{with } l_M(v) = \begin{cases} l_D(v) & \text{for } v \in V_D, \\ l_B(v) & \text{otherwise;} \end{cases}$$

$$m_M : E_M \to C_E \quad \text{with } m_M(e) = \begin{cases} m_D(e) & \text{for } e \in E_D, \\ m_B(e) & \text{otherwise.} \end{cases}$$

(9) Let M be the d-gluing of B and D along K. Then $D \subseteq M$ due to construction. Moreover, there is a graph morphism $h : B \to M$ defined by

$$h(x) = \begin{cases} d(x) & \text{for } v \in K, \\ x & \text{otherwise.} \end{cases}$$

Acknowledgment

We would like to thank P.S. Thiagarajan, Annegret Habel, and Udo Hummert for their valuable comments on an earlier version of this paper and the unknown referee for his (or her) clarifying suggestions.

References

[1] H. Ehrig, Introduction to the algebraic theory of graph grammars, *Lecture Notes in Computer Science* 73 (Springer, Berlin, 1979) 1-69.

[2] H.J. Genrich, D. Janssens, G. Rozenberg and P.S. Thiagarajan, Generalized handle grammars and their relation to Petri nets, Tech. Rept. 82-01, University of Antwerp, 1982.

[3] H.-J. Kreowski, Manipulationen von Graphmanipulationen, Ph.D. Thesis, Computer Science Department, Technical University of Berlin, 1977.

[4] H.-J. Kreowski, A comparison between Petri nets and graph grammars, *Proc. Internat. Workshop WG '80*, Bad Honnef, June 1980, Lecture Notes in Computer Science **100** (Springer, Berlin, 1980) 306-317.

[5] H.-J. Kreowski, Graph grammar derivation processes, *Proc. Internat. Workshop WG '83*, Osnabrück, June 1983 (Trauner Verlag, Linz, 1983) 136-150.

[6] H.-J. Kreowski and A. Wilharm, Processes on Petri nets and graph grammars, *Proc. Internat. Workshop WG '84*, Berlin, June 1984 (Trauner Verlag, Linz, 1984) 189-200.

[7] M. Nagl, *Graph-Grammatiken: Theorie, Implementierung und Anwendungen* (Vieweg Verlag, Braunschweig-Weisbaden, 1979).

[8] W. Reisig, An application of graph grammars to net theory, *Proc. Internat. Workshop WG '80,* Bad Honnef, June 1980, Lecture Notes in Computer Science **100** (Springer, Berlin, 1980) 318-325.

[9] W. Reisig, Netze—Eine Einführung (Springer, Berlin, 1982).

[10] E.F. Schumacher, *Small is beautiful* (London, 1976).

[11] P. Starke, Graph grammars for Petri net processes, *Elektron. Informationsverarb. Kybernet.* **19**(4, 5) (1983) 199-233.

[12] P. Wilke, Zusammenhänge und Unterschiede zwischen Graph-Grammatiken und Petri-Netzen sowie verwandter Systeme, Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung **16**(14) Universität Erlangen, 1983.