PRODUCTION NETWORKS AS COMMUNITIES OF AUTONOMOUS UNITS AND THEIR STABILITY

Sergey Dashkovskiy[1], Hans-Jörg Kreowski[2], Sabine Kuske[2], Andrii Mironchenko[1],
Lars Naujok[1], and Caroline von Totth[2]

[1]AWG Mathematical Modeling of Complex Systems
Department for Mathematics and Computer Science, University of Bremen
P.O. Box 330 440, D-28334 Bremen, Germany
e-mail: {dsn,andmir,larsnaujok}@math.uni-bremen.de
[2] Research Group Theoretical Computer Science
Department for Mathematics and Computer Science, University of Bremen
P.O. Box 330 440, D-28334 Bremen, Germany
e-mail: {kreo,kuske,caro}@informatik.uni-bremen.de

Abstract: In this paper, a discrete variant of production networks is considered. Besides the mathematical models in terms of matrices and vectors, production networks are modeled as communities of autonomous units in a rule-based, graph-transformational and visual manner. Moreover, a sufficient criterion for the stability of a production network is given where stability means that there exist suitable storage capacities at the production sites that never flow over.

AMS Subject Classification: 39A30, 90B10, 90B30, 90C35
Key Words and Phrases: Graph transformation, autonomous units, production networks, production models, deterministic network models, stability

1. Introduction

In this paper, we consider a discrete variant of production networks (see, e.g., Wiendahl et al. [1]) inspired by the work in Scholz-Reiter et al. [2], Görges et al. [3] and Dashkovskiy et al. [4] on continuous production networks and their stability. For a certain scenario it has been shown that the application of local autonomous control methods on integrated production and transport processes improves the handling of internal and external dynamics. A production network in this scenario consists of production sites, which are represented as nodes, and of links between sites, which are represented as directed edges. There is a particular input site with a continuous inflow. The production at each site runs continuously at some rates that are bounded by the maximum production rates and subject to suitable constraints. The processed product of each site is continuously distributed to the direct neighbors for further processing according to fixed distribution rates. Moreover, there is an output site with a continuous output which is computed in some suitable way. A production network is called stable if the quantity of products at each site is bounded all the time. In [4] conditions were derived by mathematical systems theory, which guarantee stability of the network. The calculation of these conditions is based on the work of Dashkovskiy et al. in [5], [6], and [7].

In the present paper, the continuity is replaced by stepwise input, production, transportation, and output. To take into account dynamic changes of the input, the input flow is not assumed to be constant. Moreover, the production rates are not determined uniquely, but may vary within certain bounds (Section 3). The discrete production networks are modeled in a graph-transformational way as communities of autonomous units (cf. Hölscher et al. [8], [9] and Kreowski et al. [10], [11]) in Section 4. These rule-based and visual models allow one to use graph-transformational tools like GrGen.NET to simulate production networks in such a way that production processes are not only statistically analyzed, but also visualized displaying their smooth running or the overflow of bottlenecks (Section 6). Each production site becomes a unit that can act independently of the other sites within certain bounds. This allows one to use decentralized decision criteria for the choice of the production and distribution rates; however, this aspect will not be further addressed in the paper, being a subject of future work.

If the input rate is constant and the production rates are chosen exhaustively, meaning that the current quantities are processed completely up to the maximum production rates in each step, then the production network becomes deterministic with a unique production process. In this case, the distribution rates and the input rate induce a system of linear equations. If this linear system is solvable, then the production network turns out to be stable, as shown in Section 7. As this result applies to the graph-transformational model of production systems, the investigation introduces a new kind of analytical problems to the area of graph transformation that may be of interest beyond the topic of this paper. The question of stability is of similar interest in the discrete case as in the continuous one, because the quantities left at the production sites may grow beyond any bound so that their storage can overflow eventually.

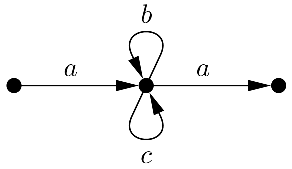


Figure 1. An edge-labeled directed graph G_0

Summarizing, the paper is structured in the following way. After the preliminaries of graph transformation as used in the paper, the discrete variant of production networks and processes is introduced in Section 3. The visual and rule-based models of production networks are specified in Section 4 in form of communities of autonomous units while a sample run is illustrated in Section 5. Section 6 describes an implementation of our production networks in the graph transformation engine GrGen.NET that gives some first ideas of the potentials of the visual modeling. In Section 7, a sufficient condition for the stability of deterministic production networks is given based on the solvability of a system of linear equations which is induced by the distribution rates and the input quantity. Section 8 concludes the paper.

2. Preliminaries

In this section, we recall the basic notions and notations of graph transformation as they are used in the paper.

2.1. Graphs. We consider the class of edge-labeled directed graphs. More precisely, let Σ be a set of labels. An *edge-labeled directed graph* over Σ is a system $G = (V, E, s, t, l)$, where V is a set of nodes, E is a set of edges, $s, t\colon E \to V$ are the *source* and *target mappings* which assign to each edge its source and target node, respectively, and $l\colon E \to \Sigma$ is a mapping assigning a label to each edge in E.

Figure 1 shows an example of a graph G_0 consisting of three nodes v_1,v_2 and v_3 (from left to right), an a-labeled edge from v_1 to v_2, a b- as well as a c-loop at v_2, and finally an a-labeled edge from v_2 to v_3.

The set of all graphs over Σ is denoted by $\mathcal{G}_\Sigma$. Loops are allowed, as are parallel edges. Moreover, a special symbol $*$ is reserved to mimic unlabeled edges and therefore omitted in drawings of graphs.

Let $G, H \in \mathcal{G}_\Sigma$ be graphs. A *graph morphism* $g\colon G \to H$ is a pair of structure-preserving mappings $g_V\colon V_G \to V_H$ and $g_E\colon E_G \to E_H$, i.e., $g_V(s_G(e)) = s_H(g_E(e))$, $g_V(t_G(e)) = t_H(g_E(e))$, and $l_H(g_E(e)) = l_G(e)$ for all $e \in E_G$. The image $g(G) \subseteq H$ is called a *match* of G in H.

Given a graph morphism $g\colon G \to H$ with inclusions g_V and g_E, G is called a *subgraph* of H, denoted by $G \subseteq H$.

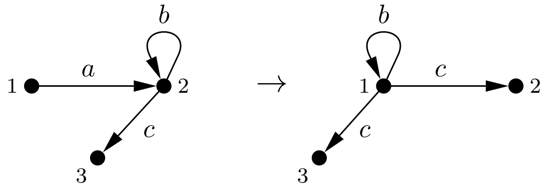


Figure 2. A graph transformation rule

2.2. Graph transformation rules. A *graph transformation rule* r is defined as $(L \supseteq K \subseteq R)$ where $L, K, R \in \mathcal{G}_\Sigma$. The graphs L, K and R are called *left-hand side*, *gluing graph* and *right-hand side* respectively. Rules are depicted in the form $L \to R$ where the nodes and edges of the gluing graph K are indicated by identical positions or by small numbers beneath the nodes.

An example of a graph transformation rule is given in Figure 2. The gluing graph consists of the numbered nodes 1, 2 and 3.

2.3. Rule application. An application of a rule r to a *host graph* G consists of the following steps:

(1) A graph morphism $g : L \to G$ is selected subject to the following two *application conditions*:
   (a) the *dangling condition*: the removal of $g(L) - g(K)$ from G yields no dangling edges, and
   (b) the *identification condition*: if two nodes or two edges of L are identified (i.e., mapped to the same graph element) in the match of L, they must be in K.

(2) $g(L) - g(K)$ is removed from G, yielding the graph Z.

(3) R is added to Z yielding H by merging K with $g(K)$.

The dangling condition guarantees that removing $g(L) - g(K)$ from G yields a subgraph. For example, the rule in Figure 2 can be applied to the graph G_0 in Figure 1 by choosing g such that node 1 is mapped to v_1, and nodes 2 and 3 are mapped to v_2, with the consequence that the c-edge is mapped to the c-loop (because its source and target are identified by g). The a-edge and its source are mapped to the a-edge and its source in G_0, respectively. Obviously, g satisfies the dangling condition because the removal of the three edges from G_0 does not produce any dangling edges. The identification condition is also satisfied, because the identified nodes 2 and 3 belong to the gluing graph.

An application of a rule r to G resulting in H is also called *direct derivation* and denoted as $G \underset{r}{\Longrightarrow} H$. Figure 3 shows a derivation where the rule of Figure 2 is applied to G_0. This way of transforming graphs

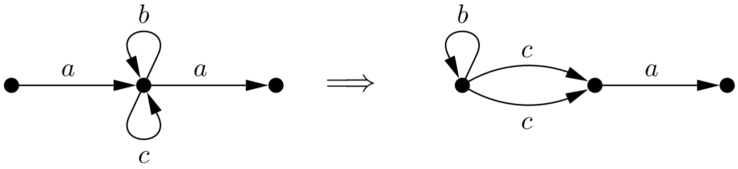


Figure 3. A derivation using the rule in Figure 2

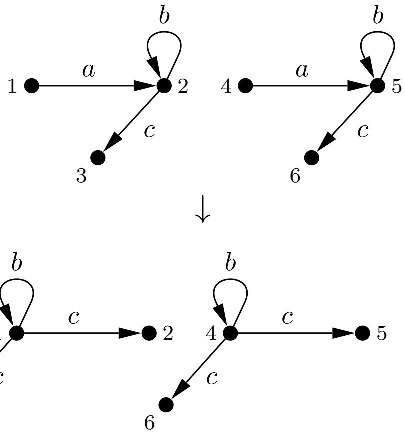


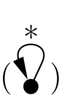
Figure 4. A parallel graph transformation rule

is based on the *double pushout approach* (see Corradini et al. [12] and Ehrig et al. [13], where a more detailed definition of rule application can be found).

Let $G_i \in \mathcal{G}_\Sigma$, $i \in \mathbb{N}$. The iteration $G_0 \underset{r_1}{\Longrightarrow} G_1 \underset{r_2}{\Longrightarrow} \cdots \underset{r_n}{\Longrightarrow} G_n$ of direct derivations is called a derivation from G_0 to G_n and may also be denoted as $G_0 \overset{r_i}{\underset{P}{\Longrightarrow}} G_n$ where $r_i \in P$, or as $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G_n$ if the number of direct derivations is of no consequence. The *application sequence* of this derivation is given by the string $r_1 \cdots r_n$. For rules $r_1, \ldots, r_n$ with $r_i = (L_i \supseteq K_i \subseteq R_i)$, their *parallel composition* $r_1 + \cdots + r_n$ yields the rule $(L_1 + \cdots + L_n \supseteq K_1 + \cdots + K_n \subseteq R_1 + \cdots + R_n)$ where $+$ denotes the disjoint union of graphs and the inclusions are the natural extensions of the inclusions in the rules $r_1, \ldots, r_n$.

For example, the parallel rule $r + r$, where r is the rule of Figure 2, corresponds to the rule in Figure 4. This rule cannot be applied to the graph G_0 because the only morphism from the left-hand side to G_0 violates the identification condition (for example, the a-edges would have to be identified by the morphism, but are not part of the gluing graph).

A *graph class expression* is a syntactic entity X, the semantics $\mathit{SEM}(X) \subseteq \mathcal{G}_\Sigma$ of which is a set of graphs. An example for a graph class expression would be

$\mathit{noloops} = \mathit{forbidden}($  $)$

where $\mathit{SEM}(\mathit{noloops})$ describes the subset of $\mathcal{G}_\Sigma$ which contains only graphs without unlabeled loops.

A *control condition* C is a syntactic entity that specifies a set of graph sequences. Control conditions cut down the nondeterminism of rule application, which would otherwise often deliver unwanted results.

A *community of autonomous units* consists of a set of autonomous units, a control condition and, in our case, a single initial environment graph. An autonomous unit provides a set of rules and a control condition. In the general case of autonomous units, the units and the community may have goals, which are not needed in this paper. Every transformation process of a community starts with the initial environment graph, on which units then act and interact according to the control condition of the community (see Kreowski et al. [11]).

2.4. Further notions and notations. The set of natural numbers is denoted by $\mathbb{N}$, $\mathbb{N}\backslash\{0\}$ is denoted by $\mathbb{N}_{>0}$ and $[k]$ denotes the subset $\{1, \ldots, k\}$ of $\mathbb{N}$. The set of real numbers is denoted by $\mathbb{R}$; we use $\mathbb{R}_+$ to describe the set of non-negative real numbers with 0. Moreover, the set of mappings from a set X to a set Y is denoted by $\langle X, Y\rangle$.

3. Production Networks and Production Processes

In this section, the notion of production networks and their processes is introduced where the input, processing, flow and output of material are not continuous, but happen step-by-step. A production network consists of production sites, which are represented as nodes, and of transportation channels between sites, which are represented as directed edges. There is one input site and one output site. In each state, the present material at each site is given as a quantity. A production step changes these quantities by distributing the production rate of each site to its direct neighbors. Moreover, the input site gets some input in each step and a part of the production rate of the output site is put out. There may be a maximum input rate, which can be chosen as ∞ if no bound is assumed. The same applies to the production rates. The distribution at a site is done according to a distribution vector, the entries of which specify which fraction of the production rate is moved to which neighbor site. The distribution vectors of all sites form a distribution matrix. A production process starts with the initial site quantities and records the changes of site quantities depending on the input, the production rates and the distribution matrix in each step.

Definition 1 (Production Network). A *production network PN* consists of

- a simple directed graph $G = ([n+1], E)$ with $E \subseteq [n+1] \times [n+1]$, the *input site* 1 and the *output site* $n+1$,
- an *initial site quantity* $q(0)\colon [n+1] \to \mathbb{R}_+$,
- a *maximum input rate* $maxin \in \mathbb{R}_+ \cup \{\infty\}$,
- a *maximum production rate* $max\colon [n+1] \to \mathbb{R}_+ \cup \{\infty\}$, and
- a *distribution matrix* $d\colon [n+1] \times [n+1] \to \mathbb{R}_+$ with $\sum\limits_{j \in [n+1]} d(i,j) = 1$ for $i \in [n]$, $\sum\limits_{j \in [n+1]} d(n+1, j) \leq 1$ and $d(i,j) = 0$ for all $(i,j) \in [n+1] \times [n+1] - E$.

3.1. Example. A sample production network is given by the components of Figure 5.

A production network specifies stepwise production processes which go on forever. If one assumes that the input rates and the production rates can be randomly chosen (within certain limits) and that the output is always the part of the production rate of the output site which is not distributed to other sites, then one gets the following production processes.

Definition 2 (Production Process). A *production process pp* (in *PN*) consists of

- an infinite sequence of *input rates in*: $\mathbb{N}_{>0} \to \mathbb{R}_+$,
- an infinite sequence of *production rates p*: $\mathbb{N}_{>0} \to \langle [n+1], \mathbb{R}_+ \rangle$,
- an infinite sequence of *output rates out*: $\mathbb{N}_{>0} \to \mathbb{R}_+$, and
- an infinite sequence of *site quantities q*: $\mathbb{N}_{>0} \to \langle [n+1], \mathbb{R}_+ \rangle$

subject to the following *production process conditions* for all $k \in \mathbb{N}_{>0}$:

$$in(k) \leq maxin, \tag{1}$$

$$p(k) \leq \min(q(k), max), \tag{2}$$

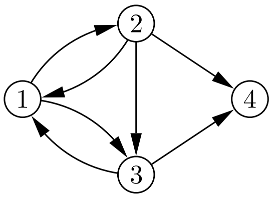
$$q(k)(j) = \begin{cases} in(k) + y & \text{if } j = 1 \\ y & \text{otherwise} \end{cases} \tag{3}$$

$$\text{where } y = \left(\sum_{i \in [n+1]} d(i,j) \cdot p(k)(i) \right)$$

SAMPLE

Graph:

$G =$



Distribution matrix:

$$d = \begin{pmatrix} 0 & .5 & .5 & 0 \\ .25 & 0 & .5 & .25 \\ .75 & 0 & 0 & .25 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Maximum input rate *maxin*:

$$maxin = 1$$

Maximum production rate *max* and initial quantity $q(0)$:

$$max = \begin{pmatrix} 3.2 \\ 1.6 \\ 2.4 \\ 1 \end{pmatrix} \quad q(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 5. An example of a production network

$$+ q(k-1)(j) - p(k)(j),$$

$$out(k) = \left(1 - \sum_{j \in [n+1]} d(n+1, j)\right) \cdot p(k)(n+1). \tag{4}$$

The first condition makes sure that no input rate exceeds the maximum input rate. The second condition requires that the processed quantity in every step is a part of the present quantity. The third condition describes the present site quantity after every step as the site quantity before the step diminished by the production rate and expanded by the quantities moved from other sites. The latter quantities are given by the fractions of the production rates due to the distribution factors. The input site gains the input rate in addition. The last condition fixes the output in each step as the part of the production rate of the output site that is not distributed to other sites.

To get shorter formulas, we use the following notational conventions for $k \in \mathbb{N}, l \in \mathbb{N}_{>0}$ and $i, j \in [n+1]$.

(1) $in_l = in(l); p_{il} = p(l)(i); d_{ij} = d(i,j); out_l = out(l); q_{il} = q(l)(i);$ $max_i = max(i)$,

(2) $In_k = \sum\limits_{j \in [k]} in_j$, $Out_k = \sum\limits_{j \in [k]} out_j$, $Q_k = \sum\limits_{i \in [n+1]} q_{ik}$ where the convention $[0] = \emptyset$ and $\sum\limits_{j \in \emptyset} = 0$ for $k = 0$ is used.

The definition of production processes has some immediate consequences:

(1) If *in* and p are given in some way, then q and *out* are uniquely determined and can be computed due to the required equations.
(2) As a particular case, one may consider constant input rates, e.g., $in_k = maxin$ for all $k \in \mathbb{N}_{>0}$.
(3) As a particular case, one may consider exhaustive production rates, e.g., $p_{ik} = \min(q_{i(k-1)}, max_i)$ for all $i \in [n+1]$ and $k \in \mathbb{N}_{>0}$.
(4) Production networks with constant input rate and exhaustive production rates have a unique production process. They are further discussed in Section 7.

3.2. Example. The production network **SAMPLE** in Figure 5 may run with constant maximum input rate and exhaustive production rates. Then there is a unique production process with output rates computed due to the respective constraints. It is not difficult to show that the site quantities of this production process never exceed the maximum production rates, so that the production rates always coincide with the site quantities.

3.3. Lossfreeness. The production process conditions (see Definition 2) guarantee that no input material gets lost, because the whole processed material is moved to other sites and put out in every step. This is formally stated in the following result.

Theorem 1. *$Q_k = Q_0 + In_k - Out_k$ for all $k \in \mathbb{N}$.*

Proof. The result is proved by induction over k.

Induction base:

$$Q_0 = Q_0 + 0 - 0 = Q_0 + \sum_{j\in[0]} in_j - \sum_{j\in[0]} out_j = Q_0 + In_0 - Out_0.$$

Induction step:

$$\begin{aligned}
Q_{k+1} &= \sum_{j\in[n+1]} q_{j(k+1)} \\
&= in_{k+1} + (\sum_{j\in[n+1]} ((\sum_{i\in[n+1]} d_{ij} \cdot p_{i(k+1)}) + q_{jk} - p_{j(k+1)}) \\
&= in_{k+1} + \sum_{j\in[n+1]} q_{jk} + \sum_{i,j\in[n+1]} d_{ij} \cdot p_{i(k+1)} - \sum_{j\in[n+1]} p_{j(k+1)} \\
&\underset{\text{Ind.}}{=} in_{k+1} + Q_0 + In_k - Out_k + \sum_{i\in[n+1]} (\sum_{j\in[n+1]} d_{ij}) \cdot p_{i(k+1)}
\end{aligned}$$

C(PN)

aut:	*input*, *j-prod*, for $j \in [n+1]$
init:	$env(PN)$
control:	$(input \mid\mid \textit{1-prod} \mid\mid \ldots \mid\mid \textit{n+1-prod}\ )^{\infty}$

Figure 6. The community C(PN) models the production network PN

$$
\begin{aligned}
&- \sum_{j\in[n+1]} p_{j(k+1)} \\
&= Q_0 + In_{k+1} - Out_k + \sum_{i\in[n]} 1 \cdot p_{i(k+1)} - \sum_{j\in[n+1]} p_{j(k+1)} \\
&\quad + (\sum_{j\in[n+1]} d_{(n+1)j}) \cdot p_{(n+1)(k+1)} \\
&= Q_0 + In_{k+1} - Out_k - (1 - \sum_{j\in[n+1]} d_{[n+1]j}) \cdot p_{(n+1)(k+1)} \\
&= Q_0 + In_{k+1} - Out_k - out_{k+1} = Q_0 + In_{k+1} - Out_{k+1}.
\end{aligned}
$$

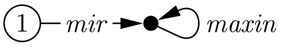
□

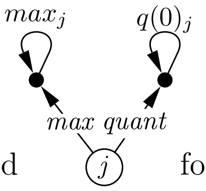
4. Production Networks as Communities of Autonomous Units

In this section, production networks are modeled as communities of autonomous units in such a way that the production processes of a network correspond to the runs of the respective community. At first sight, the production community may look more complicated than the mathematical model. But it is worth noting that the rule applications in a running step correspond directly to the multiplications and additions that define a process step in the mathematical model. Moreover, the rule-based model provides an explicit description of parallel computations. The main difference between both models is that the rule-based version provides a visual level.

4.1. The community C(PN). A production network PN is transformed into the community of autonomous units in Figure 6. There is an autonomous unit *j-prod* for each production site $j \in [n+1]$ and an extra *input* unit.

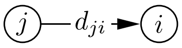
4.2. Initial environment. The initial environment graph $env(PN)$ integrates all the information of PN. The graph $env(PN)$ consists of the subgraphs

G_d, 

, and 

for $j \in [n+1]$

where G_d is obtained from G by replacing each $(j, i) \in E$ by



and the subgraphs share the nodes in $[n+1]$. Moreover, we assume that each node $j \in [n+1]$ is attached with a loop labeled by j. In drawings, the loop is omitted and its label is placed inside the node. This ensures that the node j can only be mapped to itself by a graph morphism.

In summary, the node representing the input site is labeled with the number 1; the output site is labeled with n+1. Each site j has an initial quantity q_{j0} of material, indicated by a pointer edge with the label *quant*, and a maximum production rate, denoted by a pointer with the label *max*.

The maximum input rate *maxin* is attached to the input site by a special pointer labeled *mir*. The connecting edges between sites are labeled with the distribution rates.

Additionally, if the initial quantities q_{j0} are replaced by some quantities q_j for $j \in [n+1]$, then the environment graph is denoted by $env(PN)(q)$.

A sample of such a community is discussed in Section 5, with the initial environment displayed in Figure 9.

4.3. Autonomous unit *input*. The unit *input* in Figure 7 has only one rule, which is applied exactly once in every execution step of the community. An input value *in* is chosen by some mechanism (e.g., this may be some input function like *sine* or some stochastic function or even a constant), and a *gain* pointer is added to the site, with the value of *in* attached to it by a loop edge. The choice of *in* is restricted only insofar that its value may not exceed *maxin*.

4.4. Autonomous units *j-prod*. The parallel execution of the *j-prod* units (Figure 8) together with the unit *input* model one production step of the network.

The tasks of the *j-prod* units are twofold: On one hand, they manage for each site the production and the distribution of material to neighbor

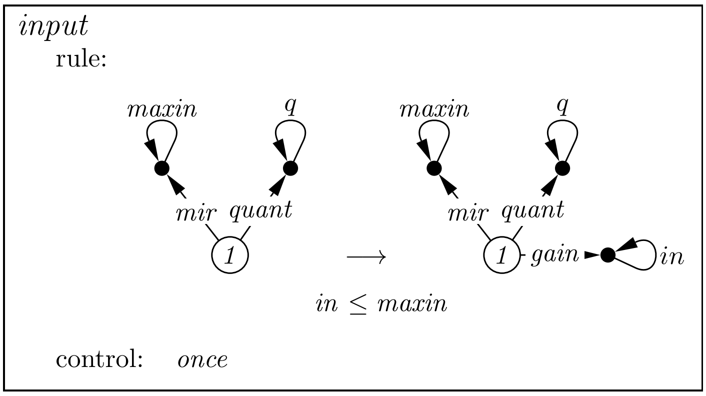


Figure 7. The unit *input*

sites. On the other hand, the fact that the distribution runs in parallel for all sites at once makes some cleanup actions necessary in preparation for the next step.

Production and distribution. The first rule of a *j-prod* unit, *produce*, chooses a production rate for site j much in the same manner as the *input* unit, by some mechanism. A *prod* pointer is added to j with the new production rate attached to it. Here, too, the choice of p is restricted by whichever is the smaller of two upper bounds: the quantity q of material present at the site and the maximal production rate max_j.

The second rule in *j-prod*, *transport*(i), is a parametric one and it is applied in one parallel step to each neighbor i of the site j. This rule moves the fraction $d_{ji} \cdot p$ of the current production rate of material at the site j to a neighboring site i, where d_{ji} is the distribution value inscribed on the edge from j to i. Rather than adding this value directly to the quantity of material already present at i, the transported value is instead attached to a *gain* pointer for the following reasons.

Cleanup and update. Depending on the configuration of the network, one production site j may receive input from many neighbor sites. To avoid conflicts generated by concurrent access to the quantity value q of j, each source generates a *gain* edge at the target site, labeled with the appropriate value. Now in order to make a next step in the production process possible, a cleanup of sorts needs to take place; this is the task of the rules *subtract* and *add*, which restore the original pointer configuration and update all values to reflect the changes that have taken place in the current production step. First, the *subtract* rule removes the amount p_j of material which is distributed by j in the current production step

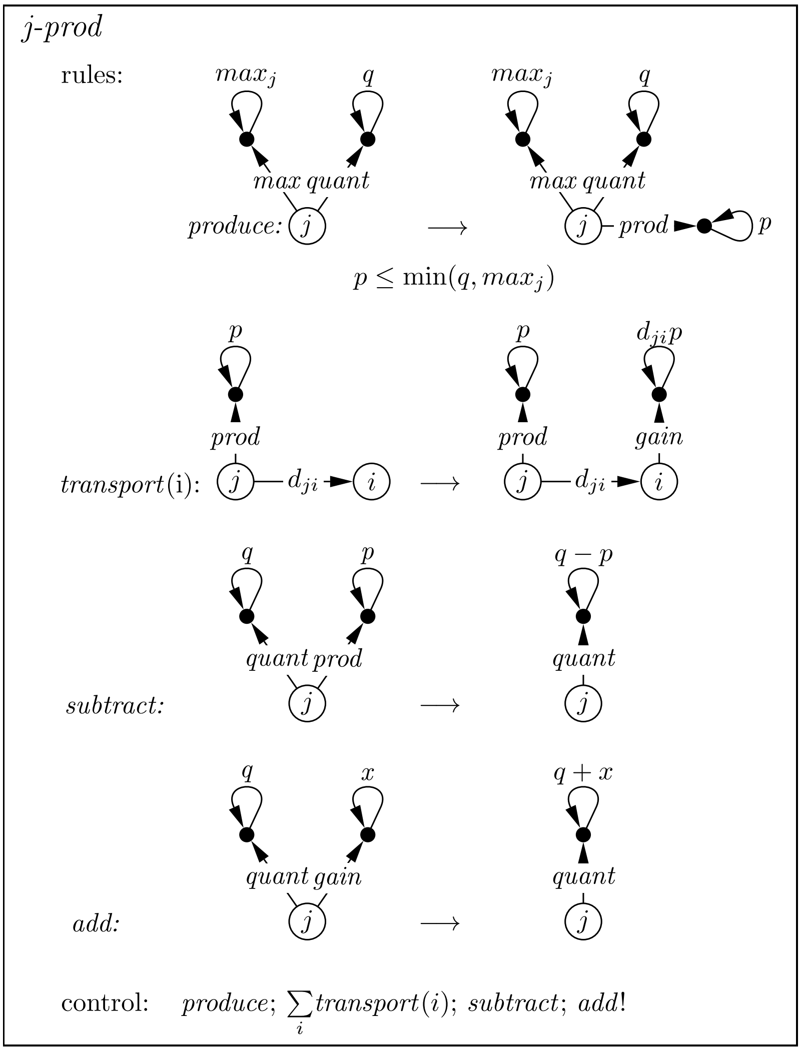


Figure 8. The unit *j-prod*

from the quantity q_j of material present at j, resulting in an intermediate quantity value $\bar{q}_j$. Implicitly, this behavior also models the output of material from the network: the difference between p_{n+1} and the amount of material the site $n+1$ sends to other sites simply disappears from the network. If, in particular, the output site has no outgoing edges, then its whole production rate leaves the network in every running step. Additionally, *subtract* also removes the *prod* pointer at each site, so that a new value for p_j can be entered into the network in the next production step. The rule *subtract* is applied exactly once, after the transport has been completed and before the application of *add*. Now, the values in the *gain* edges at each site plus the remaining quantity $\bar{q}_j$ at the site have to be consolidated into one single quantity value $q'_j = \sum d_{ij}p_i + \bar{q}_j$. This is done in the *add* rule by picking a random gain edge and adding its value to the existing quantity: the exclamation mark in the control expression *add*! requires that *add* must be applied as long as possible, i.e., until no *gain* edge remains.

4.5. Control condition. The control condition of $C(PN)$ prescribes to execute the *input* and all of the *j-prod* units in parallel (denoted by $\|$) and to iterate this ad infinitum (denoted by ∞).

Summarizing, the following observation relates a running step in the community with a process step in the mathematical model.

Observation. A running step of the community $C(PN)$ has the form $env(PN)(q) \Longrightarrow env(PN)(q')$ where q' is obtained from q by

$$q'_j = in \cdot \delta_{1j} + \sum d_{ij}p_i + q_j - p_j$$

with $\delta_{11} = 1$ and $\delta_{1j} = 0$ for $j > 1$.

As a consequence of this observation, we get the following result.

Theorem 2. *Each production process pp in PN with the sequence of site quantities* $q : \mathbb{N} \to \langle [n+1], \mathbb{R}_+ \rangle$ *corresponds to an infinite run of the community* $C(PN)$ *with the steps* $env(PN)(q_k) \Longrightarrow env(PN)(q_{k+1})$ *for all* $k \in \mathbb{N}$ *and conversely.*

This shows that $C(PN)$ models PN correctly.

5. Sample Run of a Production Community

The community C(**SAMPLE**) in Figure 9 is the graph-transformational counterpart of the production network **SAMPLE** from Section 3.2. The initial environment graph starts out with zero material at all sites and a maximum input rate of 1. The distribution values are displayed on the edges connecting the sites, and there are backflow edges leading from sites 2 and 3 to the input site. The input site has an upper bound of of 3.2 on the production rate, while at sites 2, 3 and 4 the production rates are bounded by 1.6, 2.4 and 1 respectively.

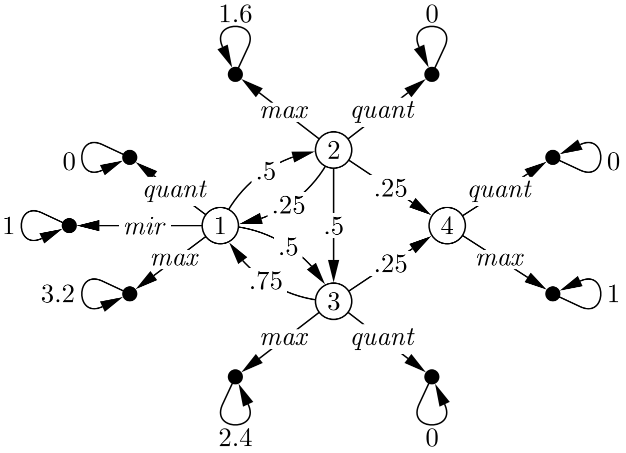
For the following example run, we assume that exhaustive rates are always chosen, and that the input rate coincides with the maximum input rate.

Step 1. Since all sites have zero material stored, in the very first step only the units *input* and 1-*prod* can do anything of note: *input* selects

C(**SAMPLE**)

aut: *input*, *j-prod*, for $j \in 1, \cdots, 4$

init: *env*(**SAMPLE**)=



control: (*input* || *1-prod* || ... || *4-prod*)$^\infty$

Figure 9. The community C(**SAMPLE**) models the production network **SAMPLE** from Section 3.2

the input rate *in* such that it does not exceed the maximum input rate of 1 and adds *in* amount of material to the input site as a *gain* pointer. None of the other sites have anything to distribute, since they started out with a load of zero material. Therefore, in this step the unit *j-prod* chooses the only possible production rate of 0 for all sites, distributes zero material to all neighbors, and then cleans up by removing all production edges and, at site 1, 1-*prod* adds the gained material of 1 to the existing quantity of 0 and removes the *gain* edge.

In the next few steps, material flows gradually through the network, while new material also flows into the network from outside in every step. Steps 2 and 3 are not shown explicitly. We will now have a detailed look at a more advanced step in the production process.

Step 4. The network in step 4, after the application of *input* in parallel with the choice of new production rates for every site (the first rule of *j-prod*), can be seen in Figure 10.

The next action, which also happens in parallel for all sites, is the distribution of material (displayed by way of example for site 3 in Figure 11). For example, site 3 has two outgoing and two incoming edges,

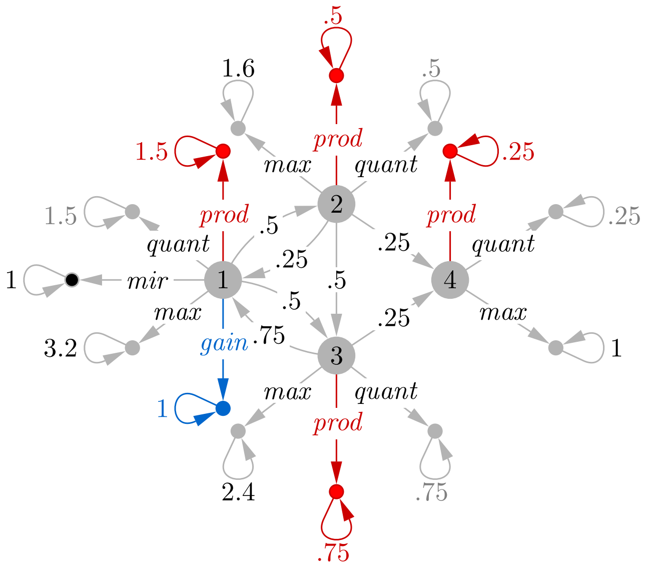


Figure 10. The community C(**SAMPLE**) in process step 4, after the application of *input*

therefore it sends 75 percent of its material to the input site 1, and the remaining 25 percent to the output site 4. At the same time, it receives half of the production rate of site 1, which amounts to a quantity of .75, and also half of the production rate of site 2, which is a quantity of .25 (nodes and edges unrelated to the distribution action are omitted from the picture).

Then the *subtract* rule is applied once, removing the distributed material from the quantity present at each site, and deleting the production rate pointer, so that a new one can be chosen in the next step. In this example, the production rate of site 4 is exactly the output rate, meaning that all material at 4 leaves the network in every step. The effect of *subtract* on site 3 can be seen in Figure 12.

Afterwards the *add* rule is applied as long as possible at every site until it has cleaned up all the *gain* edges. As can be seen in Figure 13, the mechanism is again illustrated for site 3 only, since it works the same at every site, and unrelated pointer edges are once again omitted from the illustration.

Together, these intermediate steps result in $env(PN)(q_4)$ (see Figure 14).

The following steps follow a similar pattern.

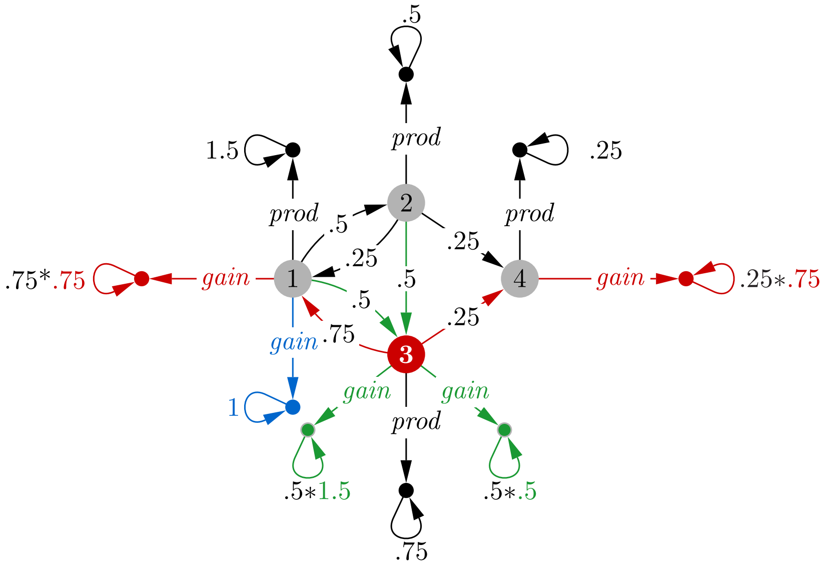


Figure 11. The community C(**SAMPLE**) in process step 4, after distribution of material from and to site **3**

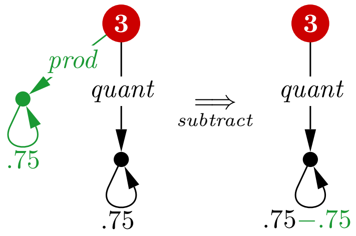


Figure 12. The community C(**SAMPLE**) in process step 4: the effect of the *subtract* rule on site **3**

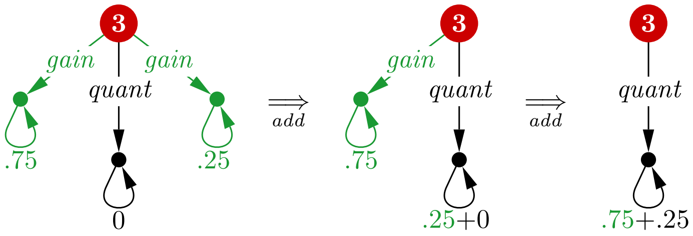


Figure 13. The community C(**SAMPLE**) in process step 4: the effect of the *add* rule on site **3**

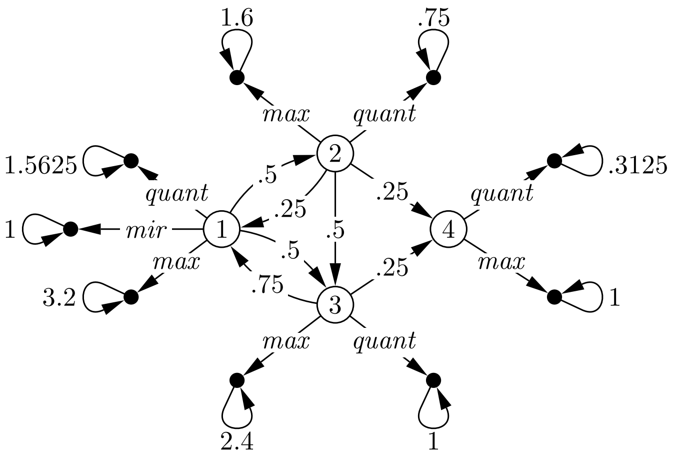
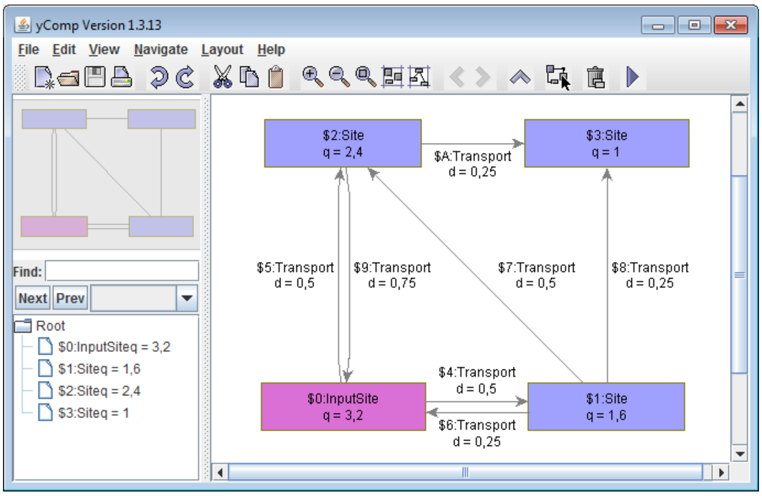


Figure 14. The community C(**SAMPLE**) after the completion of step 4



$0:InputSiteq = 3,2
$1:Siteq = 1,6
$2:Siteq = 2,4
$3:Siteq = 1

Figure 15. Community C(**SAMPLE**) in GrGen.NET: all sites have reached the saturation point (i.e., the maximal production rate) after 217 steps

6. Visual Simulation

In order to simulate runs on our sample production network as well as larger production networks, we have implemented the general production community from Section 4 using the graph transformation engine GrGen.NET (see [14]).

The GrGen.NET graph model is based on typed, attributed, directed multigraphs with inheritance. The base types at the core of this model are Node and Edge, and the primitive attribute data types int, float, double, string, boolean and object, the latter denoting a .NET object.

We made use of the subpattern matching capability of GrGen.NET, using the iterated subpattern in order to simulate parallel rule application. GrGen.NET also does not provide autonomous units; however, it allows to structure rule application by embedding imperative calls to other rules into the declarative right-hand-side of a rule. Furthermore, such calls may be controlled using, for example, regular expressions. We made use of this feature to emulate autonomous units very closely to our original specification.

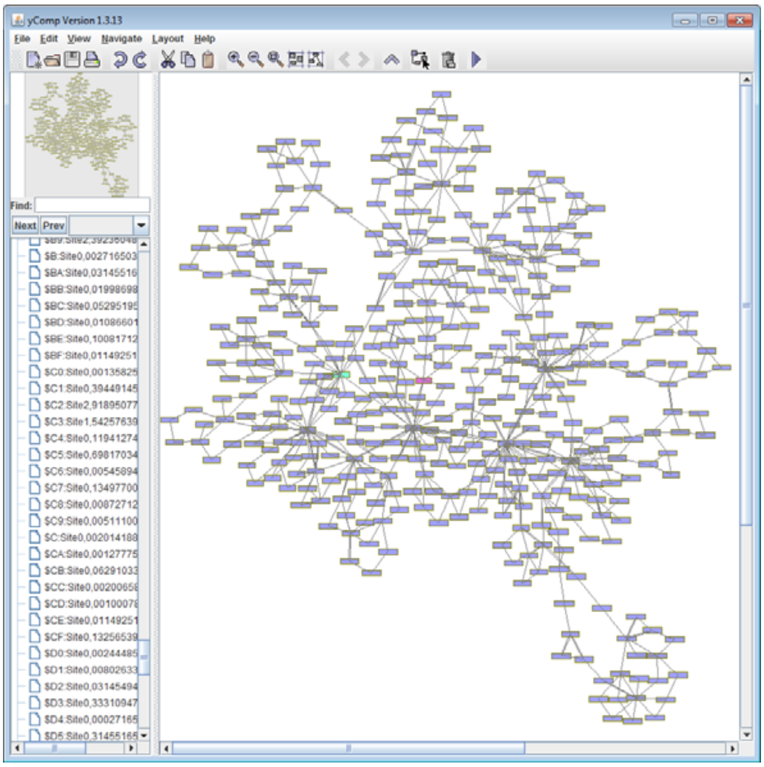
The simulation runs very fast, with our example network **SAMPLE** completing 217 steps and reaching the maximal production rate at all four sites in less than 1 millisecond (GrGen gives the time as 0 ms) on an Intel Core i5 M520 CPU with 2.40 GHz and 6 GB of RAM, having found 4340 matches and performed 4340 graph rule applications in that time.

In order to test run times on larger networks (Figure 16), we have written an additional graph grammar which creates random production networks for simulation purposes. A graph with 402 nodes is generated in 655 ms; 3000 production steps are completed after another 21840 ms (i.e., some 21 seconds), with over 4 million matches found and rewrite steps executed in that time.

The simulation is valuable because it is visual, it allows to model and debug production networks, or detect flaws in existing ones, altering them until they are stable. Additionally, the declarative nature of graph transformation rules makes the modeling less error-prone, and the production process model easily scalable, e.g., by introducing different material types, variable inflow and other extensions.

7. Deterministic Production Networks and Stability

In practice, a site in a production network has only a bounded storage capacity so that the question of stability becomes important. A production network is stable if the site quantities of each production process do not exceed a fixed bound. It will be shown in this section that deterministic production processes, which have a constant input rate and exhaustive production rates, are stable if a certain system of linear equations is solvable.



$B:Site0,002716503
$BA:Site0,03145516
$BB:Site0,01998698
$BC:Site0,05295195
$BD:Site0,01086601
$BE:Site0,10081712
$BF:Site0,01149251
$C0:Site0,00135825
$C1:Site0,39449145
$C2:Site2,91895077
$C3:Site1,54257639
$C4:Site0,11941274
$C5:Site0,69817034
$C6:Site0,00545894
$C7:Site0,13497700
$C8:Site0,00872712
$C9:Site0,00511100
$C:Site0,002014188
$CA:Site0,00127775
$CB:Site0,06291033
$CC:Site0,00200656
$CD:Site0,00100078
$CE:Site0,01149251
$CF:Site0,13256539
$D0:Site0,00244485
$D1:Site0,00802633
$D2:Site0,03145494
$D3:Site0,33310947
$D4:Site0,00027165
$D5:Site0,31455165

Figure 16. A network with 400 nodes in GrGen.NET after 3000 production steps

A production network PN is *stable* if an upper bound vector $m : [n+1] \to \mathbb{R}_+$ exists such that the following holds for each production process *pp* with the site quantity sequence $q : \mathbb{N} \to \langle\, [n+1], \mathbb{R}_+\rangle$: $q_{ik} \leq m_i$ for all $i \in [n+1]$ and $k \in \mathbb{N}$.

A production network PN is *deterministic* if the input rate is constant, i.e., $in_k = in$ for all $k \in \mathbb{N}_{>0}$ and for some $in \in \mathbb{R}_+$, and if the production rates are exhaustive, i.e., $p_{ik} = \min(q_{i(k-1)}, max_i)$ for all $k \in \mathbb{N}_{>0}$ and $i \in [n+1]$.

Let PN be a deterministic production network, and let its unique production process have – in addition – a constant site quantity, meaning that there is a quantity vector $m\colon [n{+}1] \to \mathbb{R}_+$ with $q_k = m$ for all $k \in \mathbb{N}$. Let moreover the vector m be smaller than or equal to the maximum production rate, i.e. $m \leq max$. Consequently, the production rates are also equal to m :

$$p_{ik} = \min(q_{i(k-1)}, max_i) = \min(m_i, max_i) = m_i.$$

And with a constant production rate, the outputs become constant:

$$out_k = (1 - \sum_{i\in[n+1]} d_{(n+1)i})p_{(n+1)k} = (1 - \sum_{i\in[n+1]} d_{(n+1)i})m_{n+1}.$$

Such a network is obviously stable with an upper bound being the site quantities (or more). Moreover, the production process condition 3 in Definition 2 holds, yielding the following equality for the quantities of m:

$$\begin{aligned} m_j = q_{jk} &= in \cdot \delta_{1j} + \sum_{i\in[n+1]} d_{ij}p_{ik} + q_{j(k-1)} - p_{jk} \\ &= in \cdot \delta_{1j} + \sum_{i\in[n+1]} d_{ij}m_i + m_j - m_j = in \cdot \delta_{1j} + \sum_{i\in[n+1]} d_{ij}m_i \end{aligned}$$

for all $j \in [n+1]$ where $\delta_{11} = 1$ and $\delta_{1j} = 0$ for $j \geq 2$.

If one subtracts the latter sum and denotes the transposed distribution matrix by d^t, then one gets

$$(E - d^t)m = in \cdot e_1$$

where E is the identity matrix and e_1 the first unit vector.

In other words, a deterministic production network with a constant site quantity m implies that the system of linear equations

$$(E - d^t)x = in \cdot e_1$$

has m as a solution.

Interestingly enough, the considerations work also the other way round meaning that each solution of the system of linear equations given by the constant input quantity and the constant distribution matrix gives rise to stable production networks, provided that the initial quantity is bounded by the solution and the maximum production rate equals the solution or is greater.

7.1. Example. Solving the linear system $(E - d^t_{\mathbf{SAMPLE}})m = in \cdot e_1$ for the deterministic production network **SAMPLE** from Section 3.2 results in the maximal production rate vector $m = max = \begin{pmatrix} 3.2 \\ 1.6 \\ 2.4 \\ 1 \end{pmatrix}$.

Now we state our second main result, which guarantees stability of production networks under a sufficient condition.

Theorem 3. *Let PN be a deterministic production network and $m\colon [n+1] \to \mathbb{R}_+$ be a solution of the system of linear equations*

$$(E - d^t)x = in \cdot e_1$$

with $m \leq max$ and $q_0 \leq m$. Then PN is stable.

Proof. We show by induction that the sequence of site quantities of the unique production process of PN is bounded by m, i.e. $q_k \leq m$ for all $k \in \mathbb{N}$.

Base: $q_0 \leq m$ by assumption.

Step:

$$
\begin{aligned}
&(1) \quad q_{j(k+1)} = in \cdot \delta_{1j} + \sum_{i \in [n+1]} d_{ij} p_{i(k+1)} + q_{jk} - p_{j(k+1)} \\
&(2) \qquad\qquad = in \cdot \delta_{1j} + \sum_{i \in [n+1]} d_{ij} \min(q_{ik}, max_i) + q_{jk} - \min(q_{jk}, max_j) \\
&(3) \qquad\qquad = in \cdot \delta_{1j} + \sum_{i \in [n+1]} d_{ij} q_{ik} + q_{jk} - q_{jk} \\
&(4) \qquad\qquad \leq in \cdot \delta_{1j} + \sum_{i \in [n+1]} d_{ij} m_i \\
&(5) \qquad\qquad = m_j
\end{aligned}
$$

where equality 1 is the site quantity condition, equality 2 uses the exhaustiveness, equality 3 follows from the induction hypothesis $q_k \leq m$ and the assumption $m \leq max$, the inequality 4 is again the induction hypothesis, and equality 5 uses that m solves $(E - d^t)x = in \cdot e_1$.

With the boundedness of all site quantities, the sum of them over all sites is also bounded.

□

8. Conclusion

In this paper, we have introduced and investigated a variant of production networks with step-by-step production processes. The first main result shows that production networks can be transformed into communities of autonomous units such that production processes correspond to infinite runs of the modeling communities. The second main result yields a sufficient criterion for the stability of deterministic production networks. As this is the very first attempt to relate production networks and autonomous units, future research should shed more light on the significance of this approach including the following topics:

(1) The stability results may be improved by enlarging the class of production networks for which sufficient conditions yield stability.

(2) One may also look for necessary conditions or even proper characterizations.

(3) So far, we have considered only two ways to choose the input rates and the productions: randomly on one hand and deterministically on the other. An interesting question is which other control conditions for the input unit and the production units will do to make proper use of their autonomy.

(4) To improve the behavior of a production network one may allow variable distribution rates so that further circumstances like waiting time can be considered.

(5) To make the model more flexible, one may enhance the notion of production networks by relaxing and modifying various assumptions like the following:

   - There may be more than one input site and one output site.
   - There may be an explicit control of the output rates.
   - There may be different kinds of materials and information flows through the network rather than a single homogeneous matter.
   - There may be particular time conditions for production and transportation at each site rather than the homogeneous step assumption.

   We expect that modifications like these will not be difficult to get.

(6) Another possible modification would be to assume that the produced and distributed material consists of a number of atomic items such that only integer division is possible. In this case, the graph-transformational model may be particularly suitable as the atomic items could be represented by atomic graph components explicitly.

(7) In some applications, it may not be realistic to assume that the underlying network is invariant, but it may grow or shrink due to economic circumstances. Again, the graph-transformational model may help to dynamize the structure of the production networks because the local insertion and removal of nodes and edges is just what happens if rules are applied.

Acknowledgments. *The authors would like to acknowledge that their research is partially supported by the Collaborative Research Centre 637 (Autonomous Cooperating Logistic Processes: A Paradigm Shift and Its Limitations) funded by the German Research Foundation (DFG).*

References

[1] Hans-Peter Wiendahl and Stefan Lutz. Production in Networks. *Annals of the CIRP- Manufacturing Technology*, 51(2):1–14, 2002.

[2] Bernd Scholz-Reiter, Michael Görges, Thomas Jagalski, and Afshin Mehrsai. Modelling and Analysis of Autonomously Controlled Production Networks. In *Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 09). Moscow, Russia*, pages 850–855, 2009.

[3] Bernd Scholz-Reiter, Afshin Mehrsai, and Michael Görges. Handling the Dynamics in Logistics - Adoption of Dynamic Behavior and Reduction of Dynamic Effects. *Asian International Journal of Science and Technology in Production and Manufacturing Engineering (AIJSTPME)*, 2(3):99–110, 2009.

[4] Sergey Dashkovskiy, Michael Görges, and Lars Naujok. Local Input to State Stability of Production Networks. 2009. To appear in Proceedings of the Second International Conference, LDIC 2009, Bremen, Germany, August 2009.

[5] Sergey Dashkovskiy, Björn S. Rüffer, and Fabian R. Wirth. Small gain theorems for large scale systems and construction of ISS Lyapunov functions. *SIAM Journal on Control and Optimization*, 48(6):4089–4118, 2010.

[6] Sergey Dashkovskiy, Björn S. Rüffer, and Fabian R. Wirth. Numerical verification of local input-to-state stability for large networks. In *Proceedings of the 46th IEEE Conference on Decision and Control, New Orleans, LA, USA, Dec. 12-14, 2007*, pages 4471–4476, 2007.

[7] Sergey Dashkovskiy and Björn S. Rüffer. Local ISS of large-scale interconnections and estimates for stability regions. *Systems and Control Letters*, 59(3–4):241–247, 2010.

[8] Karsten Hölscher, Hans-Jörg Kreowski, and Sabine Kuske. Autonomous units and their semantics — the sequential case. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Proc. 3rd Intl. Conference on Graph Transformations (ICGT 2006)*, volume 4178 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2006.

[9] Karsten Hölscher, Renate Klempien-Hinrichs, Peter Knirsch, Hans-Jörg Kreowski, and Sabine Kuske. Autonomous Units: Basic Concepts and Semantic Foundation. In Michael Hülsmann and Katja Windt, editors, *Understanding Autonomous Cooperation and Control in Logistics – The Impact on Management, Information and Communication and Material Flow*, pages 103–120. Springer, 2007.

[10] Hans-Jörg Kreowski and Sabine Kuske. Autonomous Units and Their Semantics - The Parallel Case. In J.L. Fiadeiro and P.Y. Schobbens, editors, *Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006*, volume 4409 of *Lecture Notes in Computer Science*, pages 56–73, 2007.

[11] Karsten Hölscher, Hans-Jörg Kreowski, and Sabine Kuske. Autonomous Units to Model Interacting Sequential and Parallel Processes. *Fundamenta Informaticae*, 92(3):233–257, 2009.

[12] Andrea Corradini, Hartmut Ehrig, Reiko Heckel, Michael Löwe, Ugo Montanari, and Francesca Rossi. Algebraic Approaches to Graph Transformation Part I: Basic Concepts and Double Pushout Approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, pages 163–245. World Scientific, Singapore, 1997.

[13] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

[14] Rubino Geiß and Moritz Kroll. GrGen.NET: A fast, expressive, and general purpose graph rewrite tool. In A. Schürr, M. Nagl, and A. Zündorf, editors, *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*, volume NN of *LNCS*. Springer, 2008. http://www.springerlink.com/content/105633/.