Requirements on the Execution of Kahn Process Networks*

Marc Geilen and Twan Basten

{*m.c.w.geilen, a.a.basten*}*@tue.nl*

Dept. of Elec. Eng., Eindhoven University of Technology, The Netherlands

Abstract. Kahn process networks (KPNs) are a programming paradigm suitable for streaming-based multimedia and signal-processing applications. We discuss the execution of KPNs, and the criteria for correct scheduling of their realisations. In [12], Parks shows how process networks can be scheduled in bounded memory; the proposed method is used in many implementations of KPNs. However, it does not result in the correct behaviour for all KPNs. We investigate the requirements for a scheduler to guarantee both correct and bounded execution of KPNs and present an improved scheduling strategy that satisfies them.

Keywords: Kahn process networks, Kahn Principle, dynamic scheduling, deadlock resolution, multi-processor architectures, media processing, signal processing, streaming

1 Introduction

Process networks are a popular model to express behaviour of data flow and streaming nature. This includes audio, video and 3D multimedia applications such as encoding and decoding of MPEG video streams. Using process networks, an application is modelled as a collection of concurrent processes communicating streams of data through FIFO channels. Process networks make task-level parallelism and communication explicit, have a simple semantics, are compositional and allow for efficient implementations without time-consuming synchronisations. There are several variants of process networks. One of the most general forms are Kahn process networks [7, 8], where the nodes are arbitrary sequential programs, that communicate via channels of the process networks with blocking read and non-blocking write operations. Although harder to analyse than more restricted models, such as synchronous dataflow networks [11], the added flexibility makes KPNs a popular programming model. Where synchronous dataflow models can be statically scheduled at compile time, KPNs must be scheduled dynamically in general, because their expressive power does not allow them to be statically analysed. A run-time system is required to schedule the execution of processes and to manage memory usage for the channels. The goal is to create an execution that is as efficient as possible w.r.t. speed and memory and that is faithful to the process network specification.

In this paper, we investigate the fundamental requirements for a run-time system for KPNs to be faithful to the semantics of the KPN specification and to use bounded memory resources when possible. In particular, we study the

* This work is supported in part by the IST-2000-30026 project, Ozone.

issues that arise when the conceptually unbounded FIFO channels of the KPN are realised using bounded FIFOs.

Related Work Reasoning about realisations of KPNs touches on the Kahn Principle, stating that the solution to the network equations formulated by Kahn [7] is the same as the behaviour of a model of (sequential) programs reading and writing tokens on channels. The Principle was introduced, but not proved, by Kahn in [7]. It was proved [5, 13] for an operational model of transition systems.

Scheduling process networks using static bounded channels is extensively addressed by Thomas Parks in [12], introducing an algorithm that uses bounded memory if possible. Based on this scheduling policy, a number of tools and libraries have been developed for executing KPNs. YAPI [9] is a C++ library for designing stream-processing applications. Ptolemy II [10] is a framework for codesign using mixed models of computation. The process-network domain is described in [6]. The Distributed Process Networks of [15] form the computational back end of the Jade/PAGIS system for processing digital satellite images. [14] covers an implementation of process networks in Java. [1] is another implementation for digital signal processing. Common among all these implementations is a multi-threading environment in which processes of the KPN execute in their own thread of control and channels are allocated a fixed capacity. Semaphores control access to channels and block the thread when reading from an empty or writing to a full channel. This raises the possibility of a deadlock when one or more processes are permanently blocked on full channels. A special thread (preempted by the other threads) is used to detect a deadlock and initiate a deadlock resolution procedure when necessary. This essentially realises the scheduling policy of [12]. The algorithm of Parks leaves some room for optimisation of memory usage by careful selection of initial channel capacities (using profiling) and clever selection of channels when the capacity needs to be increased; see [2]. [2] also introduces causal chains, also used in this paper to define deadlocks.

Contribution All work on practical implementations of KPNs that we found builds on Parks' scheduling algorithm. We show that this algorithm does not execute all KPNs correctly and propose an improved scheduling strategy.

Organisation Section 2 discusses the challenges related with implementing process networks, such as memory management, scheduling and deadlock detection and resolution. An operational semantics of process networks is introduced in Section 3 to enable reasoning about realisations of KPNs. Section 4 discusses the properties of realisations of process networks with bounded channels and Section 5 introduces a correct scheduling strategy. Section 6 concludes.

2 Implementing Kahn Process Networks

Conceptually, the FIFO communication channels of a KPN have an unbounded capacity. A realisation of a process network has to run within a finite amount of memory. Rather than dynamically allocating memory to channels when needed,

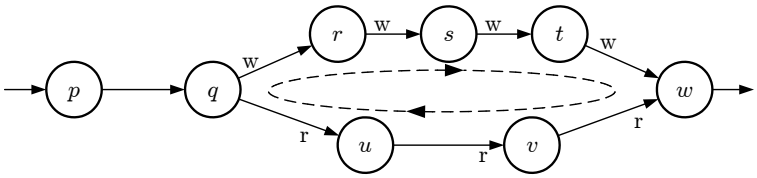

Fig. 1. An artificial deadlock

it is for reasons of efficiency better to allocate fixed amounts of memory to channels and change this amount only sporadically, if necessary. An added advantage of the fixed capacity of channels is that one can use Parks' [12] model of execution, where a write action on a full FIFO blocks until there is room again in the FIFO. This gives an efficient intermediate form of data-driven and demand-driven scheduling [12, 2] in which a process can produce output (in a data-driven way) until the channel it is writing to is full. Then it blocks and other processes take over. Unfortunately, it is undecidable in general, how much buffer capacity is needed in every channel [4]. If buffers are chosen too large, memory is wasted. If buffers are chosen too small, artificial deadlocks (i.e., processes being permanently blocked on a full channel) may occur that are not in the original KPN. Therefore, a scheduler is needed that determines the order of execution of processes and that manages buffer sizes at run-time.

In practice, such a scheduler should satisfy two requirements. Output should be complete; it must coordinate progress of all processes and manage channel capacity in such a way that the output produced by the KPN corresponds to the output predicted by the denotational semantics introduced in [7]. Secondly, it is important that this is achieved within bounded memory. Since memory usage of the individual processes is not under control of the scheduler, this amounts to keeping the FIFOs bounded, also in (conceptually) infinite computations.

An important aspect of a run-time scheduler for KPNs is dealing with deadlocks. One should discern different kinds of deadlocks. A process network is in *global deadlock* if none of the processes can make progress. In contrast, a *local deadlock* arises if some of the processes in the network cannot progress and their further progress cannot be initiated by the rest of the network. In a realisation of a KPN, processes may be blocked if an output channel is full. This is not the case for the conceptual KPNs. As a result, some of the deadlocks in realisations are *artificial* in the sense that they do not exist in the KPN. Figure 1 shows a process network in an artificial deadlock situation. Process w cannot continue because its input channel to process v is empty; it is blocked on a read action on the channel to v, denoted by the 'r' in the figure. The required input should be provided by v, but this is in turn waiting for input from u. u is waiting for q. Process q is blocked, because it is trying to output a token on the full channel to r; the block on the write action is denoted by a 'w'. Similarly processes r, s and t are blocked by a full channel. Only w could start emptying these channels, but w is blocked. The processes are in artificial deadlock (q, r, s and t would not be blocked in the KPN) and can only continue if the capacity of one of the full channels is increased. The deadlock is local as long as process p can con-

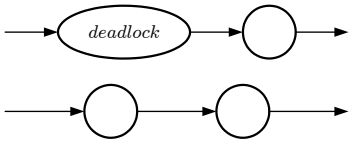

Fig. 2. A local deadlock

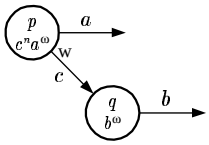

Fig. 3. Non effective network

tinue. To guarantee the correct output of a network, a run-time scheduler must detect and respond to artificial deadlocks. Parks proposes to respond to global artificial deadlocks. Detection is (in [1, 6, 9, 14, 15]) realised by detecting that all processes are blocked, some of which on a full FIFO. Although this guarantees that execution of the process network never terminates because of an artificial deadlock, it does not guarantee the production of *all* output required by the KPN semantics; output may not be complete. E.g., in the network of Figure 2, if the upper part reaches a local artificial deadlock, then the lower, independent part is not affected. Processes may not all come to a halt and the local deadlock is not detected and not resolved. The upper part may not produce the required output. Such situations exist in realistic networks. Further, when multiple KPNs are controlled by a single run-time scheduler, one entire process network may get stuck in a deadlock. A deadlock detection scheme has to detect local deadlocks.

It is well known that the Kahn Principle hinges on fair scheduling of processes [3, 7, 13]. Fairness means that all processes that can make progress should make progress at some point. This is often a tacit but valid assumption if the underlying realisation is truly concurrent, or fairly scheduled. In the context however of bounded FIFO channels where processes appear to be inactive while they are blocked for writing, fairness of a schedule is no longer evident. This issue is neglected in Parks' algorithm by responding to global deadlocks only, leading to a discrepancy with the behaviour of the conceptual KPN.

To come to an improved scheduling strategy, we must restrict our attention to particular classes of KPNs. It is observed in [12] that the scheduling requirements cannot always be met; some KPNs cannot be scheduled with bounded channel capacity meaning the second scheduling requirement cannot be achieved. Thus, the existence of a bounded execution must be required. We argue that one further restriction on KPNs is needed to meet the requirement of output completeness. A problem is posed by the production of data that is never used. This is illustrated with Figure 3. Process p writes n data elements (tokens) on channel c connecting p to process q; after that, it writes tokens to output channel a forever. q never reads tokens from c and outputs tokens to channel b forever. If the capacity of c is insufficient for n tokens, then output a will never be written to unless the capacity of c is increased. q doesn't halt and execution according to Parks' algorithm does not produce output on channel a. In the KPN however, infinite output is produced on both channels. The above suggests that a good scheduler should eventually increase the capacity of channel c so that it can contain all n tokens. However, such a scheduler fails to correctly schedule another KPN. Consider a process network with the same structure as the one of Figure 3, but this time, p continuously writes tokens on output a, mixed with infinitely many tokens to

channel c. q writes infinitely many tokens to b and reads infinitely many tokens from c, but at a different rate than p writes tokens to c. Note that a bounded execution exists; a capacity of one token suffices for channel c. If process p writes to c faster than q reads, channel c may fill up and the scheduler, not knowing if tokens on channel c will ever be read, decides to increase channel capacity. A process q exists that always postpones the read actions until after the scheduler decides to increase the capacity; the execution will be unbounded, although a bounded execution exists. The above demonstrates that KPNs producing unread tokens cannot be scheduled correctly. Our solution is to assume that every token that is written to a channel, is eventually also read. We call such KPNs *effective*.

The development of a correct scheduling algorithm is based on the use of blocking write operations to full channels as in [12]. In order to deal with local deadlocks, we build upon the notion of causal chains as introduced in [2]. Any blocked process depends for its further progress on a unique other process that must fill or empty the appropriate channel. These dependencies give rise to chains of dependencies. If such a chain of dependencies is cyclic, it indicates a local deadlock; no further progress can be made without external help. In Figure 1, such a causal chain is indicated by the dashed ellipse indicating the cyclic mutual dependencies of the processes q, r, s, t, u, v and w.

3 An Operational Model of Process Networks

The denotational semantics of KPNs of [7] is attractive from a mathematical point of view, because of its abstractness and compositionality. It (purposely) abstracts from implementation related aspects such as scheduling, performance and resources such as channel capacities. In a realisation of a process network, FIFO sizes and contents play an important role and are influenced by a runtime environment that governs the execution of the network. It is for this reason that we give a simple operational semantics of process networks, similar to [5, 13].

3.1 Labelled Transition Systems

We give an operational semantics to KPNs in the form of a labelled transition system (LTS). We use, more specifically, an LTS with an initial state, with designated input and output actions in the form of reads and writes of tokens on channels, as well as internal actions. For convenience, we assume a universal set *Chan* of channels and for every channel $c \in \mathit{Chan}$ a corresponding channel alphabet Σ_c. We use Σ to denote the union of all channel alphabets, and A^* (A^∞) to denote the set of all finite (and infinite) strings over alphabet A. If i and j are functions from channels to strings over the corresponding alphabets, we write $i \preceq j$ iff for every channel c in the domain of i, $i(c)$ is a prefix of $j(c)$.

An LTS is a tuple $(S, s_0, I, O, \mathit{Act}, \rightarrow)$ consisting of a set S of states, an initial state $s_0 \in S$, a set $I \subseteq \mathit{Chan}$ of input channels, a set $O \subseteq \mathit{Chan}$ (distinct from I) of output channels, a set Act of actions consisting of input actions $\{c?a \mid c \in I, a \in \Sigma_c\} \subseteq \mathit{Act}$, output actions $\{c!a \mid c \in O, a \in \Sigma_c\} \subseteq \mathit{Act}$ and

(possibly) internal actions (all other actions), and a labelled transition relation $\rightarrow \subseteq S \times Act \times S$. Thus, $c!a$ is a write action to channel c with token a; $c?a$ models passing of a token from input channel c to the LTS. We write $s_1 \xrightarrow{\alpha} s_2$ if $(s_1, \alpha, s_2) \in \rightarrow$ and $s_1 \xrightarrow{\alpha}$ if there is some $s_2 \in S$ such that $s_1 \xrightarrow{\alpha} s_2$.

With a write operation, the token on the output channel is determined by the LTS. With a read operation, the token that appears on the input channel is determined by the environment of the LTS. Therefore, a read operation is modelled with a set of input actions that provides a transition for every possible token of the alphabet. Kahn process networks do not exhibit non-determinism. Transitions are deterministic and if multiple actions are available at the same time, then they are truly concurrent (i.e., they can be executed in any order with an identical result). We call such LTSes determinate.

Definition 1. (Determinacy) *LTS $(S, s_0, I, O, Act, \rightarrow)$ is determinate iff for any $s, s_1, s_2 \in S$, $\alpha_1, \alpha_2 \in Act$, if $s \xrightarrow{\alpha_1} s_1$ and $s \xrightarrow{\alpha_2} s_2$, the following hold:*

1. (Determinism) *if $\alpha_1 = \alpha_2$, then $s_1 = s_2$, i.e., executing a particular action has a unique deterministic result;*
2. (Confluence) *if α_1 and α_2 are not two input actions on the same channel (i.e., instances of the same read operation), then there is some s_3 such that $s_1 \xrightarrow{\alpha_2} s_3$ and $s_2 \xrightarrow{\alpha_1} s_3$.*
3. (Input Completeness) *if $\alpha_1 = c?a$ for some $c \in I$ and $a \in \Sigma_c$, then for every $a' \in \Sigma_c$, $s \xrightarrow{c?a'}$, i.e., input tokens are completely defined by the environment, the LTS cannot be selective in the choice of tokens;*
4. (Output Uniqueness) *if $\alpha_1 = c!a$ and $\alpha_2 = c!a'$ for some $c \in O$ and $a, a' \in \Sigma_c$, then $a = a'$, i.e., output tokens are determined by the LTS.*

A sequential *LTS is a determinate LTS with the additional property that*

5. (Sequentiality) *if $\alpha_1 = c\sharp a$ ($\sharp \in \{!, ?\}$), for some $c \in I \cup O$, $a \in \Sigma_c$, then $\alpha_2 = c\sharp a'$ for some $a' \in \Sigma_c$, i.e., the LTS accepts at most one input/output operation at any point in time and no other (internal) actions.*

An execution of the transition system is a sequence $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ of states $s_i \in S$ and actions $\alpha_i \in Act$, such that $s_i \xrightarrow{\alpha_i} s_{i+1}$ for all $i \geq 0$ (up to the length of the execution). If ρ is such an execution, then we use $|\rho| \in \mathbb{N} \cup \{\infty\}$ to denote the length of the execution. $|\rho| = \infty$ if ρ is infinite and $|\rho| = n$ if $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$. For $k \leq |\rho|$, we use ρ^k to denote the prefix of the execution up to and including state k.

3.2 Operational Semantics of Process Networks

We formalise an operational notion of a KPN as an LTS.

Definition 2. (Kahn process network) *A Kahn process network is a tuple $(P, C, I, O, Act, \{LTS_p \mid p \in P\})$ that consists of the following elements.*

- *A finite set P of* processes.
- *A finite set $C \subseteq Chan$ of* internal channels, *a finite set $I \subseteq Chan$ of input channels and a finite set $O \subseteq Chan$ of output channels, all distinct.*

- *The set Act of* actions *consisting of reads and writes of tokens on the channels c in $C \cup I \cup O$: $Act = \{c?a, c!a \mid c \in C \cup I \cup O, a \in \Sigma_c\}$.*
- *Every process $p \in P$ is defined by a sequential labelled transition system $LTS_p = (S_p, s_{p,0}, I_p, O_p, Act, \xrightarrow[p]{})$, with $I_p \subseteq I \cup C$ and $O_p \subseteq O \cup C$.*
- *For every channel $c \in C \cup I$, there is exactly one process $p \in P$ that reads from it ($c \in I_p$) and for every channel $c \in C \cup O$, there is exactly one process $p \in P$ that writes to it ($c \in O_p$).*

To define the operational semantics of a KPN, we need a notion of global state of the network; this state is composed of the individual states of the processes and the contents of the internal channels.

Definition 3. (Configuration) *A configuration of a process network is a pair (π, γ) consisting of a process state π and a channel state γ, where*

- *a* process state *$\pi : P \to S = \bigcup_{p \in P} S_p$ is a function that maps every process $p \in P$ on a local state $\pi(p) \in S_p$ of its transition system;*
- *a* channel state *$\gamma : C \to \Sigma^*$ is a function that maps every internal channel $c \in C$ on a finite string $\gamma(c)$ over Σ_c.*

The set of all configurations is denoted by Confs and there is a designated initial configuration $c_0 = (\pi_0, \gamma_0)$, where π_0 maps every process $p \in P$ to its initial state $s_{p,0}$ and γ_0 maps every channel $c \in C$ to the empty string ϵ.

Definition 4. (Operational Semantics of a KPN) *We assign to a KPN an operational semantics in the form of an LTS $(Confs, c_0, I, O, Act, \to)$. The labelled transition relation $\to$ is the smallest relation satisfying the following four induction rules. For reading from and writing to internal channels:*

$$\frac{\pi(p) \xrightarrow[p]{c?a} s, \gamma(c) = a\sigma, c \in C}{(\pi, \gamma) \xrightarrow{c?a} (\pi\{s/p\}, \gamma\{\sigma/c\})} \qquad \frac{\pi(p) \xrightarrow[p]{c!a} s, \gamma(c) = \sigma, c \in C}{(\pi, \gamma) \xrightarrow{c!a} (\pi\{s/p\}, \gamma\{\sigma a/c\})}$$

(The notation $f\{y/x\}$ denotes the function with the same domain as f and identical to f, except that $f(x) = y$.) Input channels and output channels are open to the environment:

$$\frac{\pi(p) \xrightarrow[p]{c?a} s, c \in I}{(\pi, \gamma) \xrightarrow{c?a} (\pi\{s/p\}, \gamma)} \qquad \frac{\pi(p) \xrightarrow[p]{c!a} s, c \in O}{(\pi, \gamma) \xrightarrow{c!a} (\pi\{s/p\}, \gamma)}$$

It is easy to show that if the labelled transition systems of the individual processes P are sequential, then the labelled transition system of the KPN is determinate. From a given execution ρ of a KPN, we extract the consumed input and the produced output on a set $D \subseteq C \cup I \cup O$ of channels as follows.

- For a channel $c \in D$, $\rho?_c$ is a (finite or infinite) string over Σ_c that results from projecting ρ onto *read* actions on c;
- similarly, $\rho!_c$ is a (finite or infinite) string over Σ_c that results from projecting ρ onto *write* actions on c;
- finally, $\rho?_D = \{(c, \rho?_c) \mid c \in D\}$ and $\rho!_D = \{(c, \rho!_c) \mid c \in D\}$.

Thus $\rho?_I$ denotes the input consumed by the network in execution ρ and $\rho!_O$ denotes the output of the network. To reason about the input *offered* to the network (consumed or not consumed), we say that ρ is an execution with input $i : I \to \Sigma^\infty$ iff $\rho?_I \preceq i$.

In the remainder, we assume that $(P, C, I, O, Act, \{LTS_p \mid p \in P\})$ is a Kahn process network with LTS $(Confs, c_0, I, O, Act, \rightarrow)$. We can now formalise the notions of fairness, maximality, effectiveness and boundedness.

Definition 5. $\rho = (\pi_0, \gamma_0) \xrightarrow{\alpha_0} (\pi_1, \gamma_1) \xrightarrow{\alpha_1} \ldots$ *is an execution of the KPN.*

- (Fairness) *Execution ρ with input i is fair iff it is finite, or it is infinite and if at some point an input, internal or output action is enabled, it must eventually be executed, i.e.,*
  - *if for some $n \in \mathbb{N}$, $c \in C$ and $a \in \Sigma_c$, $(\pi_n, \gamma_n) \xrightarrow{c?a}$, then there is some $k \geq n$ such that $\alpha_k = c?a$;*
  - *if for some $n \in \mathbb{N}$, $c \in I$ and $a \in \Sigma_c$, $(\pi_n, \gamma_n) \xrightarrow{c?a}$, and $i(c) = (\rho^n?_c)a\sigma$ for some $\sigma \in \Sigma_c^\infty$, then there is some $k \geq n$ such that $\alpha_k = c?a$;*
  - *if for some $n \in \mathbb{N}$, $c \in C \cup O$ and $a \in \Sigma_c$, $(\pi_n, \gamma_n) \xrightarrow{c!a}$, then there is some $k \geq n$ such that $\alpha_k = c!a$.*
- (Maximality) *Execution ρ with input i is called maximal iff it is infinite or, in its last configuration, only read actions on input channels from which all input of i has been consumed are possible, i.e., if $|\rho| = n$ and $(\pi_n, \gamma_n) \xrightarrow{\alpha}$ then $\alpha = c?a$ for some $c \in I$ and $a \in \Sigma_c$ and $\rho?_c = i(c)$.*
- (Effectiveness) *Execution ρ is effective iff every token produced on an internal channel is ultimately consumed, i.e., if $\rho?_C = \rho!_C$. A KPN is called effective iff for all inputs there exists an effective fair and maximal execution.*
- (Boundedness) *Execution ρ is bounded iff for every internal channel there is an upper bound to the number of tokens that accumulate in it during the execution, i.e., $\forall\, c \in C : \exists\, n \in \mathbb{N} : \forall\, i \in \mathbb{N}, 0 \leq i \leq |\rho| : |\gamma_i(c)| \leq n$. A KPN is bounded iff for all inputs there is a bounded fair and maximal execution.*

Corollary 1. *Any two fair and maximal executions of a KPN with the same input execute the same actions.*

The question whether a KPN is bounded or effective is unfortunately undecidable in general; this follows immediately from the fact that processes are arbitrary sequential programs and thus Turing complete.

3.3 The Kahn Principle

The operational semantics given in the previous subsection is a model for a realisation of a KPN. Its behaviour corresponds to the function given by Kahn's semantics as the least solution to a set of network equations [7] for the KPN. This is referred to as the Kahn Principle. It was stated convincingly, but without proof, by Kahn in [7] and was later proved by Faustini [5] for an operational model similar to ours. Based on the operational semantics, we can derive a functional relation between inputs and outputs. This function can then be shown to correspond to the least solution of Kahn's network equations. The maximal and fair executions capture the input/output relation of the network.

Definition 6. (Input/Output Relation) *The input/output relation IO of a KPN is the relation* $\{(i, \rho!_O) \mid \rho$ *is a maximal and fair execution with input* $i\}$.

IO is a so-called continuous function, which is the basis of the following theorem.

Theorem 1. (Kahn Principle) *[7, 5] The function IO of input strings and output strings for maximal and fair executions corresponds to the least solution of Kahn's network equations of [7].*

A proof of the Kahn Principle is beyond the scope of this paper and can be found in (for instance) [5] for a similar operational model and in [13] for an operational semantics in terms of so-called concurrent transition systems.

4 Process Networks with Channel Bounds

In this section, we study the effects of imposing channel bounds on KPNs. We consider what happens if we execute a process network when offered some fixed input i. We assume that all input offered is initially available, i.e., an input action only blocks if all input has been consumed. When we mention executions, we mean executions with input i.

4.1 Bounded Channels

The bounded memory requirement is enforced on a process network by bounds on the number of tokens that can accumulate in every channel. These bounds need not be the same for all channels. We can model a realisation by an LTS, as presented in the previous section, and an execution of the realisation by an execution of this LTS. Choices are resolved by a scheduling mechanism that controls when and how processes are executed on processors and that manages memory that is used for the channels. Hence, the scheduler is a mechanism that guides the execution through the LTS.

Definition 7. (Channel Bound) *A bound* $\bar{b}$ *on channel contents is a mapping* $C \to \mathbb{N}^+$ *that maps every internal channel to the (positive) maximum number of tokens it can simultaneously contain. If* $\bar{b}_1$ *and* $\bar{b}_2$ *are both channel bounds, we write* $\bar{b}_1 \preceq \bar{b}_2$ *to denote that for every* $c \in C$, $\bar{b}_1(c) \leq \bar{b}_2(c)$.

In the remainder, we refer to a KPN and a corresponding bound on its channel sizes as a *process network with bounds (PNB)*. The operational semantics of a PNB conforms to the operational semantics of the corresponding KPN except that those configurations (π, γ) are missing that do not respect the FIFO bounds, i.e., if $|\gamma(c)| > \bar{b}(c)$ for some $c \in C$. Also transitions from or to these configurations are removed. As a consequence, where the KPN may be able to write to a channel, a PNB may not be able to do the same if the channel is full. This write may have to be postponed, until there is room in the channel, but also, artificial deadlocks may arise in the transition system as a result of this [12]. We first show that PNBs also have determinate transition systems.

Proposition 1. *A PNB is determinate.*

Proof. The corresponding KPN is determinate. The LTS of the PNB is obtained from the LTS of the KPN by removing states and transitions; thus determinism is preserved. By a simple case analysis, one can show that the transitions of the PNB are confluent, i.e., that the configuration where actions from configurations respecting the bounds converge also stays within FIFO bounds. Since input and output actions do not touch internal channels, also input completeness and output uniqueness are preserved.

Note that executions of a PNB are also executions of the corresponding KPN.

Definition 8. *An execution of a PNB is fair (maximal, effective) iff it is fair (maximal, effective) in the corresponding KPN.*

Corollary 2. *An execution of a PNB is not fair or not maximal if it permanently blocks on a full channel; in its own LTS there is no enabled write transition corresponding to the blocked write action, but such an action exists in the LTS of the KPN implying that the execution is not fair/maximal in the KPN.*

4.2 Deadlocks

A PNB behaves the same as the corresponding KPN, except for the fact that certain write actions may be disabled by full channels. This may lead to artificial deadlock situations. Since the amount of memory required for channels cannot be decided upfront [4], a scheduler must provide the means to dynamically (at run-time) increase channel capacity where needed.

If a process is blocked trying to read from an empty channel or trying to write to a full channel, this situation can only be resolved by a unique other process in the network or, in the latter case, by increasing FIFO capacity. These dependencies may give rise to a chain of blocked processes whose blocked condition depends on each other. In order for a process on this chain to make progress, a process further on the chain must first make progress. This is impossible however if the chain leads to an input and all input has been consumed, or if the chain is cyclic. In the latter case, it is clear that the processes in this cyclic causal chain are in local deadlock. If all channels in this chain are empty, the deadlock is real (i.e., also present in the KPN); if not, the deadlock is artificial and can be resolved by enlarging the capacity of a channel on that cycle.

Definition 9. (WAITING) *Let $p, q \in P$; process p is said to be waiting for process q in configuration (π, γ) of the PNB with bounds $\bar{b}$,*

- *if it can read from an empty internal channel c and q is the process that writes to channel c; i.e., $\pi(p) \xrightarrow[p]{c?a}$ for $a \in \Sigma_c$, $\gamma(c) = \epsilon$ and $c \in O_q$;*
- *if it can write to a full internal channel c and q is the process that reads from channel c; i.e., $\pi(p) \xrightarrow[p]{c!a}$ for some $a \in \Sigma_c$, $|\gamma(c)| = \bar{b}(c)$ and $c \in I_q$;*
- *if $p = q$ and it has terminated; i.e., there is no $\alpha \in Act$ such that $\pi(p) \xrightarrow[p]{\alpha}$.*

Note that in the third case the process p has terminated. Saying that in that case it is waiting for itself, simplifies further definitions. Since processes are sequential, a process is waiting for a unique other process. This gives rise to chains of waiting processes.

Definition 10. ((Complete) Causal Chain) *A causal chain (of waiting processes in configuration (π, γ) with bounds $\bar{b}$) is a sequence $\bar{p} = p_0 p_1 p_2 \dots p_k$ of different processes such that for all $0 \leq n < k$, p_n is waiting for p_{n+1} (in configuration (π, γ) with bounds $\bar{b}$). Such a causal chain is called complete if p_k can read from an input channel, or it is waiting for some p_n ($0 \leq n \leq k$) (in configuration (π, γ) with bounds $\bar{b}$).*

Based on causal chains, we can define what we consider to be a local deadlock.

Definition 11. (Local Deadlock) *A local deadlock of a PNB is a complete causal chain forming a cycle, i.e., the last process waiting for the first. A local deadlock is artificial if there is a process p in the deadlock that has an enabled write action in the corresponding KPN to a channel that is full in the PNB. A local deadlock is called real if it is not artificial, i.e., if all FIFOs on the cycle are empty. With a local deadlock, we associate its impact as the set of processes that are waiting (via a causal chain) for a process in the deadlock.*

A deadlock is an inherent property of a process network with its particular FIFO bounds combined with the input provided to the network. As a direct consequence of Corollary 1, we know that it cannot be avoided by different scheduling (unless a schedule is unfair and slows progress so that the deadlock is never reached). Since an artificial local deadlock involves a blocking write to a full channel, the following proposition follows immediately from Corollary 2.

Proposition 2. *If a fair and maximal execution of a PNB exists, then none of its executions exhibits an artificial local deadlock.*

Now we can show that artificial local deadlocks are caused by a lack of capacity in the full channels of the corresponding cycle.

Proposition 3. *If a PNB with bounds $\bar{b}$ displays an artificial local deadlock and there exists a maximal and fair execution of the corresponding PNB with bounds $\bar{b}'$, then in the deadlock there is some full channel c such that $\bar{b}'(c) > \bar{b}(c)$.*

Proof. The fact that the execution with bounds $\bar{b}'$ is maximal and fair implies that the artificial deadlock is not encountered (Proposition 2). Let causal chain $\bar{p}$ be a reachable artificial deadlock of the PNB with bounds $\bar{b}$ and let $FC \subseteq C$ be the set of full channels on this chain. Assume towards a contradiction that there is a set $\bar{b}'$ of bounds with $\bar{b}'(c) \leq \bar{b}(c)$ for every $c \in FC$, allowing for a fair and maximal execution . Now we can imagine the set $\bar{b}''$ of FIFO bounds where $\bar{b}''(c) = \max(\bar{b}(c), \bar{b}'(c))$ for all $c \in C$. Note that both $\bar{b} \preceq \bar{b}''$ and $\bar{b}' \preceq \bar{b}''$. Then the execution leading to the deadlock with bounds $\bar{b}$ is also an execution in the PNB with bounds $\bar{b}''$. In that PNB, the causal chain $\bar{p}$ will occur in that execution (the full channels have the same capacity as in $\bar{b}$). But this contradicts by Proposition 2 the fact that the network with bounds $\bar{b}''$ has a fair and maximal execution (since the network with smaller bounds $\bar{b}'$ has one).

The previous two propositions demonstrate that the channel capacity on a causal chain causing an artificial deadlock is insufficient and needs to be increased. The deadlock is not caused by wrong scheduling (Proposition 2) and could not have been avoided if other buffers outside of the deadlock were larger

(Proposition 3). It is not clear however which of the full FIFOs on the chain should be enlarged. As a consequence of Proposition 3, we do know that if there exist bounds large enough to prevent the artificial deadlock, then also for all PNBs with larger bounds, the artificial deadlock cannot occur.

Corollary 3. *If a process network with bounds $\bar{b}$ permits a maximal and fair execution, then on the same network with bounds $\bar{b}'$, with $\bar{b} \preceq \bar{b}'$, there is no execution that leads to an artificial deadlock.*

5 Schedulers

We view a scheduler for PNBs as a strategy that determines the order of execution of individual read and write actions of all processes as well as the increase (or decrease) of memory allocated to FIFOs. To define the result of a scheduling strategy, we imagine that it is applied to generate some (possibly infinite) execution. We capture a series of snapshots by repeatedly observing the PNB. The snapshots of the output ($\rho!_O$) form chains of output strings. We can define the result of the scheduling strategy to be the supremum of this chain.

Definition 12. (Scheduler Requirements) *A scheduling strategy should (if possible) satisfy the following constraints (taken from [2], adapted from [12]).*

- Output Completeness*: the output of the realisation should be equal to the output prescribed by the denotational semantics of KPNs.*
- Boundedness*: The scheduler should realise an execution where a bounded amount of memory is used for channels.*

Based upon the observation made in Section 4.2, that an artificial deadlock indicates a lack of capacity in the corresponding cyclic chain, we can devise a deadlock resolution strategy, that establishes a bounded execution if one exists. Unfortunately, we do not know which of the full buffers on the local deadlock should be enlarged. Therefore, we employ a strategy, that will (eventually) enlarge all full FIFOs on a local deadlock if necessary.

Definition 13. (Scheduling Strategy) *A correct scheduling strategy is obtained by repeating the following steps forever.*

1. *Execute the processes of a PNB in a data-driven fashion until either an artificial (local) deadlock occurs or the PNB terminates. While executing, use a scheduling policy that guarantees progress for all processes that can continue within current FIFO bounds.*
2. *Resolve all artificial deadlocks by increasing the smallest full FIFO on that deadlock by a finite amount of tokens.*

Note the difference with the algorithm of Parks, where the network is executed until *all* processes are blocked. In our scheduling strategy, deadlock resolution is activated also for local artificial deadlocks.

We proceed to show that the proposed scheduling strategy satisfies the requirements of Definition 12. These requirements can only be met if the KPN

is bounded and effective. Blocking because of a full channel must eventually be resolved when the channel is emptied by another process, or it must lead to a local artificial deadlock. To see this, consider a PNB where a process remains blocked on a full channel. The tokens in the channel are eventually read in an execution of the (effective) KPN. If this doesn't happen in the corresponding PNB, the process that reads the channel must also remain blocked, as well as the process on which this process waits, and so forth. The corresponding complete causal chain cannot end in an input; such a block can only be caused if all input has been consumed. But then the contents of the full channel will never be read contradicting effectiveness. Thus the causal chain must be cyclic, i.e., a local deadlock has occurred, and from effectiveness it follows that it must be artificial.

Lemma 1. *If a PNB of an effective KPN is scheduled according to the strategy of Definition 13 with bounds $\bar{b}$ and a process p is blocked on a write, then during step 1 of the scheduling strategy, either this write will become unblocked and subsequently scheduled, or p will eventually be in the impact of an artificial deadlock.*

Proof. Let (π, γ) be a configuration where process p is trying to write to a full channel $c \in C$, $\pi(p) \xrightarrow[p]{c!a} s$ and $|\gamma(c)| = \bar{b}(c)$. Process p is waiting for process q reading from channel c. In any execution with bounds $\bar{b}$ passing through configuration (π, γ), a read action of q on channel c precedes the write action $c!a$ if it occurs. Effectiveness implies that an effective fair and maximal execution exists on the KPN. Corollary 1 implies that, in the KPN, q performs only a finite number of read and write operations before executing a read action from c. If during scheduling step 1, this action never occurs in the PNB, then q must from some point onward be permanently blocked, waiting for some other process r. It cannot be blocked because of termination of process q or on a read operation from an input channel, because that would contradict effectiveness of the KPN. The argument can be repeated for the processes q and r and so forth. Since the number of processes is finite, this implies that there is a set of processes that remain blocked and are waiting for each other, i.e., a local deadlock. Again, this deadlock must be artificial, because effectiveness of the KPN implies that the reading of tokens from the full channel c cannot depend on a real local deadlock.

Lemma 2. *The scheduling strategy of Definition 13 applied to a PNB of a bounded and effective KPN leads a finite number of times to an artificial deadlock.*

Proof. The KPN is bounded, thus there exists a capacity b of tokens, such that the PNB with bounds $\bar{b}$, where $\bar{b}(c) = b$ for all $c \in C$, has a fair and maximal execution. Then the sum of positive differences between b and the capacities of the full FIFOs in some deadlock is a measure that decreases with every resolution of *this* deadlock (Proposition 3) and does not increase with the resolution of other deadlocks. At the latest when this measure reaches zero, this deadlock can no more occur (Corollary 3). There is only a finite number of different deadlocks and thus at some point no more artificial deadlock can occur.

Lemma 3. *The scheduling strategy of Definition 13 applied to a PNB of a bounded and effective KPN produces a fair and maximal execution.*

Proof. Step 1 of the scheduling strategy guarantees progress on all actions, except write actions to full channels. That these blocking actions cannot persist follows from the fact that a persisting blocking write action leads to an artificial local deadlock (Lemma 1). The deadlock is resolved by the scheduling strategy. A new deadlock can occur only a finite number of times (Lemma 2). Thus eventually, the blocked write actions must become enabled thereby guaranteeing fairness and maximality.

This brings us to the main result, namely that the introduced scheduling strategy is correct for the class of KPNs that are bounded and effective.

Theorem 2. *The scheduling strategy of Definition 13 applied to a PNB of a bounded and effective KPN results in an execution that satisfies both correctness requirements of Definition 12.*

Proof. (Output Completeness) The output conforms to the denotational semantics if the execution is maximal and fair (Theorem 1). That the execution is maximal and fair follows from Lemma 3. (Boundedness) First note that the strategy increases the buffer sizes with a finite amount with every deadlock detected. An unbounded schedule can only be the result of an infinite number of deadlocks. According to Lemma 2, only a finite number of times a deadlock can occur.

To conclude, we reflect on the restrictions of boundedness and effectiveness on the class of KPNs, and on the influence of fairness implicit in these restrictions.

Theorem 3. *There exists no scheduler that correctly schedules (i) all effective KPNs, (ii) all bounded KPNs, or (iii) all KPNs for which a maximal, bounded and effective (but possibly unfair) execution exists.*

Proof. (i) It is known [12] that there exist KPNs that are not bounded (but still effective). It is obvious that they cannot satisfy the boundedness and output completeness requirement. (ii) It follows from the example in Section 2 (Figure 3) that no scheduler exists that can schedule the described collection of KPNs in bounded memory *and* with complete output. (iii) A maximal, bounded and effective execution may exist that is not fair. It may be that it is bounded only because a part of the network is never scheduled. It may still be output complete if that part of the network does not produce any output. Then a bounded and effective execution may exist that doesn't execute that part of the network. A scheduler however cannot decide in general whether any part of the network contributes to the output and must schedule it, leading to unbounded memory usage.

6 Conclusions

(Kahn) process networks are a suitable model of computation and programming model for streaming-based multimedia applications. The Kahn Principle states that any operational implementation that respects some loose fairness constraints realises the behaviour specified by a KPN. The scheduling algorithm proposed in [12], and used for many implementations of KPNs [1, 6, 9, 14, 15], employs a scheduling and artificial-deadlock resolution strategy that does not guarantee fairness. In this paper, we have presented an alternative scheduling

strategy that solves this problem and we have proved that for a very broad class of KPNs (called bounded and effective), this scheduler realises the correct behaviour. As future work, we would like to study an efficient implementation and the optimisation of scheduling as done in [2] for Parks' algorithm of [12]. We would also like to investigate the implications of distributed execution of KPNs.

References

1. G.E. Allen, B. Evans, and D. Schanbacher. Real-time sonar beamforming on a UNIX workstation using process networks and POSIX threads. In *Proc. of the 32nd Asilomar Conference on Signals, Systems and Computers*, pages 1725–1729. IEEE Computer Society, 1998.
2. T. Basten and J. Hoogerbrugge. Efficient execution of process networks. In A. Chalmers, M. Mirmehdi, and H. Muller, editors, *Proc. of Communicating Process Architectures 2001, Bristol, UK, September 2001*, pages 1–14. IOS Press, 2001.
3. S. Brookes. On the Kahn principle and fair networks. Technical Report CMU-CS-98-156, School of Computer Science, Carnegie Mellon University, 1998. Presented at FMPS'98. Submitted to Theoretical Computer Science.
4. J.T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model*. PhD thesis, University of California, EECS Dept., Berkeley, CA, 1993.
5. A.A. Faustini. An operational semantics for pure dataflow. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, pages 212–224. Springer, 1982.
6. M. Goel. Process networks in Ptolemy II. Technical Memorandum UCB/ERL No. M98/69, University of California, EECS Dept., Berkeley, CA, December 1998.
7. G. Kahn. The semantics of a simple language for parallel programming. In J.L. Rosenfeld, editor, *Information Processing 74: Proceedings of the IFIP Congress 74, Stockholm, Sweden, August 1974*, pages 471–475. North-Holland, 1974.
8. G. Kahn and D.B. MacQueen. Coroutines and networks of parallel programming. In B. Gilchrist, editor, *Information Processing 77: Proceedings of the IFIP Congress 77, Toronto, Canada, August 8-12, 1977*, pages 993–998. North-Holland, 1977.
9. E.A. de Kock et al. YAPI: Application modeling for signal processing systems. In *Proc. of the 37th. Design Automation Conference, Los Angeles, CA, June 2000*, pages 402–405. IEEE, 2000.
10. E.A. Lee. Overview of the Ptolemy project. Technical Memorandum UCB/ERL No. M01/11, University of California, EECS Dept., Berkeley, CA, March 2001.
11. E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *IEEE Proceedings*, 75(9):1235–1245, September 1987.
12. T.M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, University of California, EECS Dept., Berkeley, CA, December 1995.
13. E.W. Stark. Concurrent transition system semantics of process networks. In *Proc. of the 1987 SIGACT-SIGPLAN Symposium on Principles of Programming Languages, Munich, Germany, January 1987*, pages 199–210. ACM Press, 1987.
14. R.S. Stevens, M. Wan, P. Laramie, T.M. Parks, and E.A. Lee. Implementation of process networks in Java. Technical Memorandum UCB/ERL No. M97/84, University of California, EECS Dept., Berkeley, CA, 1997.
15. J. Vayssière, D. Webb, and A. Wendelborn. Distributed process networks. Technical Report TR 99-03, University of Adelaide, Department of Computer Science, South Australia 5005, Australia, October 1999.