AUTOMATA AND CONCURRENCY

Lutz PRIESE


Communicated by M. Nivat
Received May 1981
Revised March 1982

Abstract. The purpose of this paper is twofold: to give a precise notion of a realization (or simulation) of one concurrent system by another, and to study the relations of modular concurrent systems and non-persistent (i.e. with conflicts) concurrent systems in an automata-theoretical style.

We will introduce a conception of realization that obeys three requirements:
- it allows for proper hierarchies in certain classes of concurrent systems,
- it allows for normal-form theorems, and the standard constructions of the literature remain realizations in our formal concept,
- it clarifies some counter-intuitive examples.

Further, although our realization conception is developed to translate the computational aspects of concurrent systems, it also gives a formal tool for the handling of synchronization problems. The investigated concept of APA-nets (asynchronous, parallel automata nets) is by definition a modular system of networks of (finite state) machines and seems to be quite reasonable for modeling asynchronous, distributed systems. This modular approach is in some contrast to the non-persistence of Petri-nets, as in APA-nets conflicts may arise only as some non-determinism within components of the nets but not in the structure of the nets. As a by-product we also investigate the relation of conflicts and non-determinism in concurrent systems.

Contents

1. Introduction ................................................. 221
2. I/O-systems .................................................. 223
3. Automata .................................................... 227
4. Simulation .................................................... 234
5. A hierarchy-theorem ...................................... 241
6. Modular decomposition .................................... 253
7. Modular Petri-nets ......................................... 260
8. Conclusions and discussion .............................. 261
References ..................................................... 263

1. Introduction

Research on concurrency is understood in this paper as a study on distributed systems with several activities (e.g. processors) that are not synchronized by a common clock. We research a model of concurrent systems where the components are modules with precisely defined interfaces. A ‘modular’ behavior shall mean
that any module reacts only on its state and the status of its interface but cannot react to the activities of other system components that do not influence its interface (as is sometimes done in Petri-nets, e.g., to solve hidden conflicts). We will further assume that these modules may be described as (finite or infinite) automata and will investigate asynchronous parallel automata nets (APA-nets). The relationship with Petri-nets is carefully studied. Therefore we give a precise definition of what a ‘simulation’ of a concurrent system by another shall mean. This definition is adjusted to the ‘computational abilities’ of concurrent systems. However, synchronization problems can also be managed via simulations, because our conceptions includes interfaces of modules that allow a handling of I/O-procedures.

Unfortunately a formal correct approach towards ‘concurrency’, ‘simulation’, ‘realization of I/O behavior’, ‘interfaces’, etc., needs a certain apparatus and some work. In discussions with engineers I was often confronted with the argument that a conception of a computation or realization of a concurrent system is intuitively clear: “A system, B, simulates a system, A, if B has the same ‘input-output-behavior’ as A if B is started with states attached to some states of A”.

However, such an intuitive idea is by no means clear and cannot be easily translated into a correct statement. Let us regard as a simple example the four systems A, B₁, B₂ and B₃ of Fig. 1. If one looks carefully to those systems their intuitive input-output-behavior is the same if nothing is known about the ‘delays’ of the firing of the events. For a theory of synchronized or even only ‘fuzzy’-synchronized systems one can quite easily distinguish those systems by a mere inspection of their input-output-behavior. But for concurrent systems where in general nothing is known about delays all four ‘black boxes’ cannot be distinguished from their I/O behavior. Nevertheless most engineers agreed that B₁ simulates A, B₂ does not simulate A, and for B₃ it depends on further non-clear ideas of what a simulation (or realization, we use both terms synonymous here) may be. In our formal concept B₁ is a prompt simulation, B₂ a non-prompt simulation, and B₃ no simulation of A.

To show the power of such a formal approach we will present some mathematical rigorous proofs that certain well-known classes of concurrent systems are less powerful than others.
In the final chapters we present some 'normal-to-theorems' stating that any APA-net can be (promptly) simulated by APA-nets built up by only a few and simple standardized modules. These theorems also give modular decompositions of Petri-nets. By these characterizations some insight in the power of existing classes of control modules for concurrent computations can be achieved, such as for the modules of Bruno and Altman [4], Keller [16], Yoeli and Brzozowski [38], or the MIT-modules (of Patil, Dennis) [6, 7, 26, 27], see also [34]; compare also the different modules investigated by Banning [2].

We will present several transformations that transform a given net, \( A \), into a simulating net, \( B \). If one is interested in Petri-net languages it should be noted that these transformations also preserve the \( L^A \)-languages of Hack [10], where events may be named by the empty word \( \lambda \). Most of the presented transformations are new and generalize some results of the literature. However, we will not follow the language approach but will work with a simulation conception for the computational aspects of nets that allows for some finer hierarchies results. Thus, if some readers do not like the formal apparatus of Section 4 they have to drop the proofs for the impossibility results of Section 5 but may nevertheless be interested in our hierarchy results of Sections 6 and 7 that do—of course—hold also for weaker (and simpler) realization conceptions (such as \( L^A \)-languages).

2. I/O-systems

Recently it has become quite common to study concurrency on such abstract levels as state-systems and/or transitions-systems; see, e.g., [17].

**Definition 1.** An S-system (state system) \( A \) is a tuple \( A = (S, \to) \) of a set \( S \) (of states) and a relation \( \to \subseteq S \times S \) (of transitions).

By \( \Rightarrow \) we denote the reflexive and transitive closure of \( \to \). For \( (s, s') \in \to \) we also write \( s \to s' \) and say, that "\( s' \) is direct successor of \( s'' \)" or that "\( A \) computes in one step from \( s \) to \( s'' \)" or similar.

\[ \mathcal{R}(s) := \{ s' \in S : s \Rightarrow s' \} \]

is called the 'reachability-set' of \( s \). Also: \( \mathcal{R}(K) := \bigcup_{s \in K} \mathcal{R}(s) \) for \( K \subseteq S \).

It becomes often necessary to name transitions:

**Definition 2.** A T-system (transition-system) \( A \) is a tuple \( A = (S, \Sigma, \to) \) of

(i) an S-system \( (S, \to) \), and

(ii) a finite set \( \Sigma \) (of names for transitions), such that for any \( \epsilon \in \Sigma \) there exists a relation \( \to_{\epsilon} \subseteq S \times S \) with \( \to = \bigcup_{\epsilon \in \Sigma} \to_{\epsilon} \).

We use the following notations: \( \forall s, s' \in S: \)

By induction:

\[ \forall \lambda \in \Sigma^*: \quad \forall \epsilon \in \Sigma: \]
Any T-system is by definition also an S-system. On the other hand, any S-system $A$ without interest in names for the transitions will be regarded to be the T-system $A = (S_A, \{A\}, \rightarrow)$, where all transitions receive the trivial name $A$ itself. With this canonical identification in mind all further notations and definitions for T-systems not referring to $\Sigma$ also hold for S-systems. We also use quite frequently suffixes, such as $S_A$, that should be self-explanatory.

Some common conceptions of the theory of concurrent systems may easily be stated on this level of abstraction. A few examples are given:

**Definition 3.** For any T-system $A = (S, \Sigma, \rightarrow)$, any $S' \subseteq S$ and any $e \in \Sigma$.

- $e$ is live with respect to $S'$ if $\forall S \in \mathcal{R}(S'): \exists x \in \Sigma^*: s \rightarrow e$, $S'$ is live if $\forall e \in \Sigma: e$ is live with respect to $S'$.
- $S'$ is hang-up-free if $\forall S \in \mathcal{R}(S'): \exists e \in \Sigma: s \rightarrow e$, $A$ has a dimension $d$ if $S \subseteq \mathbb{N}_0^d$.

Some coordinate $i$ of $\mathbb{N}_0^d$ of a $d$-dimensional T-system is called $k$-safe (with respect to $S'$) if $\forall S \in \mathcal{R}(S'): (s_i \cdot k$ —here $(\cdot)$ denotes the $i$th coordinate of a vector— $S'$ is safe if $i$ is $k$-safe for some $k \in \mathbb{N}_0$ (with respect to $S'$).

$A$ is called live, hang-up-free or ($k$-) safe if there exists a set $S' \subseteq S_A$, that is given from the context, such that $S'$ is live, hang-up-free, or any coordinate of $A$ is ($k$-) safe with respect to $S'$.

**Example.** Any Petri-net is a T-system in a canonical sense. Let us briefly repeat the conception of a Petri-net.

A Petri-net graph (PNG), $N$, is a directed bicolored graph $(V, F)$, $F \subseteq V \times V$, using two colors $P$ and $T$. Any vertex colored by $P$ is called a place and colored by $T$ a transition, respectively. $P_\|$, $T_\|$ are the sets of all places and transitions, respectively, of $N$. For a vertex $v$ of $N$, $v := \{v' \in V; v \in Fv\}$ denotes the set of vertices of $N$ pointing by a directed arc to $v$ and $v^\| := \{v' \in V; v^\|Fv\}$ the set of all vertices of $N$ points to. For a set $V' \subseteq V$ there holds $V' := \bigcup_{v \in V'}v$, and $V'' := \bigcup_{v \in V}v'$. As $N$ is bicolored there holds $T_\| \cup P_\| \subseteq P_\|$ and $P_\| \cup P_\| \subseteq T_\|$. For a graphical
presentation places are drawn as circles and transitions as bars, see Fig. 1, e.g. A state (often called a ‘marking’) of a PNG, \(N\), is a mapping \(s: P_N \rightarrow \mathbb{N}_0\) (where \(\mathbb{N}_0\) is the set of all (positive) integers) and \(S_N := \mathbb{N}_0^{P_N}\) is the set of all states of \(N\). As we identify finite mappings with vectors we may equivalently regard \(S_N = \mathbb{N}_0^{P_N}\). A (1-step) derivation \(\rightarrow_N\) of \(N\) is a relation \(\rightarrow_N \subseteq S_N \times S_N\) with

\[
\forall s, s' \in S_N: \forall e \in T_N: s \rightarrow s' \iff s(p) > 0 \forall p \in e,
\]

\[
\begin{cases}
  (s(p)-1, \forall p \in e-e', \\
  s'(p) = (s(p)+1, \forall p \in e-e', \\
  s(p), \text{ elsewhere})
\end{cases}
\]

and

\(\rightarrow_n := \bigcup_{\epsilon \in \mathcal{E}} r_n \rightarrow_e\).

A Petri-net (PN), \(N\), is a PNG, \(N\), together with \(S_N\) and \(\rightarrow_N\). Thus, a PN, \(N\), is by definition a T-system \(N = (S_N, T_N, \rightarrow_N)\) of dimension \(\neq P_N\).

A generalized PN (GPN) allows in addition for arc-bundles between two vertices, while a restricted PN (RPN) is a PN with no loops of lengths 2; for formal definitions see the literature, e.g. Hack [10]. If \(s \rightarrow_e s'\) holds one usually says that \(e\) fires from \(s\) to \(s'\), or similar. \(e\) is also called ‘fireable’ in some state \(s\) if \(s \rightarrow_e\) holds.

With this important model for concurrency in mind we also call the coordinates of a \(d\)-dimensional T-system \(A\) the places of \(A\) and denote by \(P_A\) the set of all places of \(A\). A state of a \(d\)-dimensional T-systems \(A\) is usually written as

\(s = (s(1), \ldots, s(p_d))\) with \(p_i \in P_A\) for \(1 \leq i \leq d\).

We need a few more definitions:

**Definition 4.** A T-system \(A = (S, \Sigma, \rightarrow)\) is called

1. locally determined \(\iff\) \(\forall e \in \Sigma: \forall s, s', s'' \in S:\)

   \((s \rightarrow_e s' \text{ and } s \rightarrow_e s'') \rightarrow s' = s'')\),

2. commutative \(\iff\) \(\forall e, e' \in \Sigma: \forall s \in S:\)

   \((s \rightarrow_{e'} s' \text{ and } s \rightarrow_{e} s'' \rightarrow s' = s'')\),

3. persistent \(\iff\) \(\forall e, e' \in \Sigma: \forall s \in S:\)

   \((e \neq e' \text{ and } s \rightarrow_{e'} s' \text{ and } s \rightarrow_{e} s'')\),

4. confluent \(\iff\) \(\forall s, s', s'' \in S:\)

   \((s \Rightarrow s' \text{ and } s \Rightarrow s'')\)

   \(\exists s' \in S: s' \Rightarrow s' \text{ and } s'' \Rightarrow s'\),

5. modular \(\iff\) \(S\) is persistent and commutative.

Only this last notation is new. It tells that any \(e \in \Sigma\) of a modular T-system behaves like a module: once \(e\) is activated in some state \(s\) \((s \rightarrow_e \text{ holds})\) then \(e\) can act independently of the activities of the further components \(e' \in \Sigma\).
As locally determined, commutative and persistent T-systems are confluent (see, e.g. Keller [17])—and confluency is certainly too restrictive for a general theory of concurrent systems—we have to deal with modular systems that cannot be locally determined.

Modularity seems to be a key for distributed systems as one usually wants all components to communicate only via well-defined interfaces. Note that events of Petri-nets are not modules in this sense, as the input-places and output-places of an event do not define the interface of this event: in order to solve possible conflicts with other events the events have to 'agree' which one fires via some invisible structure not presented in the Petri-net concept.

In the following we will leave the common approaches to concurrency in the style of S- and T-systems and will introduce I/O-places as an interface.

**Definition 5.** An I/O-system $A$ is a T-system $A = (S, \Sigma, \rightarrow)$ with some dimension $d$ and two distinguished sets $I_A, O_A \subseteq P_A$ with $I_A \cap O_A = \emptyset$ of inputs and outputs such that there holds:

$$\forall s, s' \in S: s \rightarrow s' \Rightarrow s(p) \equiv s'(p) \ \forall p \in I_A \text{ and }$$

$$s(p) \equiv s'(p) \ \forall p \in O_A.$$  

We also denote $A$ as the tuple $A = (I_A, O_A, S_A, \rightarrow_A)$ to point explicitly to the interface of $A$.

An I/O-procedure, $P$, for an I/O-system $A$, is a relation $P \subseteq S \times S$ with:

$$\forall s, s' \in S: s \stackrel{P}{\rightarrow} s' \Rightarrow s(p) \equiv s'(p) \ \forall p \in I_A,$$

$$s(p) \equiv s'(p) \ \forall p \in O_A,$$

$$s(p) = s'(p) \ \forall p \in P_A - (I_A \cup O_A).$$

We use the following notations: $W_A := P_A - (I_A \cup O_A)$ is the set of 'inner places' of $A$. For $\mathbb{N}_0 \times \mathbb{N}_0, P \subseteq P_A, s, s' \in S$, we write $sR_s' \mod P \Rightarrow \forall p \in P: s(p)R_s'(p)$.

For any I/O procedure $P$ for $A$, $s, s' \in S$, we define by induction

$$s \stackrel{1_H}{\rightarrow} s' \Rightarrow s \stackrel{A}{\rightarrow} s',$$

$$s \stackrel{A}{\rightarrow} s' \Rightarrow \exists s_1, s_2 \in S: s \stackrel{A}{\rightarrow} s_1 \text{ and } s_1 \stackrel{P}{\rightarrow} s_2 \text{ and } s_2 \stackrel{A}{\rightarrow} s'. $$

Also

$$s \stackrel{\Delta}{\rightarrow} s' \Rightarrow \exists n \in \mathbb{N}_0: s \stackrel{A^n}{\rightarrow} s'.$$

The T-system $(A, \mathbb{P}) := (S_A, \Sigma \cup \{\mathbb{P}\}, \rightarrow_A \cup \mathbb{P})$ is called the $\mathbb{P}$-closure of $A$.

$(A, \mathbb{P})$ is regarded as a T-system without inputs and outputs. Any T-system, $A$, where we are not interested in inputs and outputs, is also regarded to be an I/O-system with $I_A = O_A = \emptyset$. The interface of an I/O-system is given by its inputs.
and outputs. A can, by the above definition, take off signals from its inputs and send out signals via its outputs (property (ii)). The environment can communicate with A via I/O-procedures that put signals on the inputs of A and take signals from the outputs, but that cannot change the 'inner' state of A (this is $S_A \mid W_A$). $s \rightarrow_{A,P} s'$ means that $s'$ can be computed in A from s with exactly n applications of the I/O-procedure P, i.e., exactly n additional communications of A with its environment took place. When we operate with distributed asynchronous systems we may regard a module A to be an I/O-system with an interface $I_A, O_A$ or we may regard it as a closed system where its communication $P$ via its interface is incorporated into its transitions, $(A, P)$.

This definition of an I/O-procedure is quite general and a further theory should investigate more specific procedures.

3. Automata

Definition 6. An s-automaton (sequential automaton) $A$ is tuple $A = (I, O, S, \rightarrow)$ of

(i) finite sets $I, O$ (of input- and output-lines) with $I \cap O = \emptyset$,

(ii) a set $S$ (of states), and

(iii) a transition relation $\rightarrow \subseteq (I \times S) \times (O \times S)$.

$A$ is

finite $\iff S$ is a finite set,

determined $\iff \rightarrow$ is a functional relation.

It is customary in automata theory to code the letters of the input- and output-alphabets $I, O$ by vectors of $\{0, 1\}^l$ with $l = \lfloor \log_2 \# I_A \rfloor$, or $\lfloor \log_2 \# O_A \rfloor$, respectively. However, we have a different approach in mind: any $a \in I \cup O$ shall represent exactly one input- or output-line of the 'black box' A. I.e., we allow only for a '1-out-of-n' code. A transition $x, s \rightarrow_A y, s'$ is read: when A in state s receives a signal via its input (line) x A takes off this signal, sends out one signal to the output j and switches to state $s'$.

Note that the state concepts of S-systems and automata differ: an I/O-signal of an automaton is regarded to be a component of a state for an S-system. For finite automata we always regard S as a sub-set of $\mathbb{N}_0$ but are free to use mnemonic names for the states.

A Normed Network, N, is a net of s-automata where at most one signal may pass N. We give a definition of the semantic of Normed Networks in terms of s-automata from which the graph structure can easily be restored.

Definition 7. For some given s-automata $A_1, \ldots, A_n$ the class $NN(A_1, \ldots, A_n)$ of all Normed Networks over $A_1, \ldots, A_n$ is the smallest class of s-automata that

(i) contains $A_1, \ldots, A_n$ and

(ii) is closed under the operations feed-back and product.
Here feedback and product are defined as: For any s-automaton $A$, $x \in I_A$, $y \in O_A$, the feedback $A^y$ of the output $y$ to the input $x$ is the s-automaton

$$A^y := (I_A - \{x\}, O_A - \{y\}, S_A, \rightarrow_{A^y}),$$

with

$$\rightarrow_{A^y} := \text{Cl}(\rightarrow_A \cup \{y, s \rightarrow x, s ; s \in S_A\})$$

$$\cap \{i, s \rightarrow o, s'; i \in I_A - \{x\}, o \in O_A - \{y\}, s, s' \in S_A\}$$

and Cl denotes the transitive and reflexive closure of any relation.

For any s-automata $A$ and $B$ the product $A/B$ of $A$ and $B$ is the s-automaton

$$A/B := (I_A \cup I_B, O_A \cup O_B, S_A \times S_B, \rightarrow_{A/B}),$$

with

$$\rightarrow_{A/B} := \{x, (s_1, s_2) \rightarrow y, (s_1', s_2'); x, s_1 \rightarrow_A y, s_1', s_2 = s_2'$$

or $x, s_2 \rightarrow_B y, s_2'$ and $s_1 = s_1'\}.$

**Some examples.** $K, H, E, D, P, F$ are the following s-automata with graphical presentations as shown in Fig. 2.

$$K := ([1, 2], [3], \{a\}, \{1, a \rightarrow 3, a ; 2, a \rightarrow 3, a\}).$$

$K$ acts as a UNION of wires.

$$H := ([t, d'], \{t, t', u, d', d''\}, \{\text{up, down}\}, \rightarrow),$$

with the transitions

$$t, \text{up} \rightarrow t', \text{up}, \quad u, \text{down} \rightarrow u', \text{up}.$$
Automata and concurrency

$t, \text{down} \to t^d, \text{down}, \quad d, \text{up} \to d', \text{down},$

$u, \text{up} \to u', \text{up}, \quad d, \text{down} \to d', \text{down}.$

$H$ acts as a switch. 'Switches' are also $E$, $D$ and $P$:

$$E := (\{t, s\}, \{t'', t'^d, s'\}, \{\text{up, down}\}, \to),$$

with

$t, \text{up} \to t'', \text{up}, \quad s, \text{up} \to s', \text{down},$

$t, \text{down} \to t'^d, \text{down}, \quad s, \text{down} \to s', \text{up}.$

$$D := (\{t, s\}, \{t'', t'^d\}, \{\text{up, down}\}, \to),$$

with

$t, \text{up} \to t'', \text{up}, \quad s, \text{up} \to t'^d, \text{down},$

$t, \text{down} \to t'^d, \text{down}, \quad s, \text{down} \to s', \text{up}.$

$$P := (\{s_1, s_2\}, \{s'', s'^d\}, \{\text{up, down}\}, \to),$$

with

$s_1, \text{up} \to s'', \text{down}, \quad s_2, \text{up} \to s'^d, \text{down},$

$s_1, \text{down} \to s'^d, \text{up}, \quad s_2, \text{down} \to s'', \text{up}.$

$F$ is a kind of Flip-Flop:

$$F := (\{s\}, \{s'', s'^d\}, \{\text{up, down}\}, \to),$$

with

$s, \text{up} \to s', \text{down}, \quad s, \text{down} \to s'^d, \text{up}.$

Fig. 3 gives an example of a Normed Network over $H$ and $K$. Note that this net simulates just the $s$-automaton $E$ (in the sense of Hartmanis and Stearns [13]).

**Theorem 1.** A finite set $\mathcal{M}$ of finite, determined $s$-automata is called a base iff for any finite, determined $s$-automaton, $A$, there exists a Normed Network, $N$, over $\mathcal{M}$ that simulates $A$.

Then there holds: The sets $\{K, H\}$, $\{K, E\}$, $\{K, P\}$ and $\{K, F, D\}$ are bases.

Fig. 3. $H_1$ and $H_2$ are supposed to have the same state.
For a proof see [25] and [33].

There exists a rich (but difficult to available) literature on Normed Networks, including several thesis. On a first glimpse Normed Networks seem to be quite restrictive as only one signal is allowed in a network—however, this standardization is interesting as for Normed Networks the sequential-, concurrent- and synchronized-operation modi (trivially) coincide. Thus Normed Networks are an important subclass of most computational models, whose study gives a lot of general (and also surprising) results. Without an understanding of Normed Networks this article would have never been written. For some (easier receivable) papers on this subject see [18, 19, 23, 24, 25, 29, 30, 31, 32, 33, 35, 36, 37], e.g.

We will now generalize s-automata to asynchronous, parallel automata:

**Definition 8.** An (asynchronous, parallel) automaton $A$ is a tuple $A = (I, O, S, \rightarrow)$, with $I, O, S$ as before, but $\rightarrow$ is a relation $\rightarrow \subseteq (\Psi(I) \times S) \times (\Psi(O) \times S)$. $A$ is finite $\Leftrightarrow S$ is a finite set ($S \subseteq \mathbb{N}_0$, without loss of generality).

We again adopt the model that any $x \in I \cup O$ refers to one I/O-line. A transition $X, s \rightarrow_A Y, s'$ shall be understood that in state $s$ $A$ takes off one signal from each inputline of $X$, sends out one signal onto each output of $Y$ and switches to state $s'$. Thus, in order that a transition $X, s \rightarrow_A Y, s'$ is ‘activated’ there must be at least one signal on each input $v \in X$ and $A$ has to be in state $s$. We always identify an automaton $A$ with the following I/O-system:

$$A = (\mathbb{N}_0^n \times \mathbb{N}_0^m \times S_A, \{A\}, \rightarrow),$$

with $n = \# I_A$, $m = \# O_A$ and

$$\forall s, s' \in S: s \rightarrow s' \Leftrightarrow \exists X \subseteq I_A; \exists Y \subseteq O_A; \exists s_1, s_2 \in S_A;$$

$$X, s_1 \rightarrow_A Y, s_2,$$

$$s_1 = s \mod S_A, s_2 = s' \mod S_A,$$

$$s > 1 \mod X, \text{ and}$$

$$s' = \begin{cases} s - 1 \mod X, \\ s + 1 \mod Y, \\ s \quad \text{elsewhere.} \end{cases}$$

Although the state sets and the transitions $\rightarrow$ for $A$ regarded as an automaton or as an I/O-system are different they receive the same notation. However, their relationship is straight-forward and no confusion should arise. Note that the name for the transitions of an automaton regarded as an I/O-system is $A$ itself. Thus on the level of T-systems we will not distinguish different transitions of the same automaton by different names. Any automaton is also a modular but generally non locally determined I/O-system that consists only of one module, namely itself.
Table 1 gives several examples of automata we will use later. Also all s-automata are automata with transitions \( X, s \to Y, s' \) with sets \( X, Y \) of cardinality 1.

Table 1

<table>
<thead>
<tr>
<th>Automaton</th>
<th>Transition</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>( 1 \to 2 )</td>
<td>( a \to {2, 3}, a )</td>
</tr>
<tr>
<td>W</td>
<td>( 1 \to 2 )</td>
<td>( {1, 2}, a \to 3, a )</td>
</tr>
<tr>
<td>S</td>
<td>( 1 \to 2 )</td>
<td>( 1, \text{on} \to 1', \text{off} ), ( 2, \text{on} \to 2', \text{off} ), ( m, \text{off} \to 0, \text{on} )</td>
</tr>
<tr>
<td>SH(_m^n)</td>
<td>( i, \text{on} \to i', \text{off} )</td>
<td>( 1 \leq j &lt; n )</td>
</tr>
<tr>
<td>Seq(_n^0)</td>
<td>( i, \text{on} \to o', \text{on} )</td>
<td>( 1 \leq i &lt; n )</td>
</tr>
<tr>
<td>SEQ:=Seq(_2)</td>
<td>( i, \text{on} \to i', \text{off} )</td>
<td>( 1 \leq i &lt; 2 )</td>
</tr>
<tr>
<td>S(_m^n)</td>
<td>( s, \text{on} \to s', \text{off} )</td>
<td>( 1 \leq i &lt; 2 )</td>
</tr>
<tr>
<td>T(_2)</td>
<td>( s, \text{on} \to s', \text{off} )</td>
<td>( a_i, \text{off} \to 0, \text{on} )</td>
</tr>
<tr>
<td>R</td>
<td>sub, ( n \to \text{add}', n + 1 )</td>
<td>( \forall n \in \mathbb{N} )</td>
</tr>
<tr>
<td>A(_m^n)</td>
<td>( 1 \to 2 )</td>
<td>( {1, \ldots, n}, a \to {1, \ldots, m}, a )</td>
</tr>
</tbody>
</table>
Definition 9. An APA-net (asynchronous, parallel automata net) over some given automata \( A_1, \ldots, A_n \) is a graph with copies of \( A_1, \ldots, A_n \) as nodes and connections of inputs and outputs as directed edges with the restriction that any edge connects exactly one output of some automaton copy with one input of the same or another copy. Let \( B_1, \ldots, B_m \) be the set of all copies of \( A_1, \ldots, A_n \) occurring in \( N \). Any \( B_i \) is called a component or module of \( N \).

\( I_N \) (\( O_N \)) is the set of all inputs (outputs) of modules of \( N \) that are not connected via edges and is regarded to be the set of all inputs (outputs) of \( N \). \( S_C := S_{B_1} \times \cdots \times S_{B_m} \)—where we regard all \( B_i \) as automata and not as systems—is the set of all component-states (C-states) of \( N \).

Let \( I \) be the number of edges inside \( N \). Then \( S_W := \mathbb{N}_0^I \) is the set of all wire-states (W-states) of \( N \).

We regard \( N \) to be also the following I/O-system:

\[
N = (\mathbb{N}_0^n \times I_N \times O_N \times S_W \times S_C, \{B_1, \ldots, B_m\}, \rightarrow),
\]

where

\[
\forall s, s' \in S_N: \quad s \xrightarrow{B_i} s' \quad \text{implies} \quad \exists X \subseteq I_{B_i}, \exists Y \subseteq O_{B_i}: \exists s_1, s_2 \in S_{B_i}:
\]

\[
X, s_1 \xrightarrow{B_i} Y, s_2, \\
s = s_1 \mod S_{B_i}, s' = s_2 \mod S_{B_i}, \\
s \equiv 1 \mod X, \text{ and}
\]

\[
\begin{cases}
  s - 1 \mod X - Y, & \text{if } s \geq 1 \\
  s + 1 \mod Y - X, & \text{else}
\end{cases}
\]

where \( X \) and \( Y \) are canonically coordinates of \( I_N, O_N \) and/or \( S_W \) depending on the structure \( B_i \) is interconnected in \( N \). (\( X \cap Y \neq \emptyset \) may hold as in \( N \) an output of \( Y \) of some component may be fed back to an input of \( X \) of the same component.)

\( \text{APA}(A_1, \ldots, A_n) \) denotes the class of all APA-nets over \( A_1, \ldots, A_n \). \( \text{APA}' \) denotes the class of all APA-nets over some finite automata.

By definition any APA-net is a modular I/O-system. APA-nets are very closely related with Keller’s speedindipendent modules [16]. As a main difference we do not force APA-nets to be 1-safe I/O-systems—as Keller’s module have to be—but allow accumulation of signals on wires. This more general approach seems to be quite helpful, even for a later research of 1-safe nets.

Example. We regard the net \( N_f \) of Fig. 3 as an APA-net. The inputs, outputs and wires become coordinates of the state-vectors. We use the order as shown in Fig. 3.
Thus the state \((3,0,0,2,0,1,0,0,0,0,\text{up},\text{down})\) means that there are 3 signals on input \(s\), 2 signals on output \(r^d\), 1 signal on wire 7, \(A\) is in state 'up' and \(B\) in state 'down'. As \(#S_K - 1\) we dropped this coordinate.

If we regard \(N_E\) to be a 1-safe net (i.e., we operate only with states \(S' \subseteq S_N\) such that \(N_E\) is 1-safe with respect to \(S'\)), then \(N_E\) 'behaves the same way as \(E\) does'. However, if we drop the 1-safeness this is no longer true. Note that \(E\) is an I,O-system as shown in Fig. 4.

Two signals on input \(s\) in C-state 'up' of \(E\) result in the following computation:
\[
(2, 0, 0, 0, 0, \text{up}) \rightarrow_E (1, 0, 0, 0, 1, \text{down}) \rightarrow_E (0, 0, 0, 2, \text{up}),
\]
and the old state is restored. However, two signals on \(s\) in \(N_E\) may lead to an unwanted behavior:
\[
(2, 0, 0, 0, 0, 0, 0, 0, 0, 0, \text{up}, \text{up}) \rightarrow_A (0, 0, 0, 0, 0, 0, 2, 0, 0, 0, \text{down}, \text{down}) \rightarrow_K (0, 0, 2, 0, 0, 0, 0, 0, 0, 0, \text{down}, \text{down}),
\]
where 2 signals have correctly reached the output \(s'\) but \(N_E\) is in the incorrect C-state 'down'.

A rather simple but important difference of APA-nets and Petri-nets is

**Lemma 1.** APA-nets are modular, but generally non locally determined, T-systems. Petri-nets are locally determined, commutative, but generally non-persistent, T-systems.

A proof is trivial. This difference comes from the fact that APA-nets have no shared inputs but PN do.

Any finite automaton \(A = (I, O, S, \rightarrow)\) can easily be translated into an 'equivalent' Petri-net, \(N_A\). Simply define \(N_A\) to consist of the places \(P_{N_A} = I \cup O \cup S\), the events \(E_{N_A} = \rightarrow\), i.e., for any transition \(X, s \rightarrow Y, s'\) of \(A\) there exists an event \(e_{X,s,Y,s'} \in E_{N_A}\), and the Petri-net graph is defined in such a way that an event \(e_{X,s,Y,s'}\) has the places \(X \cup \{s\}\) as input-places and \(Y \cup \{s'\}\) as output-places. \(A\) in state \(s\) is represented
by exactly one token on place $s$ and the signals on the I/O-wires of $A$ are coded by tokens on the corresponding places. Thus the dynamic behavior of $A, \rightarrow_A$, is 'precisely translated' into $\rightarrow$ of the Petri-net $N_A$.

This simple translation is a hint that a simulation of modular systems is generally easier than a simulation of non-modular systems. The simulation of non-modular Petri-nets by our modular APA-nets becomes quite complicated, but also more interesting, and will be shown in the final sections. To do so we present in the following section a precise notion of simulation that gives a formal correct meaning of 'equivalent behavior', 'precisely translated . . .', etc.

It should be noted that we can easily corporate APA-nets and Petri-nets into a common, mixed net concept. An example is shown in Fig. 5. $N_H$ presents a mixed net whose component $H$ can only be used sequentially inside $N_H$, even if several signals accumulate on the inputs of $N_H$. This net can easily be formally defined as a Petri-net as we can translate by the above method all APA-components into Petri-net sub-parts. We will drop a formal definition for this mixed net conception here. They might be of some practical importance for a design of concurrent systems as they allow for modular APA-net subparts and non-modular Petri-net subparts for different purposes.

4. Simulation

Fortunately a definition for a simulation of concurrent systems can be given on the abstract level of S-systems which yields for many applications.
Definition 10. An S-system \( B = (S_B, \rightarrow_B) \) simulates an S-system \( A = (S_A, \rightarrow_A) \) iff

\[ \exists K: S_A \rightarrow B(S_B) - \{\emptyset\}, S_A \ni s \mapsto K, \forall s, s' \in S_A: \forall \bar{s} \in K_i: \]

(i) \( s \xrightarrow{A} s' \Rightarrow \exists \bar{s}' \in K_i: \bar{s} \xrightarrow{B} \bar{s}' \),

(ii) \( \exists \bar{s}' \in K_i: \bar{s} \xrightarrow{B} \bar{s}' \Rightarrow s \xrightarrow{A} s' \),

(iii) \( \forall \bar{s}^0 \in S_B: \bar{s} \xrightarrow{B} \bar{s}^0 \Rightarrow \exists s^+ \in K^+: \bar{s}^0 \xrightarrow{B} s^+ \),

where \( K^+ := \bigcup K(S_A) \subseteq S_B \) is the set of all attached states, and \( S_B - K^+ \) is the set of all intermediate states of \( B \). The states \( \bar{s} \in K_i \) are called attached or equivalent to \( s \).

We also say that \( B \) simulates \( A \) via (the simulation mapping) \( K \) in the above case. \( B \) simulates \( A \) \( k \)-promptly, for \( k \in \mathbb{N} \), iff in addition there holds:

(iv) \( \forall s, s' \in S_A: \forall \bar{s} \in K_i: \forall \bar{s}' \in K_i: \forall n \in \mathbb{N}: s' \in \delta^m_A(s) \) and \( \bar{s} \xrightarrow{B} \bar{s}' \)

\[ \Rightarrow \exists m \leq k \cdot n: \bar{s}' \in \delta^m_A(s) \].

Here we denote by \( \delta_C(s) \) for some S-system \( C = (S_C, \rightarrow_C) \), some state \( s \in S_C \) and some \( r \in \mathbb{N} \) the set

\[ \delta_C(s) := \{ s' \in S_C: \exists s_1, \ldots, s_r \in S_C: s = s_1, s' = s_{i+1} \text{ and } s_i \rightarrow s_{i+1} \text{ for all } i, 1 \leq i \leq r, \text{ and } \forall t \in \mathbb{N}: \forall s_1, \ldots, s_{i+1} \in S_C: (s = s_1, s' = s_{i+1} \text{ and } s_i \rightarrow s_{i+1} \text{ and } \forall i, 1 \leq i \leq t) \Rightarrow t \leq r \} \]

of all states that can be computed from \( s \) in less than \( r \) steps but not in more than \( r \) steps.

\( B \) simulates \( A \) promptly iff \( B \) simulates \( A \) \( k \)-promptly for some \( k \).

\( B \) simulates \( A \) \( n \)-safely (via \( K \)) iff, in addition to (i)-(iii), there holds:

(v) \( \forall S \subseteq S_A: A \) is \( 1 \)-safe with respect to \( S \)

\[ \Rightarrow B \) is \( n \)-safe with respect to \( \bigcup K(S) \).

Let us briefly discuss these requirements: By (i) any computation in \( A \) can also be fulfilled in \( B \), but may require more steps. All computations of \( B \) that start from some attached state and (ii) lead to an attached state again can also be done in \( A \), or (iii) if they lead to an intermediate state may always be prolonged in \( B \) to an attached state of \( K^+ \). In other words, \( B \) reflects all computations of \( A \) (via some coding \( K \) of the states) and allows for no hang-ups due to the process of simulation. However, if \( A \) has some hang-ups, then \( B \) may also have some.

Let us regard three extremely simple examples:

1. \( A = (S_A, \rightarrow_A) \) with \( S_A = \{s_0, s_1, s_2, s_3\} \) and \( \rightarrow_A \) is defined as \( s_0 \rightarrow s_i \) for \( 1 \leq i \leq 3 \).
A is a non-deterministic system (‘with degree 3 of non-determinism’). We simulate A by a system B (‘of degree 2’): \( B = (S_B, \to_B) \) with \( S_B = \{ s, \tilde{s}_0, \tilde{s}_1, \tilde{s}_2, \tilde{s}_3 \} \) with the transitions
\[ \tilde{s}_0 \to s, \quad \tilde{s}_0 \to \tilde{s}_1, \quad s \to \tilde{s}_2 \quad \text{and} \quad s \to \tilde{s}_3. \]
The state-mapping \( K \) is defined as \( K(s_i) := \{ \tilde{s}_i \} \) for \( 0 \leq i \leq 3 \). Thus \( K^* = S_B - \{ s \} \), i.e., \( s \) is a non-attached state of \( S_B \). It should be noted that there is no possible state-mapping \( K' : S_A \to \mathcal{P}(S_B) - \{ \emptyset \} \) satisfying Definition 10 such that \( K' = S_B \), i.e., \( s \) must be a non-attached state. The reason is simply that we have made an intermediate decision in \( s \), as we have excluded \( \tilde{s}_1 \) as the next state, but are still free to choose \( \tilde{s}_2 \) or \( \tilde{s}_3 \). Thus, \( s \) cannot belong to \( K_i \) for \( 0 \leq i \leq 3 \).

2. \( A = (S_A, \to_A) \) with \( S_A = \{ s_0, s_1 \} \) and \( s_0 \to s_1 \) holds. Let \( B = (S_B, \to_B) \) with \( S_B = \{ s, \tilde{s}_0, \tilde{s}_1, \tilde{s}_2, \tilde{s}_3 \} \) with \( \tilde{s}_0 \to s, \quad s \to s \) and \( s \to \tilde{s}_1 \). B simulates A, as we may define \( K(s_i) := \{ \tilde{s}_i \} \), \( 0 \leq i \leq 1 \), with \( s \) as a non-attached state. However, this simulation has some disadvantage as we may run in \( B \) on a path \( \tilde{s}_0 \to s \to s \to \cdots \to s \to \cdots \) forever. Nevertheless we fulfill all requirements of Definition 10 as we are able to reach a state of \( K^* \) from \( s \). Such a situation introduces a new ‘livelock’, a circle we may pass arbitrarily. To separate such constructions from ‘smooth’ ones we introduced the notion of a prompt simulation. In this example \( B \) simulates \( A \) but does not simulate \( A \) \( k \)-promptly for any \( k \). For promptness we require that any computation from some state \( \tilde{s} \in K^* \) to some attached state \( \hat{s} \in K(\delta_A^k(s)) \) must have reached \( \hat{K}(\delta_A^k(s)) \) in a bounded number of steps. Attention!—this does not imply that after a bounded number of steps from \( \tilde{s} \in K^* \) some state in \( \hat{K}(\delta_A^k(s)) \) for some \( n \) must have been reached, as for concurrent systems one may stay in non-attached intermediate states forever. Such an example is:

3. \( A = (S_A, \to_A) \) with \( S_A = \{ a_0, a_1 \} \times \{ h_j : j \in \mathbb{N} \} \) with the transitions
\[ (a_{i,j}, h_j) \to (a_{i,j}, h_{j+1}) \quad \forall j \in \mathbb{N} \]
and
\[ (a_{i,j}, h_j) \to (a_{i,j+1}, h_j) \quad \forall j \in \mathbb{N}, \quad 0 \leq i \leq 1. \]

\( A \) is a concurrent, distributed system with two local activities:

1. a finite activity: \( a_0 \to a_1 \).
2. an infinite activity: \( h_i \to h_{i+1}, \quad i \in \mathbb{N} \).

We define \( B = (S_B, \to_B) \) with \( S_B = \{ a, a_0, a_1 \} \times \{ h_j : j \in \mathbb{N} \} \), where the transitions result from the two local activities:

1. \( a_0 \to a \to a_1 \).
2. \( h_i \to h_{i+1}, \quad i \in \mathbb{N}. \)

\( B \) is a prompt simulation of \( A \) with the state-mapping \( \hat{K}(a_{i,j}, h_j) := \{ (a, h_j) \}, \quad 0 \leq i \leq 1, \quad j \in \mathbb{N} \), where \( (a, h_j) \) are non-attached states of \( S_B \) for all \( j \in \mathbb{N} \)— nevertheless there exists an unbounded non-attached computation in \( B \):
\[ (a_{i,j}, h_j) \to (a_{i,j}, h_{j+1}) \to (a_{i,j}, h_{j+2}) \to \cdots \to (a_{i,j}, h_{j+k}) \to \cdots. \]
If one allows for non-attached states (as one should, see our first example) and researches concurrent systems, such ‘ugly’ situations as in our previous example cannot be excluded. This forces us to be very accurate with our formal definitions and such simple things as ‘prompt’ simulations must be formalized carefully. Even this presented definition of promptness has some disadvantage: We handle only non-circular computations of $A$. If we have a state $s \in S_A$ with $s \rightarrow s \rightarrow \cdots$, then $s \not\in \delta^A_n(s)$ for all $n \in \mathbb{N}$ and we thus have made no further requirements for states $s \in K_n$. In such a situation $A$ has itself ‘non-prompt’ computations as we may run on this circle forever, and we thus allow further livelocks in $B$. A deeper research of such situations has to be done. A possible solution might be a definition of a prompt simulation via ‘unfoldings’ of both systems $A$ and $B$.

This conception has been developed by the author [31] independently from other attempts. It is quite closely related to the independent definition of a reduction in concurrent systems by Kwong [21]. The main difference is that we allow states of $A$ to be coded in $B$. Later definitions by Berthelot, Roucairol and Valk [3], Jensen [14] and Kasai and Miller (15) state very similar requirements in terms of T-systems where also the transition sequences of $A$ must be refound in some coded version. This can also be expressed in S-systems (by coding the history of a computation in the states)—but the main difference is that the mentioned authors operate with a mapping $K : S_A \rightarrow S_B$, that seems to be too restrictive. Also we are not interested in an algebraical study of the properties of simulations here but will apply this formal definition for a further study of the connections of various classes of concurrent systems. For this reason we tried to state a simulation conception in great generality, that shall hold in most models, shall allow for some proper hierarchy theorems in Petri-net classes and shall also be true for most accepted realization constructions of the literature.

However, this definition states some very general principles for simulating the computational aspects in various models—including synchronous switching circuits, classical automata theory, etc.—but is not adapted for concurrency. Thus it should not be surprising that on this general level proper hierarchy theorems could not be received—as most problems unsolvable for concurrent systems are solvable for synchronous ones. We thus have to adapt Definition 10 to specific aspects of concurrency.

The real difficulty in concurrent, asynchronous systems is the intercommunication of such systems. This general remark includes synchronization problems and the question of observability of concurrent systems. It seems to me that the handling of interfaces is a good distinction for synchronous and asynchronous systems and should be formalized in a simulation conception. We thus extend the previous definition:

**Definition 11.** An I/O-system $B = (I_B, O_B, S_B, \rightarrow_B)$ simulates an I/O-system $A = (I_A, O_A, S_A, \rightarrow_A)$ (via $K'$, iff $B$ simulates $A$ via $K$ as an S-system (i.e., the requirements of Definition 10 are fulfilled), and in addition there holds:
Let us discuss these interface coding requirements very briefly:

Firstly, it should be noted that one may require the monotony and linearity properties only for one-element sets \( P \subseteq I_A \cup O_A \), as the general case can then be shown by a simple induction proof. Also it would suffice for the sequel to state the property ‘standard coding’ only for \( P = I_A \cup O_A \).

As several ‘hand-shaking’ procedures operating with ‘ready and acknowledge’ signals are quite common in the literature as a coding of one wire by a bundle of wires—and should not be excluded as a simulation—we cannot demand \( \Phi \) to be a mapping with \( \Phi(I_A) \subseteq I_B \) and \( \Phi(O_A) \subseteq O_B \). The property zero-coding is only of technical interest and may be replaced by various different requirements in the following theory. But we need such a kind of requirement for impossibility proofs.

The properties (ii)-(iv) state certain independencies of the interface coding.

By (ii) there shall exist a certain standard coding \( * \) for the interface such that for all states of \( A \) that are equal on some parts \( P \) of the interface the standardized
coded states also equal on $\Phi(P)$. This requirement may look quite strong on a first view but it seems to be quite natural—as for asynchronous systems nothing is known about the arrival of signals on different wires, thus there should exist a coding of those signals (say on $P$) that is independent of the arrival of further signals (on $(I_A \cup O_A) - \Phi(P)$). On the other hand, this requirement makes no sense for all attached states. Suppose one would require the following property:

$$\forall P \subseteq I_A \cup O_A: \forall s_1, s_2, s_3 \in S_A: s_3 = \begin{cases} s_1 \mod P \\ s_2 \mod P_A - P \end{cases} \quad (\star)$$

Let us now regard an input $p \in I_A$ and two states $s_1, s_2$ with $s_1(p) = 1$ and $s_2(p) = 2$ with $i$ signals on input $p$ for $s_n$, $i = 1, 2$, with the standard coded attached states $s_1^*$ and $s_2^*$. However it may happen that for $s_2^*$ one signal on $\Phi(P)$ becomes assimilated in $W_H$, i.e., there may exist a state $\bar{s}_2 \in K_{s_2}$ with $s_2^* \rightarrow_H \bar{s}_2$ and in $\bar{s}_2$ only one signal can be found on $\Phi(p)$, the second being assimilated into $W_H$. Such an assimilation of signal is quite natural and cannot be avoided in general. But this situation could result in two states $s_1^*$ and $\bar{s}_2$ that equal on $\Phi(p)$. Now define

$$s = \begin{cases} s_2 \mod p \\ s_2 \mod P_A - \{p\} \end{cases}$$

By property (\star) we would conclude

$$\exists \bar{s}_3 \in K_{s_3}: \bar{s}_3 = \begin{cases} \bar{s}_2 \mod \Phi(p) \\ s_1^* \mod P_H - \Phi(p) \end{cases}$$

with the result that $\bar{s}_3 = s_1^* \in K_{s_1}$, i.e., $\bar{s}_3 \in K_{s_1}$, a contradiction.

One might try to escape this problem by defining $K: S_A \rightarrow S_H$, $s \mapsto s^*$, without further attached states to $s$. Such mappings are demanded in the mentioned simulation conceptions of the literature. However, such a requirement seems to be too restrictive. One should note that a computer $A$ together with a tape recorder that only stores the history of the computations of $A$ but does not influence these computations would not be a realization of $A$, as a state $s$ of $A$ may have different histories. But more important is that several standard transformations of a net $A$ into a net $B$ do not have the property that a state $s$ of $A$ can be mapped in only one attached state $\bar{s}$ of $B$.

Thus our requirement that $K$ maps states into sets of states is essential. As a consequence we have to be quite careful in stating requirements for all attached states. This is done in (iii) and (iv).

Monotony states that, whenever we change a state $s$ to a state $s'$ by merely adding some input signals or removing some output signals, we can find to any state $\bar{s}$ equivalent to $s$ a state $\bar{s}'$ equivalent to $s'$ where we also have only added some input signals and/or removed some output signals but had not to change the inner part $W_H$ of $\bar{s}$ or further non-involved inputs or outputs of $\bar{s}$.
Linearity states that if we can code some inputs and outputs $P$ as $\tilde{s}_1|\Phi(P)$, then we can use the same coding for a further state $\tilde{s}_2$ provided that $\tilde{s}_1 = \tilde{s}_2$ on $W_B$. This restriction is important as it requires that possibly assimilated input signals of $\tilde{s}_1$ and $\tilde{s}_2$ have to be assimilated in both states or in none.

It should be noted that such requirements are quite natural for asynchronous, concurrent systems and separate simulation conceptions for asynchronous and synchronous models as they are too restrictive for synchronized systems, where an interface coding might be more general.

We also need a translation of I/O-procedures:

**Definition 12.** For any two I/O-systems $A$ and $B$ where $B$ simulates $A$ via some $K$ and $\Phi$ and for any I/O-procedure $P$ for $A$ we define a relation $P^K_B \subseteq S_B \times S_B$ as

$$\forall \tilde{s}, \tilde{s}' \in S_B: \tilde{s} \triangleright_P \tilde{s}' \iff \exists s, s' \in S_A: \tilde{s} \in K_S: \tilde{s}' \in K'_{S'}.$$  

(i) $s \triangleright_P s'$,

(ii) $s \equiv s' \mod I_B$ and $s \equiv s' \mod O_B$.

(iii) $\tilde{s} \equiv \tilde{s}' \mod W_B \cup \Phi(P^\triangleright)$,

with $P^\triangleright := \{ p \in I_A \cup O_A: s(p) = s'(p) \}$.

**Lemma 2.** Let $A$ and $B$ be two I/O-systems such that $B$ simulates $A$ via some $K$ and $\Phi$. Then there holds for any I/O-procedure $P$ for $A$:

(i) $\forall \tilde{s}, \tilde{s}' \in S_A: \forall \tilde{s} \in K_S: \forall n \in \mathbb{N}_0: (l \triangleright P^\triangleright := P^K_B, for short)$

$$(\alpha) s \triangleright_P s' \implies \exists \tilde{s}' \in K_{S'}: \tilde{s} \triangleright_P \tilde{s}'$$

$$(\beta) s \xrightarrow{\Delta} A \triangleright s' \implies \exists \tilde{s}' \in K_{S'}: \tilde{s} \xrightarrow{\Delta} A \triangleright \tilde{s}'$$

$$(\gamma) \forall \tilde{s}' \in K_{S'}: \left( \tilde{s} \xrightarrow{\Delta} A \triangleright \tilde{s}' \implies s \xrightarrow{\Delta} A \triangleright s' \right).$$

$$(\delta) \forall s'' \in S_B: (\tilde{s} \xrightarrow{\Delta} A \triangleright s'' \implies \exists \tilde{s}' \in K_{S'}: s'' \equiv s' \mod H).$$

(ii) $P^K_B$ is an I/O-procedure for $B$.

(iii) The closed $S$-system $(B, P^K_B)$ simulates the closed $S$-system $(A, P)$ also via $K$.

Note that (ii) and (iii) are trivial consequences of (i). In order to show how to operate with this formal apparatus we will prove (i).

**Proof.** To (\alpha): $s \triangleright_P s'$ implies by definition $s \equiv s' \mod P_A - P^\triangleright$, $s \equiv s' \mod I_A \cap P^\triangleright$, $s \equiv s' \mod O_A \cap P^\triangleright$ for $P^\triangleright := \{ p \in I_A \cup O_A: s(p) \neq s'(p) \}$. By monotony we find for any given state $\tilde{s} \in K$, a state $\tilde{s}' \in K_{S'}$ with $\tilde{s} \equiv \tilde{s}' \mod P_B - \Phi(P^\triangleright)$, $\tilde{s} \equiv \tilde{s}' \mod I_B \cap \Phi(P^\triangleright)$ and $\tilde{s} \equiv \tilde{s}' \mod O_B \cap \Phi(P^\triangleright)$. Thus by Definition 12 $\tilde{s} \triangleright_P \tilde{s}'$ holds.
To (β)–(δ): Induction on n. The case n = 0 is Definition 10. n → n + 1:

(β) We have \( s \rightarrow_{A,p} s' \), \( s \in K \) fixed. There exist states \( s'' \), \( s''' \in S_A \) with \( s \rightarrow_{A,p} s'' \), \( s'' \rightarrow_{A,p} s''' \) and \( s''' \Rightarrow_A s' \). By induction there exists \( s'' \in K \) with \( s'' \Rightarrow_{A,p} s''' \). By (α) we find a state \( s'' \in K \) with \( s'' \Rightarrow_{A,p} s''' \) and by Definition 10(i), we find a state \( s' \in K \) with \( s'' \Rightarrow_{A,p} s' \). Thus \( s'' \Rightarrow_{A,p} s' \) holds.

(γ) We have \( s \rightarrow_{B,p} s' \). By Definition 12 there exist states \( s'', s''' \in K \) with \( s \rightarrow_{B,p} s'', s'' \rightarrow_{B,p} s''' \Rightarrow_B s' \). By induction and Definition 10(ii), we conclude: \( \exists s'', s''' \in S_A: s \rightarrow_{B,p} s''. \)

(δ): We have \( s \rightarrow_{B,p} s'' \). Thus: \( \exists s'', s''' \in K \). By induction and Definition 10(ii), we conclude: \( \exists s'', s''' \in S_A: s \rightarrow_{B,p} s'' \).

It should be noted that the properties zero-coding and standard-coding were not needed for this proof. There seems to be some freedom in modifying these both properties.

We will not enter a study of the properties of our simulation conception. Of course it should be noted that the simulation relation is transitive. In the sequel we will use this conception to prove some proper hierarchies in Petri-nets and to study the connections of modular, non-locally determined systems (as APA-nets are) with locally-determined, non-persistent systems (as Petri-nets are).

5. A hierarchy-theorem

As an example of the significance of such a formal approach we can precisely state a well-known—out somewhat 'mysterious' and unproven—classification theorem for Petri-nets (Theorem 3).

Definition 13. A Petri-net \((PN), N\), is called a

- **Marked Graph (MG)** \( \forall p \in P_N: \#p \leq 1 \) and \( \#p' \leq 1 \),
- **State Machine (SM)** \( \forall e \in E_N: \#e = \#e' = 1 \),
- **Free-Choice net (FC)** \( \forall p \in P_N: \forall e \in p: (\#e = 1 \text{ or } \#p = 1) \),
- **Extended FC (EFC)** \( \forall p \in P_N: \forall e_1, e_2 \in p: e_1 = e_2 \),
- **Simple Net (SN)** \( \forall e \in E_N: \forall p_1, p_2 \in e: (\#p_1 > 1 \text{ and } \#p_2 > 1 \Rightarrow p_1 \neq p_2) \).

With MG, SM, FC, EFC, and SN we denote the corresponding classes of all MG, SM, FC, EFC, and SN. Also, GPN, RPN, PN denote the classes of all GPN, RPN, and PN. For any class \( C \) of S-systems with a dimension we denote by \( k \)-safe \( C \) the class of all \( k \)-safe S-systems of \( C \).
Definition 14. For two classes $C_1, C_2$ of I/O-systems we define:

- $C_1 \subseteq C_2$ $\iff$ $\forall A \in C_1 : \exists B \in C_2 : B$ simulates $A$ promptly,
- $C_1 \not\subseteq C_2$ $\iff$ $\exists A \in C_1 : \forall B \in C_2 : B$ does not simulate $A$,
- $C_1 = C_2$ $\iff$ $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$,
- $C_1 \not\subseteq C_2$ $\iff$ $C_1 \subseteq C_2$ and $C_2 \not\subseteq C_1$.

One should note that $C_1 \not\subseteq C_2$ is not equivalent to $\neg C_1 \subseteq C_2$, as for an inclusion we require promptness of a simulation, while for non-inclusion we require the impossibility of a simulation. This is simply to make the following results as strong as possible.

To make our simulation requirements more intelligible we present a new simulation construction for $(k$-safe)$GPN = (1$-safe)$RPN$.

Theorem 2. $GPN = RPN$,

safe $GPN = 1$-safe $RPN$.

It should be noted that the construction in Theorem 5.5 in Hack [10] works also fine for S-systems (although the definition of a Petri-net language does not require hang-up-freeness). We present this (slightly modified) construction and show how to handle I/O-systems in addition and how to fulfill our axioms of a simulation.

We present only an informal outline of a proof for Theorem 2 that nevertheless should make all ideas intelligible.

Let $N$ be a GPN with some $p \in P_N$. $p = \{e_1, \ldots, e_n\}$, $p' = \{e_1', \ldots, e_{n'}\}$, where $p \not\sim p' \neq \emptyset$ is allowed.

Case 1: $p$ is $k$-safe (with respect to some set of markings we are interested in). Let $d_i$ denote the number of arcs from $e_i$ to $p$ and $d_{i'}$ the number of arcs from $e_i$ to $e_{i'}$, $1 \leq i \leq n$, $1 \leq i' \leq n$. We construct an $r$-shift-register as shown in Fig. 6(a) for $r = 1 + \max(k, d_1, \ldots, d_m, d_1', \ldots, d_{n'}')$. In Fig. 6(a) we have chosen $d_n = d_1' = d_{n'}' = 2$ for legibility.

A state $s$ is called safed iff for all $s' \in \mathcal{M}(s)$ and all place-pairs $p_i, p_i'$, for $1 \leq i \leq r$ there holds: $s'(p_i') + s'(p_i) = 1$. We say that such an $r$-shift-register $SR(p)$ for the place $p$ simulates a number $z \leq k$ in a safed state $s$ iff there are exactly $z$ places $p_i', \ldots, p_i'$, $1 \leq i \leq n$, $1 \leq j \leq z$, in $SR(p)$ with $s(p_i') = 1$. In other words, exactly $z$ tokens are on the 'positive' places, while $r - z$ tokens are on the 'negative' places, fulfilling our requirement that $s(p_i') + s(p_i) = 1$ for all $i$, $1 \leq i \leq r$.

In a safed state the tokens on the positive places may trickle down the shift-register and gather in the inferior positive places. Let $s^+$ denote such a safed state where all tokens on the positive places have gathered in the inferior part.

To construct our RPN, $M$, simulating $N$, we replace in $N$ all places $p$ by such a shift-register. If we regard $N$ and $M$ as S-systems without an interface (i.e. we have to fulfill only Definition 10 but not 11), then we have finished: $M$ obviously
simulates $N$ via the canonical state-mapping $\mathcal{X}$, where $K_\varepsilon$ is the set of all states $\bar{s}$ of $S_M$ such that $\bar{s}$ restricted on $SR(p)$ simulates the number $s(p)$ for all $p \in P_N$.

There holds obviously: Any state of $K^* = \bigcup_{s \in S_N} K_s$ is safed and $\mathcal{H}(K^*) = K^*$, i.e., there are no intermediate non-attached states for this state-mapping. Further, any event fireable in some state $s \in S_N$ may become fireable in any state $\bar{s} \in K_\varepsilon$, as $\bar{s} \Rightarrow_M s^+$ and $s^+ \rightarrow_s$ holds if $s \rightarrow_s$ holds. On the other hand, any event $e$ is fireable in some $\bar{s} \in K_\varepsilon$ only if $e$ is fireable in $s$ in $N$. This proves immediately that $M$ realizes $N k$-promptly for some $k'$. We have to choose $k'$ as the sum of all $r$ for all $r$-shift-registers in $M$. Any local activity in $M$ thus may have a delay of at most $r$ steps in a local $r$-shift-register, thus a 1-step computation in $N$ may take not longer than $k'$ steps in $M$. 

Fig. 6.
Case 2: If \( N \) is not safe an unbounded number of tokens may accumulate on \( p \). We thus have to change the previous construction a little bit. We use the notations of Case 1. Let \( d := \max(d_1, \ldots, d_n) \), \( d' := \max(d'_1, \ldots, d'_m) \). Instead of \( \text{SR}(p) \) as shown in Fig. 6(a) we operate with a shift-register \( \text{SR}'(p) \) as shown in Fig. 6(b) (where we again have chosen \( d_1 = d, d'_1 = d', d_n = d_m = 2 \) for legibility). We now proceed with \( \text{SR}'(p) \) as in the first case and result in a (possibly non safe) \( \text{RPN,M} \), simulating \( N \) promptly.

An alternative practical and hang-up-free transformation from \( \text{GPN} \) to \( \text{RPN} \) in the case of S-systems can be found also in Müller [22].

If we regard \( N \) and \( M \) as I/O-systems we must also handle an interface coding fulfilling all requirements of Definition 11. We thus regard input and output places of \( N \) that will now be transformed in a different way.

Let \( p \) be an input place of \( N \) with \( 'p = 0 \). If \( p \) is not safe we may proceed as in Fig. 6(b) where we simply drop \( e_1 \) up to \( e_n \), choose \( d := 1 \) and define \( \Phi(p) := \{ p_1 \} \). If \( p \) is \( k \)-safe we use a construction as shown in Fig. 7. We define

\[
\Phi(p) := \{ p_{\text{IN}_1}, \ldots, p_{\text{IN}_k}, p_{\text{OUT}_1}, \ldots, p_{\text{OUT}_k} \} \subseteq I_M \cup O_M
\]

and say that such a modified shift-register simulates a number \( z \leq k \) if exactly \( z \) positive places, including the positive input-places, carry a token in a safed state.

Let \( p \) be an output-place with \( 'p = 0 \). If \( p \) is not safe we again modify Fig. 6(b) and drop the inferior place-pairs \( p_{i,j}, p_{i,j} \) up to \( p_{i,j}, p_{i,j} \). The new inferior place \( p_{i,j} \) becomes an output-place and we define \( \Phi(p) := \{ p_{i,j} \} \). If \( p \) is \( k \)-safed we have to modify the 'vertical' shift-register output-part of Fig. 6(a) into a 'horizontal' one as shown in Fig. 8, with \( d := \max(d_1, \ldots, d_n) \). Obviously we define

\[
\Phi(p) := \{ p_{\text{OUT}_1}, \ldots, p_{\text{OUT}_k}, p_{\text{IN}_1}, \ldots, p_{\text{IN}_k} \} \subseteq I_M \cup O_M.
\]

We say that such a modified output shift-register (\( \text{OSR}(p) \)) simulates a number \( z < k \)

(i) weakly iff exactly \( z \) positive places (including the output-places) carry a token in a safed state, and

(ii) strongly iff exactly \( z \) of its positive output-places carry a token, and none of its inner positive places, in a safed state.
The state-mapping $s \rightarrow K$, must now be defined in such a way that for an output place $p \in O_N$ and a state $s \in S_N$ a state $\tilde{s} \in K$, restricted on $OSR(p)$ simulates $s(p)$ strongly. This is simply to fulfill our requirement of ‘zero-coding’ of Definition 11. The remaining states $\tilde{s}$ of $S_M$, where $\tilde{s} \mid ORS(p)$ simulates $s(p)$ weakly, become now intermediate states of $S_{\mathbf{M}} \setminus K^+$. We thus also result in an I/O-system, $M$, simulating $N$ as an i/O-system promptly (and, of course, hang-up-free). 

We now will investigate the Petri-net sub-classes of Definition 13. In a first part we state some ‘positive’ results (Lemma 3) and will then turn our attention to the quite interesting negative impossibility results.

**Lemma 3.**

$MG = APA(V, W)$,

$SM = APA(K, I)$,

$FC = APA(K, I, V, W) = EFC$,

$SN = APA(K, V, W, S')$.

For a definition of the used modules see Table 1. A proof of the first three lines is a simple exercise where one needs only some very simple techniques, as to simulate modules $A^\infty$, by some trees of $V$ and $W$ modules, e.g., to translate places into wires, tokens into signals, etc. There arise no problems as the Petri nets in $MG \cup SM \cup FC \cup EFC$ are also a kind of ‘modular’ nets: a conflict in $FC$ can be interpreted as a non-determinism. However, this is not true in $SN$. Nevertheless a transformation from $SN$ into $APA(K, V, W, S')$ is quite simple and a typical example is given in Fig. 9, that should be self-explanatory.
The other direction of Lemma 3, to translate the given APA-net classes into Petri-net classes, is a very trivial exercise.

Regarding the languages generated by Petri-nets, $FC, SN$ and $PN$ are equivalent, as for any Petri-net there exists a FC generating the same language, see, e.g., (10). On the other hand APA($K, V, W, S$) is certainly less powerful than general APA-nets with respect to their synchronization or computing abilities. This fact, in terms of $SN$ was first noticed by Patil [28], who proved that $SN$ is ‘less powerful’ than $PN$. Compare this with Kosaraju’s result [20] that inhibitory Petri-nets (iPN) are ‘more powerful’ than PN. In inhibitory Petri-nets there are special ‘inhibitory’ arcs from some places to some events allowed that prohibit such an event to fire if one of its ‘inhibitory’ input-places carries a token. We require here that no inhibitory arcs are allowed from an input-place $p \in I_N$ of $N$. Any PN is also an iPN—without inhibitory arcs.) The above apostrophes have to be used as both authors gave no definition or clear idea of what ‘powerful’ shall mean.

We shall now restate both results, and some more, in precisely defined terms with rigorous mathematical proofs (without any ‘mysteriousness’).

**Theorem 3**

$$SM \not\subseteq MG \subseteq FC \subseteq SN \subseteq PN \subseteq iPN.$$  

**Proof.** The inclusions are trivial by definition. We have to show that they are proper and will proceed by a series of lemmata.
Lemma 4. $PN \preceq iPN$.

Proof. We call an iPN, $N$, testable \( \iff \)
\[ \exists P \subseteq O_N : \forall n, m \in \mathbb{P} : \exists s'_m, s''_m \in S_N : \forall s_{n,m} \in S_N : \]
\[ s_{n,m} = \begin{cases} s'_m \mod I_N \cup O_N, & \text{and} \\ s''_m \mod W_N \end{cases}, \]
(i) \( n \leq m \quad \Rightarrow \quad \forall s \in \mathbb{R}(s_{n,m}) : \forall p \in P : s(p) = 0, \)
(ii) \( n > m \quad \Rightarrow \quad \exists s \in \mathbb{R}(s_{n,m}) : \exists p \in P : s(p) > 0. \)

Step 1: There exists a testable iPN.

Regard the iPN, $N$, of Fig. 10. $N$ is an unbounded register with a test. We define $s^s := (0, 0, 0, 0, 0, 0, 0)$, $s^t := (0, 0, m, 0, 0, 0)$ and $P := \{ p_n \}$. Thus $N$ is testable. (There are more simple examples of testable iPNs.)

![Fig. 10.](image)

Step 2: Any iPN, $M$, simulating a testable iPN, $N$, is testable itself.

Let $N$ be testable with $P, s'_m, s''_m, s_{n,m}$ as defined above. Let $M$ simulate $N$ via $K$ and $\Phi$. We define for $M$: $p := (D(P) \cap \Phi(M) \cap O_M$. By the standard coding we find for all $n, m \in \mathbb{N}$ states $s^*_{n,m} \in K_{s_m}, s''_{n,m} \in K_{s_m}, s'_{n,m} \in K_{s_m}$ with $s'^*_{n,m} = s''_{n,m} \mod I_M \cup O_M$ and $s''_{n,m} \mod W_M$.

For some $n, m \in \mathbb{N}$ with $n \leq m$ and $s'' \in S_M$ with $s''_m \Rightarrow_M s''$ we find states $s \in S_N, \bar{s} \in K$, with $s''_m \Rightarrow_M \bar{s}$ (Definition 10(i)) and thus $s''_{n,m} \Rightarrow_M \bar{s}$ and $s_{n,m} \Rightarrow_N s$ (by (ii)).

As $N$ is testable we know that $s(p) = 0 \forall p \in P$. Thus by zero-coding we know that also $\bar{s}(p) = 0 \forall p \in \Phi(p) \cap O_M = \bar{p}$. As these places are output-places of $M$ we conclude from $s''_m \Rightarrow_M \bar{s}$ that also $s''_m = 0 \mod \bar{p}$.

For $n, m \in \mathbb{N}$ with $n > m$ we find a state $s \in S_N, p \in P$ with $s_{n,m} \Rightarrow_N s$ and $s(p) \cdot > 0$. Thus by Definition 10(i) and zero-coding there exist $\bar{s} \in K_{s_m}, \bar{p} \in \Phi(p)$ with $s''_{n,m} \Rightarrow_M \bar{s}$ and $\bar{s}(\bar{p}) > 0$. Thus $M$ is testable.

Step 3: There exists no testable PN.

Suppose there exists a testable PN, $N$, with $P, s'_n, s''_m, s_{n,m}$ as above. Define $S^* := \{ s''_m, m \in \mathbb{N} \}$. $S^*$ is an infinite set of \#$P_N$-vectors. By a little generalization
of Kosaraju's argument [20] we find $m_1, m_2 \in \mathbb{N}$ with $m_1 < m_2$ and $s''_{m_1} \equiv s''_{m_2}$. Thus there exists a vector $s'$ with $s''_{m_2} = s''_{m_1} + s'$. For $s^+ := s' \mod W_N$ and $s^+ := 0 \mod I_N \cup O_N$ there holds $s_{n,m_2} = s_{n,m_1} + s^+ \forall n \in \mathbb{N}$. For $n := m_1 + 1 \leq m_2$ we conclude: $\exists s \in S_N: \exists p \in P: s_{n,m_1} \Rightarrow N s$ and $s(p) > 0$. As $N$ is a PN we conclude: $s_{n,m_2} = s_{n,m_1} + s^+ \Rightarrow N s + s^+ \Rightarrow (s + s^+)(p) > 0$, a contradiction.

By Step 1 up to 3 not all iPN can be simulated by PN. This proves Lemma 4. □

We will discuss why we prohibited inhibitory input-places of $I_N$ for an iPN $N$ after the proof of Theorem 4.

Lemma 5. $SN \subseteq PN$.

Proof. We call a PN,$N$, SN-hard ⇔

$\exists P \subseteq O_N: \exists s_0 \in S_N: \exists I/O$-procedures $P_1, \mathbb{P}$ for $N$:

(i) $P_1 \subseteq \mathbb{P}$,

(ii) $\forall s' \in S_N: s_0 \xrightarrow{N, P_1} s'$

(1) $s'(p) = 0 \forall p \in P$,

(2) $\exists s^* \in S_N: \exists \tilde{p} \in P:

\[ s' \xrightarrow{N, \mathbb{P}} s^* \text{ and } s^*(\tilde{p}) > 0, \]

(iii) $\forall s, s' \in S_N: \forall p \in I_N \cup O_N: s \mathbb{P} s' \Rightarrow \exists s'_1 \in S_N:

\[ s \mathbb{P} s'_1 \text{ and } s'(p) = s'_1(p). \]

Step 1: There exists an SN-hard PN.

We may choose Patil's '3-smoker-net' [28]: $N$ is the Petri net of Fig. 11. We define $s_0 := (0, 0, 0, 0, 0, 0), P := \{p_6\}, P_1$ is the relation: $\forall i, j, k \in \mathbb{N}_0$:

(0, 0, 0, i, j, k) $P_1(1, 1, 0, 0, 0, 0),$

(0, 0, 0, i, j, k) $P_1(0, 1, 1, 0, 0, 0),$

(0, 0, 0, i, j, k) $P_1(0, 0, 0, 0, 0, 0),$

and for $\mathbb{P}$ holds in addition

(0, 0, 0, i, j, k) $\mathbb{P}(1, 0, 1, 0, 0, 0).$

Thus $N$ is SN-hard.

Fig. 11.
Step 2: Any PN, $M$, simulating an SN-hard PN, $N$, is SN-hard itself.

Suppose $N$ is SN-hard with $P, s_0, P, \Phi$ as above and $M$ simulates $N$ via $K$ and $\Phi$. We define for $M$: $\tilde{P} := \Phi(P) \cap O_M$, $\tilde{s}_0$ is some state of $K_{s_0}$, $\tilde{P}_1 := \Phi(S)$ and $\tilde{P}_1 := \Phi(S)$. Then there holds:

(i) $\tilde{P}_1 \subseteq \tilde{P}$, obviously.

(ii) For some $s_0 \in S_M$ with $s_0 \rightarrow_{M, \tilde{P}}, s_0$ we find $s' \in S_N$ and $\tilde{s}' \in K_{\tilde{P}}$ (by $\gamma$ and $\delta$ of Lemma 2) with $s' \rightarrow_{N, \tilde{P}}, s'$ As $N$ is SN-hard we conclude that $s'(p) = 0 \mod P$, this implies by zero-coding $\tilde{s}' = 0 \mod \tilde{P}$ and thus also $s_0 = 0 \mod \tilde{P}$. Also we find a state $s' \in S_N$ with $s' \rightarrow_{N, \tilde{P}}, s'$ and a $p \in P$ with $s'(p) > 0$. By Lemma 2 we thus find a $\tilde{s}' \in K_{\tilde{P}}$, a $\tilde{p} \in \tilde{P}(p)$ with $\tilde{s}' \rightarrow_{M, \tilde{P}}, \tilde{s}'$ and $\tilde{s}'(\tilde{p}) > 0$. Thus also $s_0 \rightarrow_{M, \tilde{P}}, s_0$.

(iii) Given are $s, s' \in S_M$, $p \in P, s \rightarrow_{M, \tilde{P}}, s$, $s' \rightarrow_{N, \tilde{P}}, s'$ with $s' \rightarrow_{N, \tilde{P}}, s'$ and $s \rightarrow_{M, \tilde{P}}, s$. As $N$ is SN-hard we find $s_1$ with $s \rightarrow_{M, \tilde{P}}, s_1$ and $s \rightarrow_{M, \tilde{P}}, s_1$ mod $P$. Thus there also exists a state $\tilde{s}_1 \in K_{\tilde{P}}$, with $\tilde{s}_1 \rightarrow_{M, \tilde{P}}, \tilde{s}_1$ and thus:

$$s' = s_1 \mod W_N, \quad \tilde{s}' = \tilde{s}_1 \mod W_M,$$

$$s' = s_1 \mod P, \quad \tilde{s}' = \tilde{s}_1 \mod M - \{p\}.$$

By linearity we conclude:

$$\exists \tilde{s}_1 \in K_{\tilde{P}}: \tilde{s}_1 = \tilde{s}' \mod \Phi(p) \text{ and } \tilde{s}_1 = \tilde{s}' \mod M - \Phi(p).$$

This implies also $s \rightarrow_{M, \tilde{P}}, s_1$ and $s_1(p) = \tilde{s}'(\tilde{p})$. Thus $M$ is SN-hard.

Step 3: There exists no SN-hard SN.

It should be noted that a simpler version of this Step 3 is the non-formally used idea in Patil’s proof in [28]. This step generalizes and formalizes Patil’s idea.

Proof of Step 3: Suppose there exists an SN-hard Simple net, $N$, with $P, s_0, P, \Phi$ as above. We define for this proof: An $s_0$-derivation, $d$, in $(N, P)$—for any $1/O$-procedure $P$ for $N$—is a finite sequence $d = (s_n, s_{n+1}, \ldots, s_0)$ with $s_i \rightarrow_{N, s_{i+1}}$ or $s_i \rightarrow_{P, s_{i+1}} \forall i, 1 \leq i < n$. The length $l(d)$ of $d$ shall be $n$ and $s_d$ denotes the final state $s_n$ in $d$. We say that $d$ puts token on the places in $B(d) := \{p \in P_N: s_d(p) > 0\}$. Let $P'/d$ be the set of $s_0$-derivations in $(N, P')$ that may be prolonged from $d: d' \in P'/d$ if there exist states $s_n, \ldots, s_{n+m}$ with $d' = (s_n, \ldots, s_{n+m})$, $m = 0$ is allowed. Also $P'/d$ is the set of all $s_0$-derivations in $(N, P')$ with $d', d, d' \in P'/d, d, d'$ allowed. iff $d$ is an $s_0$-derivation in $(N, P)$ with at most one application of $P$ in $d$. $B(P'/d) := \{p \in P_N: d' \in P'/d: p \in B(d')\}$ is the set of all places that may receive a token by prolonging $d$ in $(N, P')$.

We define inductively $s_0$-derivations $d_n$ in $(N, P_1)$:

1. $n = 0$: $d_0 := (s_0)$.
2. $n \rightarrow n + 1$: Let $d_n$ be the $s_0$-derivation constructed in the $n$th step. Define for all $p \in P_N - B(P'/d_n)$:

$$L(p) := \begin{cases} \infty & \text{if } p \notin B(P'/d_n), \\ \min \{(d'): p \in B(d') \text{ and } d' \in P'/d_n\}. \end{cases}$$

The property (ii) of SN-hardness ensures that there exists a $p \in P_N$ with $L(p) < \infty$. 

Define $L := \min \{ L(p); p \in P_N - B(P_1/d_n) \}$, and let $\tilde{p}$ and $\tilde{d}$ be a place and an $s_0$-derivation with $L(\tilde{p}) = L$ and $l(\tilde{d}) = L$ and $\tilde{p} \in B(\tilde{d})$ and $\tilde{d} \in P^{-1}/d_n$.

There holds: $l(\tilde{d}) > l(d_n)$, as $\tilde{p} \not\in B(d_n)$. Thus the $s_0$-derivation $d^+$ with $l(d^+) = l(d^+) + 1$ and $\tilde{d} \in P^{-1}/d_n$, i.e.: $\tilde{d} - (d^+, s^d)$, is also in $P/d_n$. We regard the final state $s^+ := s^{d^+}$ in $d^+$. There holds either $s^+ \rightarrow_{s} s$, as $s^+ \rightarrow_{s} s$, or $s^+ \rightarrow_{s} s$. Suppose $s^+ \rightarrow_{s} s$. This implies $s^+(\tilde{p}) \geq s(\tilde{p})$, as $\tilde{p}$ cannot be an input-place of $N$, according to $\tilde{p} \not\in B(P_1/d_n)$ and $\tilde{p} \not\in B(P^{-1}/d_n)$ and (iii) of SN-hardness. Thus we found a $s_0$-derivation $d^+$ with $l(d^+) < l(d)$, in contradiction to the definition of $L$. As a consequence $s^+ \rightarrow_{s} s$ must hold with $s^+(\tilde{p}) = 0$. There exists an event $e \in E_N$ with $\tilde{p} \in e$, $e$ fires from state $s^+$ to $\tilde{s}$, and thus $s^+(p) > 0 \forall p \in e$. We assume $e = \{i_1, i_2\}$, the general case is analogical. Assume $i_1 \not\in B(P_1/d_n)$. This contradicts to the definition of $L$ as $i_1 \not\in B(d^+)$. This holds also for $i_2$ with the consequence that $i_1, i_2 \in B(P_1/d_n)$. As $N$ is in SN we assume that only $i_2$ is a shared input-place of $e$, i.e., $i_1 = \{e\}$ holds. Let $d_0$ be the shortest $s_0$-derivation in $P_1/d_n$ that puts a token on $i_1$, i.e. such that $d_0(i_1) > 0$. There are two possibilities:

(i) $i_2 \in B(P_1/d_1)$,
(ii) $i_2 \not\in B(P_1/d_1)$.

For (i) we choose a shortest $s_0$-derivation $d_1 \in P_1/d_1 \subseteq P_1/d_n$ with $i_2 \in B(d_1)$. As $i_1$ is not shared in $N$ we know that also $i_1 \subseteq B(d_1)$ must hold. Thus in $s_1$ $i_1$ and $i_2$ carry a token, and by a firing of $e$ we get $\bar{p} \in B(P_1/d_1) \subseteq (P_1/d_n)$, a contradiction to the definition of $\bar{p}$.

Thus (i) is true. We define the $(n + 1)$st $s_0$-derivation $d_{n+1} := d_0$. As $B(P_1/d_n) - B(P_1/d_{n+1})$ contains $i_2$ (case (ii)) we conclude that $B(P_1/d_{n+1}) \subseteq B(P_1/d_n)$ and this induction must lead to a contradiction as $P_N$ is a finite set.

All main ideas of this step can be found in Patil [28]. □

Lemma 6. FC $\equiv$ SN.

Proof. We say that a state $s$ of a PN $N$, can always reach a set $P \subseteq O_N \iff$

$$\forall s' \in \mathcal{H}(s): \exists s'' \in \mathcal{H}(s') \exists p \in P: s''(p) > 0.$$ 

A PN $N$, is called FC-hard $\iff \exists P \subseteq O_N: \exists s, s_1 \in S_N: s_1 \equiv s \mod I_N$ and $s_1 = s \mod W_N \cup P$ such that $s$ can always reach $P$ but $s_1$ cannot.

Step 1: There exists an FC-hard SN.

Regard the SN of Fig. 12 with $s := (0, 1, 0, 0), s_1 := (1, 1, 0, 0)$ and $P := \{p_4\}$. 

![Fig. 12](image-url)
**Step 2:** As PN, $M$, simulating an FC-hard PN, is itself FC-hard.

**Proof.** As in the previous lemmata. Use monotony.

**Step 3:** There exists no FC-hard FC.

We present a more general characterization for FC:

**Lemma 7.** A PN, $N$, is called strictly monotone if

$$\forall P \subseteq O_N: \forall s, s_1 \in S_N: s_1 \geq s \Rightarrow (s \text{ can always reach } P \Rightarrow s_1 \text{ can always reach } P).$$

All Free-Choice-nets are strictly monotone.

Note that from Lemma 7 it follows immediately that no FC can be FC-hard. A proof of Lemma 7 follows from the nice theorem of Klemens Döpp:

**Theorem 4** (Klemens Döpp [8]). Let $N$ be an FC and $E \subseteq \Psi(E_N)$ a set of sets of events. Let $S \subseteq S_N$ be a set of states with the property:

$$\forall s, s' \in S_N: s \in S \text{ and } s_1 \geq s \Rightarrow s_1 \in S, \text{ and } S = \mathbb{R}(S).$$

Let $s$ be a state of $S_N - S$ with

$$\forall s_1 \in \mathbb{R}(s) - S: \exists e \in E: \forall e \in E: \exists s_2 \in \mathbb{R}(s_1): s_2 \rightarrow e.$$

Then there holds:

$$\forall s^* \in S_N: s^* \geq s \Rightarrow \forall s_1^* \in \mathbb{R}(s^*) : \exists e \in E: \forall e \in E: \exists s_2^* \in \mathbb{R}(s_1^*) : s_2^* \rightarrow e \text{ or } s_2^* \in S.$$

We drop a proof of this theorem here. Also due to Klemens Döpp [8] are the following nice applications:

**Lemma 8.** Let $N$ be an FC, $s, s_1 \in S_N$ with $s_1 \geq s$. Then there holds:

(i) $\forall P \subseteq O_N: s \text{ can always reach } P \Rightarrow s_1 \text{ can always reach } P$,

(ii) $s \text{ is live } \Rightarrow s_1 \text{ is live},$

(iii) $s \text{ is hang-up-free } \Rightarrow s_1 \text{ is hang-up-free},$

(iv) $\forall e_0 \in E_N: e_0 \text{ is live in } s \Rightarrow e_0 \text{ is live in } s_1.$

**Proof.** Apply Theorem 3: For (i) we define $\mathcal{C} := \{e\}; e \in \mathcal{P}$, $s := \{s \in S_N: \exists p \in P: s(p) > 0\}$. For (ii) define $\mathcal{C} := \{E_N\}, S := \emptyset$, and for (iii): $\mathcal{C} := \{e\}; e \in E_N$, $S := \emptyset$. For (iv): $\mathcal{C} := \{e_0\}$ and $S := \emptyset$. 

It is quite interesting that FC may have the same reachability-sets as PN, however their 'always-reachability' properties differ. This gives also a structure-free (i.e., not regarding events) characterization of FC.
To complete Theorem 3 we have to prove $MG \not\preceq SM$ and $SM \not\preceq MG$, which also implies $SM \preceq FC$ and $MG \preceq FC$. We leave both cases to the reader who may try to find some invariants in the style of the previous proofs himself. □

It might be somewhat surprising that we excluded inhibitory input-places of $I_N$ in an iPN $N$. However, the reason is quite simple: if one simulates an inhibitory input-place $p \in I_N$ its image in a simulating system $M$ will in general not belong to the interface of $M$. Let us for example regard the standard simulation construction of safe iPN by PN. Fig. 13 gives an example of a 1-safe iPN $N$ and its simulating PN $M$.

Thus in the subnet $N'$ of $N$ of Fig. 14, where $p_1$ is an input-place, the translated place pair $p'_1, p_1$ in $M'$ does not belong to $I_M \cup O_M$. This is true for any attempt to simulate $N'$ by a PN.

Let $iPN^*$ be the class of all 1-safe iPN with inhibitory input-places allowed. There holds:

**Lemma 9.** $iPN^* \not\cong PN$.

A proof is quite simple: We call an iPN $N$, strong \( \Leftrightarrow \)

\[
\exists P \in O_N : \exists s_0 \in S_N : \exists s' \exists \text{I/O-procedure for } N:
\]

(i) \( s_0 = 0 \mod P \),

(ii) \( \exists s' \in S_N : \exists p \in P : s_0, s' \Rightarrow s' \) and \( s'(p) > 0 \),

(iii) \( \exists s'' \in S_N : s', s'' \Rightarrow s'' \forall s'' \in \mathcal{M}(s'') : s'' = 0 \mod P \).
As can easily be seen $N'$ of Fig. 14 is strong: Define $s_0 := (0, 1, 0)$, $s'' = (1, 1, 0)$, 
$p := \{p_3\}$ and $(0, 1, 0)P'(1, 1, 0)$.

With the previous techniques it is easily shown that any \(iPN,M\), simulating a strong \(iPN,N\), is itself strong and obviously no Petri-net can be strong (as \(sPs''\) implies that \(s'' \equiv s \mod P_N - O_N\)).

As for our simulation conception it is also true that any 1-safe \(iPN\) can be simulated by a 1-safe \(PN\) the previous lemma has the somewhat counterintuitive implication that \(iPN^*\) cannot be simulated by 1-safe \(PN\). Thus the \(PN\) \(N'_1\) of Fig. 14 is no simulation of \(N'\). At a first glimpse this seems to be a disadvantage of our simulation conception—but if one compares \(N'\) and \(N'_1\) more carefully it turns out that their behaviors are quite different: An input signal on \(p_1\) of \(N'\) prohibits \(e\) to become active immediately, while an input signal on \(p_1\) of \(N'_1\) only prohibits \(e\) from activity eventually. This results in quite different computations of both systems. As a consequence of this observation a criticism should not contain our simulation but the hidden synchronizations implied in the logic of inhibitory places.

6. Modular decomposition

We now present a simulation of Petri-nets by certain classes of \(APA\)-nets.

**Theorem 5.** Let \(W\) be any base for Normed Networks (see Theorem 1). Any \(PN\) can be promptly, 2-safely simulated by an \(APA\)-net over \(K, V, W, T_2, SEQ\) and \(W\).

**Sketch of a proof.** One may define ‘modular blocks’ for Petri-nets as subnets where all events are ‘conflict-connected’. We say that two events \(e, e'\) are in conflict if they share a common input place and that they are conflict-connected if there exist events \(e_1, \ldots, e_n\) such that \(e = e_1, e' = e_n\) and for all \(i, 1 \leq i < n, e_i\) and \(e_{i+1}\) are in conflict. Any Petri net can be regarded to be composed of modular blocks where these blocks are connected with each others in exactly the same was as \(APA\)-nets are. (But we may drop the \(K\)-modules needed in \(APA\)-nets in these modular blocks, as several arcs may lead into the same place.)

Thus it suffices to simulate modular blocks. We will outline this simulation by one simple example that nevertheless shows all inherent difficulties.

Let us regard the Petri net, \(N\), of Fig. 15 that is a modular block itself. \(N\) is simulated by the \(APA\)-net, \(A_N\), of Fig. 15. CONTROL is a sequential automaton that will be explained in the sequel. Any token on any place \(i, 1 \leq i \leq 5\), of \(N\) becomes a signal on wire \(i\) of \(A_N\). \(A_N\) is in a standard coded state \(s^*\) if there is a signal on all wires that are signed by a cross in our figure, i.e., the wires \(t\) and \(1a\) up to \(5a\), all \(T_2\)-modules are in ‘off’, \(SH\) is in state ‘on’, CONTROL is in some distinguished initial state and in \(s^*\) there are exactly as many signals on the corresponding wires as in \(s\) are tokens on some places. The signal on wire \(t\) spreads in four signals, each trying to pass a cascade of \(T_2\) modules. Such a cascade
corresponds to an event $A$, $B$, $C$ or $D$ of $N$. Any cascade $X, X \in \{A, B, C, D\}$, can only be passed if the corresponding event $X$ is fireable. If there are several fireable events several cascades can be passed, but only one signal may also pass SH, telling CONTROL what event shall be fired. Let us suppose that a signal reaches CONTROL via input $t_R$. Thus CONTROL knows that two signals have to be removed from the places 2 and 3, or, in $N_A$, the $T_2$-modules $(A, 2), (A, 3), (B, 2), (B, 3), (C, 2)$ and $(C, 3)$ have to be switched back into state ‘off’.

Fig. 15.
However, the difficulty rises that signals from wire 2 and 3 may not have changed all of these $T_2$-modules into state ‘on’. To handle this problem CONTROL sends out one signal via one wire of the $a_i$-combinations that spreads after passing SH in such a way that all $(X, i)$-components for $X \in \{A, B, C, D\}, i \in \{1, 4, 5\}$ receive one signal via their $a_i$-input. Now in principle all $T_2$-modules may be switched into state ‘on’. Thus all remaining signals from the original signal on the $t$-wire may also pass their $T_2$-cascades. As SH is reset to ‘on’ one further of these test-signals may reach CONTROL. Via $r_1$ and $r_2$ CONTROL can always reset SH to ‘on’—thus waiting to receive all test-signals. By the help of $r'_1$ and $r'_2$ the signals from $r_1$ and $r_2$ are then withdrawn (to avoid an accumulation of signals in $A_N$). Now by sending out one signal via output $s$ and waiting for an arrival of this signal on input $s'$ CONTROL can reset all $T_2$-modules to ‘off’. CONTROL now sends out one signal via its output $O_B$ that simulates the action of the event $B$, switches SH back to state ‘on’ (this is not shown in the figure for legibility) and resets a signal on the wires $t$, $2a$ and $3a$ such that another ‘input’ signal may enter via the lines 2 and 3.

As CONTROL is a sequential machine that is never used by several signals in $A_N$ concurrently we may replace CONTROL by Theorem 1 by a Normed Network over any base. We can replace any SH'' in $A_N$ by a net over $\text{Seq''}$, $V, W, K$, as shown in Fig. 16. Further any $\text{Seq''}$ can easily be replaced by a tree of $\text{SEQ}$-modules, see Fig. 16. $\square$

In addition, if $N$ is $k$-safe, than also all wires of $A_N$ are $k$-safe except of the $a_2$-wires of the $T_2$ modules that may be $k + 1$-safe. This case is handled in the next theorem.
**Theorem 6.** Let $\mathcal{M}$ be any base for Normed Networks. Any PN can be promptly 1-safely simulated by an APA-net over $K, V, W, SEQ$ and $\mathcal{M}$

**Proof.** We have to replace $T_2$ in the previous construction. If we do not care about 1-safeness, we may use the net $N_T$ of Fig. 17. Here and in the sequel we use the following graphical notation to describe a distinguished state of the net in a diagram:

A cross on a wire denotes a signal on this wire, and a dot inside a module denotes that this module is in such a state that the output signed with a dot must be used in a test.

For our net of Fig. 17 this implies that a signal via the $t$-input may pass $N_T$ to the output $t'$ iff $N_1$ is a state 'on' (that is expressed by a signal on the wire $@$). A signal entering $N_T$ via $s$ removes such a signal on $@$ and leaves via $s'$, restoring the old state of $H$. As in $A_N$ of Fig. 15 $t$ and $s$ have to used sequentially but not concurrently our construction works fine. Further, as $H$ is only used sequentially and never concurrently we may substitute $H$ by a Normed Network over any base. However, two signals may accumulate on the wire $@$.

Settling this problem requires some effort and leads to the net $N_T$ of Fig. 18 that shall now replace each $T_2$-module of $A_N$.

Let us briefly discuss $N_T$: In state 'off' of $T_2$ all $H$-modules of $N_1$ are in such a state that any test-signal would use the output-line marked by a dot. $T_2$ can be switched to state 'on' by a signal via an $a_1$ or $a_2$ input. $H_2$ stores what input was used. Assume an $s$-signal wants to switch $N_T$ to state 'off'. This signal withdraws a signal from wire $@$ (that defines the state 'on'), switches $H_1$ to its correct state and asks $H_2$ whether $a_1$ or $a_2$ was used. If $a_1$ was used the signal reaches CONTROL via line $s'_1$. CONTROL knows whether it has sent a signal on the $a_2$-input of $N_T$. If not, a signal is sent back to $N_T$ via $s''_1$' that simply switches $SH$ to its correct state 'on'. In the other case a signal is sent back to $N_T$ via $s''_1'$ that changes $H_3$.
and also sets $SH$ back to 'on'. As only one signal via $a_1$ or $a_2$ could pass $SH$ and $s'_1$ was used the $a_2$-signal has not passed $SH$. It now can do so and reaches $W$, thus switches $H_2$ and $H_3$ and $SH$ to their distinguished states and leaves $N_T$ correctly in state 'on' (as both inputs $a_1$ and $a_2$ have been used) via $s'$.

Again, the $H$-modules are only sequentially used (and may thus be replaced by a Normed Network). This can be seen easily, as the $s$-input must be used after a successful passing of a signal via the $t\mathbin{-}t'$-line—see our construction of Theorem 5—where $T_2$ is for sure in the state 'on'. This implies that a signal must have entered $N_T$ via $a_1$ or $a_2$ switching the sequential hull $SH$ into the state 'off', with a result that not two signal may try to pass the $H$-modules of $N_T$ concurrently. □

**Theorem 7.** Let $\mathcal{M}$ be any base for Normed Networks. Any PN can be simulated

(i) promptly, 1-safely by an APA-net over $K$, $V$, $W$, $S$ and $\mathcal{M}$,
(ii) promptly, 2-safely by an APA-net over $K$, $E$, $V$ and $W$.

**Proof.** For (i) we have to replace SEQ by $S$. Fig. 19 shows how this can be managed.

A proof for (ii) is quite complicated and not straightforward. We present an outline: Let us regard the net $N$ of Fig. 20. $N$ is a kind of a SEQ-module. One signal from 1 or 2 may pass to the output $0'$ and a signal via 0 will reach $1'$ or $2'$. $2'$ must be used if a signal from input 2 has meanwhile entered. Thus, whilst a signal from input 1 may pass to $0'$, a signal from 0 may nevertheless reach $2'$ if a 2-input has meanwhile entered $N$. It should be noted that therefore $N$ is no simulation of SEQ, but nevertheless we may construct with $N$-modules a module $M$ according to the two construction steps of Fig. 16. This module $M$ again behaves like a sequential hull module $SH$: some input signals via the inputs 1, ..., $n$ may
enter $M$ and exactly one of them (but not necessarily the first one that has entered!) may pass $M$ in state 'on' and will switch to the next state 'off'. Again, $M$ is no simulation of an SH-module, but we may use $M$ instead of an SH-module in our main construction of Fig. 15 and everything in this figure will work fine. As $K$ and $E$ form a base, see Theorem 1, we may drop $R$. □

in our figures we have signed wires, where two signals may accumulate, by (2).

**Theorem 8.** Any PN can be promptly, 2-safely simulated by an APA-net over $K$, $V$, $W$ and $S$.

**Proof.** Fig. 21 shows how to replace SEQ and Fig. 22 gives a prompt, 2-safe simulation of the sequential automaton $P$ (compare Fig. 2).

As $K$ and $P$ form a base, see Theorem 1, the proof is finished. □

Without a proof (that yields similar construction) we state that any inhibitory Petri-net can be simulated as described in Theorem 5 up to 8 if one always adds the module $R$ (of Table 1) to the APA-net classes.
As any APA-net over finite state automata is trivially promptly and 1-safely simulated by a Petri-net these results also give a decomposition theorem for APA-nets.

**Conclusion.** Theorem 2 and the previous theorems imply:

\[
\begin{align*}
\text{safe } PN &= \text{safe APA}^{+} = 1\text{-safe } PN = 1\text{-safe APA}^{+} \\
&= 1\text{-safe APA}(\mathcal{M} \cup \{K, V, W, S\}) \\
&= 1\text{-safe APA} (K, E, V, W, S) \\
&= 2\text{-safe APA} (K, F, V, W) \\
&= 2\text{-safe APA}(K, V, W, S^{+}),
\end{align*}
\]

especially,

\[
\begin{align*}
PN &= \text{APA}^{+} = \text{APA}(\mathcal{M} \cup \{K, V, W, S\}) = \text{APA}(K, E, V, W) \\
&= \text{APA}(K, V, W, S^{+});
\end{align*}
\]

where \(\mathcal{M}\) may be any base of Normed Networks.

Without a proof we state some more results on non-prompt simulation:

**Theorem 9.** Any PN can be simulated

(i) 1-safely by an APA-net over \(K, V, W\) and \(E\),

(ii) 2-safely by an APA-net over \(K, V, W, F\) and \(D\).
These results present our first decomposition theorems on APA-nets and, thus, on Petri-nets. The most surprising result is the existence of a very simple base—namely $K$ and $E$—for sequential finite state machines such that this base together with only $V$ and $W$ is also a base for arbitrary Petri-nets.

There arise at once two open questions:

1. Given any base for sequential finite state machines (i.e. for Normed Networks), does this base together with $V$ and $W$ form also a base for Petri-nets?
   (Conjecture: No.)

2. Is it decidable whether a set of modules forms a base for Normed Networks (or for Petri-nets)?

7. Modular Petri-nets

Let us regard the Petri-nets of Table 2. These Petri-nets are also automata. Also, any APA-net built up from these components is by its structure a Petri-net. We call these Petri-nets 'modular decomposed Petri-nets' and with $PN(A_1, \ldots, A_n)$ we simply denote $APA(A_1, \ldots, A_n)$ for $A$, from Table 2. As a consequence from the previous theorems we get a theorem on the modular decomposition of Petri-nets:

**Theorem 10**

\[
MG = PN(V, W),
\]

\[
SM = PN(K, I),
\]

\[
FC = PN(K, I, V, W),
\]

\[
SN = PN(K, S, V, W),
\]

\[
PN = PN(K, U, V, W),
\]

\[
tPN = PN(K, R, U, V, W).
\]
Proof. It remains to simulate the sequential automaton $E$ in $\text{PN}(K, U, V, W)$ as a consequence of Theorem 7. All other cases are trivial. Also, we leave this simple remaining construction to the reader. □

Note that, as any $k$-safe $R$-module can easily be simulated by an APA-net over $K$, $V$, $W$ and $E$, we thus have an alternative proof that bounded (safe) iPN can be promptly simulated by 1-safe PN.

8. Conclusions and discussion

Such an automata-theoretical approach to modular, concurrent systems should also give some more insight into older attempts to design control structures for asynchronous systems.

Let us as an example discuss an open problem in Yoeli and Brzozowski [38]. The structure diagram of Fig. 23 can be found there and they put the question how 'powerful' the classes of control structures build from the modules of the diagram are.

There modules $U$ (UNION of Wires), $F$ (FORK of signals), $J$ (JOIN of signals) and $C$ (CHOICE of outputs) are just our modules $K$, $V$, $W$ and $I$. However, $D$ (DECIDER) is no module at all. It is a decision element that decides according to the status of some data (lines) (or some predicate), thus $D$ 'looks' into other parts of the net and its behavior is not restricted to its state and the status of its interface. However, if one assumes that data is always a finite object, descriptive
by finite states (on data-lines, e.g.) it turns out that $D$ can be replaced by $E$ (or better by a net over $K$, $V$, $W$ and $E$), and vice versa. In this sense their element $T$ can also be replaced by a construction similar to Fig. 19 with inhibitory Petri net parts and thus by an appropriate APA-net class with $R$—but compare Lemma 9: $T$ must not be an ‘input element’ as $T$ allows for inhibitory input lines. On the other hand, the register $R$ is easily substituted by $T$. We may thus reformulate their question and ask about the corresponding relations of the APA-net classes over $K,E (= U, D)$, $K,E,I (= U, D, C)$, $K,E,V,W (= U, D, F, J)$, $K,E,V,W,I (= U, D, F, J, C)$, $K,E,V,W,R (= U, D, F, J, T)$ and $K,E,V,W,I,R (= U, D, F, J, T, C)$. However, all these questions are solved in this paper: \text{APA}(K, E, V, W, R) are (always understood modulo prompt simulation) the inhibitory Petri-nets. The same holds for \text{APA}(K, E, V, W, R, I). \text{APA}(K, E, V, W)$ and $\text{APA}(K, E, V, W, I)$ are the Petri-nets. In the case of $K, E$ and $K, E, I$ it seems to be reasonable to regard them as Normed Networks—as there is no split or join of signals we suppose that there is only one signal operating—with the result that $\text{NN}(K, E)$ are the deterministic, finite state machines and $\text{NN}(K, E, I)$ are the finite state machines (possibly non-deterministic). Thus we receive the diagram of Fig. 24. It should be noted that Yoeli and Bizoño allow only for 1-safe markings—but this hierarchy is also classified here.

It should further be noted that Keller’s serial modules are our Normed Networks and his speedindependent modules are our 1-safe APA-nets, see Keller [16]. Thus we answered all open questions of Keller and improved his results. The well-known modules of Bruno and Altmann [4] correspond to our APA-nets without $K$-module and with pairs of ready-acknowledge wires. The result of Section 7 on modular decomposed Petri nets can also be found in Furtek [9]. However it is not too clear to me what his ‘implementation’ concept means and what effects his solving of conflicts by a fixed total ordering on some places has.
It is my belief that such a simple, straightforward approach with concurrently operating finite state machines clarifies certain problems of the theory of modular control structures (note, e.g., that there is no need for a 'hand-shaking' I/O-connection between any two connected modules) and may be an alternative treatment of concurrency. Also this approach, that one tries to define what properties a simulation of concurrent computations should preserve instead of trying to define directly what a concurrent computation is, seems to be promising and might lead to a deeper understanding of concurrency.

Acknowledgment

For a critical review and fruitful discussions that led to some improvements of this paper I would like to thank Frau Anne Brüggemann, Frau Cäcilie Hesse, Herrn Andreas Podelski and Herrn Rüdiger Schätz of the University of Münster.

References


[34] Record of the Project MAC Conference on Concurrent Systems and Parallel Computation ACM (1970).

