Algorithmic Aspects of Concurrent Automata

Peter Deussen

Brandenburg Technical University Cottbus
Computer Science Institute
—Data Structures and Software Dependability—
email: pd@informatik.tu-cottbus.de
tel: (+49-355) 69 3826, fax: (+49-355) 69 3820
Postbox 10 13 44, 03013 Cottbus, Germany

ABSTRACT. Partial order semantics of Petri nets have a long history. In this paper, we describe a formalism which combines partial order semantics with the usual notion of markings of a Petri net. We call this formalism *concurrent automata*. We present a generation algorithm for concurrent automata. We show that our algorithm is correct in the sense of semi language equivalence: The generated automaton recognizes essentially the same set of semiwords as the associated Petri net.

Key words. Concurrent automata, Petri Nets, Partial Order Semantics, Semiwords, Semi Languages.

1. Introduction

Partial order semantics of formalisms designed to describe concurrent systems have a long history. Concentrating on Petri nets, instances of those semantics are processes [**1**], (prime) event structures [**11**], partial words [**6**] and semiwords [**12, 17**], or branching processes [**3**]. Especially a finite representation of a branching process of a 1-bounded Petri net, called the *finite prefix of the maximal branching process of a Petri net* (*prefix*, for short) has turned out as extraordinary useful for analysis goals (see [**9, 4**] for available analysis techniques).

In this paper, we describe another formalism which combines partial order semantics with the usual notion of markings of a Petri net. We call this formalism *concurrent automata*. Concurrent automata have the benefit that concurrency of transitions does not necessarily lead to the state explosion problem. Moreover, since global states (markings) are maintained, it is possible to rejoin branching behaviour in opposite to branching processes.

Concurrent automata were originally introduced by Ulrich [**15, 16**]. Ulrich uses the term *behaviour machine*. We prefer the term concurrent automaton because it seems to meet the crucial point somewhat better. In [**15**], an algorithm for the construction of a behaviour machine is presented. This algorithm uses the prefix of a safe Petri net as input. As noted in [**16**], the algorithm does not work correctly if the Petri net under consideration contains dead markings. Another disadvantage is that the input prefix has to have certain structural properties, namely that each *cut-off event* and at least one of its associated events have to be in the same *local configuration*

(see [**5**] for the meaning of the terms *cut-off event* and *local configuration*—a detailed explanation is far beyond the scope of this paper). This implies that only the most inefficient cut-off criterion for the generation of the prefix can be applied (see [**5**]).

In this paper, we present another generation algorithm which does not depend on the prefix of a Petri net. We show that our algorithm is correct in the sense of semi language equivalence: The generated automaton recognizes an essential subset of the set of semiwords generated by the associated Petri net. Our algorithm is however restricted to safe Petri nets.

This paper is organized as follows: Section 2 describes the basic notations used in the following, especially semiwords are introduced. In section 3, Petri nets are introduced and semiwords are used to define partial order semantics for Petri nets. In dealing with Petri nets, we adopt mostly the notations given in [**14**]. Section 4 addresses the existence and uniqueness of least sequential semiwords, i. e. those semiwords which express the causal relation of the transitions of a Petri nets mostly adequate. In section 5, concurrent automata are introduced. The semi language of a concurrent automaton (w. r. t some Petri net) is defined. Definitions for the correctness and completeness of a concurrent automaton are given. In Section 6, a construction algorithm is described. It is shown that this algorithm is complete and correct in the sense of the previous section 5. Finally, section 7 addresses open problems and further works.

An extended version of this paper will be published as [**2**], where also proofs omitted here can be found.

2. Semiwords

To avoid tedious notions, we fix the following convention: If a structure $S = \langle A, B, \ldots \rangle$ is introduced, the components of S will always be denoted by $A_S, B_S, \ldots$.

We use the following notations: $\mathbb{N}$ and $\mathbb{Z}$ denote the sets of non-negative integers and integers, respectively. For some set A, $\mathscr{P}(A)$ denotes the set of all subsets of A. For $R \subseteq A \times B$ and $a \in A$, we denote the *image* of a under b by $R(a) =_{\mathrm{df}} \{b \in B : a\ R\ b\}$. This notation is extended to subsets $C \subseteq A$ by $R(C) =_{\mathrm{df}} \bigcup_{a \in C} R(a)$. $R^{-1} \subseteq B \times A$ denotes the *inverse* of R, i. e, $b\ R^{-1}\ a \Leftrightarrow_{\mathrm{df}} a\ R\ b$. For each set A, $\mathbf{id}_A \subseteq A \times A$ denotes the *identity relation* on A, i. e. $a\ \mathbf{id}_A\ b \Leftrightarrow_{\mathrm{df}} a = b$. $R^+ \subseteq A \times A$ denotes the least transitive relation containing $R \subseteq A \times A$. Let $R \subseteq A \times A$ be a binary relation on A and $B \subseteq A$. B is called *pre-closed* with respect to R iff $B = R^{-1}(B)$. A *preorder* on a set A is a irreflexive and transitive relation $< \subseteq A \times A$. A *partial order* on A is an asymmetric preordering on A. If $<$ ($\prec$) denotes a preordering on A, then $\leqslant =_{\mathrm{df}} < \cup\, \mathbf{id}_A$ ($\preccurlyeq =_{\mathrm{df}} \prec \cup\, \mathbf{id}_A$). Throughout this paper, we do not distinguish between mappings $f : A \to B$ and their graphs $\{\langle a, b\rangle \in A \times B : b = f(a)\}$.

Let T be an alphabet. A *(finite) labelled partial order* (lpo) over T is a tuple $a = \langle E, <, \lambda\rangle$, where E is a finite set of *events*, $< \subseteq E \times E$ is a partial order, and $\lambda : E \to T$ is a labelling function.

Let a be a lpo over an alphabet T. We use the following notations:

1. The relation $\mathbf{co}_a \subseteq E_a \times E_a$ is defined by $e_1\ \mathbf{co}_a\ e_2 \Leftrightarrow_{\mathrm{df}} \neg(e_1 \leqslant_a e_2)\ \&\ \neg(e_2 \leqslant_a e_1)$. A set $C \subseteq E$ is called a *co-set* iff we have $e_1\ \mathbf{co}_a\ e_2$ for all $e_1, e_2 \in C$ such that $e_1 \neq e_2$.
2. If $t \in T$, the lpo $\langle \{0\}, \varnothing, \{\langle 0, t\rangle\}\rangle$ is called a *letter*. If no confusion can occur, we use t both to denote an element of T and its letter.
3. The *empty lpo* is $\varepsilon =_{\mathrm{df}} \langle \varnothing, \varnothing, \varnothing\rangle$.
4. A *semiorder* is a lpo a where for all $e_1, e_2 \in E_a$, $e_1\ \mathbf{co}_a\ e_2 \Rightarrow \lambda_a(e_1) \neq \lambda_a(e_2)$. $\mathbf{so}(T)$ denotes the class of semiorders over T.

DEFINITION 2.1 (Prefix and Sequentialization). Let a and b be lpo's.

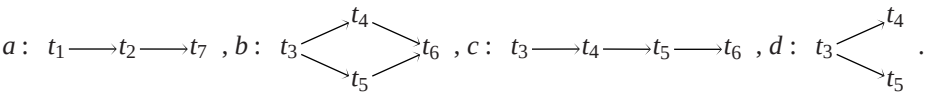

FIGURE 1. Some semiorders over $T = \{t_i : 1 \leqslant i \leqslant 7\}$.

1. A mapping $h : E_b \to E_a$ is called a *pre-homomorphism* iff $e_1 <_b e_2$ implies $h(e_1) <_a h(e_2)$ for all $e_1, e_2 \in E_b$ and furthermore, $\lambda_a = \lambda_b \circ h$. It is called a *homomorphism* iff it is a pre-homomorphism with the property $h(\leqslant_b^{-1}(e)) = \leqslant_a^{-1}(h(e))$.
2. b is called a *prefix* of a, denoted by $b \leqslant a$, iff there is an injective homomorphism $h : E_b \to E_a$ We write $b \equiv a$, if $b \leqslant a$ and $a \leqslant b$ holds.
3. a is called a *sequentialization* of b, denoted by $b \preccurlyeq a$, iff there is a bijective pre-homomorphism $h : E_b \to E_a$.
4. If $D \subseteq E_a$, then we denote by $a[D]$ the *lpo generated by D in a*: $a[D] =_{\mathrm{df}} \langle D, <_a \cap (D \times D), \lambda_a \cap (D \times T)\rangle$. Clearly, if $D = \leqslant_a^{-1}(D)$, then $a[D] \leqslant a$. The required homomorphismus is just $\mathbf{id}_D$. □

The notions introduced above are illustrated by the examples in figure 1. Event names are omitted, only their labels are shown. The order relation is figured by arrows. Transitive arcs are omitted. We have $b \preccurlyeq c$ and $d \leqslant b$.

DEFINITION 2.2 (Partial words and Semiwords). A *partial word* is an equivalence class (w. r. t. $\equiv$) of lpo's. A semiword is an equivalence class of semiorders. We write $[a] = [E_a, <_a, \lambda_a]$ to denote the equivalence class of a lpo a. The same notion applies to semiwords. A *semi language* is a set of semiwords. $\mathbf{sw}(T)$ denotes the class of semiwords over T. □

We fix the following conventions: If $a, b, c, \ldots$ are lpo's or semiorders, then we use boldfaced lowercase letters $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$, to denote its equivalence class $[a], [b], [c], \ldots$. Hence, for instance, E_a will always refer to the event set of a representant of the partial word $\boldsymbol{a} = [a]$. The equivalence class of ε will also be denoted by ε. Especially, if t is a letter, we use $\boldsymbol{t}$ to denote the semiword $[t] = [\{0\}, \varnothing, \{\langle 0, t\rangle\}]$.

It is easy to prove the following lemma:

LEMMA 2.3. *Both $\leqslant$ and $\preccurlyeq$ are preorderings on the class of lpo's. $\equiv$ is an equivalence relation. $a \equiv b$ iff $a \preccurlyeq b$ and $b \preccurlyeq a$. If we put $\boldsymbol{a} \leqslant \boldsymbol{b} \Leftrightarrow_{df} a \leqslant b$, and $\boldsymbol{a} \preccurlyeq \boldsymbol{b} \Leftrightarrow_{df} a \preccurlyeq b$ for all $a \in \boldsymbol{a}$, $b \in \boldsymbol{b}$, then $\leqslant$ and $\preccurlyeq$ are reflexive partial orderings on partial words and semiwords, respectively.*

3. Partial Order Semantics of Petri Nets

A *net* $\langle P, T, F\rangle$ consists of non-empty, finite sets P and T such that $P \cap T = \varnothing$, where the elements of P and T are called *places* and *transitions*, respectively, and a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$. We assume a net to be connected, i. e. $P \cup T = (F \cup F^{-1})^+(x)$ for each $x \in P \cup T$. A *state* or *marking* of a net $\langle P, T, F\rangle$ is a mapping $m : P \to \mathbb{N}$. If $\langle P, T, F\rangle$ is a net and m is a marking of N, we call the tuple $N = \langle P, T, F, m\rangle$ a *Petri net*. m is called the *initial marking* of N.

Figure 2 gives an example of a Petri net. Places are figured as circles, transitions as rectangles, and the flow relation is indicated by arcs. Black dots (tokens) are used to indicate a marking m of the net. In this example, $m(p_1) = 1$, and $m(p) = 0$ for all places different from p_1.

The mappings $(\cdot)^-, (\cdot)^+ : T_N \to (P_N \to \mathbb{N})$, and $\Delta : T_N \to (P_N \to \mathbb{Z})$ are defined to be

$$t^-(p) \quad =_{\mathrm{df}} \quad \begin{cases} 1, & \text{if } p\, F_N\, t, \\ 0, & \text{otherwise,} \end{cases} \qquad t^+(p) \quad =_{\mathrm{df}} \quad \begin{cases} 1, & \text{if } t\, F_N\, p \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{and} \quad \Delta t \quad =_{\mathrm{df}} \quad t^+ - t^- \quad \text{(component wise)}.$$

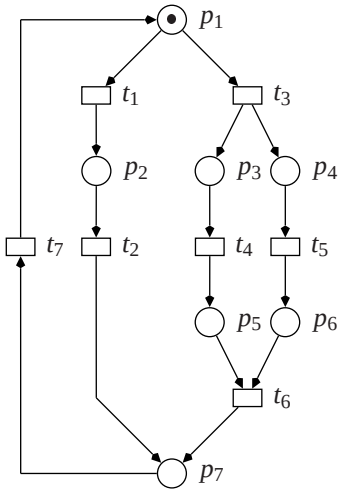

FIGURE 2. A Petri net N

Let a be a semiorder over the transition set T_N of a Petri net N. For each $e \in E_a$ we define $e^- =_{\mathrm{df}} \lambda_a(e)^-$, $e^+ =_{\mathrm{df}} \lambda_a(e)^+$, and $\Delta e =_{\mathrm{df}} \Delta\lambda_a(e)$. For $D \subseteq E_a$ we define $D^- =_{\mathrm{df}} \sum_{e\in D} \lambda_a(e)^-$, $D^+ =_{\mathrm{df}} \sum_{e\in D} \lambda_a(e)^+$, and $\Delta D =_{\mathrm{df}} \sum_{e\in D} \Delta\lambda_a(e)$, where $\sum$ is understood component wise.

A lpo a is called *enabled* at a marking m of N iff for all co-set C of a we have: $C^- \leqslant m + \Delta(<_a^{-1}(C))$. We write $m \stackrel{a}{\Longrightarrow}$, if a is enabled at m. The *successor marking* m' of m and a is $m' = m + \Delta E_a$. In this case, we write $m \stackrel{a}{\Longrightarrow} m'$. By

$$\mathscr{L}_N(m) \quad = \quad \left\{ \boldsymbol{a} \in \mathbf{sw}(T_N) : m \stackrel{a}{\Longrightarrow} \right\}$$

we denote the *semi language* of a Petri net N and a marking m of N. The set

$$\mathscr{R}_N(m) \quad = \quad \left\{ m' \in (P_N \to \mathbb{N}) : \exists \boldsymbol{a} \in \mathscr{L}_N(m) \left(m \stackrel{a}{\Longrightarrow} m' \right) \right\}$$

denotes the set of markings reachable from a marking m of N. A Petri net N is called *k-bounded* iff there is some $k \geqslant 0$ such that $m(p) \leqslant k$ for all $m \in \mathscr{R}_N(m_N)$ and $p \in P_N$. It is called *safe* if it is 1-bounded. A marking $m \in \mathscr{R}_N(m_N)$ is called *dead* iff $m \not\geqslant t^-$ for all $t \in T_N$. A transition t is called life in N iff for all $m \in \mathscr{R}_N(m_N)$ there is some $m' \in \mathscr{R}_N(m)$ such that $m' \geqslant t^-$.

The semiorders shown in figure 1 are all enabled at the initial marking of the Petri net in figure 2.

THEOREM 3.1 (Starke [**13**]). *If $m \stackrel{a}{\Longrightarrow}$, then $m \stackrel{b}{\Longrightarrow}$ for each $\boldsymbol{a} \leqslant \boldsymbol{b}$, i. e. $\mathscr{L}_N(m)$ is pre-closed w. r. t. $\leqslant$. If $m \stackrel{a}{\Longrightarrow} m'$, then $m \stackrel{b}{\Longrightarrow} m'$ for each $\boldsymbol{a} \preccurlyeq \boldsymbol{b}$, i. e. $\mathscr{L}_N(m)$ is pre-closed w. r. t. $\preccurlyeq^{-1}$.*

4. Least Sequential Semi Languages

We now address the following problem: If $\mathscr{L}_N(m_N)$ is the semi language of a Petri net N, is it possible to find another semi language $\mathscr{S} \subseteq \mathscr{L}_N(m_N)$ such that $\preccurlyeq(\mathscr{S}) = \mathscr{L}_N(m_N)$? Is there a uniquely defined minimal semi language $\mathscr{S}$ with this property? These questions will be answered positively in the following; this *least sequential semi language* will be denoted by $\mathscr{S}_N(m_N)$.

Another problem we have to solve is: Given two least sequential semiwords $\boldsymbol{a}$ and $\boldsymbol{b}$ such that $m \xrightarrow{a} m' \xrightarrow{b}$ holds for reachable markings m, m' of a Petri net N, can we find an operation $\odot$ on semiwords such that $\boldsymbol{a} \odot \boldsymbol{b}$ is again the least sequential semiword with $m \xrightarrow{a \odot b}$?

Let N be a safe Petri net and let $\boldsymbol{a} \in \mathscr{L}_N(m_N)$. Define $A(e) =_{\mathrm{df}} F_N(\lambda_a(e)) \cup F_N^{-1}(\lambda_a(e))$ for each $e \in E_a$.

LEMMA 4.1. *Let N be a safe Petri net and let $\boldsymbol{a} \in \mathscr{L}_N(m_N)$. Then for all $e, e' \in E_a$ we have $A(e) \cap A(e') \neq \varnothing \Rightarrow e \leqslant_a e' \vee e' \leqslant_a e$,*

PROOF (Sketch). Assume $e, e' \in E_a$ such that $A(e) \cap A(e') \neq \varnothing$ and $e \,\mathbf{co}_a\, e'$, i. e. $C = \{e, e'\}$ is a co-set. If $F_N^{-1}(\lambda_a(e)) \cap F_N^{-1}(\lambda_a(e')) \neq \varnothing$, we would have $(m_N + \Delta{<_a^{-1}}(C))(p) > 1$ for each $p \in F_N^{-1}(\lambda_a(e)) \cap F_N^{-1}(\lambda_a(e'))$, If $F_N(\lambda_a(e)) \cap F_N(\lambda_a(e')) \neq \varnothing$, then $(m_N + \Delta{\leqslant_a^{-1}}(C))(p) > 1$ for each $p \in F_N(\lambda_a(e)) \cap F_N(\lambda_a(e'))$. Finally, $F_N(\lambda_a(e)) \cap F_N^{-1}(\lambda_a(e')) \neq \varnothing$ implies $(m_N + \Delta{\leqslant_a^{-1}}(\{e\}))(p) > 1$ for each $p \in F_N(\lambda_a(e)) \cap F_N^{-1}(\lambda_a(e'))$. In any case, the safety of N would be contradicted. □

LEMMA 4.2. *If $\boldsymbol{a}$ is a semiword over the transition set T_N of a safe Petri net N and $m \in \mathscr{R}_N(m_N)$ is a reachable marking of N, then $m \xrightarrow{a}$ iff for each $e \in E_a$ we have $m + \Delta{<_a^{-1}}(e) \geqslant e^-$.*

PROOF. ($\Rightarrow$) Assume $m \xrightarrow{a}$. Let C be a co-set of a. Then $m + \Delta{<_a^{-1}}(C) \geqslant C^-$, which implies $m + \Delta{<_a^{-1}}(C) \geqslant e^-$ for each $e \in C$. By lemma 4.1 we have $A(e) \cap A(C - {\leqslant_a^{-1}}(e)) = \varnothing$, which implies $(m + \Delta{<_a^{-1}}(C - {\leqslant_a^{-1}}(e)))(p) = m(p)$ for each $p \in A(e)$. We conclude $(m + \Delta{<_a^{-1}}(C))(p) = (m + \Delta{<_a^{-1}}(e))(p)$. Then $m + \Delta{<_a^{-1}}(e) \geqslant e^-$ by assumption.

($\Leftarrow$) Assume $m + \Delta{<_a^{-1}}(e) \geqslant e^-$ for each $e \in E_a$. Let C be a co-set of a. First, we may note that $C^-(p) \leqslant 1$ for each $p \in P_N$, because $C^-(p) > 1$ would imply that there are events $e_1, e_1 \in C$, $e_1 \neq e_2$, such that $p \in F_N^{-1}(\lambda_a(e_1)) \cap F_N^{-1}(\lambda_a(e_2))$, which implies $A(e_1) \cap A(e_2) \neq \varnothing$. But then $e_1 <_a e_2$ or $e_2 <_a e_1$ by lemma 4.1, and C would not be a co-set. Suppose $(m + \Delta{<_a^{-1}}(C)) \not\geqslant C^-$, i. e., there is some place $p \in P_N$ such that $(m + \Delta{<_a^{-1}}(C))(p) < C^-(p)$, i. e. $(m + \Delta{<_a^{-1}}(C))(p) \leqslant 0$ and $C^-(p) = 1$. By assumption we have $(m + \Delta{<_a^{-1}}(e))(p) \geqslant 1$, i. e there must be at least some event $e' \in {<_a^{-1}}(C - {\leqslant_a^{-1}}(e))$ such that $p \in F_N^{-1}(\lambda_a(e'))$. But this is impossible because of lemma 4.1. We conclude $m + \Delta{<_a^{-1}}(C) \geqslant C^-$. □

Lemma 4.2 provides an inductive proof method for enabledness proofs. To prove that a semiorder $\boldsymbol{a}$ is enabled at a marking m of a safe Petri net N, it has to be shown that for each $e \in E_a$ the implication $\forall e' \in E_a (e' <_a e \;\&\; m + \Delta{<_a^{-1}}(e') \geqslant e'^-) \Rightarrow m + \Delta{<_a^{-1}}(e) \geqslant e^-$ is true. As an application of the principle of Noetherian Induction, it is allowed to conclude $m + {<_a^{-1}}(e) \geqslant e^-$ for each $e \in E_a$, hence, by lemma 4.2, $m \xrightarrow{a}$.

The following theorem states that if we consider safe Petri nets N, for each member of the semi language of N it exists a uniquely defined least sequential semiword. This is not true for non-safe nets, as the example in figure 3 shows: The semiword $\boldsymbol{c}$ is a sequentialization of both $\boldsymbol{a}$ and $\boldsymbol{b}$, but neither $\boldsymbol{a} \preccurlyeq \boldsymbol{b}$ nor $\boldsymbol{b} \preccurlyeq \boldsymbol{a}$ holds.

The theorem resembles (the second part of) theorem 2.2.9 in [**17**]. Vogler uses a *process semantics* for Petri nets, which is not considered in this paper. We give a direct proof.

THEOREM 4.3. *Let N be a safe Petri net and let $\boldsymbol{a} \in \mathscr{L}_N(m_N)$. Then the set $\preccurlyeq^{-1}(\boldsymbol{a}) \cap \mathscr{L}_N(m_N)$ contains a uniquely defined least element with respect to $\preccurlyeq$, namely the semiword $\boldsymbol{a}^{\downarrow} =_{df} [E_a, R^+, \lambda_a]$, where $e \,R\, e' \Leftrightarrow_{df} e <_a e' \;\&\; A(e) \cap A(e') \neq \varnothing$.*

PROOF. Clearly, the operation $(\cdot)^{\downarrow}$ is well-defined. We have to prove the following properties of $\boldsymbol{a}^{\downarrow}$:

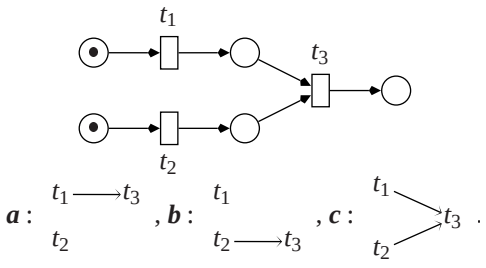

FIGURE 3. A Petri net N and some of its semiwords. $\boldsymbol{a} \preccurlyeq \boldsymbol{c}$ and $\boldsymbol{b} \preccurlyeq \boldsymbol{c}$, but neither $\boldsymbol{a} \preccurlyeq \boldsymbol{b}$ nor $\boldsymbol{b} \preccurlyeq \boldsymbol{a}$.

(1) $<_{a^\downarrow}$ is a partial order. This follows from the fact that $<_a$ is a partial order.

(2) $\boldsymbol{a}^\downarrow \in \mathscr{L}_N(m_N)$. Because of lemma 4.2 it is enough to show that for all $e \in E_a$, if $[a^\downarrow[<_{a^\downarrow}^{-1}(e)]] \in \mathscr{L}_N(m_N)$, then $[a^\downarrow[\leqslant_{a^\downarrow}^{-1}(e)]] \in \mathscr{L}_N(m_N)$. Let $e \in E_a$ and assume $[a^\downarrow[<_{a^\downarrow}^{-1}(e)]] \in \mathscr{L}_N(m_N)$. We have

$$\{e' \in A : e' <_{a^\downarrow} e \,\&\, A(e) \cap A(e') \neq \varnothing\} \quad = \quad \{e' \in A : e' <_a e \,\&\, A(e) \cap A(e') \neq \varnothing\}$$

by lemma 4.1, which implies $m + \Delta <_{a^\downarrow}^{-1}(e) = m + \Delta <_a^{-1}(e)$. Then $m + \Delta <_a^{-1}(e) \geqslant e^-$ implies $m + \Delta <_{a^\downarrow}^{-1}(e) \geqslant e^-$.

(3) $\boldsymbol{a}^\downarrow$ is least sequential. Clearly $\boldsymbol{a}^\downarrow \preccurlyeq \boldsymbol{a}$. Suppose some $\boldsymbol{b} \in \preccurlyeq^{-1}(\boldsymbol{a}) \cap \mathscr{L}_N(m_N)$. Then there are bijective pre-homomorphisms $h : E_b \to E_a$ and $g : E_{a^\downarrow} \to E_a$. Let $f : E_{a^\downarrow} \to E_b = h^{-1} \circ g$. We want to show that f is a bijective pre-homomorphism, i. e., $\boldsymbol{a}^\downarrow \preccurlyeq \boldsymbol{b}$. Bijectivity follows from the bijectivity of h and g. Obviously, $\lambda_b = \lambda_{a^\downarrow} \circ f$.

Since f, g, and h are bijective, we must have $g = h \circ f$. Let $e_1, e_2 \in E_{a^\downarrow}$ and assume $e_1 <_{a^\downarrow} e_2$. Then $A(e_1) \cap A(e_2) \neq \varnothing$. $e_1 <_{a^\downarrow} e_2$ implies $g(e_1) <_a g(e_2)$. On the other hand, $A(e_1) = A(f(e_1))$ and $A(e_2) = A(f(e_2))$, which implies $f(e_1) <_b f(e_2)$ or $f(e_2) <_b f(e_1)$ by lemma 4.1. $f(e_2) <_b f(e_2)$ is impossible because this would imply $g(e_1) = h(f(e_2)) <_a h(f(e_1)) = g(e_1)$. We conclude $f(e_1) <_b f(e_2)$. □

DEFINITION 4.4. Let N be a safe Petri net and let $m \in \mathscr{R}_N(m_N)$. We put

$$\mathscr{S}_N(m) \quad =_{\mathrm{df}} \quad \left\{ \boldsymbol{a}^\downarrow \in \mathbf{sw}(T_N) : \boldsymbol{a} \in \mathscr{L}_N(m) \right\}.$$

Furthermore, we define an operation $\odot_N : \mathbf{so}(T_N) \times \mathbf{so}(T_N) \to \mathbf{so}(T_N)$ on semiorders over T_N by $a \odot_N b =_{\mathrm{df}} \left\langle E_a \cup E_b, (<_a \cup <_b \cup R)^+, \lambda_a \cup \lambda_b \right\rangle$, where $E_a \cap E_b = \varnothing$ is assumed. $R \subseteq E_a \times E_b$ is defined by $e \,R\, e' \Leftrightarrow_{\mathrm{df}} A(e) \cap A(e') \neq \varnothing$. For semiwords $\boldsymbol{a}, \boldsymbol{b}$, we put $\boldsymbol{a} \odot_N \boldsymbol{b} =_{\mathrm{df}} [a \odot_N b]$. □

Clearly, $\odot_N$ (on semiwords) is a well-defined operation.

COROLLARY 4.5. *If N is a safe Petri net and $m \in \mathscr{R}_N(m_N)$, then $\preccurlyeq(\mathscr{S}_N(m)) = \mathscr{L}_N(m)$.*

COROLLARY 4.6. *Let N be a safe Petri net and let $\boldsymbol{a} \in \mathscr{S}_N(m)$, $\boldsymbol{b} \in \mathscr{S}_N(m')$ be semiwords such that $m \stackrel{a}{\Longrightarrow} m' \stackrel{b}{\Longrightarrow}$ for markings $m, m' \in \mathscr{R}_N(m_N)$. Then $\boldsymbol{a} \odot_N \boldsymbol{b} \in \mathscr{S}_N(m)$.*

PROOF. We have $\boldsymbol{a} \odot_N \boldsymbol{b} = (\boldsymbol{a} \odot_N \boldsymbol{b})^\downarrow$ due to theorem 4.3. □

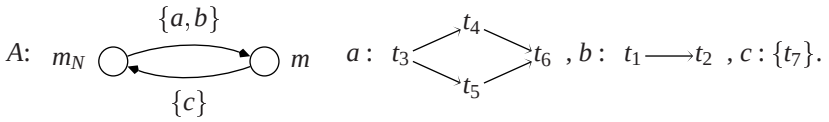

FIGURE 4. A concurrent automaton A to the net in figure 2.

5. Concurrent Automata

DEFINITION 5.1 (Concurrent Automaton). A *concurrent automaton* over an alphabet T is a tuple $A = \langle M, R, \Lambda, m\rangle$ comprising a finite set M of *states*, a *transition relation* $R \subseteq M \times M$, an *arc labelling function* $\Lambda : R \to \mathscr{P}(\mathbf{so}(T))$, and an *initial state* $m \in M$. A *concurrent automaton* of a Petri net N is a concurrent automaton over T_N such that $M \subseteq \mathscr{R}_N(m_N)$ and $m = m_N$ holds. □

Figure 4 shows a concurrent automaton of the Petri net in figure 2. The marking m is given by $m(p_7) = 1$ and $m(p) = 0$ for all places different from p_7.

DEFINITION 5.2 (Semi Language of a Concurrent Automaton). Let A be a concurrent automaton of a Petri net N. A *path* through A is a finite sequence of states $\alpha = m_A m_1 m_1 \dots m_n$ ($n \geqslant 0$) such that $m_A\, R_A\, m_1$, $m_1\, R_A\, m_2, \dots, m_{n-1}\, R_A\, m_n$. Let $P(A)$ denote the set of pathes through A.

If α is a path through A as given above, then $\mathscr{S}_A(\alpha)$ is defined by

$$\begin{aligned} \boldsymbol{a} \in \mathscr{S}_A(\alpha) \quad \Leftrightarrow_{\mathrm{df}} \quad & \boldsymbol{a} = \boldsymbol{a}_1 \odot_N \boldsymbol{a}_2 \odot_N \cdots \odot_N \boldsymbol{a}_n \\ & \& \; a_1 \in \Lambda_A(m_A, m_1) \;\&\; a_2 \in \Lambda_A(m_1, m_2) \;\&\; \cdots \;\&\; a_n \in \Lambda_A(m_{n-1}, m_n) \end{aligned}$$

The *semi language* of A is the set $\mathscr{L}(A) =_{\mathrm{df}} \bigcup_{\alpha \in P(A)} \mathscr{L}(\alpha)$. □

DEFINITION 5.3 (Correctness and Completeness). A concurrent automaton of a Petri net N is called *complete* iff $\mathscr{L}(A) \supseteq \mathscr{S}_N(m_N)$ holds. It is called *correct* iff we have $\mathscr{L}(A) \subseteq \mathscr{S}_N(m_N)$. □

The following lemma states that the reachability graph of a Petri net N can be considered as a concurrent automaton of N.

LEMMA 5.4. *For some safe Petri net N, define A to be a concurrent automaton of N with the components $M_A = \mathscr{R}_N(m_N)$, $m_1 R_A m_2 \Leftrightarrow_{df} \exists t \in T_N\, (m_1 \geqslant t^- \;\&\; m_2 = m_1 + \Delta t)$, $\Lambda_A(m_1, m_2) = \{t \in \mathbf{so}(T_N) : m_1 \geqslant t^- \;\&\; m_2 = m_1 + \Delta t\}$, and $m_A = m_N$. Then A is complete and correct.*

The following lemma is obvious:

LEMMA 5.5 (Preservation of Dead States and Liveness). *Let N be a Petri net and let A be a correct and complete concurrent automaton of N.*

1. *$m \in \mathscr{R}_N(m_N)$ is dead iff $m \in M_A$ and $|R_A(m)| = 0$.*
2. *A transition t is life in N iff for each terminal strongly connected component[1] U of A it holds: $\exists \langle m_1, m_2\rangle \in R_A \cap (U \times U)\, (\exists a \in \Lambda_A(m_1, m_2)\, (t \in \lambda_a(E_a)))$*

6. Algorithm

In this section we discuss a basic algorithm to generate a concurrent automaton A of a safe Petri net N. Algorithm 1 resembles the basic reachability graph construction algorithm. It works as follows: It starts by introducing the initial state $m_A = m_{N^*}$ of A into the set Q, which contains

[1] If $G = \langle V, R\rangle$ is a directed graph with node set V and edge relation $R \subseteq V \times V$, then a *strongly connected component* $U \subseteq V$ is maximal set of nodes such that $v \neq w \Rightarrow v\, R^+\, w \;\&\; w\, R^+\, v$ for all $v, w \in U$. U is called *terminal* iff for $\forall w \in V(\exists v \in U(v\, R^+\, w) \Rightarrow w \in U)$. A *strongly connected component* of a concurrent automaton A is a strongly connected component of the graph $\langle M_A, R_A\rangle$.

algorithm *generate* **is**
  input N, a Petri net;
  output A, a concurrent automaton;
begin
(1) $m_A \leftarrow m_{N^*}$; $M_A \leftarrow \varnothing$; $R_A \leftarrow \varnothing$; $\Lambda_A \leftarrow \varnothing$; $Q \leftarrow \{m_A\}$;
(2) **while** $Q \neq \varnothing$ **do**
(3) select $m \in Q$; $Q \leftarrow Q - \{m\}$;
(4) **foreach** $C \in max\_steps(enabled(m))$ **do**
(5) $a \leftarrow so(C)$; $m' \leftarrow m + \Delta C$; $extend(a, m')$;
(6) **if** $m' \notin M_A$ **then** $Q \leftarrow Q \cup \{m'\}$; $M_A \leftarrow M_A \cup \{m'\}$ **fi**;
(7) $R \leftarrow R \cup \{\langle m, m' \rangle\}$; $\Lambda_A(m, m') \leftarrow \Lambda_A(m, m') \cup \{a\}$
(8) **od**
(9) **od**
end *generate*;

ALGORITHM 1. Concurrent automata generation—basic algorithm.

unprocessed states (the meaning N^* will be explained later—for now, assume $N^* = N$). If a state m is considered (lines 2–9), a set of semiorders enabled at m is generated and appropriate arcs are added to A (lines 4–8). If a new state m' is encountered by the firing of a at m, m' is added to M_A and Q (line 6). The algorithm terminates if all states in Q have been completely processed.

We have to consider the following problems:

1. If m is a state of A already generated, how do we construct an appropriate set of semiorders enabled at m?
2. If a is a semiorder under construction enabled at a state m, do we add another event to a or do we stop extending a and add an arc labelled with a to A?

To solve problem 1, let us discuss the following strategy: For a state m under consideration, let T be the set of enabled transitions at m. Define the *forward conflict relation* $C_f \subseteq T_N \times T_N$ by

$$t_1 \, C_f \, t_2 \quad \Leftrightarrow_{df} \quad F_N^{-1}(t_1) \cap F_N^{-1}(t_2) \neq \varnothing \,\&\, t_1 \neq t_2.$$

Now we generate the set of all maximal steps in T, i. e. the set of cliques[2] in T according to the relation $t_1 \, I \, t_1 \Leftrightarrow_{df} \neg(t_1 \, C_f \, t_2) \,\&\, t_1 \neq t_2$. If C is such a clique, we construct a semiorder a, which contains an event e labelled with t for each $t \in C$ and an empty ordering relation $<_a$. Now, events for each transition t enabled at $m + \Delta E_a$ are added repeatedly, until some termination criterion holds (problem 2).

This strategy fails to yield a complete concurrent automaton for N. Consider the Petri net N in figure 5 and the two concurrent automata A_1 and A_2: At the shown marking m_N, we have the maximal step $C = \{t_1, t_3\}$. Using the strategy described above, the concurrent automaton A_1 would be generated, which is incomplete in opposite to A_2, which is complete.

We therefore propose another strategy: Instead of using the relation I to compute cliques in the set T of enabled transitions at a state m, we compute cliques according to the relation $t_1 \, I_T \, t_2 \Leftrightarrow_{df} t_1 \, I \, t_2 \,\&\, C_f(t_1) \subseteq T \,\&\, C_f(t_2) \subseteq T$.

Now let us discuss problem 2. At first, we suppose N to be extended by an *initialization part*, i. e. if N is a safe Petri net, we define a Petri net N^* with the components $P_{N^*} =_{df} P_N \cup \{p_I\}$ ($p_I \notin P_N$), $T_{N^*} =_{df} T_N \cup \{t_I\}$ ($t_I \notin T_N$), $F_{N^*} =_{df} F_N \cup \{\langle t_I, p \rangle : m_N(p) = 1\} \cup \{\langle p_I, t_I \rangle\}$, $m_{N^*}(p_I) =_{df} 1$ and

[2] For some set T, a *clique* according to a symmetric and irreflexive relation $I \subseteq T \times T$ is a maximal set $C \subseteq T$ such that $t \neq t' \Rightarrow t \, I \, t'$ for all $t, t' \in C$.

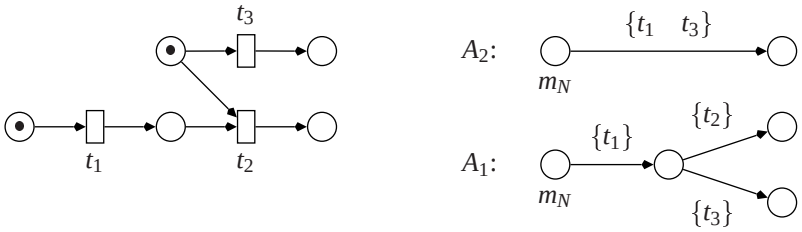

FIGURE 5. A Petri net N and two concurrent automata of N.

$m_{N^*}(p) =_{\mathrm{df}} 0$ for all $p \neq p_I$. Obviously, the extension of N to N^* does not change the behaviour of the net significantly. We have $\mathscr{L}_N(m_N) = \mathscr{L}_{N^*}(m_N)$, and $m_N = m_{N^*} + \Delta t_I$.

Define for some Petri net N the *backward conflict relation* $C_{\mathrm{b}} \subseteq T_N \times T_N$ by

$$t_1 \, C_{\mathrm{b}} \, t_2 \quad \Leftrightarrow_{\mathrm{df}} \quad F_N(t_1) \cap F_N(t_2) \neq \varnothing \,\&\, t_1 \neq t_2.$$

We prove the following property of semiwords $\boldsymbol{a} \in \mathscr{S}_{N^*}(m)$:

THEOREM 6.1. *Let N be a safe Petri net and let $m \in \mathscr{R}_{N^*}(m_N)$. Let $a_0, a_1, a_2 \ldots$ be an infinite sequence of semiorders such that $a_0 = \varepsilon$ and for all $i \geqslant 0$, $a_i \in \mathscr{S}_{N^*}(m)$ and $a_{i+1} = a_i \odot_N t_i$ for some $t_i \in T_N$, i. e. we have $a_i < a_k$ for $i < k$. Then there is some $j \geqslant 0$ such that a_j fulfills the following: for each $e \in E_{a_j}$ there is either some $e' \in E_{a_j}$ such that $e <_{a_j} e'$ and $C_{\mathrm{b}}(\lambda_{a_j}(e')) \neq \varnothing$ or $\leqslant_{a_k}(h_j^k(e)) = \varnothing$ for each $k \geqslant j$, where $h_j^k : E_{a_j} \to E_{a_k}$ is an injective homomorphism.*

PROOF. Choose some a_n ($n \geqslant 0$) as follows: If $e \in \max_{<_{a_n}}(E_{a_n})$, then either $|\leqslant_{a_n}^{-1}(e)| > |T_{N^*}|$ or $\leqslant_{a_k}(h_n^k(e)) = \varnothing$ for each $k \geqslant n$. $\leqslant_{a_k}(h_n^k(e)) = \varnothing$ for each $k \geqslant n$ and for all $e \in \max_{<_{a_n}}(E_{a_n})$ is impossible since we would have $a_n \equiv a_k$, which contradicts $a_n < a_k$ for $n < k$. Hence suppose some $e \in \max_{<_{a_n}}(E_{a_n})$ such that $|\leqslant_{a_n}^{-1}(e)| > |T_{N^*}|$ and $\leqslant_{a_k}(h_n^k(e)) \neq \varnothing$. Then there must be an event $e' \in E_{a_n}$, $e' \neq e$, such that $\lambda_{a_n}(e') = \lambda_{a_n}(e)$, say $\lambda_{a_n}(e) = t$. Since a_n is a semiorder and $e \in \max_{<_{a_n}} E_{a_n}$, we must have $e' <_{a_n} e$. Let $D =_{\mathrm{df}} \{\hat{e} \in E_{a_n} : e' \leqslant_{a_n} \hat{e} \leqslant_{a_n} e\}$ and define $Q =_{\mathrm{df}} \bigcup_{\hat{e} \in D} A(\hat{e})$. It is easy to check that $Q = \varnothing$ implies that N^* and also N are either disconnected or non-safe. $t_I \in \lambda_{a_n}(D)$ is impossible, since $t = t_I$ means that t_I fires twice in a_n, and $t_I = \lambda_{a_n}(\hat{e})$ for some $\hat{e} \in D$ different from e and e' means that $\hat{e}$ has some predecessor $\hat{e}'$ such that $p_I \in A(\hat{e}')$. This implies $p_I \notin Q$.

Suppose $|F_{N^*}^{-1}(p)| \leqslant 1$ for each $p \in Q$, i. e. $C_{\mathrm{b}}(\lambda_{a_n}(\hat{e})) = \varnothing$ for each $\hat{e} \in E$. Then $F_{N^*}^{-1}(Q) = E \subseteq F_{N^*}(Q)$, i. e. Q is a *deadlock* in the sense of Petri net theory.[3] But because of $p_I \notin Q$, we have $\sum_{p \in Q} m(p) = 0$, i. e a_n would not be enabled at m. We conclude $C_{\mathrm{b}}(\lambda_{a_n}(\hat{e})) \neq \varnothing$ for some $\hat{e} \in E$.

Now we are ready to determine the semiorder a_j. Let

$$H \quad =_{\mathrm{df}} \quad \max_{\leqslant_{a_n}} \{e \in E_{a_n} : C_{\mathrm{b}}(\lambda_{a_n}(e)) \neq \varnothing \vee \forall k \geqslant n (\leqslant_{a_k}(h_n^k(e)) = \varnothing)\},$$

and define $a = a_n\left[\leqslant_{a_n}^{-1}(H)\right]$. Then a has the required properties. It remains to show that $a \equiv a_j$ for some $j \geqslant 0$. But clearly, $a \equiv t_0 \odot_N t_1 \odot_N \cdots \odot_n t_j$ for some $j \geqslant 0$ because of theorem 4.3, which implies $a \equiv a_j$. □

Now we have solved problem 2. If a is a semiorder under consideration enabled at a state m of the concurrent automaton which we want to construct, a new event e labelled with some transition t is only added if the following conditions hold:

[3] A *deadlock* of a Petri net N is a non-empty set $Q \subseteq P_N$ of places such that $F_N^{-1}(Q) \subseteq F_N(Q)$. For deadlocks the following holds: if $m, m' \in \mathscr{R}_N(m_N)$ such that $m \xrightarrow{a} m'$ for some semiorder a, then $\sum_{p \in Q} m(p) \geqslant \sum_{p \in Q} m'(p)$.

procedure *extend*(a : **in out so**(T_{N^*}); m : **in out** $\mathscr{P}(P_N) \to \{0,1\}$) **is**
begin
(1) $T \leftarrow addable(a,m)$;
(2) **while** $T \neq \varnothing$ **do**
(3) select $t \in T$; $T \leftarrow T - \{t\}$; $a \leftarrow a \odot_{N^*} t$; $m \leftarrow m + \Delta t$;
(4) $T \leftarrow addable(a,m)$
(5) **od**
end *extend*;

ALGORITHM 2. Concurrent automata generation—procedure *extend*.

T1. $m + \Delta E_a \geqslant t^-$;
T2. $C_{\mathrm{f}}(t) = \varnothing$;
T3. if for some $e \in E_{a\odot_{N^*}t}$ we have $C_{\mathrm{b}}(\lambda_{a\odot_{N^*}t}(e)) \neq \varnothing$, then $e \in \max_{<_{a\odot_{N^*}t}}(E_{a\odot_{N^*}t})$.

Theorem 6.1 makes sure that this procedure finally terminates.

Algorithm 1 makes use of the following subroutines:

1. *enabled*(m) returns the set of enabled transitions at a marking m of N.
2. *max_steps*(T) returns for a transition set T the set of all cliques in T according to I_T.
3. *so*(C) returns a semiorder a with empty ordering for the transition set C, i. e. if $C = \{t_1, t_2, \ldots, t_n\}$, then $a = \langle \{1,2,\ldots,n\}, \varnothing, \{\langle i, t_i\rangle : 1 \leqslant i \leqslant n\}\rangle$
4. *extend*(a,m) is shown in fig 2. It extends a semiorder a computed by *so*(C) as large as possible and simultaniously updates the marking m to $m + \Delta E_a$.
5. *addable*(a,m) returns a set T of transitions such conditions T1, T2, and T3 are satisfied for each $t \in T$.

THEOREM 6.2. *For each safe Petri net N, algorithm 1 terminates.*

PROOF. The termination of algorithm 2 follows from theorem 6.1. But clearly, a safe Petri net has only finitely many reachable markings, namely $|\mathscr{R}_N(m_N)| \leqslant 2^{|P_N|}$. □

THEOREM 6.3. *Let A be a concurrent automaton of a safe Petri net N generated by algorithm 1. Then A is correct.*

PROOF. For each pair $\langle m, m'\rangle \in R_A$ and for all $a \in \Lambda_A(m,m')$ we have $m \overset{a}{\Longrightarrow} m'$ by corollary 4.6. Now the theorem follows by a simple induction on the length n of a path $\alpha = m_A m_1 \ldots m_n$ through A. □

THEOREM 6.4. *Let A be a concurrent automaton of a safe Petri net N generated by algorithm 1. Then A is complete.*

PROOF. Assume $\mathscr{S}_{N^*}(m_{N^*}) - \mathscr{L}(A) \neq \varnothing$. Let $\boldsymbol{a} \in \min_{\leqslant}(\mathscr{S}_{N^*}(m_{N^*}) - \mathscr{L}(A))$. Choose some $\boldsymbol{b} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{a}) \cap \mathscr{L}(A))$. Then neither $\boldsymbol{a} = \varepsilon$ nor $\boldsymbol{b} = \varepsilon$, since $\boldsymbol{t}_I \leqslant \boldsymbol{c}$ for each $\boldsymbol{c} \in \mathscr{L}(A) \cup \mathscr{S}_{N^*}(m_{N^*})$. Because of $\boldsymbol{b} \in \mathscr{L}(A)$, there is a path $\alpha = m_A m_1 m_2 \ldots m_n$ through A such that there is some $\boldsymbol{c} \in \mathscr{L}(\alpha)$ with $\boldsymbol{c} \leqslant \boldsymbol{b}$. Put α to be of maximal length such that $\boldsymbol{c} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{b}) \cap \mathscr{L}(A))$.

Now let $\boldsymbol{b} = \boldsymbol{c} \odot_{N^*} \boldsymbol{d}$. and $\boldsymbol{a} = \boldsymbol{c} \odot_{N^*} \boldsymbol{d}'$. $\boldsymbol{d} = \varepsilon = \boldsymbol{d}'$ is impossible since this would imply $\boldsymbol{a} = \boldsymbol{b}$ in contradiction to $\boldsymbol{a} \notin \mathscr{L}(A)$. Suppose $\boldsymbol{d} = \varepsilon$ and $\boldsymbol{d}' \neq \varepsilon$. Let $C = \lambda_{d'}(\min_{\leqslant_{d'}}(E_{d'}))$. Then C is an enabled step at m_n, and therefore there is a step $C' \subseteq T_{N^*}$ generated by the procedure *max_steps* with $C' \cap C \neq \varnothing$. This implies $[c \odot_{N^*} so(C')] \in \mathscr{L}(A)$, and also $[c \odot_{N^*} so(C' \cap C)] \in \mathscr{L}(A)$ Then $\boldsymbol{b} \leqslant [c \odot_{N^*} so(C' \cap C)] \leqslant \boldsymbol{a}$, which contradicts $\boldsymbol{b} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{a}) \cap \mathscr{L}(A))$.

Hence $\boldsymbol{d} \doteq \varepsilon$ and $\boldsymbol{d}' \doteq \varepsilon$. We have $\boldsymbol{d} \leqslant \boldsymbol{d}'$. Let $m = m_n + \Delta E_d$. Note that m cannot be in M_A, since otherwise the maximality of the path α would be contradicted. Put $\tilde{\boldsymbol{d}}$ such that $\boldsymbol{a} = \boldsymbol{c} \odot_{N^*} \boldsymbol{d} \odot_{N^*} \tilde{\boldsymbol{d}}$. $\tilde{\boldsymbol{d}} = \varepsilon$ is impossible since this would imply $\boldsymbol{a} = \boldsymbol{b}$.

Let $F = \{\boldsymbol{f} \in \mathbf{sw}(T_{N^*}) : \boldsymbol{f} \doteq \varepsilon \,\&\, \boldsymbol{b} \odot_{N^*} \boldsymbol{f} \in \mathscr{L}(A)\}$ be the set of continuations of $\boldsymbol{b}$ in $\mathscr{L}(A)$. Note that each $\boldsymbol{f} \in F$ is enabled at m. $F \doteq \varnothing$, since otherwise m would be a dead state. Therefore, $m \in M_A$ by lemma 5.5, which contradicts the maximality of α. $\boldsymbol{a} \in \min_{\leqslant}(\mathscr{S}_{N^*}(m_{N^*}) - \mathscr{L}(A))$ implies $\lambda_a(\min_{\leqslant_a}(E_a)) \cap \lambda_f(\min_{\leqslant_f}(E_f)) = \varnothing$ for all $\boldsymbol{f} \in F$. Let $t \in \lambda_{\tilde{d}}(\min_{\leqslant_{\tilde{d}}}(E_{\tilde{d}}))$. Assume $C_{\mathrm{f}}(t) = \varnothing$. If $C_{\mathrm{b}}(t) = \varnothing$ or $C_{\mathrm{b}}(t) \doteq \varnothing$ such that condition T3 is satisfied for $[d \odot_{N^*} t]$, then t could be added to d by line 3 of algorithm 2. Then $[b \odot_{N^*} t] \in \mathscr{S}_A(m_A)$ in contradiction to $\boldsymbol{b} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{a}) \cap \mathscr{L}(A))$. On the other hand, if $C_{\mathrm{b}}(t) \doteq \varnothing$ and condition T3 is not satisfied, then d wound be extended by algorithm 2 to some semiorder d^*, and a new arc $\langle m_n, m^* \rangle$ with $m^* = m_n + \Delta E_{d^*}$ will be added to A such that $d^* \in \Lambda_A(m_n, m^*)$. Since algorithm 2 adds only transitions with empty forward conflict relation to d^*, we have that $m \leqslant t^-$ implies $m^* \leqslant t^-$. Then, a step C containing t would be constructed at m^* in line 4 of algorithm 1, i. e $[d \odot_{N^*} t] \leqslant [d^* \odot_{N^*} t] \in \mathscr{S}_{N^*}(m_{N^*})$. This implies $[b \odot_{N^*} t] \in \mathscr{S}_{N^*}(m_{N^*})$, a contradiction to $\boldsymbol{b} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{a}) \cap \mathscr{L}(A))$.

We conclude $C_{\mathrm{f}}(t) \doteq \varnothing$. Repeating the argument above, d will be extended to d^* by algorithm 2, and an arc $\langle m_n, m^* \rangle$ such that $m^* = m_n + \Delta E_{d^*}$ and $d^* \in \Lambda_A(m_n, m^*)$ will be introduced. Since algorithm 2 adds only transitions with empty forward conflict relation to d^*, we have that $m \leqslant t^-$ implies $m^* \leqslant t^-$. Then, a step C containing t would be constructed at m^* in line 4 of algorithm 1, i. e $[d \odot_{N^*} t] \leqslant [d^* \odot_{N^*} t] \in \mathscr{S}_{N^*}(m_{N^*})$. This implies $[b \odot_{N^*} t] \in \mathscr{S}_{N^*}(m_{N^*})$. Again, $\boldsymbol{b} \in \max_{\leqslant}(\leqslant^{-1}(\boldsymbol{a}) \cap \mathscr{L}(A))$ would be contradicted. This concludes the proof. □

A *conflict cluster* of a Petri net N is a maximal set $D \subseteq T_N$ such that $t \doteq t' \Rightarrow t \, C_{\mathrm{f}} \, t$ for all $t, t' \in D$ holds. Let $\mathscr{C}_N$ be denote the set of conflict clusters of N.

THEOREM 6.5. *The time effort of the computation of a concurrent automaton A for a Petri net N by algorithm 1 is* $O\left(2^{|P_{N^*}|} \times \left(2^k + |T_N|\right)\right)$, *where* $k = \max\{|D| : D \in \mathscr{C}_{N^*}\}$.

PROOF. The first factor, $2^{|P_{N^*}|}$, is simply the maximal number of reachable states of N^*. $2^k + |T_{N^*}|$ is obtained as follows: For each reachable marking m of N^*, the procedure *max_steps* computes at most $c \times 2^k$ steps for some constant $c > 0$. (examples where equality holds can be easily constructed). For each of this steps, at most $|T_{N^*}|$ events are added by algorithm 2, hence the time effort to process a state completely is $O(2^k + |T_N|)$. □

7. Summary and Further Works

We have introduced the formalism of a concurrent automaton for a safe Petri net. The use of concurrent automata instead of reachability graphs has the benefit that concurrency of transition does not necessarily lead to the state explosion problem. Moreover, since global states (markings) are maintained, it is possible to rejoin branching behaviours of N. In some cases, this will yield a smaller representation of the behaviour of N than the prefix.

A basic algorithm for the generation of a concurrent automaton has been presented. It has been shown, that the concurrent automaton constructed by this algorithm is correct and complete w. r. t. semi language equivalence in the sense of section 5.

However, it is unlikely that this basic algorithm has acceptable run-times for other then small Petri net examples. An inspection of some medium-sized Petri nets [**7, 8**] leads to the observation, that almost every transition t in these examples is involved in a forward conflict ($C_{\mathrm{f}}(t) \doteq \varnothing$) or a backward conflict ($C_{\mathrm{b}}(t) \doteq \varnothing$), i. e. algorithm 1 will be in practice not significantly faster then usual reachability graph generation.

Concentrating on forward conflicts, we may improve algorithm 1 in the following way: Instead of using the relation I_T to construct steps by the procedure *max_steps*, we may use the relation J_T defined by $t_1\, J_T\, t_2 \Leftrightarrow_{\mathrm{df}} t_1\, I_T\, t_2\ \&\ \exists m \in \mathscr{R}_N(m_N)(m \geqslant t_1\ \&\ m \geqslant t_2)$. In practice, J_T cannot be determined without an exhaustive net analysis such as reachability graph generation or prefix generation. However, any relation I' such that $J_T \subseteq I' \subseteq I_T$ will also do the job.

Then the problem can be restated as follows: Let t_1 and t_2 be transitions of a Petri net N such that $t_1\, C_{\mathrm{f}}\, t_2$ holds. If for every marking $m \in \mathscr{R}_N(m_N)$ there is some place p such that $m(p) = 0$ and $(t_1^- + t_2^-)(p) > 0$, then put $t_1\, I'\, t_2$, i. e. we have to compute whether a marking m with $m \geqslant t_1^-$ and $m \geqslant t_2^-$ is unreachable in N.

There are several methods of classical Petri net theory to prove the unreachability of a marking m, for instance net analysis be means of place invariants, the state equation (see [**10**] for details), or by traps or deadlocks.

Another question is which net properties can be determined using concurrent automata. Lemma 5.5 states that the existence or absence of dead states can be proved by inspection of a concurrent automaton very easily. The same applies to liveness of transitions. Techniques for more sophisticated analysis goals are in preparation. Especially, it seems to be likely that it is possible to determine so-called *partial order properties* by means of concurrent automata, i. e those properties related to the concurrency of transitions.

References

1. E. Best and C. Fernández, *Nonsequential processes*, EATCS, vol. 13, Springer, 1988.
2. P. Deussen, *Concurrent automata*, Tech. Report 1-05/1998, Brandenburg Techn. Univ. Cottbus, 1998, to appear.
3. J. Engelfriet, *Branching processes of Petri nets*, Acta Inf. **25** (1991), pp. 575–591.
4. J. Esparza, *Model checking using net unfoldings*, Science of Computer Programming **23** (1994), pp. 151–195.
5. J. Esparza, S. Römer, and W. Vogler, *An improvement of McMillan's unfolding algorithm*, Tech. Report SFB-Report 342/12/95 A, Techn. Univ. of München, 1995.
6. J. Grabowski, *On partial languages*, Fund. Inform **4** (1981), no. 2, pp. 427–498.
7. M. Heiner, *Verifaction and optimization of control programs by Petri nets without state explosion*, Proc. 2nd Int. Workshop on Manufacturing and Petri Nets held at Int. Conf. on Application and Theory of Petri Nets (ICATPN '97) (1997), pp. 69–84.
8. M. Heiner, P. Deussen, and S. Spranger, *A case study in design and verification of manufacturing system control software with hierarchical Petri nets*, The Int. Journal of Advanced Manufacturing Technology, special issue on Petri Net Applications in Advanced Manufacturing (1998), to appear.
9. K. L. McMillan, *Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits*, Proc. of the 4th Workshop on Computer Aided Verification (Montreal), 1992, pp. 164–174.
10. S. Melzer and J. Esparza, *Verification of system properties via integer programming*, Programming Languages and Systems—ESOP'96 (H.R. Nielson, ed.), LNCS, vol. 1058, Springer-Verlag, 1996, pp. 250–264.
11. M. Nielsen, G. Plotkin, and G. Winskel, *Petri nets, event structures and domains, Part I*, Theoretical Computer Science **13** (1981), pp. 85–108.
12. P. H. Starke, *Processes in Petri nets*, J. Inf. Process. Cybern. EIK **17** (1981), no. 8/9, pp. 389–416.
13. ———, *Graph grammars for Petri net processes*, J. Inf. Process. Cybern. EIK **19** (1983), no. 4/5, pp. 199–233.
14. ———, *Analyse von Petri-Netz-Modellen*, G. B. Teubner, Stuttgart, 1990.
15. A. Ulrich, *A description model to support test suite derivation for concurrent systems*, Tech. Report I-06/1996, Brandenburg Techn. Univ. Cottbus, 1996.
16. ———, *A description model to support test suite derivation for concurrent systems*, Kommunikation in verteilten Systemen, GI/ITG-Fachtagung (KiVS'97), Springer-Verlag, 1997, pp. 151–166.
17. W. Vogler, *Modular construction and partial order semantics of Petri nets*, LNCS, vol. 625, Springer-Verlag, 1992.