An Efficient Polynomial-Time Algorithm to Decide Liveness and Boundedness of Free-Choice Nets

Peter Kemper and Falko Bause

Informatik IV
Universität Dortmund
Postfach 500 500
4600 Dortmund 50
Germany

Abstract. In [3] J. Esparza presented an interesting characterization of structurally live and structurally bounded Free-Choice Nets (LBFC-Nets). Exploiting this characterization in combination with new results and refined algorithms the authors formulate an $O(|P||T||F|)$ algorithm deciding whether a Free-Choice Net is a LBFC-Net or not. Furthermore the algorithm contains a simple and efficient test to ensure that the initial marking of a LBFC-Net is live. This test is based on a simplified characterization of liveness for LBFC-Nets.

1 Introduction

Petri Nets have been successfully used for modelling discrete concurrent systems. Petri Net theory offers a variety of analysis algorithms, which often have exponential worst case time complexity as far as liveness and boundedness of a Petri Net is concerned. Polynomial-time algorithms for examination of liveness and boundedness do exist for certain subclasses obeying restrictions on the net's structure. The border between polynomial-time and exponential-time algorithms seems to lead directly through the class of Free-Choice Nets (FC-Nets). In [6] it is proven that the liveness problem for FC-Nets is Co-NP-complete while in [4] a polynomial-time algorithm determining liveness of bounded FC-Nets is exhibited. This algorithm is based on a linear algebraic characterization of deadlocks and traps given in [7] employing linear programming techniques. Its worst case time complexity is $O(|P||T||F|^{6})$. In this paper an algorithm deciding whether a Free-Choice-Net is structurally live and structurally bounded in $O(|P||T||F|)$ is described. It is based on a characterization of LBFC-Nets given in [3] and employs an efficient graph-orientated algorithm to recognize State-Machine-Decomposable Nets (SMD-Nets). State-Machine-Decomposability can be checked efficiently due to the close relationship between S-components and minimal deadlocks in LBFC-Nets.

This article is structured as follows: Sect. 2 contains basic definitions. An outline of the complete algorithm and its theoretical background is presented in Sect. 3. Sections 4 - 7 contain algorithms for certain subproblems left open in the outline of the algorithm. Section 4 summarizes an algorithm from [2] computing a minimal deadlock, which contains a certain place p of a strongly connected FC-Net. A question left open in this algorithm is solved by the algorithm described in Sect. 5, which altogether leads to an efficient algorithm to compute a minimal deadlock for a certain place p. Section 6 contains an algorithm which checks a minimal deadlock for generating an S-component. In Sect. 7 we exploit our simplified characterization of liveness for LBFC-Nets to describe a simple and efficient algorithm testing liveness of an initial marking. Finally the algorithms of all subproblems are employed for the formulation of the complete algorithm. This section also contains the calculation of the algorithm's time complexity.

2 Basic Definitions

Definition 1 Net.
A net is a triple $N = (P, T, F)$ where

1. P and T are non-empty, finite sets and $|P| = n$, $|T| = m$,
2. $P \cap T = \emptyset$,
3. $F \subseteq (P \times T) \cup (T \times P)$.

The next definition summarizes some well known notions for nets.

Definition 2.
Let $N = (P, T, F)$ be a net.

1. Let $x \in P \cup T$. The preset $\bullet x$ and postset $x\bullet$ are given by
$\bullet x := \{y \in P \cup T | (y, x) \in F\}$, $x\bullet := \{y \in P \cup T | (x, y) \in F\}$
The preset (postset) of a set of nodes is the union of presets (postsets) of its elements.
2. W denotes the characteristic function of F. The incidence matrix C of N is given by $c_{ij} := W(t_j, p_i) - W(p_i, t_j)$. Rank(C) denotes the rank of the incidence matrix.
3. A path of a net N is a sequence $(x_1, \ldots, x_n)$ of nodes $x_i \in P \cup T, i \in \{1, \ldots, n\}$, such that $(x_i, x_{i+1}) \in F$ for all $i \in \{1, \ldots, n-1\}$.
4. N is a S-graph iff $\forall t \in T : |\bullet t| = |t \bullet| = 1$.
5. N is a Free-Choice Net (FC-Net) iff $\forall p \in P : |p \bullet| > 1 \Rightarrow \bullet(p\bullet) = \{p\}$.
6. A net $N' = (P', T', F')$ is a subnet of N, $(N' \subseteq N)$, iff $P' \subseteq P, T' \subseteq T$ and $F' = F \cap ((S' \times T') \cup (T' \times S'))$.
7. The subnet $N' = (P', T', F')$ of N generated by $P' \subseteq P$ is given by $T' = \bullet P' \cup P'\bullet$ and $F' = F \cap ((P' \times T') \cup (T' \times P'))$.
8. $N' \subseteq N$ is an S-component of N iff N' is a strongly connected S-graph and $T' = \bullet S' \cup S'\bullet$.
9. N is State Machine Decomposable (SMD) iff it is covered by S-components, i.e. every node belongs to an S-component of N.
10. $P' \subseteq P$ is a deadlock of N iff $P' \neq \emptyset$ and $\bullet P \subseteq P\bullet$. A deadlock is minimal iff it does not contain a deadlock as a proper subset. A deadlock P' is strongly connected iff $N' = (P', \bullet P', F')$ with $F' = F \cap ((P' \times \bullet P') \cup (\bullet P' \times P'))$ is strongly connected.
11. $P' \subseteq P$ is a trap of N iff $P' \neq \emptyset$ and $P\bullet \subseteq \bullet P$. A trap is minimal iff it does not contain a trap as a proper subset.

Definition 3 Place/Transition Nets, Petri Nets and related definitions. A function $M : P \longrightarrow \mathbb{N}$ is called marking. A Place/Transition Net or Petri Net is a tuple (N, M_0) where N is a net and M_0 is a marking named initial marking.

1. $P' \subseteq P$ is marked at M iff $\exists p \in P' : M(p) > 0$.
2. $t \in T$ is enabled at M iff $\forall p \in \bullet t : M(p) > 0$.
3. A transition t enabled at M can fire and thereby create a new marking M' given by $M'(p) := M(p) + W(p,t) - W(t,p)\ \forall p \in P$. This is denoted by $M[t > M'$ and M' is called reachable from M.
4. The reflexive and transitive closure of reachable markings from marking M_0 for (N, M_0) is called reachability set and denoted by $R(N, M_0)$.
5. (N, M_0) is bounded iff $\exists k \in \mathbb{N} : \forall p \in P, M \in R(N, M_0) : M(p) \leq k$. (N, M_0) is safe iff it is bounded with $k = 1$.
6. N is structurally bounded iff $\forall M_0 \in [P \longrightarrow \mathbb{N}] : \exists k \in \mathbb{N} : (N, M_0)$ is bounded.
7. (N, M_0) is live iff $\forall t \in T, M \in R(N, M_0) : \exists M' \in R(N, M)$: t is enabled at M'.
8. N is structurally live iff $\exists M_0 \in [P \longrightarrow \mathbb{N}] : (N, M_0)$ is live.

Structurally live and structurally bounded FC-Nets are denoted by LBFC-Nets.

3 Deciding Liveness and Boundedness of FC-Nets

This section presents an outline of a new algorithm to decide liveness and boundedness of FC-Nets. In [6] it is shown that the liveness problem for FC-Nets is Co-NP-complete and in [4] a polynomial-time algorithm to decide liveness for bounded FC-Nets is presented. Our new algorithm does not decide liveness independently from boundedness. It checks whether the net satisfies both properties, liveness and boundedness, or not. The well known fact that all live and bounded Petri nets are strongly connected can be exploited as a precondition.

Theorem 4 [9].
Let (N, M_0) be a Petri Net.
(N, M_0) is live and bounded $\Rightarrow$N is strongly connected.

In the following we regard strongly connected nets. The new algorithm is based on the following characterization:

Theorem 5 Characterization of LBFC-Nets [3].
Let $N = (P, T, F)$ be a FC-Net with incidence matrix C and $a = |F \cap (P \times T)|$. N is structurally live and structurally bounded iff N is SMD and rank$(C) = |P| + |T| - a - 1$. (a is the number of arcs leading from a place to a transition.)

Obviously LBFC-Nets are structurally live and SMD. In [5] it is proven that a live and safe initial marking exists for a structurally live SMD-FC-Net. FC-Nets with a live and safe initial marking are denoted by LSFC-Nets. The following theorems taken from [1] hold for LSFC-Nets as well as for LBFC-Nets.

Theorem 6 [1].
Let $N = (P, T, F)$ be a LSFC-Net (LBFC-Net).
$D \subseteq P$ is a minimal deadlock $\Rightarrow$D generates an S-component.

Theorem 7 [1].
Let $N = (P, T, F)$ be a LSFC-Net (LBFC-Net).
$D \subseteq P$ is a minimal deadlock $\Rightarrow D$ is a minimal trap.

Let us consider Theorems 5 and 6 first. Checking the condition $rank(C) = |P| + |T| - a - 1$ in Theorem 5 is trivial due to well known algorithms with tolerable time complexity, cf. [10]. The crux of this characterization is to decide the SMD property of a FC-Net efficiently. Naturally this problem can be solved by searching one S-component containing a place p for any $p \in P$. Theorem 6 implies that searching for a minimal deadlock will result in an S-component for LBFC-Nets. J. Esparza gave in [2] an outline of an algorithm to find a minimal deadlock in a strongly connected FC-Net containing a certain place. This algorithm is summarized in Sect. 4. If a minimal deadlock not generating an S-component is found, N is not structurally live and structurally bounded. Obviously at most $|P|$ minimal deadlocks have to be found and checked for generating an S-component in order to decide the SMD property for N.

If a net is a LBFC-Net it is still left open whether a given initial marking is a live marking. J. Esparza suggested to check this by solving a linear programming problem, cf. [3], Proposition 4.3. We suggest to check this property by exploiting a simplified characterization of liveness for LBFC-Nets which follows from Theorem 7 and

Theorem 8 [5].
Let $N = (P, T, F)$ be a FC-Net with an initial marking M_0.
(N,M_0) is live iff every minimal deadlock contains a marked trap.

Theorems 7 and 8 allow the following

Conclusion 9 Simplified characterization of liveness for LBFC-Nets.
Let $N = (P, T, F)$ be a LBFC-Net. M_0 is a live marking iff all minimal deadlocks are marked at M_0.

In order to check liveness for a given initial marking of a LBFC-Net we simply try to find a deadlock which is a subset of the set of unmarked places.

All these theorems give reason for the following outline of a new algorithm:

Input (N, M_0) and N is a FC-Net
Output *Yes* N is LBFC and M_0 is a live initial marking
No otherwise
step1 Check the net for being strongly connected.
If the net is not strongly connected Stop with *No* due to Theorem 4.
step2 For all places p find an S-component which contains p by
1. finding a minimal deadlock D containing p.
   Such a minimal deadlock exists due to an algorithm in [2].
2. checking D for generating an S-component.
   If D does not generate a subnet being an S-component Stop with *No* due to Theorem 6.

After successful completion of step2 all places are covered by at least one S-component and the net is SMD.

step3 Check $rank(C) = m + n - a - 1$.

If this condition is not satisfied Stop with *No* due to Theorem 5. Otherwise the net is LBFC-Net and the liveness of the initial marking has to be checked in step4.

step4 Check existence of an unmarked deadlock.

If an unmarked deadlock exists Stop with *No* due to Theorem 8. Otherwise the net is marked live due to Conclusion 9.

In the following sections we present efficient algorithms for these steps apart from steps 1 and 3, because efficient algorithms are well known for the computation of the strongly connected components of a directed graph and for the calculation of a matrix rank, see [8] and [10] for example.

4 Calculation of Minimal Deadlocks in FC-Nets

This section summarizes an algorithm described in [2] computing a minimal deadlock in a strongly connected FC-Net containing a given place p. The following characterization of minimal deadlocks in FC-Nets is exploited for this algorithm.

Theorem 10 [2].
Let $N = (P, T, F)$ be a FC-Net and $D \subseteq P$ a deadlock in N.
D is minimal iff D is strongly connected and $\forall t \in \bullet D : |\bullet t \cap D| = 1$.

The central idea for searching a minimal deadlock is to find a handle for any place of the minimal deadlock (starting with p) that has an input transition not being an output transition.

Definition 11 Handle.
Let $N = (P, T, F)$ be a net with two non-empty sets $S, S' \subseteq P \cup T$, $S \cup S' = P \cup T$ and $S \cap S' = \emptyset$. A path $H = (x_0, x_1, \ldots, x_{n-1}, x_n)$ in N is a handle iff $x_0, x_n \in S$, $x_1, \ldots, x_{n-1} \in S'$, $(x_i, x_{i+1}) \in F$, $\forall i \in \{0, \ldots, n-1\}$ and furthermore $x_i \neq x_j$, $\forall i, j \in \{1, \ldots, n-1\}, i \neq j$.

Given two such sets S, S' a handle always exists for strongly connected nets. The following algorithm computes minimal deadlocks by searching handles for all places belonging to the minimal deadlock in demand. The set $\hat{\text{P}}$ of places belonging to the minimal deadlock is initiated with $\{p\}$ and successively all places of the computed handles are inserted into $\hat{\text{P}}$. The algorithm terminates if all places in $\hat{\text{P}}$ have handles covering all of their input transitions.

Algorithm 12. get-minimal-deadlock(P,T,F,p,T_D), [2]

Input:
$N = (P, T, F)$ strongly connected FC-Net with $p \in P$.

Output:
1. Minimal deadlock $D \subseteq P$ containing p and
2. Set T_D which is $\bullet D$

Initiate:
$\hat{P} = \{p\}$; $\hat{T} = \emptyset$;

Function: get-handle(S,S',F,p,t), cf. Sect. 5
This function computes a handle $(x_0, x_1, \ldots, x_{n-2}, t, p)$ with $x_0, p \in S$
and $x_1, \ldots, x_{n-2}, t \in$ S'. It returns the handle as a set $\{x_0, x_1, \ldots, x_{n-2}, t, p\}$.

Program:
begin
    while $(\exists p' \in \hat{P} : \ \exists t \in \bullet p' : \ t \notin \hat{T})$
    begin
        H := get-handle$((\hat{P} \cup \hat{T}), (P \cup T) - (\hat{P} \cup \hat{T}), F, p', t)$;
        $\hat{P} := \hat{P} \cup (H \cap P)$;
        $\hat{T} := \hat{T} \cup (H \cap T)$;
    end
    D := $\hat{\mathsf{P}}$; $T_D := \hat{T}$;
end

The correctness of this algorithm follows straight forward from the following four properties holding at every stage of the algorithm:

1. $\hat{N} = (\hat{P}, \hat{T}, \hat{F})$ with $\hat{F} = F \cap ((\hat{P} \times \hat{T}) \cup (\hat{T} \times \hat{P}))$ is a subnet of N.
2. $\hat{N}$ is strongly connected in terms of $\hat{F}$.
3. Every transition in $\hat{T}$ has exactly one incoming $\hat{F}$-arc.
4. After termination of the algorithm, if $\hat{p} \in \hat{P}$ then all incoming arcs of $\hat{p}$ in F are also in $\hat{F}$. Thus $\bullet\hat{p} \subseteq \hat{T}$.

These properties ensure that output D is a minimal deadlock due to Theorem 10. For details proving correctness see [2].

The time complexity of Algorithm 12 is clearly dominated by the effort for function get-handle. The computation of a minimal deadlock cannot cause more than $|T|$ get-handle calls. In [2] it is left open, how to compute a handle and therefore no time complexity of Algorithm 12 is presented there. We show in Sect. 5, that the computation of a single handle is possible in $O(|P|+|T|+|F|)$. This results in a worst case time complexity for the computation of a minimal deadlock in $O(|T|(|P|+|T|+|F|))$.

5 Computing a Handle in $O(|P| + |T| + |F|)$

In this section we suggest an algorithm that solves the open problem in Algorithm 12. The precise problem is:

Definition 13 Compute handle $(x_0, \ldots x_{n-2}, t, p)$**.**
Let $N = (P, T, F)$ be a net and S, S' two non-empty sets with the property $S \subseteq P \cup T$ and $S' = (P \cup T) - S$. Let $p \in S$ and $t \in \bullet p, t \in S'$. Compute a handle $H = (x_0, \ldots, x_n)$ satisfying $x_0 \in S$, $x_n = p$, $x_{n-1} = t$, $x_i \in S'$ and $x_i \neq x_j$, $\forall i, j \in \{1, \ldots, n-1\}, i \neq j$.

The handle, we are looking for, is a path of the net starting at an arbitrary node[1] in S and ending at place p. All other nodes on this path are elements of S' and appear exactly once. From an algorithmic point of view the search of a handle obviously starts at the fixed end and follows possible paths backwards (!) in the net. The new algorithm suggested here follows Depth-First-Search (DFS) to visit all nodes reachable from a start node. Within this search a path is followed in depth as far as possible. Each visited node gets an individual DFS-number num. This number exhibits the node's type. Assume a node v is reached by the DFS-algorithm, the numbers assigned to v are explained in the following:

1. num(v) = -1
   v belongs to S and is a node the handle can start from, thus search of a handle terminates at v. For all nodes in S the number -1 is assigned as an initial value and is never changed.
2. num(v) = 0
   v belongs to S' and has not been visited so far. This node is a candidate for a node of the handle we are looking for and search will continue with this node. Zero is the initial value for all nodes in S' and num(v) is set to a value greater than zero as soon as a node is reached by DFS-search.
3. num(v) > 0
   v belongs to S' and has been visited before. num(v) represents two different situations leading to the same consequence:
   (a) v was completely checked before and no handle was found. Thus it is not necessary to check v again.
   (b) v has been checked and is reached again. Because a handle does contain a node of S' exactly once, this node cannot be part of a handle.

   Altogether this node is not checked furthermore and search has to continue with the predecessor node of v w.r.t. DFS.

Application of these assignments to all nodes leads to the following algorithm searching a handle backwards in the net.

Algorithm 14. get-handle(S, S', F, p, t)

Input:
$N = (P, T, F)$ is a strongly connected FC-Net, with $S, S' \subseteq P \cup T$
$S \cup S' = P \cup T$, $S \cap S' = \emptyset$, $p \in S$, $t \in S'$ and $t \in \bullet p$, cf. Definition 13

Output:
Handle $H = (x_0, \ldots, x_{n-2}, t, p)$ or
Message "No handle exists"

Initiate:
i := 1 ;
Stack := empty-stack ;
num(x) := 0, $\forall x \in S'$;
num(x) := -1, $\forall x \in S$;

[1] The node x_0 must be a place in order to find a deadlock being an S-component.

```
Function: dfs(v)
begin
    num(v) := i ; i := i + 1 ; push(Stack,v) ;
    forall ( w ∈ •v )
    begin
        if (num(w) = -1 )                { start node of the handle }
        then push(Stack,w) ; return Yes ;
    end
    forall ( w ∈ •v )
    begin
        if (num(w) = 0 )                 { a new node is reached }
        then if (dfs(w) = Yes )
            then return Yes ;
    end
    pop(Stack,v) ;
    return No ;
end

Program:
begin
    push(Stack,p) ;
    if (dfs(t) = No )
    then Stop with Message "No handle exists" ;
    else Stop with Output Stack ;
end
```

Remarks:
The stack is used to store nodes which might belong to a handle. First p is pushed on the stack and not popped until termination, because p is surely member of a handle, if a handle exists at all. A node v is pushed on the stack if dfs(v) is called initiating a depth-first-search from v. If this search is successful, v is not popped at the end of dfs(v), because dfs(v) returns with Yes. If no handle is found on the search starting at v, v is not element of a handle, which might be found lateron and is therefore popped at the end of dfs(v) returning No.

Theorem 15. *Algorithm 14 terminates and is correct.*

Proof Termination. The forall loops terminate due to the finiteness of all sets. The recursion in dfs terminates because of two effects:

1. A call of function dfs with parameter x can only occur if num(x) = 0.
2. Within dfs(x) num(x) is set to a value greater than 0 and is not changed anywhere else in the algorithm.

Thus dfs(x) can only be called once for any $x \in P \cup T$. □

Proof Correctness. Because of termination dfs(t) returns either Yes or No.
The algorithm is correct iff
1. dfs(t) = No $\Longrightarrow$ no handle exists.
2. dfs(t) = Yes $\Longrightarrow$ the stack contains a handle.
at 1)
Assumption: dfs(t) = No and handle $(x_0, x_1, \ldots, x_{n-2}, t, p)$ exists.
dfs(t) = No $\Rightarrow$all dfs calls have returned with No
$\Rightarrow$num$(x_1) = 0$
$\Rightarrow \forall v \in x_1\bullet$: num(v) = 0
$\Rightarrow$num$(x_2) = 0$
$\Rightarrow$This argumentation can be continued successively yielding num$(x_{n-2}) = 0$
$\Rightarrow$num(t) = 0, which is a contradiction due to the explicit call for dfs(t) setting num(t) = 1.
at 2)
If dfs(t) terminates with Yes, the stack contains a handle $(x_0, x_1, \ldots, x_{n-2}, t, p)$, because
1. $x_0, p \in S$ and $x_1, \ldots, x_{n-2}, t \in S'$ by construction
2. no node of S' occurs twice on the stack,
because any $v \in S'$ on the stack must have been pushed in dfs(v) and dfs is only called once with parameter v (cf. proof of termination).
3. $(w, v) \in F$, because dfs(w) occurs only if $w \in \bullet v$.

□

Example 1. The sequence of Figs. 1 - 5 show a LBFC-Net and a possible computation of a handle for place a. $S = \{a\}$ and $S' = P - \{a\}$. Letters are identifying different nodes and numbers attached to the nodes represent their actual value of `num`. Figure 1 displays the situation starting with dfs(b). Following the incoming arcs with depth-first-search, node b is reached again in Fig. 2. num(b) = 1 causes a successive return to node f. From node f an incoming arc to node l is checked which leads to a revisit of node j. This situation is presented in Fig. 3. Returning to node c searching through m leads to k again causing a return to node b, which is shown in Fig. 4 before returning to node b. Finally a successful search is started from b through n and o reaching node a, cf. Fig. 5. The output stack contains nodes a,b,n,o and a on top.

This example demonstrates the worst case for searching a handle, which can cause a depth-first-search starting from any node at most once.

The efficiency of Algorithm 14 is based on the fact that nodes are not checked several times. Obviously the effort of this algorithm depends on the number of calls of function dfs. For any $v \in S'$ dfs is called only once and any arc leading to v is checked at most twice. The net has $(|P| + |T|)$ nodes and $|F|$ arcs, thus obviously the worst case time complexity is $O(|P| + |T| + |F|)$.

6 Checking a Minimal Deadlock for Generating an S-component

This section describes an efficient test for a minimal deadlock generating an S-component or not.

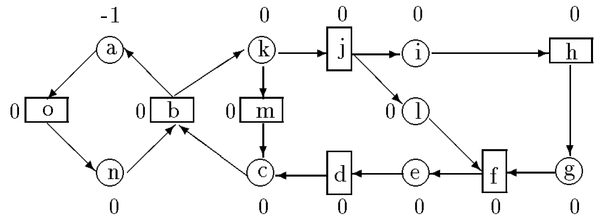

Fig. 1. Stack: a

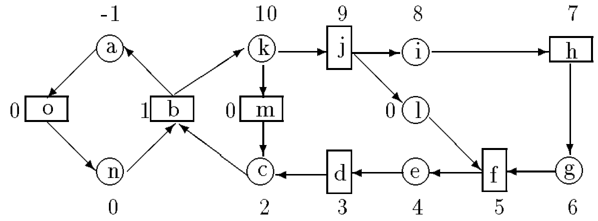

Fig. 2. Stack: a b c d e f g h i j k

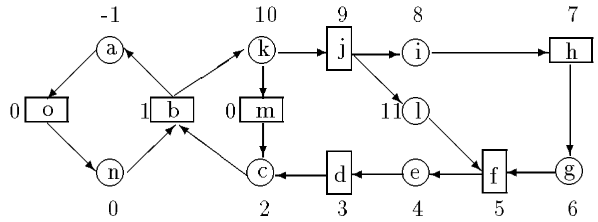

Fig. 3. Stack: a b c d e f l

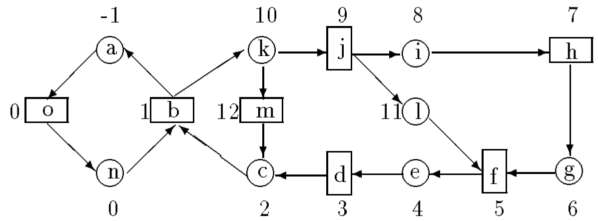

Fig. 4. Stack: a b c m

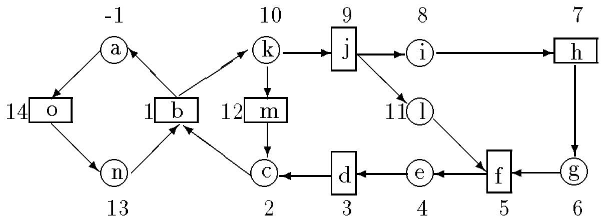

Fig. 5. Stack: a b n o a

A deadlock D in a net N generates a subnet $\hat{N} = (\hat{P}, \hat{T}, \hat{F})$ with

$$\hat{P} = D$$
$$\hat{T} = \bullet D \cup D\bullet$$
$$\hat{F} = F \cap ((\bullet D \times D) \cup (D \times D\bullet))$$

For being an S-component, $\hat{\text{N}}$ must satisfy the following conditions

1. $\hat{\text{N}}$ is strongly connected.
2. $\forall t \in \hat{T} : |\bullet t \cap D| = |t \bullet \cap D| = 1$ ($\bullet$-operation w.r.t. N)

Minimal deadlocks satisfy these properties partly. Theorem 10 ensures that any minimal deadlock is strongly connected and fulfills $\forall t \in \bullet D : |\bullet t \cap D| = 1$. The conditions still in question are summarized by the following

Conclusion 16.
Let $N = (P, T, F)$ be a FC-Net with a minimal deadlock $D \subseteq P$.
D generates an S-component iff $\bullet D = D\bullet$ and $\forall t \in \bullet D$: $|t \bullet \cap D| = 1$
($\bullet$-operation w.r.t. N).

One way to check these conditions is to count the arcs $(t, s) \in F$ of any transition $t \in \hat{T}$ leading to an $s \in D$. In the following algorithm the vector num is used for this purpose.

Algorithm 17. check-s-component(T,F,D,T_D)

```
Input:
    T,F of a FC-Net N=(P,T,F)
    D ⊆ P is a minimal deadlock of N
    T_D = •D ⊆ T
Output:
    Yes ,D generates an S-component.
    No , D does not generate an S-component.
Initiate:
    num(x) := 0, ∀x ∈ T − T_D ;
    num(x) := 1, ∀x ∈ T_D ;
Program:
begin
    forall(s ∈ D)
        begin
        forall(t ∈ T)
            begin
                if ((t,s) ∈ F)
                then num(t) := num(t) - 1 ;
                if (num(t) < 0)
                then return No ;
                if ((s,t) ∈ F ∧ t ∈ T − T_D)
                then return No ;
            end
```

```
        end
    return Yes ;
end
```

Termination is ensured by the finiteness of sets D and T.

Theorem 18.
1. Algorithm 17 terminates with No $\Rightarrow D$ *does not generate an S-component.*
2. Algorithm 17 terminates with Yes $\Rightarrow D$ *generates an S-component.*

In the following proofs $\bullet D$ is used instead of T_D to improve readability.

Proof 1. part. Two cases can cause a termination with *No* :
1. $\exists t \in T$: num(t)<0
2. $\exists (s,t) \in F : s \in D \wedge t \in T - \bullet D$
at 1) a) If $t \in \bullet D$
$\Rightarrow$num(t) = 1 by initiation
$\Rightarrow$num(t) is decremented at least twice to get num(t)<0
$\Rightarrow \exists s, s' \in D, s \neq s' : (t,s) \in F \wedge (t,s') \in F$
$\Rightarrow |t \bullet \cap D| > 1$
$\Rightarrow$D does not generate an S-component due to Conclusion 16
at 1) b) If $t \in T - \bullet D$
$\Rightarrow$num(t) = 0 by initiation
$\Rightarrow \exists s \in D : (t,s) \in F$, because of termination due to num(t) < 0
$\Rightarrow t \in \bullet D$, contradicting assumption $t \in T - \bullet D$.
Thus, if Algorithm 17 terminates with *No* due to the existence of a transition t with num(t)<0, this transition $t \notin T - \bullet D$.
at 2) $t \in D \bullet$ and $t \notin \bullet D$
$\Rightarrow \bullet D \neq D \bullet$
$\Rightarrow$D does not generate an S-component due to Conclusion 16. □

Proof 2. part. Assumption: Algorithm 17 terminates with *Yes* and D does not generate an S-component.
Because of Conclusion 16 only two reasons might prevent D from generating an S-component:
1. $\bullet D \neq D \bullet$
2. $\exists t \in \bullet D : |t \bullet \cap D| \neq 1$.
at 1) D is a minimal deadlock.
$\Rightarrow \exists t \in D \bullet : t \notin \bullet D$
Initial value of num(t) = 0
num(t) is never decremented because of $\forall s \in D : (t,s) \notin F$,
but $\exists (s,t) \in F \wedge t \in T - \bullet D$ ensuring return of *No* contradicting the assumption.
at 2) Thus initial value of num(t) = 1 and $\exists s, s' \in D, s \neq s' : (t,s), (t,s') \in F$
$\Rightarrow$num(t) is decremented twice which causes return of *No* contradicting the assumption.
The case $|t \bullet \cap D| < 1$ is not possible for minimal deadlocks, because they are strongly connected. □

The worst case time complexity is obviously $O(|D||T|)$ caused by the nested forall loops.

7 How to Ensure Liveness for LBFC-Nets at the Initial Marking

This section presents an algorithm to decide whether a structurally live and structurally bounded FC-Net is marked live by an initial marking M_0. Theorem 8 formulates a necessary and sufficient condition for liveness in FC-Nets[2]. Conclusion 9 shows that it is sufficient and necessary for LBFC-Nets to check all minimal deadlocks for being marked in order to fulfill the deadlock/trap-property of Theorem 8.

This conclusion can be exploited for an efficient algorithm. The main idea is to take the set of places U, that are unmarked at M_0, and delete successively those places preventing U from satisfying the deadlock property $\bullet U \subseteq U\bullet$. If finally $U = \emptyset$, M_0 is a live marking. Otherwise Theorem 8 is not satisfied by (N, M_0) and the net is not live. A formulation in pseudocode is:

Algorithm 19. check-initial-marking(N, M_0)

```
Input:
    N = (P,T,F) is LBFC-Net with initial marking M0
Output:
    Yes , M0 is a live marking.
    No , M0 is not a live marking.
Initiate:
    U := {p|p ∈ P ∧ M0(p) = 0} ;
    T' := •U ;
    done := False ;

Program:
begin
    while ( U ≠ ∅ and not done )
    begin
        if ( ∃t ∈T' : U ∩t• ≠ ∅∧ U ∩ • t = ∅ )
        then U := U −t• ; T' = T' - {t} ;
        else done := True ;
    end
    if (U = ∅)
    then Stop with Yes
    else Stop with No
end
```

U is the set of unmarked places, which are reduced to a set being a deadlock. T' is the set of transitions, which have output places in U. The termination is

[2] Note that we only regard strongly connected nets due to Theorem 4.

ensured by the finiteness of U and the fact that in each iteration of the while-loop at least one element is eliminated from U. The correctness follows directly from Conclusion 9. Time complexity is clearly dominated by efforts concerning the while loop. Initiating U and T costs at most $O(|P||T|)$. There are at most $|T|$ iterations within the while-loop, because once a transition's output places are not in U this property will hold until termination. Because at least one place is eliminated from U per iteration, altogether at most $min(|T|, |P|)$ iterations are possible. Finding $t \in T'$ with $U \cap t\bullet \neq \emptyset \wedge U \cap \bullet t = \emptyset$ costs $O(|P|)$ comparisons to match U with $t\bullet$ and $\bullet t$ *per(!)* $t \in$ T'. The size of T' decreases on any iteration and has $|T|$ as its maximal cardinality leading to a maximum effort of $O(|P||T|)$ per iteration. Thusly a rough estimation for the algorithm's worst case time complexity is $O(|P|^2|T|)$. [3]

8 Algorithm to Decide Liveness and Boundedness for FC-Nets

Now we exploit the algorithms of Sects. 4 - 7 to formulate the complete algorithm for deciding liveness and boundedness of FC-Nets.

Algorithm 20.

Input:
    $N = (P, T, F)$ is a FC-Net with incidence matrix C and initial marking M_0
Output is one of the following messages:
    1: "N is not strongly connected"
    2: "N contains a minimal deadlock not generating an S-component"
    3: "N is SMD but not structurally live"
    4: "N is a LBFC-Net but not live at M_0"
    5: "N is a LBFC-Net and M_0 is a live marking"
Initiate:
    Uncovered := P ;
    $a := |F \cap (P \times T)|$;
    $T_D := \emptyset$;

Function: check-str-connected(N), see [8]
This function checks, if a directed graph is strongly connected.

Function: get-minimal-deadlock(P,T,F,p,T_D), cf. Sect. 4
This function finds a minimal deadlock containing place p.

Function: check-s-component(T,F,D,T_D), cf. Sect. 6
This function checks deadlock D being an S-component.

Function: rank(C), see [10]
This function computes the rank of matrix C.

[3] or to be more precise: $O(min(|T|, |P|)|P||T|)$

```
Program:
begin
    if ( check-str-connected(N) = No )
    then Stop with "N is not strongly connected" ;
    while (Uncovered ≠ ∅)
    begin
        choose p ∈ Uncovered ;
        T_D := ∅ ;
        D := get-minimal-deadlock(P,T,F,p,T_D) ;
            {T_D is set in get-minimal-deadlock}
        if ( check-s-component(T,F,D,T_D) = No )
        then Stop with
            "N contains a minimal deadlock not generating an S-component" ;
        else Uncovered := Uncovered - D ;
    end
    if ( rank(C) ≠ |P| + |T| - a - 1)
    then Stop with "N is SMD but not structurally live" ;
    if ( check-initial-marking(N,M_0) = No )
    then Stop with "N is a LBFC-Net but not live at M_0" ;
    else Stop with "N is a LBFC-Net and M_0 is a live marking" ;
end
```

The set Uncovered is used to store places not covered by S-components calculated so far. T_D holds the preset of deadlock D. This set is computed in get-minimal-deadlock and exploited in check-s-component.

Termination of Algorithm 20 is ensured by the finiteness of P being the initial set for Uncovered and by the fact, that in each iteration of the while-loop at least the chosen place p is deleted from Uncovered or the algorithm stops with output 2 respectively. The correctness of Algorithm 20 follows directly the argumentation of the algorithm's outline in Sect. 3.

For determining worst case time complexity of Algorithm 20, worst case time complexities of all functions are listed below:

check-str-connected	$O(\|P\| + \|T\| + \|F\|)$
get-minimal-deadlock	$O(\|T\|(\|P\| + \|T\| + \|F\|))$
check-s-component	$O(\|P\|\|T\|)$
rank	$O(\|P\|^2\|T\|)$
check-initial-marking	$O(\|P\|^2\|T\|)$

Functions check-str-connected, rank and check-initial-marking are called at most once. The number of calls of get-minimal-deadlock and check-s-component is determined by the number of iterations within the while-loop. This number is at most $|P|$, because at least one place is eliminated from Uncovered per iteration. Thusly worst case time complexity for Algorithm 20 is given by

$$O([|P| + |T| + |F|] + |P|[|T|(|P| + |T| + |F|) + |P||T|] + 2|P|^2|T|) =$$
$$O(|P|\ |T|\ (|P| + |T| + |F|)).$$

9 Conclusions

We have described a polynomial-time algorithm deciding if a FC-Net is structurally live and structurally bounded and has a live initial marking. This algorithm has a worst case time complexity of $O(|P||T|(|P| + |T| + |F|))$, which can be estimated as $O(n^4)$ with $n = max(|P|, |T|)$. The algorithm combines and refines ideas and theorems of J.Esparza in a new way and adds a new idea to prove in an efficient way, if an initial marking is live given a structurally live and structurally bounded FC-Net. A polynomial-time algorithm of comparable functionality is described in [4]. It is based on the linear algebraic characterization of deadlocks and traps in [7] and exploits linear programming techniques. Its worst case time complexity is estimated as $O(|P||T||F|^6)$. In order to show a direct comparison our algorithm's time complexity can be estimated by $\mathbf{O}(|\mathbf{P}||\mathbf{T}||\mathbf{F}|)$, because for strongly connected nets $|P| + |T| \leq |F|$ holds. Therefore we regard our algorithm, exploiting a graph orientated approach, as a major improvement of FC-Net analysis.

References

1. Best,E.: *Some Classes of Live and Safe Petri Nets,* in K.Voss, H.J.Genrich, G.Rozenberg: "Concurrency and Nets, Advances of Petri Nets", Springer, Berlin 1987.
2. Esparza,J.; Best,E.; Silva,M.: *Minimal Deadlocks in Free Choice Nets*, Hildesheimer Informatikberichte 1/89, Institut für Informatik, Universität Hildesheim.
3. Esparza,J.: *Synthesis Rules for Petri Nets, and How They Lead to New Results*, Hildesheimer Informatikberichte 5/90, Institut für Informatik, Universität Hildesheim.
4. Esparza,J.;Silva,M.: *A Polynomial-Time Algorithm to Decide Liveness of Bounded Free Choice Nets*, Hildesheimer Informatikberichte 12/90, Institut für Informatik, Universität Hildesheim.
5. Hack,M.H.T.: *Analysis of Production Schemata by Petri Nets,* TR-94, MIT, Boston 1972 corrected june 1974
6. Jones,N.;Landweber,L.;Lien,Y.: *Complexity of some Problems in Petri Nets,* Theoretical Computer Science, Vol 4, pp. 277-299, 1977.
7. Lautenbach,K.: *Linear Algebraic Calculation of Deadlocks and Traps* in K.Voss, H.J.Genrich, G.Rozenberg: "Concurrency and Nets, Advances in Petri Nets", Springer, Berlin 1987.
8. Mehlhorn,K.: *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, EATCS Monographs on Theoretical Computer Science, Springer, Berlin Heidelberg 1984.
9. Starke,P.: *Analysetechniken von Petri-Netz-Modellen,* Teubner, Stuttgart 1990 (in German).
10. Stewart,G.W.: *Introduction to Matrix Computations,* Academic Press, New York, 1973.