Approximating Petri Net Reachability Along Context-free Traces

Mohamed Faouzi Atig[1] and Pierre Ganty[2]

1 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se
2 **IMDEA Software Institute, Spain**
pierre.ganty@imdea.org

Abstract

We investigate the problem asking whether the intersection of a context-free language (CFL) and a Petri net language (PNL) (with reachability as acceptance condition) is empty. Our contribution to solve this long-standing problem which relates, for instance, to the reachability analysis of recursive programs over unbounded data domain, is to identify a class of CFLs called the finite-index CFLs for which the problem is decidable. The k-index approximation of a CFL can be obtained by discarding all the words that cannot be derived within a budget k on the number of occurrences of non-terminals. A finite-index CFL is thus a CFL which coincides with its k-index approximation for some k. We decide whether the intersection of a finite-index CFL and a PNL is empty by reducing it to the reachability problem of Petri nets with *weak* inhibitor arcs, a class of systems with infinitely many states for which reachability is known to be decidable. Conversely, we show that the reachability problem for a Petri net with weak inhibitor arcs reduces to the emptiness problem of a finite-index CFL intersected with a PNL.

1998 ACM Subject Classification D.2.4 Software Engineering /Program Verification

Keywords and phrases Petri nets, Context-free Grammars, Reachability Problem

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.152

1 Introduction

Automated verification of infinite-state systems, for instance programs with (recursive) procedures and integer variables, is an important and a highly challenging problem. Pushdown automata (or equivalently context-free grammars) have been proposed as an adequate formalism to model procedural programs. However pushdown automata require finiteness of the data domain which is typically obtained by abstracting the program's data, for instance, using the predicate abstraction techniques [3, 9]. In many cases, reasoning over finite abstract domains leads to too coarse an analysis and is therefore not precise. To palliate this problem, it is natural to model a procedural program with integer variables as a pushdown automaton manipulating counters. In general, pushdown automata with counters are Turing powerful which implies that basic decision problems are undecidable (this is true even for the case finite-state automata with counters).

Therefore one has to look for restrictions on the model which retain sufficient expressiveness while allowing basic properties like reachability to be algorithmically verified. One such restriction is to forbid the test of a counter and a constant for equality. In fact, forbidding test for equality implies the decidability of the reachability problem for the case of finite-state automata with counters (i.e. Petri nets [13, 15]).

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 152–163

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The verification problem for pushdown automata with (restricted) counters boils down to check whether a context-free language (CFL) and a Petri net language (PNL) (with reachability as acceptance condition) are disjoint or not. We denote this last problem $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$.

The decidability of $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$ is open and lies at the very edge of our comprehension of infinite-state systems. We see two breakthroughs contributing to this question. First, determining the emptiness of a PNL was known to be decidable as early as the eighties. Then, in 2008, Reinhardt [15] lifted this result to an extension of PN with inhibitor arcs (that allow to test if a counter equals 0) which must satisfy some additional topological conditions. By imposing a topology on the tests for zero, Reinhardt prevents his model to acquire Turing powerful capabilities. We call his model PNW and their languages PNWL.

Our contribution to the decidability of $\mathsf{PNL} \cap \mathsf{CFL} \stackrel{?}{=} \emptyset$ comes under the form of a partial answer which is better understood in terms of underapproximation. In fact, given a PNL L_1 and a language L of a context-free grammar, we under-approximate L by a subset L' which is obtained by discarding from L all the words that cannot be derived within a given budget $k \in \mathbb{N}$ on the number of non-terminal symbols. (In fact, the subset L' contains any word of L that can be generated by a derivation of the context-free grammar that contains at most k non-terminal symbols at each derivation step.) We show how to compute L' by annotating the variables of the context-free grammar for L with an allowance. What is particularly appealing is that the coverage of L increases with the allowance. Approximations induced by allowances are non-trivial: every regular or linear language is captured exactly with an allowance of 1, L' coincides with L when the allowance is unbounded, and under commutativity of concatenation L' coincides with L for some allowance $k \in \mathbb{N}$.

We call finite-index CFL, or fiCFL for short, a context-free language where each of its words can be derived within a given budget. In this paper, we prove the decidability of $\mathsf{PNL} \cap \mathsf{fiCFL} \stackrel{?}{=} \emptyset$ by reducing it to the emptiness problem of PNWL. We also prove the converse reduction; showing those two problems are equivalent. Hence, we offer a whole new perspective on the emptiness problem for PNWL and $\mathsf{PNL} \cap \mathsf{CFL}$.

To conclude the introduction let us mention the recent result of [2] which builds on [13] to give an alternative proof of Reinhardt's result (PNW reachability is decidable) for the particular case where one counter only can be tested for zero.

2 Preliminaries

2.1 Context-Free Languages

An *alphabet* Σ is a finite non-empty set of *symbols*. A *word* w over an alphabet Σ is a finite sequence of symbols of Σ where the empty sequence is denoted ε. We write Σ^* for the set of words over Σ. Let $L \subseteq \Sigma^*$, L defines a *language*.

A *context-free grammar* (CFG) G is a tuple $(\mathcal{X}, \Sigma, \mathcal{P})$ where $\mathcal{X}$ is a finite non-empty set of *variables* (*non-terminal letters*), Σ is an alphabet of *terminal letters*, and $\mathcal{P} \subseteq \big(\mathcal{X} \times (\mathcal{X}^2 \cup \Sigma \cup \{\epsilon\})\big)$ is a finite set of *productions* (the production (X, w) may also be denoted by $X \to w$). For every production $p = (X, w) \in \mathcal{P}$, we use $head(p)$ to denote the variable X. Observe that the form of the productions is restricted, but it has been shown in [12] that every CFG can be transformed, in polynomial time, into an equivalent grammar of this form.

Given two strings $u, v \in (\Sigma \cup \mathcal{X})^*$ we define the relation $u \Rightarrow v$, if there exists a production $(X, w) \in \mathcal{P}$ and some words $y, z \in (\Sigma \cup \mathcal{X})^*$ such that $u = yXz$ and $v = ywz$. We use $\Rightarrow^*$ for the reflexive transitive closure of $\Rightarrow$. Given $X \in \mathcal{X}$, we define the language $L_G(X)$, or

simply $L(X)$ when G is clear form the context, as $\{w \in \Sigma^* \mid X \Rightarrow^* w\}$. A language L is *context-free* (CFL) if there exists a CFG $G = (\mathcal{X}, \Sigma, \mathcal{P})$ and $A \in \mathcal{X}$ such that $L = L_G(A)$.

2.2 Finite-index Approximation of Context-Free Languages

Let $k \in \mathbb{N}$, $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a CFG and $A \in \mathcal{X}$. A derivation from A given by $A = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n$ is *k-index bounded* if for every $i \in \{0, \ldots, n\}$ at most k symbols of α_i are variables. We denote by $L^{(k)}(A)$ the subset of $L(A)$ such that for every $w \in L^{(k)}(A)$ there exists a k-index bounded derivation $A \Rightarrow^* w$. We call $L^{(k)}(A)$ the *k-index approximation* of $L(A)$ or more generically we say that $L^{(k)}(A)$ is a *finite-index approximation* of $L(A)$.[1]

Let us now give some known properties of finite-index approximations. Clearly $\lim_{k\to\infty} L^{(k)}(A) = L(A)$. Moreover, let L be a regular or linear language[2], then there exists a CFG G', and a variable A' of G' such that $L(A') = L = L^{(1)}(A')$. Also Luker showed in [14] that if $L(A) \subseteq L(w_1^* \cdots w_n^*)$ for some $w_i \in \Sigma^*$, then $L^{(k)}(A) = L(A)$ for some $k \in \mathbb{N}$. More recently, [6, 8] showed some form of completeness for finite-index approximation when commutativity of concatenation is assumed. It shows that there exists a $k \in \mathbb{N}$ such that $L(A) \subseteq \Pi(L^{(k)}(A))$ where $\Pi(L)$ denotes the language obtained by permuting symbols of w for every $w \in L$. As an incompleteness result, Salomaa showed in [16] that for the Dyck language $L_{D_1^*}$ over 1-pair of parentheses there is no CFG G', variable A' of G' and $k \in \mathbb{N}$ such that $L^{(k)}(A') = L_{D_1^*}$.

Inspired by [5, 7, 6] let us define the CFG $G^{[k]}$ which annotates the variables of $\mathcal{X}$ with a positive integer. With this annotation we can capture precisely finite-index approximations of $L(A)$ as given in Lem. 2.

▶ **Definition 1.** Let $G^{[k]} = (\mathcal{X}^{[k]}, \Sigma, \mathcal{P}^{[k]})$ be the context-free grammar defined as follows: $\mathcal{X}^{[k]} = \left\{X^{[i]} \mid 0 \leq i \leq k \wedge X \in \mathcal{X}\right\}$, and $\mathcal{P}^{[k]}$ is the smallest set such that:

- For every $X \to Y\,Z \in \mathcal{P}$, $\mathcal{P}^{[k]}$ has the productions $X^{[i]} \to Y^{[i-1]}Z^{[i]}$ and $X^{[i]} \to Y^{[i]}Z^{[i-1]}$ for every $i \in \{1, \ldots, k\}$.
- For every $X \to \sigma \in \mathcal{P}$ with $\sigma \in \Sigma \cup \{\epsilon\}$, $X^{[i]} \to \sigma \in \mathcal{P}^{[k]}$ for all $i \in \{0, \ldots, k\}$.

What follows is a consequence of several results from different papers by Esparza *et al.* Because of space constraints the proof is given in [1].

▶ **Lemma 2.** *Let $X \in \mathcal{X}$. We have $L(X^{[k]}) = L^{(k+1)}(X)$.*

2.3 Petri nets with Inhibitor Arcs

Let Σ be a finite non-empty set, a *multiset* (or a marking) $\mathbf{m} \colon \Sigma \to \mathbb{N}$ over Σ maps each symbol of Σ to a natural number. Let $\mathbb{M}[\Sigma]$ be the set of all multisets over Σ.

Sometimes, we use $\mathbf{m} = [\![q_1, q_1, q_3]\!]$ to denote the multiset $\mathbf{m} \in \mathbb{M}[\{q_1, q_2, q_3, q_4\}]$ such that $\mathbf{m}(q_1) = 2$, $\mathbf{m}(q_2) = \mathbf{m}(q_4) = 0$, and $\mathbf{m}(q_3) = 1$. The empty multiset is denoted $\varnothing$.

Given $\mathbf{m}, \mathbf{m}' \in \mathbb{M}[\Sigma]$ we define $\mathbf{m} \oplus \mathbf{m}' \in \mathbb{M}[\Sigma]$ to be the multiset such that $\forall a \in \Sigma \colon (\mathbf{m} \oplus \mathbf{m}')(a) = \mathbf{m}(a) + \mathbf{m}'(a)$, we also define the natural partial order $\preceq$ on $\mathbb{M}[\Sigma]$ as follows: $\mathbf{m} \preceq \mathbf{m}'$ iff there exists $\mathbf{m}^{\Delta} \in \mathbb{M}[\Sigma]$ such that $\mathbf{m} \oplus \mathbf{m}^{\Delta} = \mathbf{m}'$. We also define $\mathbf{m} \ominus \mathbf{m}' \in \mathbb{M}[\Sigma]$ as the multiset such that $(\mathbf{m} \ominus \mathbf{m}') \oplus \mathbf{m}' = \mathbf{m}$ provided $\mathbf{m}' \preceq \mathbf{m}$.

A *Petri net* with inhibitor arcs (PNI for short) $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ consists of a finite non-empty set S of *places*, a finite set T of *transitions* disjoint from S, a tuple

[1] Finite-index approximations were first studied in the 60's.
[2] See [11] for definitions.

$F = \langle Z, I, O\rangle$ of functions $Z\colon T \to 2^S$, $I\colon T \to \mathbb{M}[S]$ and $O\colon T \to \mathbb{M}[S]$, and an *initial marking* $\mathbf{m}_\iota \in \mathbb{M}[S]$. A marking $\mathbf{m}$ ($\in \mathbb{M}[S]$) of N assigns to each place $p \in S$ $\mathbf{m}(p)$ *tokens*.

A transition $t \in T$ is *enabled at* $\mathbf{m}$, written $\mathbf{m}\,[t\rangle$, if $I(t) \preceq \mathbf{m}$ and $\mathbf{m}(p) = 0$ for all $p \in Z(t)$. A transition t that is enabled at $\mathbf{m}$ can be *fired*, yielding a marking $\mathbf{m}'$ such that $\mathbf{m}' = (\mathbf{m} \ominus I(t)) \oplus O(t)$. We write this fact as follows: $\mathbf{m}\,[t\rangle\,\mathbf{m}'$. We extend enabledness and firing inductively to finite sequences of transitions as follows. Let $w \in T^*$. If $w = \varepsilon$ we define $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ iff $\mathbf{m}' = \mathbf{m}$; else if $w = u \cdot v$ we have $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ iff $\exists \mathbf{m}_1 \colon \mathbf{m}\,[u\rangle\,\mathbf{m}_1 \wedge \mathbf{m}_1\,[v\rangle\,\mathbf{m}'$.

A marking $\mathbf{m} \in \mathbb{M}[S]$ is *reachable from* $\mathbf{m}_0$ if and only if there exists $w \in T^*$ such that $\mathbf{m}_0\,[w\rangle\,\mathbf{m}$. Given a language $L \subseteq T^*$ over the transitions of N, the *set of reachable markings from* $\mathbf{m}_0$ *along* L, written $[\mathbf{m}_0\rangle^L$, is defined by $\{\mathbf{m} \mid \exists w \in L \colon \mathbf{m}_0\,[w\rangle\,\mathbf{m}\}$. Incidentally, if L is unspecified then it is assumed to be T^* and we simply write $[\mathbf{m}_0\rangle$ for the set of markings reachable from $\mathbf{m}_0$. To avoid ambiguities, we sometimes explicit the PNI, e.g. $\mathbf{m}_1 \in [\mathbf{m}_0\rangle^L_N$.

A Petri net with *weak* inhibitor arcs (PNW for short) is a PNI $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ such that there is an index function $f\colon S \to \mathbb{N}$ with the property:

$$\forall p, p' \in S \colon f(p) \leq f(p') \to (\forall t \in T \colon p' \in Z(t) \to p \in Z(t)) \ . \tag{1}$$

A Petri net (PN for short) can be seen as a subclass of Petri nets with *weak* inhibitor arcs where $Z(t) = \emptyset$ for all transitions $t \in T$. In this case, we shorten F as the pair $\langle I, O\rangle$.

The *reachability* problem for a PNI $N = (S, T, F = \langle Z, I, O\rangle, \mathbf{m}_\iota)$ is the problem of deciding, for a given marking $\mathbf{m}$, whether $\mathbf{m} \in [\mathbf{m}_\iota\rangle$ holds. It is well known that reachability for Petri nets with inhibitor arcs is undecidable [10]. However, the following holds:

▶ **Theorem 3.** *[15] The reachability problem for* PNW *is decidable.*

2.4 The reachability problem for Petri nets along finite-index CFL

Let us formally define the problem we are interested in. Given: (1) a Petri net $N = (S, T, F, \mathbf{m}_\iota)$ where $T \neq \emptyset$; (2) a CFG $G = (\mathcal{X}, T, \mathcal{P})$ and $A \in \mathcal{X}$; (3) a marking $\mathbf{m}_f \in \mathbb{M}[S]$; and (4) a value $k \in \mathbb{N}$.

$$\text{Does } \mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L^{(k)}(A)} \text{ hold ?}$$

In what follows, we prove the interreducibility of the reachability problem for PN along finite-index CFL and the reachability problem for PNW.

3 From PN reachability along fiCFL to PNW reachability

In this section, we show that the reachability problem for Petri nets along finite-index CFL is decidable. To this aim, let us fix an instance of the problem: a Petri net $N = (S, T, F, \mathbf{m}_\iota)$ where $T \neq \emptyset$, a CFG $G = (\mathcal{X}, T, \mathcal{P})$, $\mathbf{m}_f \in \mathbb{M}[S]$, and a natural number $k \in \mathbb{N}$. Moreover, let $G^{[k]} = (\mathcal{X}^{[k]}, T, \mathcal{P}^{[k]})$ be the CFG given by def. 1.

Lemma 2 shows that $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L^{(k+1)}(A)}$ if and only if $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$. Then, our decision procedure, which determines if $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$, proceeds by reduction to the reachability problem for PNW and is divided in two steps. First, we reduce the question $\mathbf{m}_f \in [\mathbf{m}_\iota\rangle^{L(A^{[k]})}$ to the existence of a successful execution in the program of Alg. 1 which, in turn, is reduced to a reachability problem for PNW. Let us describe Alg. 1.

Part 1. Alg. 1 gives the procedure *traverse* in which $\mathbf{M}_\mathsf{i}$ and $\mathbf{M}_\mathsf{f}$ are global arrays of markings with index ranging from 0 to k (i.e., for every $j \in \{0,\dots,k\}$, $\mathbf{M}_\mathsf{i}[j], \mathbf{M}_\mathsf{f}[j] \in \mathbb{M}[S]$). We say that a call $traverse(X^{[\ell]})$ *successfully returns* if there exists an execution which eventually reaches line 22 (i.e., no assert fails) and the postcondition $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for every $j \in \{0,\dots,\ell\}$ holds. Moreover we say that a call $traverse(X^{[\ell]})$ is *proper* if $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $0 \leq j < \ell$. Let $\ell \in \{0,\dots,k\}$, we shall now demonstrate that a proper call $traverse(X^{[\ell]})$ successfully returns if and only if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$.
The formal statement is given at Lem. 4. We give some explanations about Alg. 1 first.

Algorithm 1: *traverse*

Input: A variable $X^{[\ell]} \in \mathcal{X}^{[k]}$ of $G^{[k]}$

1 **begin**
2 Let $p \in \mathcal{P}^{[k]}$ such that $head(p) = X^{[\ell]}$
3 **switch** p **do**
4 **case** $X^{[\ell]} \to \sigma$ /* $\sigma \in \Sigma \cup \{\epsilon\}$ */
5 $\mathbf{M}_\mathsf{i}[\ell] := (\mathbf{M}_\mathsf{i}[\ell] \ominus I(\sigma)) \oplus O(\sigma)$
6 Choose non det $qty \in \mathbb{M}[S]$
7 $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus qty$
8 **case** $X^{[\ell]} \to B^{[\ell]}C^{[\ell-1]}$
9 $transfer\_from\_to(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{f}[\ell-1])$
10 Choose non det $qty \in \mathbb{M}[S]$
11 $(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) := (\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) \oplus qty$
12 $traverse(C^{[\ell-1]})$
13 assert $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$
14 $traverse(B^{[\ell]})$
15 **case** $X^{[\ell]} \to B^{[\ell-1]}C^{[\ell]}$
16 $transfer\_from\_to(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{i}[\ell-1])$
17 Choose non det $qty \in \mathbb{M}[S]$
18 $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) \oplus qty$
19 $traverse(B^{[\ell-1]})$
20 assert $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$
21 $traverse(C^{[\ell]})$
22 **return**

Instructions of the form $(var_1, var_2) := (var_1, var_2) \odot qty$ where $qty \in \mathbb{M}[S]$ and $\odot \in \{\oplus, \ominus\}$ stand for the two instructions $var_1 := var_1 \odot qty; var_2 := var_2 \odot qty$. Observe that, given two markings $\mathbf{m}, \mathbf{m}'$, the subtraction operation $\mathbf{m} \ominus \mathbf{m}'$ can be performed only when $\mathbf{m}' \preceq \mathbf{m}$ (i.e., assume every occurence of $\mathbf{m} \ominus \mathbf{m}'$ is preceded by assert $\mathbf{m}' \preceq \mathbf{m}$).

The procedure *transfer_from_to* proceeds as follows: (1) non deterministically choose a marking $qty \in \mathbb{M}[S]$, (2) add qty to tgt, and (3) subtract qty from src. Intuitively, it transfers an arbitrary sub-marking qty of src to tgt.

Algorithm 2: *transfer_from_to*

Input: src, tgt
Choose non det. $qty \in \mathbb{M}[S]$
$tgt := tgt \oplus qty$
$src := src \ominus qty$

Intuitively, $traverse(X^{[\ell]})$ simulates the execution of N along a sequence of transitions w such that $X^{[\ell]} \Rightarrow^* w$. However as *traverse* simulates the derivation of w, it does not necessarily follows a leftmost order but instead an order which guarantees that a bounded amount of memory only is needed to derive w. This is needed for the translation to PNW.

To understand the correctness argument of Alg. 1, let us see why the call $traverse(X^{[\ell]})$ successfully returns if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$.

Let us start by assuming that $X^{[\ell]} \Rightarrow u \Rightarrow^* w$ with $u \in (\Sigma \cup \mathcal{X}^{[k]})^*$ and $\mathbf{M}_\mathsf{i}[\ell] \,[w\rangle_N\, \mathbf{M}_\mathsf{f}[\ell]$. An execution of $traverse(X^{[\ell]})$ is such that at line 2, some $p = (X^{[\ell]}, u) \in \mathcal{P}^{[k]}$ is picked. The choice of p yields three case studies.

The first case is given by $p = (X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$ (with $\sigma \in \Sigma \cup \{\epsilon\}$) which yields the case of line 4 to be executed. It follows that $X^{[\ell]} \Rightarrow u = w = \sigma$. Since $\mathbf{M}_\mathsf{i}[\ell] \,[\sigma\rangle\, \mathbf{M}_\mathsf{f}[\ell]$ holds by assumption there exists a marking qty such that $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus qty$

has the effect to empty $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$. Finally, upon reaching line 22 we find that the post condition $\mathbf{M}_\mathsf{i}[\ell] = \varnothing = \mathbf{M}_\mathsf{f}[\ell]$ for every $j \in \{0, \ldots, \ell\}$ holds. Therefore the call to $\mathit{traverse}(X^{[\ell]})$ successfully returns.

The second case is $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$ which yields line 9 is executed. We further assume that $\mathbf{M}_\mathsf{i}[\ell] \, [w_1 w_2\rangle \, \mathbf{M}_\mathsf{f}[\ell]$ where $w_1 \in L(B^{[\ell]})$ and $w_2 \in L(C^{[\ell-1]})$. Hence, we find that there is $\mathbf{m} \in \mathbb{M}[S]$ such that $\mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m} \, [w_2\rangle \, \mathbf{M}_\mathsf{f}[\ell]$ which, by monotonicity of PN, is equivalent to:

$$\exists \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3 \in \mathbb{M}[S] \colon \mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m}_1 \wedge \mathbf{m}_2 \, [w_2\rangle \, \mathbf{m}_3 \wedge \mathbf{m}_1 = \mathbf{m}_2 \oplus (\mathbf{M}_\mathsf{f}[\ell] \ominus \mathbf{m}_3) \ . \tag{2}$$

Observe that, for (2) to be valid, we need $\mathbf{m}_3 \preceq \mathbf{M}_\mathsf{f}[\ell]$ to hold.

Let us resume the execution of $\mathit{traverse}(X^{[\ell]})$ which now executes the call to the procedure $\mathit{transfer\_from\_to}(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{f}[\ell-1])$ of line 9. We assume the transfer is given by the marking $\mathbf{m}_3$ so that when returning from Alg. 2 we have $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_3$. Next the instruction $(\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) := (\mathbf{M}_\mathsf{f}[\ell], \mathbf{M}_\mathsf{i}[\ell-1]) \oplus \mathit{qty}$ of line 11 executes. The effect is to add an arbitrary value, say $\mathbf{m}_2$, to the markings $\mathbf{M}_\mathsf{f}[\ell]$ and $\mathbf{M}_\mathsf{i}[\ell-1]$. Therefore, $\mathbf{M}_\mathsf{i}[\ell-1]$ is updated to $\mathbf{m}_2$ and $\mathbf{M}_\mathsf{f}[\ell]$ to $\mathbf{m}_1$ $(= \mathbf{m}_2 \oplus (\mathbf{M}_\mathsf{f}[\ell] \ominus \mathbf{m}_3))$.

Now, a recursive proper call $\mathit{traverse}(C^{[\ell-1]})$ takes place (see line 12) to determine if there exists a word $w' \in L(C^{[\ell-1]})$ such that $\mathbf{M}_\mathsf{i}[\ell-1] \, [w'\rangle \, \mathbf{M}_\mathsf{f}[\ell-1]$. We conclude from above that w_2 is such a word: $(\mathbf{M}_\mathsf{i}[\ell-1] =)\mathbf{m}_2 \, [w_2\rangle \, \mathbf{m}_3 (= \mathbf{M}_\mathsf{f}[\ell-1])$ holds. Therefore the call $\mathit{traverse}(C^{[\ell-1]})$ successfully returns and we find that $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j < \ell$ (assuming Alg. 1 is correct). This implies that the assert statement at line 13 succeeds.

Then, a recursive proper call $\mathit{traverse}(B^{[\ell]})$ takes place (see line 14) to determine if there exists a word $w' \in L(B^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell] \, [w'\rangle \, \mathbf{M}_\mathsf{f}[\ell]$. We conclude from above that w_1 is such a word: $\mathbf{M}_\mathsf{i}[\ell] \, [w_1\rangle \, \mathbf{m}_1 (= \mathbf{M}_\mathsf{f}[\ell])$ holds. Therefore $\mathit{traverse}(B^{[\ell]})$ successfully returns (again assuming Alg. 1 is correct) and so is $\mathit{traverse}(X^{[\ell]})$ and we are done.

The third case given by $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]}) \in \mathcal{P}^{[k]}$ is treated similarly.

It is worth pointing that the control flow of *traverse* matches the traversal of a parse tree of $G^{[k]}$ such that at each node *traverse* goes first to the subtree which carries the least index. The tree traversal is implemented through recursive calls in *traverse*. To see that the traversal goes first in the subtree of least index, it suffices to look at the ordering of the recursive calls to *traverse* in the code of Alg. 1, e.g. in case the of line 8, $\mathit{traverse}(C^{[\ell-1]})$ is called before $\mathit{traverse}(B^{[\ell]})$. Moreover, we have that the proper call $\mathit{traverse}(X^{[\ell]})$ returns iff there exists a parse tree t of $G^{[k]}$ with root variable $X^{[\ell]}$ such that the sequence of transitions given by the yield of t is enabled from the marking stored in $\mathbf{M}_\mathsf{i}[\ell]$ and its firing yields the marking stored in $\mathbf{M}_\mathsf{f}[\ell]$. Because of the least index first tree traversal, it turns out that the arrays $\mathbf{M}_\mathsf{i}$ and $\mathbf{M}_\mathsf{f}$ provide enough space to manage all the intermediary results.

Also, we observe that when the procedure $\mathit{traverse}(X^{[\ell]})$ calls itself with the parameter, say $B^{[\ell]}$, the call is a *tail recursive call*. This means that when $\mathit{traverse}(B^{[\ell]})$ returns then $\mathit{traverse}(X^{[\ell]})$ immediately returns. It is known from programming techniques how to implement tail recursive call without consuming space on the call stack. In the case of Alg. 1, we can do so by having a global variable to store the parameter of *traverse* and by replacing tail recursive calls with **goto** statements. For the remaining recursive calls (line 12 and 19), because the index of the callee is one less than the index of the caller, we conclude that a bounded space consisting of k frames suffices for the call stack.

Those two insights (two arrays with k entries and a stack with k frames) will be the key to show, in Part 2, that *traverse* can be implemented as a PNW.

▶ **Lemma 4.** *Let* $\ell \in \{0, \ldots, k\}$, $X^{[\ell]} \in \mathcal{X}^{[k]}$, *and* $\mathbf{m}, \mathbf{m}' \in \mathbb{M}[S]$. *Then, the proper call*

traverse$(X^{[\ell]})$ *with* $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ *and* $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ *successfully returns if and only if there exists* $w \in L(X^{[\ell]})$ *such that* $\mathbf{m}\,[w\rangle_N\,\mathbf{m}'$.

Proof. If. We prove that if there exists $w \in L(X^{[\ell]})$ such that $\mathbf{m}\,[w\rangle\,\mathbf{m}'$ then the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ and $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ successfully returns.

Our proof is done by induction on the length n of the derivation of $w \in L(X^{[\ell]})$. For the case $n = 1$, we necessarily have $X^{[\ell]} \Rightarrow w = \sigma$ for some $(X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$. In this case, the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$ and $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ executes as follows: $p = (X^{[\ell]}, \sigma)$ is picked and the case of line 4 executes successfully since $\mathbf{m} = \mathbf{M}_\mathsf{i}[\ell]\,[\sigma\rangle\,\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ holds. In fact, after the assignment of line 5 we have $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{M}_\mathsf{f}[\ell]$. Hence, by choosing the right *qty*, the instruction $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell]) \ominus \mathit{qty}$ of line 7 empties $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$ which shows that *traverse*$(X^{[\ell]})$ successfully returns.

For the case $n > 1$, we have $X^{[\ell]} \Rightarrow^n w$ which necessarily has the form $X^{[\ell]} \Rightarrow B^{[\ell]}C^{[\ell-1]} \Rightarrow^{n-1} w$ or $X^{[\ell]} \Rightarrow B^{[\ell-1]}C^{[\ell]} \Rightarrow^{n-1} w$ by def. of $G^{[k]}$. Assume we are in the latter case. Thus there exists w_1 and w_2 such that $X^{[\ell]} \Rightarrow B^{[\ell-1]}C^{[\ell]} \Rightarrow^i w_1C^{[\ell]} \Rightarrow^j w_1w_2 = w$ with $i + j = n - 1$ and $\exists \mathbf{m}_1 \colon \mathbf{m}\,[w_1\rangle\,\mathbf{m}_1\,[w_2\rangle\,\mathbf{m}'$. Observe that $w_1 \in L(B^{[\ell-1]})$ and $w_2 \in L(C^{[\ell]})$ and so by induction hypothesis we find that the proper call *traverse*$(B^{[\ell-1]})$ with $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_1$ successfully returns. And so does, by induction hypothesis, the proper call *traverse*$(C^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}_1$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$. Therefore let us consider the proper call *traverse*$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$. We show it successfully returns.

First observe that the call to the procedure *traverse*$(X^{[\ell]})$ is proper. Next, at line 2, pick $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]})$. Then the call *transfer_from_to*$(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{i}[\ell-1])$ of line 16 executes such that $\mathbf{M}_\mathsf{i}[\ell]$ is updated to $\varnothing$ and $\mathbf{M}_\mathsf{i}[\ell-1]$ to $\mathbf{m}$. Next the non determisnistic choice of *qty* and the instruction $(\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) := (\mathbf{M}_\mathsf{i}[\ell], \mathbf{M}_\mathsf{f}[\ell-1]) \oplus \mathit{qty}$ execute such that both $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell-1]$ are updated to $\mathbf{m}_1$. Recall that $\mathbf{m}\,[w_1\rangle\,\mathbf{m}_1\,[w_2\rangle\,\mathbf{m}'$.

Finally we showed above that the proper call *traverse*$(B^{[\ell-1]})$ successfully returns, the assert that follows too and finally the proper call *traverse*$(C^{[\ell]})$. Moreover it is routine to check that upon completion of *traverse*$(C^{[\ell]})$ (and therefore *traverse*$(X^{[\ell]})$) we have $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j \leq \ell$.

The left case (i.e. $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]}) \in \mathcal{P}^{[k]}$) is treated similarly.

Only If. Here we prove that if the proper call *traverse*$(X^{[\ell]})$ successfully returns then there exists $w \in L(X^{[\ell]})$ such that $\mathbf{M}_\mathsf{i}[\ell]\,[w\rangle_N\,\mathbf{M}_\mathsf{f}[\ell]$.

Our proof is done by induction on the number n of times line 2 is executed during the execution of *traverse*$(X^{[\ell]})$. In every case, line 2 is executed at least once. For the case $n = 1$, the algorithm necessarily executes the case of line 4. In this case, the definition of $G^{[k]}$ shows that along a successful execution of *traverse*$(X^{[\ell]})$, the non deterministic choice of line 2 necessarily returns a production of the form $p = (X^{[\ell]}, \sigma) \in \mathcal{P}^{[k]}$. Therefore, a successful execution must execute line 5 to 7 and then 22 after which the postcondition $\mathbf{M}_\mathsf{i}[j] = \mathbf{M}_\mathsf{f}[j] = \varnothing$ for all $j \leq \ell$ holds. Because the postcondition holds, we find that $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{M}_\mathsf{f}[\ell]$ holds before executing line 7, hence that $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{M}_\mathsf{i}[\ell] \ominus I(\sigma) \oplus O(\sigma)$ before executing line 5, and finally $\mathbf{M}_\mathsf{i}[\ell]\,[\sigma\rangle\,\mathbf{M}_\mathsf{f}[\ell]$ by semantics of transition σ and we are done.

For the case $n > 1$, the first non deterministic choice of line 2 necessarily picks $p \in \mathcal{P}^{[k]}$ of the form $(X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$ or $(X^{[\ell]}, B^{[\ell-1]}C^{[\ell]})$. Let us assume $p = (X^{[\ell]}, B^{[\ell]}C^{[\ell-1]})$, hence that the case of line 8 is executed. Let $\mathbf{m}$ and $\mathbf{m}'$ be respectively the values of $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$ when *traverse*$(X^{[\ell]})$ is invoked. Now, let $\mathbf{m}_3, \mathbf{m}_\Delta$ be such that $\mathbf{m}' = \mathbf{m}_3 \oplus \mathbf{m}_\Delta$ and such that upon completion of the call to *transfer_from_to* at line 9 we have that $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}_\Delta$ and $\mathbf{M}_\mathsf{f}[\ell-1] = \mathbf{m}_3$. Moreover, let $\mathbf{m}_2$ be the marking such that $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}_2$ upon completion of the assignment at line 11. Therefore we find that $\mathbf{M}_\mathsf{f}[\ell]$ is updated to $\mathbf{m}_\Delta \oplus \mathbf{m}_2$. Next consider the successful proper call *traverse*$(C^{[\ell-1]})$ of line 12 with $\mathbf{M}_\mathsf{i}[\ell-1] = \mathbf{m}_2$,

$\mathbf{M}_{\mathsf{f}}[\ell-1] = \mathbf{m}_3$. Observe that because the execution of $traverse(X^{[\ell]})$ yields the calls $traverse(C^{[\ell-1]})$ and $traverse(B^{[\ell]})$, we find that the number of times line 2 is executed in $traverse(C^{[\ell-1]})$ and $traverse(B^{[\ell]})$ is strictly less than n. Therefore, the induction hypothesis shows that there exists w_2 such that $w_2 \in L(C^{[\ell-1]})$ and $\mathbf{m}_2 \,[w_2\rangle\, \mathbf{m}_3$. Then comes the successful assert of line 13 followed by the successful proper call $traverse(B^{[\ell]})$ of line 14 with $\mathbf{M}_{\mathsf{i}}[\ell] = \mathbf{m}$ and $\mathbf{M}_{\mathsf{f}}[\ell] = \mathbf{m}_\Delta \oplus \mathbf{m}_2$. Again by induction hypothesis, there exists w_1 such that $w_1 \in L(B^{[\ell]})$ and $\mathbf{m} \,[w_1\rangle\, (\mathbf{m}_\Delta \oplus \mathbf{m}_2)$.

Next we conclude from the monotonicity property of PN that since $\mathbf{m}_2 \,[w_2\rangle\, \mathbf{m}_3$ then $(\mathbf{m}_2 \oplus \mathbf{m}_\Delta) \,[w_2\rangle\, (\mathbf{m}_3 \oplus \mathbf{m}_\Delta)$, hence that $\mathbf{m} \,[w_1\rangle\, (\mathbf{m}_2 \oplus \mathbf{m}_\Delta) \,[w_2\rangle\, (\mathbf{m}_3 \oplus \mathbf{m}_\Delta)$ and finally that $\mathbf{m} \,[w_1\, w_2\rangle\, \mathbf{m}'$ because $\mathbf{m}' = \mathbf{m}_3 \oplus \mathbf{m}_\Delta$. Finally since $w_1 w_2 \in L(X^{[\ell]})$ we conclude that $\mathbf{m}' \in [\mathbf{m}\rangle^{L(X^{[\ell]})}$ and we are done.

The left case (i.e. $p = (X^{[\ell]}, B^{[\ell-1]}C^{[\ell]}) \in \mathcal{P}^{[k]}$) is treated similarly. ◀

Part 2. In this section, we show that it is possible to construct a PNI N' such that the problem asking if the call to $traverse(A^{[k]})$ successfully returns can be reduced to a reachability problem for N'. Incidentally, we show that N' is a PNW, hence that the reachability problem for PN along finite-index CFL is decidable.

To describe N' we use a generalization of the net program formalism introduced by Esparza in [4] which enrich the instruction set with the test for 0 of a variable.

A *net program* is a finite sequence of *labelled commands*. Those commands have the following form, where $\ell, \ell', \ell_1, \ldots, \ell_k$ are *labels* taken from some arbitrary set, and x is a variable over the natural numbers, also called a *counter*.

ℓ: $x := x - 1$	ℓ: **return**
ℓ: $x := x + 1$	ℓ: **goto** ℓ_1 **or** $\cdots$ **or goto** ℓ_k (where $k \geq 1$)
ℓ: **assert** $x = 0$	ℓ: **gosub** ℓ'

A net program is *syntactically correct* if the labels of commands are pairwise different, and if the destinations of the **goto** and **gosub** commands corresponds to existing labels. (**goto** commands correspond to a possibly non deterministic jump while **gosub** commands correspond to a subroutine call.) A subroutine is a subsequence of the program commands which has a unique *entry label* identified by a *subroutine name*, and a unique *exit command* of the form ℓ: **return**. Also every command of the program belongs to exactly one subroutine. No **goto** commands leaves its enclosing subroutine. Finally, we require the existence of a level assignment to subroutines such that each subroutine only calls lower-level subroutines, which in turn only call lower-level subroutines, etc so as to prevent recursion.

A net program can only be executed once its variables have received initial values which we assume here to be 0. The semantics of net programs can be defined in a straightforward manner from the syntax (see [4] for more information). The only point to be remarked is that the command ℓ: $x := x - 1$ *fails* if $x = 0$, and causes abortion of the program.

The compilation of a syntactically correct net program to a PNI is straightforward and omitted due to space constraints. See [4] for the compilation.

At Alg. 3, 4, 5 and 6 is the net program that implements Alg. 1. In what follows assume S, the set of places of the underlying Petri net, to be $\{1, \ldots, d\}$ for $d \geq 1$. The counter variables of the net program are given by $\{x^{[i]}\}_{0 \leq i \leq k, x \in \mathcal{X}}$ and $\mathbf{M}_{\mathsf{f}}[0..k][1..d]$ $\mathbf{M}_{\mathsf{i}}[0..k][1..d]$ which arranges counters into two matrices of dimension $(k+1) \times d$. For clarity, our net programs use some abbreviations whose semantics is clear from the syntax, e.g. $\mathbf{M}_{\mathsf{i}}[\ell] := \mathbf{M}_{\mathsf{i}}[\ell] \oplus \mathbf{m}$ stands for $\mathbf{M}_{\mathsf{i}}[\ell][1] := \mathbf{M}_{\mathsf{i}}[\ell][1] + \mathbf{m}(1); [\ldots]; \mathbf{M}_{\mathsf{i}}[\ell][d] := \mathbf{M}_{\mathsf{i}}[\ell][d] + \mathbf{m}(d)$.

Let us now make a few observations of Alg. 3, 4, 5 and 6

• the execution starts with the subroutine **main** which sets up $\mathbf{M}_\mathsf{i}[\ell]$ and $\mathbf{M}_\mathsf{f}[\ell]$, then simulates the call *traverse*$(X^{[\ell]})$ and finally checks that the postcondition holds (label $\mathbf{0}_1$) before returning (label **success**).

• in subroutines $\mathbf{traverse}_j$, [...] stands for the code which is given at Alg. 5 and 6 according to the different cases that may occur. The code for the case $\mathbf{p}_i^j = (X^{[j]}, B^{[j-1]}C^{[j]})$ has been omitted for space reasons but it is easily inferred.

• the counter variables $\{x^{[i]}\}_{0\leq i\leq k, x\in\mathcal{X}}$ record the parameters of the calls to **traverse**. For instance, a call to *traverse*$(X^{[j]})$ is simulated in the net program by incrementing counter $x^{[j]}$ (which records that the parameter of *traverse* is $X^{[j]}$) and then calling subroutine $\mathbf{traverse}_j$. When the call executes, the corresponding variable is decremented.

• the **goto** command at label $\mathbf{traverse}_j$ simulates the non deterministic selection of a production rule $\mathbf{p}_i^j = (X^{[j]}, w)$ which will be fired next (if enabled else the program fails).

Algorithm 3: **main** & $\mathbf{traverse}_{i=\ell,\ldots,0}$

```
main:          M_i[ℓ] := M_i[ℓ] ⊕ m;
               M_f[ℓ] := M_f[ℓ] ⊕ m';
               x^[ℓ] := x^[ℓ] + 1;
               gosub traverse_ℓ;
0_1            assert
               M_i[0..ℓ] = ∅ = M_f[0..ℓ];
success:       return;
traverse_ℓ:    goto p_1^ℓ or ··· or goto p_{n_ℓ}^ℓ;
p_1^ℓ:         [...];
               ⋮
p_{n_ℓ}^ℓ:     [...];
exit^ℓ:        return;
               ⋮
traverse_0:    goto p_1^0 or ··· or goto p_{n_0}^0;
p_1^0:         [...];
               ⋮
p_{n_0}^0:     [...];
exit^0:        return;
```

Algorithm 4: $\mathbf{tr\_f}_\ell\mathbf{\_f}_{\ell-1}$

```
tr_f_ℓ_f_{ℓ-1}:  goto out or t_1 or ... or t_d ;
t_1:             M_f[ℓ][1] := M_f[ℓ][1] − 1;
                 M_f[ℓ−1][1] := M_f[ℓ−1][1]+1;
                 goto tr_f_ℓ_f_{ℓ−1};
                 [...];
t_d:             M_f[ℓ][d] := M_f[ℓ][d] − 1;
                 M_f[ℓ−1][d] := M_f[ℓ−1][d]+1;
                 goto tr_f_ℓ_f_{ℓ−1};
out:             return;
```

Algorithm 5: if $\mathbf{p}_i^j = (X^{[j]}, \sigma)$ then

```
p_i^j:   x^[j] := x^[j] − 1;
         M_i[j] := M_i[j] ⊖ I(σ);
         M_i[j] := M_i[j] ⊕ O(σ);
loop:    goto exit^j or s_1 or ... or s_d ;
s_1:     M_i[j][1] := M_i[j][1] − 1;
         M_f[j][1] := M_f[j][1] − 1;
         goto loop;
         [...];
s_d:     M_i[j][d] := M_i[j][d] − 1;
         M_f[j][d] := M_f[j][d] − 1;
         goto loop;
```

Algorithm 6: if $\mathbf{p}_i^j = (X^{[j]}, B^{[j]}C^{[j-1]})$ then

```
p_i^j:   x^[j] := x^[j] − 1;
         gosub tr_f_j_f_{(j−1)};
loop:    goto exitloop or s_1 or ... or s_d ;
s_1:     M_i[j][1] := M_i[j][1] + 1;
         M_f[j − 1][1] := M_f[j − 1][1] + 1;
         goto loop;
         [...];
s_d:     M_i[j][d] := M_i[j][d] + 1;
         M_f[j − 1][d] := M_f[j − 1][d] + 1;
         goto loop;
exitloop: c^[j−1] := c^[j−1] + 1;
         gosub traverse_{(j−1)};
0_2      assert
         M_i[0..j − 1] = ∅ = M_f[0..j − 1];
l1:      b^[j] := b^[j] + 1;
         goto traverse_j;
```

• the program is syntactically correct. First, observe that no **goto** commands leaves its enclosing subroutine. Second, we assign levels to subroutines as follows: **main** has level $\ell + 1$, $\mathbf{traverse}_j$ has level j for every $0 \leq j \leq \ell$ and $\mathbf{tr\_f}_j\mathbf{\_f}_{j-1}$ has level $j - 1$. Then it is routine to check that this level assignment satisfies the requirement. Moreover, thanks to the programming techniques that allow to implement the tail recursive call as a **goto** instead of **gosub** we find that the program is syntactically correct. (If we had used **gosub** everywhere, then the net program would be syntactically incorrect because of the recursion).

• the **assert** commands at labels $\mathbf{0}_1$ and $\mathbf{0}_2$ have a particular structure matching the level of the subroutines (level $\ell+1$ for $\mathbf{0}_1$ and j for $\mathbf{0}_2$). So, after compilation of the net program into a PNI N', if we set a mapping f from the places of N' to $\mathbb{N}$ such that c is mapped to i if $c \in \{\mathbf{M}_\mathsf{i}[i][j] \mid j \in \{1,\ldots,d\}\} \cup \{\mathbf{M}_\mathsf{f}[i][j] \mid j \in \{1,\ldots,d\}\}$ and every other place is mapped to $\ell+2$ then we find that N' is a PNW. Clearly, deciding whether **main** returns (i.e. reaches **success**) reduces to PNW reachability. Therefore, by Thm. 3, it is decidable whether **main** returns.

▶ **Lemma 5.** *Let $\ell \in \{0,\ldots,k\}$, $X^{[\ell]} \in \mathcal{X}^{[k]}$, and $\mathbf{m},\mathbf{m}' \in \mathbb{M}[S]$. Then the proper call traverse$(X^{[\ell]})$ with $\mathbf{M}_\mathsf{i}[\ell] = \mathbf{m}$, $\mathbf{M}_\mathsf{f}[\ell] = \mathbf{m}'$ successfully returns iff* **main** *returns.*

Hence from Lem. 2, 4 and 5, we conclude the following.

▶ **Corollary 6.** *The reachability problem for* PN *along finite-index* CFL *can be reduced to the reachability problem for* PNW*.*

4 From PNW reachability to PN reachability along fiCFL

In this section, we show that the reachability problem for PNW can be reduced to the reachability problem of PN along finite-index CFL. To this aim, let $N = (S,T,F = \langle Z,I,O\rangle,\mathbf{m}_\imath)$ be a PNW, $\mathbf{m}_f \in \mathbb{M}[S]$ a marking, and $f\colon S \to \mathbb{N}$ an index function such that (1) holds.

Let $S = \{s_1,\ldots,s_{n+1}\}$ and $T = \{t_1,\ldots,t_m\}$. Because it simplifies the presentation we will make a few assumptions that yield no loss of generality. (i) For every $i \in \{1,\ldots,n\}$, we have $f(s_i) \leq f(s_{i+1})$, (ii) $\mathbf{m}_\imath = [\![s_{n+1}]\!]$, $\mathbf{m}_f = \varnothing$, (iii) $Z(t_1) \subseteq Z(t_2) \subseteq \cdots \subseteq Z(t_m) \subseteq \{s_1,\ldots,s_n\}$, and (iv) for every $t \in T$, if $s \in Z(t)$ then $O(t)(s) = 0$ (see [15], Lemma 2.1). Notice that the Petri net N can not test if the place s_{n+1} is empty or not.

In the following, we show that it is possible to construct a Petri net (without inhibitor arcs) N', a marking $\mathbf{m}'_f$, and a finite-index CFL L such that: $\mathbf{m}_f \in [\mathbf{m}_\imath\rangle_N^{T^*}$ iff $\mathbf{m}'_f \in [\mathbf{m}'_\imath\rangle_{N'}^{L}$.

Constructing the Petri net N': Let $N' = (S',T',F' = \langle I',O'\rangle,\mathbf{m}'_\imath)$ be a PN which consists in $n+1$ unconnected PN widget: the widget N_0 given by N without tests for zero (i.e. $Z(t)$ is set to $\varnothing$ for every $t \in T$) and the widgets $N_1,\ldots,N_n$ where each $N_i = (\{r_i\},\{p_i,c_i\},F_i,\varnothing)$ where $F_i(p_i) = \langle\varnothing,[\![r_i]\!]\rangle$ and $F_i(c_i) = \langle[\![r_i]\!],\varnothing\rangle$. N_i is depicted as follows: 

. Finally, define $\mathbf{m}'_\imath \in \mathbb{M}[S']$ to be $\mathbf{m}'_\imath(s) = \mathbf{m}_\imath(s)$ for $s \in S$ and 0 elsewhere; and $\mathbf{m}'_f = \varnothing$.

Since we have the ability to restrict the possible sequences of transitions that fire in N', we can enforce the invariant that the sum of tokens in s_i and r_i stays constant. To do so it suffices to force that whenever a token is produced in s_i then a token is consumed from r_i and vice versa. Call L the language enforcing that invariant. Then, let $\mathbf{m}$ be a marking such that $\mathbf{m}(s_i) = \mathbf{m}(r_i) = 0$, observe that by firing from $\mathbf{m}$ a sequence of the form: (i) p_i repeated n times, (ii) any sequence $w \in L$ and (iii) c_i repeated n times; the marking $\mathbf{m}'$ that is reached is such that $\mathbf{m}'(s_i) = \mathbf{m}'(r_i) = 0$. This suggests that to simulate faithfully a transition t_0 of N that does test s_i for 0 we allow the occurrence of the counterpart of t_0 in N_0 right before (i) or right after (iii) only. In what follows, we build upon the above idea the language L_n which, as we we will show, coincides with the finite-index approximation of some CFG.

We need the following notation. Given a word $v \in \Sigma^*$ and $\Theta \subseteq \Sigma$, we define $v|_\Theta$ to be the word obtained from v by erasing all the symbols that are not in Θ. We extend it to languages as follows: Let $L \subseteq \Sigma^*$. Then $L|_\Theta = \{u|_\Theta \mid u \in L\}$.

Constructing the language L_n: For every $j \in \{1,\ldots,m\}$, let $u_j = p_1^{i_1}p_2^{i_2}\cdots p_n^{i_n}$ and $v_j = c_1^{k_1}c_2^{k_2}\cdots c_n^{k_n}$ be two words over the alphabet T' such that $i_\ell = I(t_j)(s_\ell)$ and $k_\ell =$

$O(t_j)(s_\ell)$ for all $\ell \in \{1,\dots,n\}$. Observe that firing the sequence of transitions u_j (resp. v_j) will produce in (resp. consume from) the place r_ℓ (with $1 \leq \ell \leq n$) the same number of tokens that the transition t_j will consume from (resp. produce in) the place s_ℓ. Therefore, firing the sequence of transitions $v_j t_j u_j$ keeps unchanged the total number of tokens in $\{s_i, r_i\}$ for each $i \in \{1,\dots,n\}$.

Let L_0 be a regular language over the alphabet T' defined as follows:

$$L_0 = \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \emptyset \text{ in the Petri net } N\}^* \ .$$

Next, we define the CFL $L_1,\dots,L_n$ such that for every $\ell \in \{1,\dots,n\}$ we have:

$$L_\ell = \left(\{p_\ell^i \cdot v \cdot c_\ell^i \mid i \in \mathbb{N}, v \in L_{\ell-1}\} \cup \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \{s_1,\dots,s_\ell\} \text{ in } N\}\right)^* \ .$$

Observe that, for every $j \in \{1,\dots,n\}$, the sum of tokens in the places s_j and r_j is preserved after firing a sequence of transitions in L_ℓ. Hence, the places r_ℓ and s_ℓ are empty after firing a sequence of transitions $w \in L_\ell$ from a marking where these places are empty. Also notice that these places can become non-empty during the execution of w. For instance, if w fires $p_\ell^i \cdot v \cdot c_\ell^i$ which first produces i tokens in r_ℓ, then executes v and finally consumes i tokens from r_ℓ. It is worth pointing that along v transitions which produce and/or consume tokens in s_ℓ can be fired. However, since $v \in L_{\ell-1}$ no transition t such that $s_\ell \in Z(t)$ is allowed, that is no test of s_ℓ for 0 is allowed along v. The language L_ℓ imposes that the place s_ℓ can only be tested for 0 along $v_j \cdot t_j \cdot u_j \in L_\ell \setminus L_{\ell-1}$. The underlying idea is that L_ℓ allows to test s_ℓ for 0 provided the places s_ℓ and r_ℓ (and inductively all the places s_j and r_j for $j \leq \ell$) are empty.

It is routine to check that $L_0 \subseteq L_1 \subseteq \cdots \subseteq L_n$ (since $L_{\ell-1} \subseteq \{p_\ell^i \cdot v \cdot c_\ell^i \mid i \in \mathbb{N}, v \in L_{\ell-1}\}$) and $L_n|_T = T^*$ (since $L_n \supseteq \bigcup_{i=0}^n \{v_j \cdot t_j \cdot u_j \mid Z(t_j) = \{s_1,\dots,s_i\}\}$[3]). Also, L_0 is a regular language and therefore there exists a CFG G_0 and a variable A_0 of G_0 such that $L^{(1)}(A_0) = L_0$. Now, let us assume that for L_i there exists a CFG G_i and a variable A_i such that $L^{(i+1)}(A_i) = L_i$. From the definition of L_{i+1} it is routine to check that there exists a CFG G_{i+1} and a variable A_{i+1} such that $L^{(i+2)}(A_{i+1}) = L_{i+1}$. Finally we find that L_n can be captured by the $n+1$-index approximation of a CFG.

Let us make a few observations about the transitions of N' which were carrying out 0 test in N. In L_ℓ no transition t such that $s_{\ell+1} \in Z(t)$ is allowed, that is no test of place $s_{\ell+1}$ for 0 is allowed along any word of L_ℓ. The language L_ℓ imposes that the place s_ℓ can only be tested for 0 along T_ℓ. The intuition is that L_ℓ allows to test s_ℓ for 0 provided all places s_j and r_j for $j \leq \ell$ are empty.

The relation between the reachability problem for N and the reachability problem for N' along L_n is given by the following lemma (whose proof can be found in [1]):

▶ **Lemma 7.** $\mathbf{m}_f(=\varnothing) \in [\mathbf{m}_\iota\rangle_N$ *if and only if* $\mathbf{m}'_f(=\varnothing) \in [\mathbf{m}'_\iota\rangle_{N'}^{L_n}$.

As an immediate consequence of Lemma 7, we obtain the following result:

▶ **Corollary 8.** *The reachability problem for* PNW *can be reduced, in polynomial time, to the reachability problem for* PN *along finite-index* CFL.

5 Conclusion

In this paper, we have shown that the problem of checking whether the intersection of a finite-index context-free language and a Petri net language is empty is decidable. This result is obtained through a non-trivial reduction to the reachability problem for Petri nets with

[3] Note that if $i = 0$ then $\{s_1,\dots,s_i\} = \emptyset$.

weak inhibitor arcs. On the other hand, we have proved that the reachability problem for Petri nets with weak inhibitor arcs can be reduced, in polynomial time, to the emptiness problem of the language obtained from the intersection of a finite-index context-free language and a Petri net language.

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating petri net reachability along context-free traces. *CoRR*, abs/1105.1657, 2011.

2 Rémi Bonnet. The reachability problem for vector addition systems with one zero-test. In *MFCS '11: Proc. 36th Int. Symp. on Mathematical Foundations of Computer Science*, volume 6907 of *LNCS*, pages 145–157. Springer, 2011.

3 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252. ACM Press, 1977.

4 Javier Esparza. Decidability and complexity of petri net problems – an introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998.

5 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Information Processing Letters*, 111:614–619, 2011.

6 Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newton's method for ω-continuous semirings. In *ICALP '08*, volume 5126 of *LNCS*, pages 14–26. Springer, 2008. Invited paper.

7 Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM*, 57(6):33:1–33:47, 2010.

8 Pierre Ganty, Benjamin Monmege, and Rupak Majumdar. Bounded underapproximations. In *CAV '10*, volume 6174 of *LNCS*, pages 600–614. Springer, 2010.

9 Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *CAV '97*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

10 Michel Henri Théodore Hack. Decidability questions for petri nets. Technical Report 161, MIT, 1976.

11 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, third edition, July 2006.

12 Martin Lange and Hans Leiß. To CNF or not to CNF ? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2008-2010.

13 Jérôme Leroux. Vector addition system reachability problem (a short self-contained proof). In *POPL '11*, pages 307–316. ACM, 2011.

14 Mark Luker. A family of languages having only finite-index grammars. *Information and Control*, 39(1):14–18, 1978.

15 Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Electr. Notes Theor. Comput. Sci*, 223:239 – 264, 2008. RP '08.

16 Arto Salomaa. On the index of a context-free grammar and language. *Information and Control*, 14(5):474 – 477, 1969.