# On the Upward/Downward Closures of Petri Nets[*]

**Mohamed Faouzi Atig[1], Roland Meyer[†2], Sebastian Muskalla[3], and Prakash Saivasan[4]**

1 **Uppsala University, Sweden**
mohamed_faouzi.atig@it.uu.se

2 **TU Braunschweig, Germany**
roland.meyer@tu-braunschweig.de

3 **TU Braunschweig, Germany**
s.muskalla@tu-braunschweig.de

4 **TU Braunschweig, Germany**
p.saivasan@tu-braunschweig.de

## Abstract

We study the size and the complexity of computing finite state automata (FSA) representing and approximating the downward and the upward closure of Petri net languages with coverability as the acceptance condition. We show how to construct an FSA recognizing the upward closure of a Petri net language in doubly-exponential time, and therefore the size is at most doubly exponential. For downward closures, we prove that the size of the minimal automata can be non-primitive recursive. In the case of BPP nets, a well-known subclass of Petri nets, we show that an FSA accepting the downward/upward closure can be constructed in exponential time. Furthermore, we consider the problem of checking whether a simple regular language is included in the downward/upward closure of a Petri net/BPP net language. We show that this problem is EXPSPACE-complete (resp. NP-complete) in the case of Petri nets (resp. BPP nets). Finally, we show that it is decidable whether a Petri net language is upward/downward closed. To this end, we prove that one can decide whether a given regular language is a subset of a Petri net coverability language.

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** Petri nets, BPP nets, downward closure, upward closure

## 1 Introduction

Petri nets are a popular model of concurrent systems [19]. Petri net languages (with different acceptance conditions) have been extensively studied during the last years, including deciding their emptiness (which can be reduced to reachability) [36, 28, 30, 31], their regularity [44, 14], their context-freeness [43, 32], and many other decision problems (e.g. [22, 2, 20]). In this paper, we consider the class of Petri net languages with coverability as the acceptance condition (i.e. the set of sequences of transition labels occurring in a computation reaching a marking greater than or equal to a given final marking).

We address the problem of computing the *downward* and the *upward* closure of Petri net languages. The downward closure of a language $\mathcal{L}$, denoted by $\mathcal{L}\downarrow$, is the set of all subwords, all words that can be obtained from words in $\mathcal{L}$ by deleting letters. The upward closure of $\mathcal{L}$, denoted by $\mathcal{L}\uparrow$, is the set of all superwords, all words that can be obtained

[*] The conference version of this paper has been published in the proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017 [8].

[†] A part of this work was carried out when the author was at Aalto University.

from words in $\mathcal{L}$ by inserting letters. It is well-known that, for any language, the downward and upward closure are regular and can be described by a *simple regular expression (SRE)*. However, such an expression is in general not computable, e.g. for example, it is not possible to compute the downward closure of languages recognized by lossy channel systems [38].

In this paper, we first consider the problem of constructing a finite state automaton (FSA) accepting the upward/downward closure of a Petri net language. We give an algorithm that computes an FSA of doubly-exponential size for the upward closure in doubly-exponential time. This is done by showing that every minimal word results from a computation of length at most doubly exponential in the size of the input. Our algorithm is also optimal since we present a family of Petri net languages for which the minimal finite state automata representing their upward closure are of doubly-exponential size.

Our second contribution is a family of Petri net languages for which the size of the minimal finite state automata representing the downward closure is non-primitive recursive. To prove this, we resort to a construction due to Mayr and Meyer [37]. It gives a family of Petri nets whose language, albeit finite, is astronomically large: It contains Ackermann many words. The downward closure of Petri net languages has been shown to be effectively computable [22]. The algorithm is based on the Karp-Miller tree [27], which has non-primitive recursive complexity.

Furthermore, we consider the SRE inclusion problem which asks whether the language of a simple regular expression is included in the downward/upward closure of a Petri net language. The idea behind SRE inclusion is to stratify the problem of computing the downward/upward closure: Rather than having an algorithm computing all information about the language, we imagine to have an oracle (e.g. an enumeration) making proposals for SREs that could be included in the downward/upward closure. The task of the algorithm is merely to check whether a proposed inclusion holds. We show that this problem is EXPSPACE-complete in both cases. In the case of upward closures, we prove that SRE inclusion boils down to checking whether the set of minimal words of the given SRE is included in the upward closure. In the case of downward closures, we reduce the problem to the simultaneous unboundedness problem for Petri nets, which is EXPSPACE-complete [14].

We also study the problem of checking whether a Petri net language actually is upward or downward closed. This is interesting as it means that an automaton for the closure, which we can compute with the aforementioned methods, is a precise representation of the system's behavior. We show that the problem of being upward/downward closed is decidable for Petri nets. The result is a consequence of a more general decidability that we believe is of independent interest. We show that checking whether a regular language is included in a Petri net language (with coverability as the acceptance condition) is decidable. Here, we rely on a decision procedure for trace inclusion due to Esparza et al. [26].

Finally, we consider *BPP*[1] *nets* [16], a subclass of Petri nets defined by a syntactic restriction: Every transition is allowed to consume at most one token in total. We show that we can compute finite state automata accepting the upward and the downward closure of a BPP net language in exponential time. The size of the FSA is also exponential. Our algorithms are optimal as we present a family of BPP net languages for which the minimal FSA representing their upward/downward closure have exponential size. Furthermore, we consider the SRE inclusion problem. We show that, in the case of BPP nets, it is NP-complete for both, inclusion in the upward and in the downward closure. To prove the upper bound, we reduce to the satisfiability problem for existential Presburger arithmetic

[1] BPP stands for *basic parallel processes*, a notion from process algebra.

(which is known to be NP-complete [42]). The hardness is by a reduction from SAT to the emptiness of BPP net languages, which in turn reduces to SRE inclusion.

The following table summarizes our results.

| | Petri nets | BPP nets |
|---|---|---|
| Computing the upward closure | Doubly exponential* | Exponential* |
| Computing the downward closure | Non-primitive recursive* | Exponential* |
| SRE in downward closure | EXPSPACE-complete | NP-complete |
| SRE in upward closure | EXPSPACE-complete | NP-complete |
| Being downward/upward closed | Decidable | |
| Containing regular language | Decidable | |

$^*$: Time for the construction and size of the resulting FSA, optimal.

### Related Work

Several constructions have been proposed in the literature to compute finite state automata recognizing the downward/upward closure. In the case of Petri net languages (with various acceptance conditions including reachability), it has been shown that the downward closure is effectively computable [22]. With the results in this paper, the computation and the state complexity have to be non-primitive recursive. For the languages generated by context-free grammars, effective computability of the downward closure is due to [45, 21, 11, 9]. For the languages recognized by one-counter automata, a strict subclass of the context-free languages, it has been shown how to compute in polynomial time a finite state automaton accepting the downward/upward closure of the language [7]. The effective computability of the downward closure has also been shown for stacked counter automata [48]. In [47], Zetzsche provides a characterization for a class of languages to have an effectively computable downward closure. It has been used to prove the effective computability of downward closures of higher-order pushdown automata and higher-order recursion schemes [23, 10]. The downward closure of the languages of lossy channel systems is not computable [38].

The computability results discussed above have been used to prove the decidability of verification problems and to develop approximation-based program analysis methods (see e.g. [6, 5, 4, 29, 35, 49]). Throughout the paper, we will give hints to applications in verification.

## 2 Preliminaries

In this section, we fix some basic definitions and notations that will be used throughout the paper. For every $i, j \in \mathbb{N}$, we use $[i..j]$ to denote the set $\{k \in \mathbb{N} \mid i \leqslant k \leqslant j\}$.

### Languages and Closures

Let $\Sigma$ be a finite alphabet. We use $\Sigma_\varepsilon$ to denote $\Sigma \cup \{\varepsilon\}$. The length of a word $u$ over $\Sigma$ is denoted by $|u|$, where $|\varepsilon| = 0$. Let $k \in \mathbb{N}$ be a natural number, we use $\Sigma^k$ (resp. $\Sigma^{\leqslant k}$) to denote the set of all words of length equal (resp. smaller or equal) to $k$. A language $\mathcal{L}$ over $\Sigma$ is a (possibly infinite) set of finite words over $\Sigma$.

Let $\Gamma$ be a subset of $\Sigma$. Given a word $u \in \Sigma^*$, we denote by $\pi_\Gamma(u)$ the projection of $u$ over $\Gamma$, i.e. the word obtained from $u$ by erasing all the letters that are not in $\Gamma$.

The *Parikh image* of a word [39] counts the number of occurrences of all letters while forgetting about their positioning. Formally, the function $\Psi : \Sigma^* \mapsto \mathbb{N}^\Sigma$ takes a word $w \in \Sigma^*$ and gives the function $\Psi(w) : \Sigma \to \mathbb{N}$ defined by $(\Psi(w))(a) = \left|\pi_{\{a\}}(w)\right|$ for all $a \in \Sigma$.

The *subword relation* $\preceq\ \subseteq \Sigma^* \times \Sigma^*$ [25] between words is defined as follows: A word $u = a_1 \dots a_n$ is a subword of $v$, denoted $u \preceq v$, if $u$ can be obtained by deleting letters from $v$ or, equivalently, if $v = v_0 a_1 v_1 \dots a_n v_n$ for some $v_0, \dots, v_n \in \Sigma^*$.

Let $\mathcal{L}$ be a language over $\Sigma$. The *upward closure* of $\mathcal{L}$ consists of all words that have a subword in the language, $\mathcal{L}\!\uparrow = \{v \in \Sigma^* \mid \exists u \in \mathcal{L}\colon u \preceq v\}$. The *downward closure* of $\mathcal{L}$ contains all words that are dominated by a word in the language, $\mathcal{L}\!\downarrow = \{u \in \Sigma^* \mid \exists v \in \mathcal{L}\colon u \preceq v\}$. Higman showed that the subword relation is a well-quasi ordering [25], which means that every set of words $\mathcal{L} \subseteq \Sigma^*$ has a finite *basis*, a finite set of *minimal elements* $v \in \mathcal{L}$ such that $\nexists u \in \mathcal{L} : u \neq v, u \preceq v$. With finite bases, $\mathcal{L}\!\uparrow$ and $\mathcal{L}\!\downarrow$ are guaranteed to be regular for every language $\mathcal{L} \subseteq \Sigma^*$ [24]. Indeed, they can be expressed using the subclass of simple regular languages defined by so-called *simple regular expressions* [1].

These SREs are choices among *products* $p$ that interleave single letters $a$ or $(a + \varepsilon)$ with iterations over letters from subsets $\Gamma \subseteq \Sigma$ of the alphabet:

$$\mathit{sre} ::= p \;\mid\; \mathit{sre} + \mathit{sre} \qquad p ::= a \;\mid\; (a+\varepsilon) \;\mid\; \Gamma^* \;\mid\; p.p \quad .$$

Note that this is an extension of the classical definition that we introduce so that we can also represent upward closures. The syntactic size of an SRE *sre* is denoted by $|\mathit{sre}|$ and defined as expected, every piece of syntax contributes to it.

## Finite State Automata

A *finite state automaton (FSA)* $A$ is a tuple $(\Sigma, Q, \rightarrow, q_{init}, Q_f)$ where $Q$ is a finite non-empty set of states, $\Sigma$ is the finite input alphabet, $\rightarrow \subseteq Q \times \Sigma_\varepsilon \times Q$ is the non-deterministic transition relation, $q_{init} \in Q$ is the initial state, and $Q_f \subseteq Q$ is the set of final states. We represent a transition $(q, a, q') \in \rightarrow$ by $q \xrightarrow{a} q'$ and generalize the relation to words in the expected way. The language of finite words accepted by $A$ is denoted by $\mathcal{L}(A)$. The size of $A$, denoted $|A|$, is defined by $|Q| + |\Sigma|$. An FSA is *minimal* for its language $\mathcal{L}(A)$ if there is no FSA $B$ with $\mathcal{L}(A) = \mathcal{L}(B)$ with a strictly smaller number of states.

## Petri Nets

A *(labeled) Petri net* is a tuple $N = (\Sigma, P, T, F, \lambda)$ [41]. Here, $\Sigma$ is a finite alphabet, $P$ a finite set of *places*, $T$ a finite set of *transitions* with $P \cap T = \emptyset$, $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ a *flow function*, and $\lambda : T \mapsto \Sigma_\varepsilon$ a labeling function. When convenient, we will assume that the places are ordered, $P = [1..\ell]$ for some $\ell \in \mathbb{N}$. For a place or transition $x \in P \cup T$, we define the *preset* to consist of the elements that have an arc to $x$, ${}^\bullet x = \{y \in P \cup T \mid F(y, x) > 0\}$. The *postset* is defined similarly, $x^\bullet = \{y \in P \cup T \mid F(x, y) > 0\}$.

To define the semantics of Petri nets, we use *markings* $M : P \to \mathbb{N}$ that assign to each place a number of *tokens*. A marking $M$ *enables* a transition $t$, denoted $M[t\rangle$, if $M(p) \geqslant F(p, t)$ for all $p \in P$. A transition $t$ that is enabled may be *fired*, leading to the new marking $M'$ defined by $M'(p) = M(p) - F(p, t) + F(t, p)$ for all $p \in P$, i.e. $t$ consumes $F(p, t)$ tokens and produces $F(t, p)$ tokens in $p$. We write the firing relation as $M[t\rangle M'$. A *computation* $\pi = M_0[t_1\rangle M_1 \cdots [t_m\rangle M_m$ consists of markings and transitions. We extend the firing relation to transition sequences $\sigma \in T^*$ in the straightforward manner and also write

$\pi = M_0[\sigma\rangle M_m$. A marking $M$ is *reachable* from an initial marking $M_0$ if $M_0[\sigma\rangle M$ for some $\sigma \in T^*$. A marking $M$ covers another marking $M_f$, denoted $M \geqslant M_f$, if $M(p) \geqslant M_f(p)$ for all $p \in P$. A marking $M_f$ is *coverable* from $M_0$ if there is a marking $M$ reachable from $M_0$ that covers $M_f$, $M_0[\sigma\rangle M \geqslant M_f$ for some $\sigma \in T^*$.

A *Petri net instance* $(N, M_0, M_f)$ consists of a Petri net $N$ together with an initial marking $M_0$ and final marking $M_f$ for $N$. Given a Petri net instance $(N, M_0, M_f)$, the associated *covering language* is

$$\mathcal{L}(N, M_0, M_f) = \{\lambda(\sigma) \mid \sigma \in T^*,\ M_0[\sigma\rangle M \geqslant M_f\}\ ,$$

where the labeling function $\lambda$ is extended to sequences of transitions in the straightforward manner. Given a natural number $k \in \mathbb{N}$, we define

$$\mathcal{L}_k(N, M_0, M_f) = \left\{\lambda(\sigma) \;\middle|\; \sigma \in T^{\leqslant k},\ M_0[\sigma\rangle M \geqslant M_f\right\}$$

to be the set of words accepted by computations of length at most $k$.

Let $max(F)$ denote the maximum of the range of $F$. The size of the Petri net $N$ is

$$|N| = |\Sigma| + |P| \cdot |T| \cdot (1 + \lceil log_2(1 + max(F))\rceil)\ ,$$

Similarly, the size of a marking $M$ is

$$|M| = |P| \cdot (1 + \lceil log_2(1 + max(M))\rceil)\ ,$$

where $max(M)$ denotes the maximum of the range of $M$. The size of a Petri net instance $(N, M_0, M_f)$ is $|(N, M_0, M_f)| = |N| + |M_0| + |M_f|$. This means we consider the the binary encoding of numbers occurring in markings and the flow function. In contrast, we define the *token count* $tc(M) = \Sigma_{p \in P} M(p)$ of a marking $M$ to be the sum of all tokens assigned by $M$, i.e. the size of the unary encoding of $M$.

A Petri net $N$ is said to be a *BPP net* (or *communication-free*) if every transition consumes at most one token from one place (i.e. $\Sigma_{p \in P} F(p, t) \leqslant 1$ for every $t \in T$).

## 3 Upward Closures

We consider the problem of constructing a finite state automaton accepting the upward closure of a Petri net and a BPP net language, respectively. The upward closure offers an over-approximation of the system behavior that is useful for verification purposes [35].

**Computing the upward closure**
**Given:** A Petri net instance $(N, M_0, M_f)$.
**Compute:** An FSA $A$ with $\mathcal{L}(A) = \mathcal{L}(N, M_0, M_f)\uparrow$.

### 3.1 Petri Nets

We prove a doubly-exponential upper bound on the size of the finite state automaton representing the upward closure of a Petri net language. Then, we present a family of Petri net languages for which the minimal finite state automata representing their upward closure have a size doubly exponential in the size of the input.

### Upper Bound

Fix the Petri net instance $(N, M_0, M_f)$ of interest and let $n$ be its size.

▶ **Theorem 1.** *One can construct an FSA of size $\mathcal{O}\Big(2^{2^{poly(n)}}\Big)$ for $\mathcal{L}(N, M_0, M_f)\uparrow$.*

The remainder of the section is devoted to proving the theorem. We will show that every minimal word results from a computation of length at most $O(2^{2^{\text{poly}(n)}})$. Let us call such computations the minimal ones. Let $k$ be a bound on the length of the minimal computations. This means the language $\mathcal{L}_k(N, M_0, M_f)$ contains all minimal words of $\mathcal{L}(N, M_0, M_f)$. Furthermore, $\mathcal{L}_k(N, M_0, M_f) \subseteq \mathcal{L}(N, M_0, M_f)$ and therefore the equality $\mathcal{L}_k(N, M_0, M_f)\uparrow = \mathcal{L}(N, M_0, M_f)\uparrow$ holds. Now we can use the following lemma to construct a finite automaton whose size is $\mathcal{O}\Big(2^{2^{\text{poly}(|n|)}}\Big)$ and that accepts $\mathcal{L}_k(N, M_0, M_f)$. Without an increase in size, this automaton can be modified to accept $\mathcal{L}_k(N, M_0, M_f)\uparrow$: Add for each state $q$ and for each symbol $a \in \Sigma$ a loop $q \xrightarrow{a} q$.

▶ **Lemma 2.** *For every $k \in \mathbb{N}$, one can construct an FSA of size $\mathcal{O}\big((k+2)^{poly(n)}\big)$ that accepts $\mathcal{L}_k(N, M_0, M_f)$.*

**Proof.** If there is a word $w \in \mathcal{L}_k(N, M_0, M_f)$, then there is a run of the form $M_0[\sigma\rangle M'$ with $M' \geqslant M_f$ and $|\sigma| \leqslant k$. Any place $p$ can have at most $M_0(p) + k \cdot 2^n$ tokens in $M'$. Note that $M_0(p) \leqslant 2^n$. With this observation, we construct the required finite state automaton $A = (\Sigma, Q, \rightarrow, q_{init}, Q_f)$ as follows.

The set of states is $Q = (P \rightarrow [0..(k+1)\cdot 2^n]) \times [0..k]$. The first component stores the token count for each place $p \in [1..\ell]$ (i.e. a marking), the second component counts the number of steps that have been executed so far. For each transition $t \in T$ of the Petri net and each state $(M, i)$ with $M(p) \geqslant F(p, t)$ for all $p$ and $i < k$, there is a transition from $(M, i)$ to $(M', i+1)$ in $\rightarrow$, where $M[t\rangle M'$. It is labeled by $\lambda(t)$. The initial state is $q_{init} = (M_0, 0)$, and a state $(M', i)$ is final if $M'$ covers $M_f$. By the construction of the automaton, it is clear that $(M_0, 0) \xrightarrow{w} (M', j)$ with $(M', j)$ final iff there is a $\sigma \in T^{\leqslant k}$ such that $M_0[\sigma\rangle M'$ with $M' \geqslant M_f$. Hence we have $\mathcal{L}(A) = \mathcal{L}_k(N, M_0, M_f)$. We estimate the size of $Q$ by

$$\begin{aligned}|Q| &= |(P \rightarrow [0..(k+1)\cdot 2^n])| \cdot |[0..k]| \\ &= ((k+1)\cdot 2^n)^\ell \cdot (k+1) \leqslant ((k+1)\cdot 2^n)^n \cdot (k+1) = (k+1)^{n+1} \cdot 2^{n^2} \\ &\leqslant (k+2)^{n+1} \cdot (k+2)^{n^2} \leqslant (k+2)^{n+1+n^2} \in \mathcal{O}\Big((k+2)^{\text{poly}(n)}\Big)\ .\end{aligned}$$

Since we can assume $|\Sigma| \leqslant n$, we have that $|A| = |Q| + |\Sigma|$ is in $\mathcal{O}\big((k+2)^{\text{poly}(n)}\big)$. ◀

It remains to show that every minimal word results from a computation of length at most doubly exponential in the size of the input. This is the following proposition.

▶ **Proposition 3.** *For every computation $M_0[\sigma\rangle M \geqslant M_f$, there is $M_0[\sigma'\rangle M' \geqslant M_f$ with $\lambda(\sigma') \preceq \lambda(\sigma)$ and $|\sigma'| \leqslant 2^{2^{cn\log n}}$, where $c$ is a constant.*

Our proof is an adaptation of Rackoff's technique to show that coverability can be solved in EXPSPACE [40]. Rackoff derives a bound (similar to ours) on the length of the shortest computations that cover a given marking. Rackoff's proof has been generalized to different settings, e.g. to BVAS in [15]. Lemma 5.3 in [33] claims that Rackoff's original proof already implies Proposition 3. This is not true as shown by the following example.

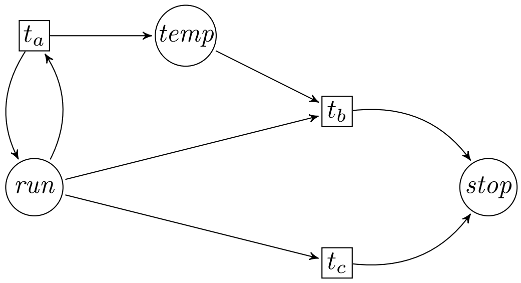

**Figure 1.** The flow relation of the Petri net $N_{ce}$.

▶ **Example 4.** Consider the Petri net $N_{ce} = (\{a, b, c\}, \{run, temp, stop\}, \{t_a, t_b, t_c\}, F, \lambda)$, where the flow relation is given by Figure 1 and we have $\lambda(t_a) = a$, $\lambda(t_b) = b$, and $\lambda(t_c) = c$. Consider the initial marking $M_0 = (1, 0, 0)$ with one token on *run* and no token elsewhere, and the final marking $M_f = (0, 0, 1)$ that requires one token on *stop*. We have $\mathcal{L}(N, M_0, M_f) = a^+b \cup a^*c$ and thus $\mathcal{L}(N, M_0, M_f)\!\uparrow = \Sigma^*a\Sigma^*b\Sigma^* \cup \Sigma^*c\Sigma^*$.

We may compute Rackoff's bound [40], and obtain that if there is a computation covering $M_f$ from $M_0$, then there is one consisting of at most one transition. Indeed, the computation $M_0[t_c\rangle M_f$ is covering.

However, computations whose length is within Rackoff's upper bound do not necessarily generate all minimal words: We have that *ab* is a minimal word in $\mathcal{L}(N, M_0, M_f)\uparrow$, but the shortest covering computation that generates *ab* is $M_0[t_a\rangle(1, 1, 0)[t_b\rangle M_f$, consisting of 3 markings and 2 transitions.

To handle labeled Petri nets, Rackoff's proof needs two amendments. First, it is not sufficient to consider the shortest covering computations. Instead, we have to consider computations long enough to generate all minimal words. Second, Rackoff's proof splits a firing sequence into two parts and replaces the second part by a shorter one. In our case, we need that the shorter word is a subword of the original one.

We now elaborate on Rackoff's proof strategy and give the required definitions, then we explain in more detail our adaptation, and finally give the technical details.

We assume that the places are ordered, i.e. $P = [1..\ell]$. Rackoff's idea is to relax the definition of the firing relation and allow for negative token counts on the last $i + 1$ to $\ell$ places. With a recurrence over the number of places, he then obtains a bound on the length of the computations that keep the first $i$ places positive.

Formally, an *pseudo-marking* of $N$ is a function $M : P \to \mathbb{Z}$. For $i \in [1..\ell]$, a pseudo-marking marking $M$ *i-enables* a transition $t \in T$ if $M(j) \geqslant F(j, t)$ for all $j \in [1..i]$. Firing $t$ yields a new pseudo-marking $M'$, denoted $M[t\rangle_i M'$, with $M'(p) = M(p) - F(p, t) + F(t, p)$ for all $p \in P$. A computation $\pi = M_0[t_1\rangle_i M_1 \ldots [t_m\rangle_i M_m$ is *i-non-negative* if for each marking $M_k$ with $k \in [0..m]$ and each place $j \in [1..i]$, we have $M_k(j) \geqslant 0$. We assume a fixed marking $M_f$ to be covered. The computation $\pi$ is *i-covering* (wrt. $M_f$) if $M_m(j) \geqslant M_f(j)$ for all $j \in [1..i]$. Given two computations $\pi_1 = M_0[t_1\rangle_i \cdots [t_k\rangle_i M_k$ and $\pi_2 = M'_0[t'_1\rangle_i \cdots [t'_s\rangle_i M'_s$ such that $M_k(j) = M'_0(j)$ for all $j \in [1..i]$, we define their *i-concatenation* $\pi_1 \cdot_i \pi_2$ to be the computation $M_0[t_1\rangle_i \cdots [t_k\rangle_i M_k[t'_1\rangle_i M''_{k+1} \cdots [t'_s\rangle_i M''_{k+s}$.

Rackoff's result provides a bound on the length of the shortest $i$-covering computations. Since we have to generate all minimal words, we will specify precisely which computations to consider (not only the shortest ones). Moreover, Rackoff's bound holds independent of the initial marking. This is needed, because the proof of the main lemma splits a firing sequence into two parts and then considers the starting marking of the second part as the new initial marking. The sets we define in the following will depend on some unrestricted initial marking $M$, but we then quantify over all possible markings to get rid of the dependency.

Let $\mathit{Paths}(M,i)$ be the set of all $i$-non-negative and $i$-bounded computations from $M$,

$$\mathit{Paths}(M,i) = \{\sigma \in T^* \mid \pi = M[\sigma\rangle_i M',\ \pi \text{ is } i\text{-non-negative and } i\text{-covering}\}\ .$$

Let $\mathit{Words}(M,i) = \{\lambda(\sigma) \mid \sigma \in \mathit{Paths}(M,i)\}$ be the corresponding set of words, and let $\mathit{Basis}(M,i) = \{w \in \mathit{Words}(M,i) \mid w \text{ is } \preceq\text{-minimal}\}$ be its minimal elements. The central definitions is $\mathit{SPath}(M,i)$, the set of shortest paths yielding the minimal words in $\mathit{Basis}(M,i)$,

$$\mathit{SPath}(M,i) = \left\{ \sigma \in \mathit{Paths}(M,i) \;\middle|\; \begin{array}{l} \lambda(\sigma) \in \mathit{Basis}(M,i), \\ \nexists\ \sigma' \in \mathit{Paths}(M,i)\colon\ \ |\sigma'| < |\sigma|,\ \lambda(\sigma') = \lambda(\sigma) \end{array} \right\}\ .$$

Define $m(M,i) = \max\{|\sigma| + 1 \mid \sigma \in \mathit{SPath}(M,i)\}$ to be the length $(+1)$ of the longest path in $\mathit{SPath}(M,i)$, or $m(M,i) = 0$ if $\mathit{SPath}(M,i)$ is empty. Note that $\mathit{Basis}(M,i)$ is finite and therefore only finitely many different lengths occur for sequences in $\mathit{SPath}$, i.e. $m(M,i)$ is well-defined. To remove the dependency on $M$, define

$$f(i) = \max\{m(M,i) \mid M\colon P \to \mathbb{Z}\}$$

to be the maximal length of an $i$-covering computation, where the maximum is taken over all unrestricted initial markings. The well-definedness of $f(i)$ is not clear yet and will be a consequence of the next lemma. A bound on $f(\ell)$ will give us a bound on the maximum length of a computation accepting a minimal word from $\mathcal{L}(N, M_0, M_f)$. To derive the bound, we prove that $f(i+1) \leqslant (2^n f(i))^{i+1} + f(i)$ using Rackoff's famous case distinction [40].

▶ **Lemma 5.** *$f(0) = 1$ and $f(i+1) \leqslant (2^n f(i))^{i+1} + f(i)$ for all $i \in [1..\ell-1]$.*

**Proof.** To see that $f(0) = 1$, note that $\varepsilon \in \mathit{Basis}(M,0)$ for any $M \in \mathbb{Z}^\ell$, and the empty firing sequence is a 0-covering sequence producing $\varepsilon$.

For the second claim, we show that for any $M \in \mathbb{Z}^\ell$ and any $w \in \mathit{Words}(M,i+1)$, we can find $\sigma \in \mathit{Paths}(M,i+1)$ with $|\sigma| < (2^n f(i))^{i+1} + f(i)$ and $\lambda(\sigma) \preceq w$. Let $\sigma' \in \mathit{Paths}(M,i+1)$ be a shortest firing sequence of transitions such that $\lambda(\sigma') = w$. If $|\sigma'| < (2^n f(i))^{i+1} + f(i)$, we have nothing to do. Assume now $|\sigma'| \geqslant (2^n f(i))^{i+1} + f(i)$. We distinguish two cases.

**1$^{\text{st}}$ Case:** Suppose $\sigma'$ induces the $(i+1)$-non-negative, $(i+1)$-covering computation $\pi'$, in which for each occurring marking $M$ and for each place $p \in [1..i+1]$, $M(p) < 2^n \cdot f(i)$ holds. We extract from $\pi'$ an $(i+1)$-non-negative, $(i+1)$-covering computation $\pi$ where no two markings agree on the first $(i+1)$ places: Whenever such a repetition occurs in $\pi'$, we delete the transitions between the repeating markings to obtain a shorter computation that is still $(i+1)$-covering. Iterating the deletion yields the sequence of transition $\sigma$. The computation $\sigma$ satisfies

$$|\sigma| < (2^n f(i))^{i+1} \leqslant (2^n f(i))^{i+1} + f(i)\ .$$

The strict inequality holds as a computation of $h$ markings has $(h-1)$ transitions. Moreover, $\sigma$ is a subword of the original $\sigma'$, and hence $\lambda(\sigma) \preceq \lambda(\sigma') = w$.

**2$^{\text{nd}}$ Case:** Otherwise, $\sigma'$ is the path of an $(i+1)$-non-negative, $(i+1)$-covering computation $\pi'$, in which a marking occurs that assigns more than $2^n \cdot f(i)$ tokens to some place $p \in [1..i+1]$. Then, we can decompose $\pi'$ as follows:

$$\pi' = M[\sigma_1'\rangle_{i+1} M_1 [t\rangle_{i+1} M_2 [\sigma_2'\rangle_{i+1} M'$$

so that $M_2$ is the first marking that assigns $2^n \cdot f(i)$ or more tokens to some place, say wlog. place $i+1$. We may assume that $|\sigma_1'| < (2^n f(i))^{i+1}$. Otherwise, we can replace $\sigma_1'$ by a repetition-free sequence $\sigma_1$ as in the first case, where $M_0[\sigma_1\rangle_{i+1} M_1'$ such that $M_1'$ and $M_1$ agree on the first $i+1$ places.

Note that $\pi_2' = M_2[\sigma_2'\rangle_{i+1} M'$ is also an $i$-non-negative, $i$-covering computation. By the definition of $f(i)$, there is an $i$-non-negative, $i$-covering computation $\pi_2$ starting from $M_2$ such that the corresponding path $\sigma_2$ satisfies $|\sigma_2| < f(i)$ and $\lambda(\sigma_2) \preceq \lambda(\sigma_2')$. Since the value of place $i+1$ is greater or equal $2^n f(i)$, it is easy to see that $\pi_2$ is also an $(i+1)$-non-negative, $(i+1)$-covering computation starting in $M_2$: Even if all the at most $f(i)-1$ transitions subtract $2^n$ tokens from place $i+1$, we still end up with $2^n$ tokens. The concatenation $\sigma_1' \cdot_i t \cdot_i \sigma_2'$ is then an $(i+1)$-non-negative, $(i+1)$-covering run starting in $M$ of length at most $((2^n f(i))^{i+1} - 1) + 1 + (f(i) - 1) < (2^n f(i))^{i+1} + f(i)$. ◀

**Proof of Proposition 3.** As in [40], we define the function $g$ inductively by $g(0) = 2^{3n}$ and $g(i+1) = (g(i))^{3n}$. It is easy to see that $g(i) = 2^{((3n)^{(i+1)})}$. Using Lemma 5, we can conclude $f(i) \leqslant g(i)$ for all $i \in [0..\ell]$. Furthermore,

$$f(\ell) \leqslant g(\ell) \leqslant 2^{((3n)^{(\ell+1)})} \leqslant 2^{((3n)^{n+1})} \leqslant 2^{2^{cn \log n}}$$

for some suitable constant $c$.

Let $M_0[\sigma\rangle M \geqslant M_f$ be a covering computation of the Petri net. By the definitions, $\sigma \in \mathit{Paths}(M_0, \ell)$ and $\lambda(\sigma) \in \mathit{Words}(M_0, \ell)$. There is a word $w \in \mathit{Basis}(M_0, \ell)$ with $w \preceq \lambda(\sigma)$, and $w$ has a corresponding computation $\sigma' \in \mathit{SPath}(M_0, \ell)$ (i.e. $\lambda(\sigma') = w$). By the definition of $f(\ell)$, we have $|\sigma'| < m(M_0, \ell) \leqslant f(\ell) \leqslant 2^{2^{cn \log n}}$. ◀

## Lower Bound

We present a family of Petri net languages for which the minimal finite state automata representing the upward closure are of size doubly exponential in the size of the input. We rely on a construction due to Lipton [34] that shows how to calculate in a precise way (including zero tests) with values up to $2^{2^n}$ in Petri nets.

▶ **Lemma 6.** *For every number $n \in N$, we can construct a Petri net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings $M_0, M_f$ of size polynomial in $n$ such that $\mathcal{L}(N(n), M_0, M_f) = \left\{a^{2^{2^n}}\right\}$ .*

**Proof.** We rely on Lipton's proof [34] of EXPSPACE-hardness of Petri net reachability. Lipton shows how a counter machines in which the counters are bounded by $2^{2^n}$ can be simulated using a Petri net of polynomial size. We will use the notations as in [17].

Lipton defines *net programs* (called *parallel programs* in [34]) to encode Petri nets. For the purpose of proving this lemma, we will recall the syntax of net programs and also some of the subroutines as defined in [34, 17].

We will use the following commands resp. the following established subroutines from [17] in the program.

| | |
|---|---|
| $l : x := x - 1$ | decrement a variable $x$ |
| $l : \texttt{gosub}\ s$ | call the subroutine $s$ |
| $l : \texttt{Inc}_n(x)$ | sets variable $x$ to exactly $2^{2^n}$ |
| $l : \texttt{Test}(x, l_{=0}, l_{\neq 0})$ | jumps to $l_{=0}$ if $x = 0$ and to $l_{\neq 0}$ if $x \neq 0$ |

Note that all commands can be encoded using a Petri net of size polynomial in $n$. The fact that a test for zero can be implemented (by the subroutine $\texttt{Test}(x, l_{=0}, l_{\neq 0})$) relies on the counters being bounded by $2^{2^n}$. It is not possible to encode zero tests for counter machines with unbounded counters using Petri nets.

We assume that in the Petri net encoded by these commands, all transitions are labeled by $\varepsilon$. We consider an additional command $\texttt{Action}(a)$ to accept the input $a$, which can be encoded using a set of transitions such that exactly one of them is labeled by $a$.

Consider the following net program.

```
$l_1$ : gosub $\texttt{Inc}_n(x)$
$l_2$ : $x := x - 1$
$l_3$ : $\texttt{Action}(a)$
$l_4$ : gosub $\texttt{Test}(x, l_5, l_2)$
$l_5$ : Halt
```

In any halting computation, the program performs $\texttt{Action}(a)$ exactly $2^{2^n}$ times. The required Petri net $N(n)$ is the one equivalent to this net program. ◀

The upward closure $\mathcal{L}(N(n), M_0, M_f)\uparrow$ is $\left\{a^k \mid k \geqslant 2^{2^n}\right\}$ and needs at least $2^{2^n}$ states.

## 3.2 BPP Nets

We establish an exponential upper bound on the size of the finite automata representing the upward closure of BPP net languages. Then, we present a family of BPP net languages for which the minimal finite automata representing their upward closure are of size at least exponential in the size of the input.

### Upper Bound

Assume that the net $N$ in the Petri net instance $(N, M_0, M_f)$ of size $n$ is a BPP net.

▶ **Theorem 7.** *One can construct an FSA of size $O(2^{poly(n)})$ for $\mathcal{L}(N, M_0, M_f)\uparrow$.*

We will show that every minimal word results from a computation whose length is polynomially dependent on the number of transitions and on the number of tokens in the final marking (which may be exponential in the size of the input). Let $k$ be a bound on the length of the minimal computations. With the same argument as before and using Lemma 2, we can construct a finite state automaton of size $O(2^{\text{poly}(n)})$ that accepts $\mathcal{L}_k(N, M_0, M_f)\uparrow$.

▶ **Proposition 8.** *Consider a BPP net $N$. For every computation $M_0[\sigma\rangle M \geqslant M_f$ there is $M_0[\sigma'\rangle M' \geqslant M_f$ with $\lambda(\sigma') \preceq \lambda(\sigma)$ and $|\sigma'| \leqslant tc(M_f)^2 \cdot |T|$.*

The key to proving the proposition is to consider a structure that makes the concurrency among transitions in the BPP computation of interest explicit. Phrased differently, we give a true concurrency semantics (also called partial order semantics and similar to Mazurkiewicz traces) to BPP computations. Since BPPs do not synchronize, the computation yields a forest where different branches represent causally independent transitions. To obtain a subcomputation that covers the final marking, we select from the forest a set of leaves that corresponds exactly to the final marking. We then show that the number of transitions in the minimal forest that generates the selected set of leaves is polynomial in the number of tokens in the final marking and in the number of transitions.

To make the proof sketch for Proposition 8 precise, we use (and adapt to our purposes) unfoldings, a true concurrency semantics for Petri nets [18]. The unfolding of a Petri net is the true concurrency analogue of the computation tree – a structure that represents all computations. Rather than having a node for each marking, there is a node for each token in the marking. To make the idea of unfoldings formal, we need the notion of an *occurrence net*, an unlabeled BPP net $O = (P', T', F')$ that is acyclic and where each place has at most one incoming transition and each transition creates at most one token per place: $\sum_{t' \in T'} F(t', p') \leqslant 1$ for every $p' \in P'$. Two elements $x, y \in P' \cup T'$ are *causally related*, $x \trianglelefteq y$, if there is a path from $x$ to $y$. We use $\lfloor x \rfloor = \{y \in P' \cup T' \mid y \trianglelefteq x\}$ to denote the predecessors of $x \in P' \cup T'$. The $\trianglelefteq$-minimal places are denoted by $\mathit{Min}(O)$. The initial marking of $O$ is fixed to having one token in each place of $\mathit{Min}(O)$ and no tokens elsewhere. So occurrence nets are 1-safe and we can identify markings with sets of places $P'_1, P'_2 \subseteq P'$ and write $P'_1[t'\rangle P'_2$. To formalize that $O$ captures the behavior of a BPP net $N = (\Sigma, P, T, F, \lambda)$ from marking $M_0$, we define a *folding homomorphism* $h : P' \cup T' \to P \cup T$ satisfying

(1) Initiation: $h(\mathit{Min}(O)) = M_0$.
(2) Consecution: For all $t' \in T'$, $h({}^\bullet t') = {}^\bullet h(t')$, and all $p \in P$, $(h(t'^\bullet))(p) = F(h(t'), p)$. Here, $h(P'_1)\colon P \to \mathbb{N}$ with $P'_1 \subseteq P'$ is a function with $(h(P'_1))(p) = |\{p' \in P'_1 \mid h(p') = p\}|$.
(3) No redundancy: For all $t'_1, t'_2 \in T'$, with ${}^\bullet t'_1 = {}^\bullet t'_2$ and $h(t'_1) = h(t'_2)$, we have $t'_1 = t'_2$.

The pair $(O, h)$ is called a *branching process* of $(N, M_0)$. Branching processes are partially ordered by the prefix relation which, intuitively, states how far they unwind the BPP. The limit of the unwinding process is the *unfolding* $\mathrm{Unf}(N, M_0)$, the unique (up to isomorphism) maximal branching process. It is not difficult to see that there is a one to one correspondence between the firing sequences in the BPP net and the firing sequences in the unfolding. Note that $\mathrm{Unf}(N, M_0)$ will usually have infinitely many places and transitions, but every computation will only use places up to a bounded distance from $\mathit{Min}(O)$. With this, we are prepared to prove the proposition.

**Proof of Proposition 8.** Consider a computation $M_0[\sigma\rangle M$ with $M \geqslant M_f$ in the given BPP net $N = (\Sigma, P, T, F, \lambda)$. Let $(O, h)$ with $O = (P', T', F')$ be the unfolding $\mathrm{Unf}(N, M_0)$. Due to the correspondence in the firing behavior, there is a sequence of transitions $\tau$ in $O$ with $h(\tau) = \sigma$ and $\mathit{Min}(O)[\tau\rangle P'_1$ with $h(P'_1) = M$. Since $M \geqslant M_f$, we know that for each place $p \in P$, the set $P'_1$ contains at least $M_f(p)$ many places $p'$ with $h(p') = p$. We arbitrarily select a set $X_p$ of size $M_f(p)$ of such places $p'$ from $P'_1$. Let $X = \bigcup_{p \in P} X_p$ be the union for all $p \in P$.

The computation $\tau$ induces a forest in $O$ that consists of all places that contain a token after firing $\tau$ and their predecessors. We now construct a subcomputation by restricting $\tau$ to the transitions leading to the places in $X$. Note that the transitions leading to $X$ are contained in $\lfloor X \rfloor$, which means we can define the subcomputation as $\tau_1 = \pi_{\lfloor X \rfloor}(\tau)$, i.e. the projection of $\tau$ onto $\lfloor X \rfloor$. In $\tau_1$, we mark all $\trianglelefteq$-maximum transitions $t'$ that lead to two different places in $X$. Formally, if there are $x, y \in X$ with $t' \in \lfloor x \rfloor \cap \lfloor y \rfloor$ and there is no $t'' \in \lfloor x \rfloor \cap \lfloor y \rfloor$ with $t' \trianglelefteq t''$, then we mark $t'$. We call the marked $t'$ the *join transitions*.

Assume that $t' \neq t''$ are two join transitions that occur on the same branch of the forest. Note that for two places in $X$, there is either no join transition or a unique one leading to these two places. Consequently, $t'$ and $t''$ have to lead to different places of $X$. Let $t't^1 \ldots t^m t''$ be the transitions on the branch in between $t'$ and $t''$. We assume that $t'$ and $t''$ are adjacent join transitions, i.e. none of the $t^i$ is a join transition.

Since $t', t''$ occur in $\tau_1$, all $t^i$ also have to occur in $\tau_1$. If there are indices $j < k$ such that $t^j = t^k$, we may delete $t^{j+1} \ldots t^k$ from $\tau_1$ while keeping a transition sequence that covers $X$. It will cover $X$ as none of the deleted transitions was a join transition, i.e. we will only lose leaves of the forest that are not in $X$. We repeat this deletion process until there are no more repeating transitions between adjacent join transitions. Let the resulting transition sequence be $\tau_2$. First, note that for any $x \in X$, there are at most $tc(M_f)$ many join transitions on the branch from the corresponding minimal element to $x$: In the worst case, for each place in $X \setminus \{x\}$, there is a join transition on the branch, and $|X| = tc(M_f)$. Between any two adjacent join transitions along such a path, there are at most $|T|$ transitions (after deletion). Hence, the number of transitions in such a path is bounded by $tc(M_f) \cdot |T|$. Since we have $tc(M_f)$ many places in $X$, the total number of transitions in $\tau_2$ is bounded by $tc(M_f)^2 \cdot |T|$. ◀

### Lower Bound

We present a family of BPP net languages for which the minimal FSA representing the upward closure are exponential in the size of the input. The idea is to rely on the binary encoding of numbers, which allows us to handle $2^n$ using a polynomially sized net.

▶ **Lemma 9.** *For all numbers $n \in \mathbb{N}$, we can construct a BPP net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings $M_0, M_f$ of size polynomial in $n$ such that $\mathcal{L}((N(n), M_0, M_f) = \{a^{2^n}\}$ .*

**Proof.** The BPP net $N(n)$ consists of three places $p_0, p_1, p_f$ and two transitions $t, t_a$. Transition $t$ is $\varepsilon$-labeled, consumes one token from $p_0$, and creates $2^n$ tokens on $p_1$. Transition $t_a$ is labeled by $a$ and moves one token from $p_1$ to $p_f$. Formally, $\lambda(t) = \varepsilon, F(p_0, t) = 1, F(t, p_1) = 2^n$, $\lambda(t_a) = a, F(p_1, t_a) = 1, F(t_a, p_f) = 1$. All other values for $F$ are 0. The initial marking $M_0$ places one token on $p_1$ and no tokens elsewhere, the final marking $M_f$ requires $2^n$ tokens on $p_f$ and no tokens elsewhere. Note that $F$ as well as $M_f$ have polynomially-sized encodings. There is a unique covering computation in $N(n)$, namely the computation $M_0[\sigma\rangle M_f$, where $\sigma = t.\underbrace{t_a \ldots t_a}_{2^n \text{ times}}$. Thus, the language of $(N(n), M_0, M_f)$ is as required. ◀

## 4 Downward Closures

We consider the problem of constructing a finite state automaton accepting the downward closure of a Petri net and a BPP net language, respectively. The downward closure often has the property of being a precise description of the system behavior, namely as soon as asynchronous communication comes into play: If the components are not tightly coupled, they may overlook commands of the partner and see precisely the downward closure of the other's computation. As a result, having a representation of the downward closure gives the possibility to design exact or under-approximate verification algorithms.

**Computing the downward closure**
**Given:** A Petri net instance $(N, M_0, M_f)$.
**Compute:** An FSA $A$ with $\mathcal{L}(A) = \mathcal{L}(N, M_0, M_f)\downarrow$.

### 4.1 Petri Nets

The downward closure of Petri net languages has been shown to be effectively computable in [22]. The algorithm is based on the Karp-Miller tree [27], which can be of non-primitive recursive size. We now present a family of Petri net languages that are already downward closed and for which the minimal finite automata have to be of non-primitive recursive size in the size of the input. Our result relies on a construction due to Mayr and Meyer [37]. It gives a family of Petri nets whose computations all terminate but, upon halting, may have produced Ackermann many tokens on a distinguished place.

We first recall the definition of the Ackermann function.

▶ **Definition 10.** The Ackermann function is defined inductively as follows:

$$\begin{aligned} \mathit{Acker}_0 \quad (x) &= x+1 \\ \mathit{Acker}_{n+1} \quad (0) &= \mathit{Acker}_n(1) \\ \mathit{Acker}_{n+1}(x+1) &= \mathit{Acker}_n(\mathit{Acker}_{n+1}(x)) \ . \end{aligned}$$

▶ **Lemma 11.** *For all $n, x \in \mathbb{N}$, there is a Petri net $N(n) = (\{a\}, P, T, F, \lambda)$ and markings $M_0^{(x)}, M_f$ of size polynomial in $n+x$ such that $\mathcal{L}\Big(N(n), M_0^{(x)}, M_f\Big) = \big\{a^k \mid k \leqslant \mathit{Acker}_n(x)\big\}$.*

Our lower bound is an immediate consequence of this lemma.

▶ **Theorem 12.** *There is a family of Petri net languages for which the minimal finite automata representing the downward closure are of non-primitive recursive size.*

This hardness result relies on a weak computation mechanism of very large numbers that is unlikely to show up in practical examples. The SRE inclusion problem studied in the following section can be understood as a refined analysis of the computation problem for downward closures.

It remains to prove Lemma 11. We start by defining a preliminary version of the required nets. The construction is inductive and imitates the definition of the Ackermann function.

▶ **Definition 13.** We define the Petri net $AN_0$ to be

$$\begin{aligned} AN_0 &= (\{a\}, P^0, T^0, F^0, \lambda^0) \quad \text{with} \\ P^0 &= \big\{\text{in}^0, \text{out}^0, \text{start}^0, \text{stop}^0, \text{copy}^0\big\}, \\ T^0 &= \big\{t^0_{start}, t^0_{stop}, t^0_{copy}\big\}, \\ \lambda^0(t) &= \varepsilon \text{ for all } t \in T^0 \ . \end{aligned}$$

The flow relation is given by Figure 2, where each edge carries a weight of 1.
For $n \in \mathbb{N}$, we define $AN_{n+1}$ inductively by

$$\begin{aligned} AN_{n+1} &= (\{a\}, P^{n+1}, T^{n+1}, F^{n+1}, \lambda^{n+1}) \quad \text{with} \\ P^{n+1} &= P^n \cup \{\text{in}^{n+1}, \text{start}^{n+1}, \text{copy}^{n+1}, \text{out}^{n+1}, \text{stop}^{n+1}, \text{swap}^{n+1}, \text{temp}^{n+1}, \} \\ T^{n+1} &= T^n \cup \{t^{n+1}_{start}, t^{n+1}_{copy}, t^{n+1}_{stop}, t^{n+1}_{restart}, t^{n+1}_{in}, t^{n+1}_{swap}, t^{n+1}_{temp}\}, \\ \lambda^{n+1}(t) &= \varepsilon \text{ for all } t \in T^{n+1}. \end{aligned}$$

The flow relation is given by Figure 3, where again each edge carries a weight of 1.

Let us furthermore define for each $x \in \mathbb{N}$ the marking $M_0^{(x)}$ of $AN_n$ that places one token on start$^n$, $x$ tokens on in$^n$ and no token elsewhere.

We prove that $AN_n$ indeed can weakly compute $\mathit{Acker}_n(m)$.

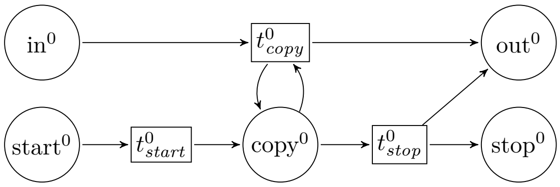

**Figure 2.** The flow relation of the Petri net $AN_0$.

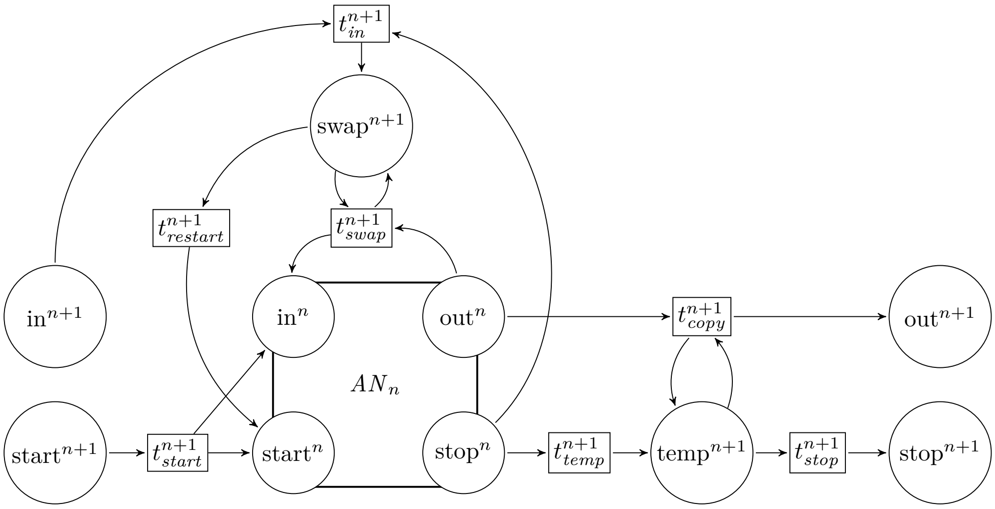

**Figure 3.** The flow relation of the Petri net $AN_{n+1}$.

▶ **Lemma 14.** *For all $n, x \in \mathbb{N}$:*
*(1) There is $M_0^{(x)}[\sigma\rangle M$ of $AN_n$ such that $M(out^n) = Acker_n(x)$, $M(stop^n) = 1$.*
*(2) There is no computation starting in $M_0$ that creates more than $Acker_n(x)$ tokens on $out^n$.*

**Proof.** We proof both statements simultaneously by induction on $n$.
**Base case**, $n = 0$: We have $Acker_0(x) = x + 1$. The only transition that is enabled in $AN_0$ is the starting transition $t^0_{start}$. Firing it leads to one token on the copy place copy$^0$. Now we can fire the copy transition $t^0_{copy}$ $x$ times, leading to $x$ tokens on out$^0$. Finally, we fire the stopping transition $t^0_{stop}$, which leads to one token on stop$^0$ and in total $x+1$ tokens on out$^0$. This is the computation maximizing the number of tokens on out. Firing $t^0_{copy}$ less than $x$ times or not firing $t^0_{stop}$ leads to less tokens on out$^0$.
**Inductive step,** $n \mapsto n+1$: Initially, we can only fire the starting transition $t^{n+1}_{start}$, creating one token on in$^n$ and one token on start$^n$. We can now execute the computation of $AN_n$ that creates $Acker_n(1)$ tokens on out$^n$ and one token on stop$^n$, which exists by induction. We consume one token from in$^{n+1}$ and the token on stop$^n$ to create a token on swap$^{n+1}$ using $t^{n+1}_{in}$. This token allows us to swap all $Acker_n(1)$ tokens from out$^n$ to in$^n$ using $t^{n+1}_{swap}$. After doing this, we move the token from swap$^{n+1}$ to $t^n_{start}$ using the restart transition $t^{n+1}_{restart}$. We iterate the process to create

$$Acker_n(Acker_n(1)) = Acker_n(Acker_{n+1}(0)) = Acker_{n+1}(1)$$

tokens on out$^{n+1}$, which we can then swap again to in$^n$.

We iterate this process $x$-times, which creates $Acker_{n+1}(x)$ tokens on out$^n$, since

$$Acker_{n+1}(x) = \underbrace{Acker_n(\dots Acker_n}_{x+1 \text{ times}}(1)) \ .$$

Note that it is not possible to create more than $Acker_{n+1}(x)$ on out$^n$. Having $y$ tokens on in$^n$, we cannot create more than $Acker_n(y)$ tokens on out$^n$ by induction. Furthermore, if we do not execute $t_{swap}$ as often as possible, say we leave $y'$ out of $y$ tokens on out$^n$, we end up with $Acker_n(y - y') + y' \leqslant Acker_n(y)$ tokens on out$^n$, since

$$Acker_n(k) + k' \leqslant Acker_n(k + k')$$

for all $k, k' \in \mathbb{N}$. Finally, we move the token on $t^{n+1}_{stop}$ to temp$^{n+1}$ using $t^{n+1}_{temp}$, and then move all $Acker_n(x)$ tokens to out$^{n+1}$ using $t^{n+1}_{copy}$. We end the computation by firing $t^{n+1}_{stop}$ to move the token on temp$^{n+1}$ to stop$^{n+1}$. We now have one token on stop$^{n+1}$ and $Acker_n(x)$ tokens on out$^{n+1}$ as required.

This maximizes the number of tokens on out$^{n+1}$: As already argued, we cannot create more than $Acker_n(x)$ tokens on out$^n$, and firing $t^{n+1}_{copy}$ less than $Acker_n(x)$ times will lead to less tokens on out$^{n+1}$. ◀

We are now prepared to tackle the proof of Lemma 11.

**Proof of Lemma 11.** Let $n \in \mathbb{N}$. We define $N(n)$ to be the Petri net that is obtained from $AN_n$ by adding a place final and a transition $t_{final}$ that is labeled by $a$ and moves one token from out$^n$ to final.is defined as before. The final marking $M_f$ is zero on all places.

By Lemma 14, we can create at most $Acker_n(x)$ tokens on out$^n$. We can then move a part of these tokens to final, producing up to $Acker_n(x)$ many $a$s in the process. This proves $\mathcal{L}\Big(N(n), M_0^{(x)}, M_f\Big) = \big\{a^k \mid k \leqslant Acker_n(x)\big\}$.

Note that the size of $N(n)$ is a constant plus the size of $AN_n$, which is linear in $n$. The final and initial marking are linear in $n + \log x$. ◀

## 4.2 BPP Nets

We prove an exponential upper bound on the size of the finite automata representing the downward closure of BPP languages. Then, we present a family of BPP languages for which the minimal finite automata representing their downward closure are exponential in the size of the input BPP nets.

### Upper Bound

Assume that the net $N$ in the Petri net instance $(N, M_0, M_f)$ of size $n$ is a BPP net.

▶ **Theorem 15.** *We can construct a finite automaton of size $O(2^{poly(n)})$ for $\mathcal{L}(N, M_0, M_f)\!\downarrow$.*

The key insight for simulating $N$ by a finite automaton is the following: If during a firing sequence a marking occurs that has more than $c$ tokens (where $c$ is specified below) in some place $p$, then there has to be a *pump*, a subsequence of the firing sequence that can be repeated to produce arbitrarily many tokens in $p$. The precise statement is this, where we use $m = max(F)$ to refer to the maximal multiplicity of an edge.

▶ **Lemma 16.** *Let $M_0[\sigma\rangle M$ such that for some place $p \in P$, we have $M(p) > c$ with*

$$c = tc(M_0) \cdot (|P| \cdot m)^{(|T|+1)} \ .$$

*Then for each $j \in \mathbb{N}$, there is $M_0[\sigma_j\rangle M_j$ such that*
*(1) $\sigma \preceq \sigma_j$, (2) $M \leqslant M_j$, and (3) $M_j(p) > j$.*

**Proof.** We consider the unfolding $(O, h)$ of $N$, as defined in Subsection 3.2. Let $\sigma'$ be a firing sequence of $O$ induced by $\sigma$, i.e. $h(\sigma') = \sigma$ (where $h$ is extended to sequences of transitions in the obvious way), and $\sigma'$ can be fired from the marking $M_0'$ of $O$ corresponding to $M_0$.

Executing $\sigma'$ leads to a marking $M'$, such that the sum of tokens in places $p'$ of $O$ with $h(p') = p_N$ is equal to $M(p_N)$ for each place $p_N$ of $N$. In particular, this holds for the place $p$ on which we exceed the bound $c$:

$$\sum_{\substack{p' \in P',\\ h(p')=p}} M'(p') = M(p) > c \ .$$

Recall that $O$ is a forest, and each tree in it has a minimal place $r' \in \mathit{Min}(O)$ that corresponds to a token assigned to a place of $N$ by $M_0$, i.e. $M'_0(r') = 1$. We fix the root node $r$ of the tree $\mathcal{T}$ with a maximal number of leaves that correspond to place $p$ (called *$p$-leaves* in the following), i.e. the root node $r$ such that the number of places $p'$ with $h(p') = p$, $M'(p') = 1$ in the corresponding tree is maximal. Note that the number of $p$-leaves in this tree is at least

$$c_1 = \frac{c}{tc(M_0)} = (\ell \cdot m)^{(|T|+1)} .$$

Since there are only $tc(M_0)$ many trees in the forest, if all trees have strictly less than $c_1$ many $p$-leaves, the whole forest cannot have $c$ many $p$-leaves.

We now consider the subtree $\mathcal{T}_p$ of $\mathcal{T}$ that is defined by the $p$-leaves in $\mathcal{T}$, i.e. the tree one gets by taking the set of $p$-leaves $X$ in $\mathcal{T}$ and all places and transitions $\lfloor X \rfloor$ that are their predecessors. This tree has the following properties:

(i) Its leaves are exactly the $p$-leaves in $\mathcal{T}$.

(ii) Each place in it has out-degree 1 if it is not a $p$-leaf. That the out-degree is at most 1 is clear by how $O$ was defined: Since each place only carries at most one token, it can be consumed by at most one transition during the run, and we don't consider the transitions that are not fired in the run in $\mathcal{T}_p$. That the out-degree of the places that are not $p$-leaves is exactly 1 is because we only consider transitions leading to $p$-leaves in $\mathcal{T}_p$.

(iii) Each transition has out-degree at most $m \cdot \ell$. (In the worst case, each transition creates $m$ tokens in each of the $\ell$ places, which is modeled in $O$ by having one place for each token that is produced.)

We will call a transition in $\mathcal{T}_p$ of out-degree at least 2 a *join-transition*, since it joins (at least) two branches of the tree that lead to a $p$-leaf. Our goal is to show that there is a branch of $\mathcal{T}_p$ in which at least $|T| + 1$ many join-transitions occur.

**Claim:** Let $\mathcal{T}$ be a tree with $x$ leaves in which all nodes have out-degree at most $k$. Then $\mathcal{T}$ has a branch with at least $\log_k x$ nodes of out-degree at least 2.

Towards a proof of the claim, assume that the maximal number of nodes of out-degree greater than 2 in any branch of the tree is $h < \log_k x$. To maximize the number of leaves, we assume that the number of such nodes is exactly $h$ in every branch, and all nodes (but the leaves) have out-degree $k$. The number of leaves in this tree is $k^h$, but since $h < \log_k x$, this is less than $x$: $k^h < k^{\log_k x} = x$.

Instantiating the claim for $x = c_1$ and $k = m \cdot |M_0|$ yields that $\mathcal{T}_p$ has a branch with at least $c_2 = \log_{m \cdot tc(M_0)} c_1$ many join-transitions, and by the definition of $c$, $c_2 = |T| + 1$. Since the original BPP net has only $|T|$ many different transitions, there have to be join-transitions $\tau$ and $\tau'$ in the same branch with $h(\tau) = h(\tau') = t$ for some transition $t$ of the original net.

Since $\tau$ was a join-transition, it has at least two child branches $b_1$ and $b_2$ that lead to a $p$-leaf. We assume without loss of generality that $b_1$ is the branch on which $\tau'$ occurs, and $b_2$ is another branch. We consider the sequence of transitions $\sigma_p^{pump}$ that occur on $b_1$ when going from $\tau$ to $\tau'$ (including $\tau$, not including $\tau'$). We also consider the sequence of transitions $\sigma_p^{gen}$ that occur on $b_2$ when going from $\tau$ to a $p$-leave (not including $\tau$). Let $\sigma^{pump}$ and $\sigma^{gen}$ be the corresponding sequences in the original BPP net, i.e. $\sigma^{pump} = h(\sigma_p^{pump}), \sigma^{gen} = h(\sigma_p^{gen})$.

We now modify the run $\sigma$ in the original net to obtain the desired amount of $j$ tokens. We decompose $\sigma = \sigma_1.t.\sigma_2$, where $t$ is the transition that corresponds to $\tau$ in $\mathcal{T}_p$. We extend the run to

$$\sigma_j = \sigma_1.\underbrace{\sigma^{pump}\ldots\sigma^{pump}}_{j\text{ times}}.t.\sigma_2.\underbrace{\sigma^{gen}\ldots\sigma^{gen}}_{j\text{ times}} \ .$$

Obviously, $\sigma$ is a subsequence of $\sigma_j$, and therefore satisfies the required Property (1). We have to argue why $\sigma_j$ is a valid firing sequence, why firing $\sigma_j$ leads to a marking greater than $M$ (Property (2)) and why it generates at least $j$ tokens in place $p$ (Property (3)).

The latter is easy: $\sigma^{gen}$ corresponds to a branch of $\mathcal{T}_p$ leading to a $p$-leaf, i.e. firing it creates one additional token in $p$ that will not be consumed by another transition in $\sigma$.

Note that up to $\sigma_1$, $\sigma$ and $\sigma_j$ coincide. Since $t$ could be fired after $\sigma_1$, $\sigma^{pump}$ can be fired after $\sigma_1$: $t$ corresponds to transition $\tau$ in $O$, and so does the first transition in $\sigma^{pump}$. Since $\sigma^{pump}$ was created from a branch in the tree $\mathcal{T}_p$, each transition will consume the token produced by its immediate predecessor. The last transition creates a token in the place that feeds transition $\tau'$, but $h(\tau) = t = h(\tau')$, so after firing $\sigma^{pump}$, we can fire $t$ again. (Either as the first transition of the next $\sigma^{pump}$, or after all pumps have been fired.) Furthermore, $t$ corresponds to the join-transition $\tau$, i.e. it does create a token in the place that is the starting point of $\sigma^{gen}$. This token will not be consumed by any other transition in the sequence $t.\sigma_2$, so after firing $\sigma^{pump}$ $j$ times, we can indeed fire $\sigma^{gen}$ $j$ times.

As argued above, firing $\sigma^{gen}$ and $\sigma^{pump}$ has a non-negative effect on the marking, so the marking one gets by firing $\sigma'$ is indeed greater than the marking $M$. ◀

It remains to use Lemma 16 to prove Theorem 15.

**Proof of Theorem 15.** We will state the construction of the automaton, prove its soundness and that the size of the automaton is as required. The automaton for $\mathcal{L}(N, M_0, M_f)\!\downarrow$ is the state space of $N$ with token values beyond $c$ set to $\omega$. For every transition, we also have an $\varepsilon$-variant to obtain the downward closure.

More formally, $A = (\Sigma, Q, \rightarrow_A, q_{init}, F)$ is defined as follows: Its set of states is $Q = P \rightarrow ([0..c] \cup \{\omega\})$, where $c = tc(M_0) \cdot (|P| \cdot m)^{(|T|+1)}$ as in Lemma 16. This means each state is a marking that will assign to each place a number of tokens up to $c$ or $\omega$. For each transition $t$ of the BPP net $N$ and each state $q \in Q$ such that $q(p) \geqslant F(p,t)$ (where we define $\omega > k$ to be true for all $k \in \mathbb{N}$), $\rightarrow_A$ contains two transitions $(q, \lambda(t), q')$ and $(q, \varepsilon, q')$. Here, $q'$ is defined by

$$q'(p) = (q(p) \ominus F(p,t)) \oplus F(t,p) \ ,$$

for all $p \in P$, where $\oplus$ and $\ominus$ are variants of $+$ and $-$ that treat $\omega$ as infinity: $x \oplus y = x + y$ if $x + y < c$, $x \oplus y = \omega$ otherwise. Similarly, $x \ominus y = x - y$ if $x \neq \omega$. Note that if $t$ was already labeled by $\varepsilon$, the two transitions coincide. The initial state is defined by $q_{init}(p) = M_0(p)$ for all $p \in P$. A state $q \in F$ is final if it covers the final marking $M_f$ of $N$, i.e. $q(p) \geqslant M_f(p)$ for all places $p$. Again, we assume $\omega > k$ to hold for all $k \in \mathbb{N}$.

We prove that indeed $\mathcal{L}(A) = \mathcal{L}(N, M_0, M_f)\!\downarrow$ holds. First assume $w \in \mathcal{L}(N, M_0, M_f)\!\downarrow$. Then there is a computation $\pi = M_0[\sigma\rangle M$ of $N$ such that $M \geqslant M_f$ and $w \preceq \lambda(\sigma)$. We can delete transitions in $\sigma$ to obtain a sequence of transitions $\tau$ with $\lambda(\tau) = w$. (One may not be able to fire $\tau$.) We construct a run $\rho$ of the automaton $A$ starting in $q_0$ by replacing transitions in $\sigma$ by a corresponding transition of the automaton. For the transitions $t$ present in $\tau$, we pick the variant of the transitions labeled by $\lambda(t)$, for the ones not present in $\tau$, we pick the $\varepsilon$-labeled variant. Note that $\rho$ is a valid run of $A$ because $\pi$ was a computation of

$N$. The run $\rho$ ends in a state $q_\rho$ such that for each place $p$, either $q_\rho(p) = M(p)$ holds, or $q(p) = \omega$. Since $M \geqslant M_f$, this means that $q_\rho$ is final. We have constructed an accepting run of $A$ that produces the word $w$.

Now assume that $w \in \mathcal{L}(A)$ is a word accepted by the automaton. Let $\rho$ be an accepting run and let $q_0, q_1, \ldots, q_s$ be the states occurring during $\rho$. We prove that there is a computation $\pi = M_0[\sigma\rangle M$ of $N$ such that $M \geqslant M_f$ and $w \preceq \lambda(\sigma)$. Assume the final state $q_s$ does not assign $\omega$ to any place, $q_s(p) \neq \omega$ for all $p \in P$. Note that in this case, we have $q_i(p) \neq \omega$ for all $i \in [0..s]$ and all $p \in P$, since $q_i(p) = \omega$ implies $q_j(p) = \omega$ for all $j \in [i..s]$. In this case, we can easily construct a sequence of transitions $\sigma$ corresponding to $\rho$: For each transition in $\rho$, we take the corresponding transition of $N$. Note that the transition in $\rho$ can be labeled by $\varepsilon$ while the transition of $N$ is not labeled by $\varepsilon$. Still, $w \preceq \lambda(\sigma)$ will hold. Furthermore, $q_s$ is a marking for $N$ (since $\omega$ does not occur), and since $q_s$ was final, $q_s \geqslant M_f$ has to hold.

Now assume that there is a unique place $p$ such that $q_s(p) = \omega$. Let $i \in [0..s]$ be the first index such that $q_i(p) = \omega$. We decompose the run $\rho = \rho_1.\rho_2$, where $\rho_1$ is the prefix that takes the automaton from state $q_0$ to $q_i$. As in the previous case, we may obtain sequences of transitions $\sigma_1$ and $\sigma_2$ that correspond to $\rho_1$ and $\rho_2$. In particular, we have $w \preceq \lambda(\sigma_1).\lambda(\sigma_2)$. The first sequence $\sigma_1$ is guaranteed to be executable, i.e. $\pi_1 = M_0[\sigma_1\rangle M_1$ is a valid computation for some $M_1$. Since $q_i(p) = \omega$, the transition relation of the automaton guarantees that $M_1(p) > c$.

It might not be possible to fire $\sigma_2$ from $M_1$, because $\sigma_2$ may consume more than $c$ tokens from place $p$. Let

$$d = \sum_{j \in [1..|\sigma_2|]} F(p, t_j) + M_f(p) \ .$$

where $\sigma_2 = t_1.t_2 \ldots t_{|\sigma_2|}$. The number $d$ is certainly an upper bound for the number of tokens needed in place $p$ to be able to fire $\sigma_2$ and end up in a marking $M_2$ such that $M_2(p) \geqslant M_f(p)$. We apply Lemma 16 to obtain a supersequence $\sigma_1'$ of $\sigma_1$ with $M_0[\sigma_1'\rangle M_1'$ where $M_1' \geqslant M_1$ and $M_1(p) \geqslant d$. Now consider the concatenation $\sigma = \sigma_1'.\sigma_2$. Since the marking $M_1'$ has enough tokens in place $p$, $\sigma$ is executable and $M_0[\sigma\rangle M$, where $M \geqslant M_f$.

If the final state $q_p$ assigns $\omega$ to several places, the above argumentation has to be applied iteratively to all such places.

It remains to argue argue that the size of the automaton is in $\mathcal{O}(2^{\mathrm{poly}(n)})$. The size of the automaton is certainly polynomial in its number of states $|Q|$. We have

$$\begin{aligned} |Q| &= |P \to ([0..c] \cup \{\omega\})| = |[0..c] \cup \{\omega\}|^{\ell} = (c+2)^{\ell} \\ &= \Big(tc(M_0)(|P| \cdot m)^{(|T|+1)} + 2\Big)^{\ell} \leqslant \Big((\ell \cdot 2^n)(\ell \cdot 2^n)^{(|T|+1)} + 2\Big)^{\ell} \\ &\leqslant \Big((2^{(n+1)})^{(n+2)} + 2\Big)^{n} = \Big(2^{(n+1)\cdot(n+2)} + 2\Big)^{n} \\ &\leqslant (2^{(n+1)\cdot(n+2)})^{n} \cdot 2^{n} \leqslant 2^{(n+1)\cdot(n+2)\cdot n+1} \in \mathcal{O}\Big(2^{\mathrm{poly}(n)}\Big) \ . \end{aligned}$$

◀

### Lower Bound

Consider the family of BPP nets from Lemma 9 with $\mathcal{L}(N(n), M_0, M_f) = \{a^{2^n}\}$ for all $n \in \mathbb{N}$. The minimal finite state automata recognizing the downward closure $\{a^i \mid i \leqslant 2^n\}$ has at least $2^n$ states.

# 5 SRE Inclusion in Downward Closure

The downward closure of a Petri net language is hard to compute. We therefore propose to under-approximate it by an SRE as follows. Assume we have a heuristic coming up with a candidate SRE that is supposed to be an under-approximation in the sense that its language is included in the downward closure of interest. The problem we study is the algorithmic task of checking whether the inclusion indeed holds. If so, the SRE provides reliable (must) information about the system's behavior, behavior that is guaranteed to occur. This information is useful for finding bugs.

**SRE Inclusion in Downward Closure (SRED)**
**Given:** A Petri net instance $(N, M_0, M_f)$, an SRE $sre$.
**Decide:** $\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$?

## 5.1 Petri Nets

▶ **Theorem 17.** SRED *is* EXPSPACE-*complete for Petri nets.*

Hardness is due to the hardness of coverability [34].

▶ **Lemma 18.** SRED *is* EXPSPACE-*hard for Petri nets.*

**Proof.** We reduce the EXPSPACE-complete coverability problem for Petri nets. Given an Petri net instance $(N, M_0, M_f)$, where $N$ is an unlabeled net, we equip $N$ with the labeling $\lambda(t) = \varepsilon$ for all transitions $t$. We have that $M_f$ is coverable from $M_f$ if and only if $\mathcal{L}(N, M_0, M_f) = \{\varepsilon\}$. If $M_f$ is not coverable, the language is empty) Thus, we have that $M_f$ is coverable iff $\{\varepsilon\} \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$. To conclude the proof, note that $\{\varepsilon\}$ is the language of the SRE $\emptyset^*$. ◀

For the upper bound, we take inspiration from a recent result of Zetzsche [47]. He has shown that, for a large class of models, computing the downward closure is equivalent to deciding an unboundedness problem. We use a variant of this problem that comes with a complexity result. The *simultaneous unboundedness problem for Petri nets* (SUPPN) is, given a Petri net $N$, an initial marking $M_0$, and a subset $X \subseteq P$ of places, decide whether for each $n \in \mathbb{N}$, there is a computation $\sigma_n$ such that $M_0[\sigma_n\rangle M_{\sigma_n}$ with $M_{\sigma_n}(p) \geqslant n$ for all places $p \in X$. In [14], Demri has shown that this problem is EXPSPACE-complete.

▶ **Theorem 19** ([14]). SUPPN *is* EXPSPACE-*complete.*

We turn to the reduction of the inclusion problem SRED to the unboundedness problem SUPPN. Since SREs are choices among products, an inclusion $\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ holds iff $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ holds for all products $p$ in $sre$. Since $\mathcal{L}(N, M_0, M_f)\!\downarrow$ is downward closed, we can further simplify the products by removing choices. Fix a total ordering on the alphabet $\Sigma$. Such an ordering can be represented by a word $w_\Sigma$. We define the *linearization operation* that takes a product and returns a regular expression:

$$\begin{aligned} lin(a+\varepsilon) &= a & lin(a) &= a \\ lin(\Gamma^*) &= (\pi_\Gamma(w_\Sigma))^* & lin(p_1.p_2) &= lin(p_1).lin(p_2)\ . \end{aligned}$$

For example, if $\Sigma = \{a, b, c\}$ and we take $w_\Sigma = abc$, then $p = (a+c)^*(a+\varepsilon)(b+c)^*$ is turned into $lin(p) = (ac)^*a(bc)^*$. The discussion justifies the following lemma.

▶ **Lemma 20.** *$\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ if and only if for all products $p$ in sre we have $\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$.*

**Proof.** $\mathcal{L}(lin(p)) \subseteq \mathcal{L}(sre)$ holds, so one direction is clear.

For the other direction, we show that every word in $\mathcal{L}(sre)$ is a subword of a word in $\mathcal{L}(lin(p))$. From this, if $\mathcal{L}(lin(p))$ is included in the downward closure, then all its subwords will be contained in the downward closure. In particular, all words in $\mathcal{L}(sre)$ will be contained in the downward closure. Towards proving that every word in $\mathcal{L}(sre)$ is a subword of a word in $\mathcal{L}(lin(p))$, note that for any word in $\mathcal{L}((a+\varepsilon))$, the letter $a$ may or may not occur, while in $lin((a+\varepsilon)) = a$, it is forced to occur. Furthermore, given $v \in \Gamma^*$, we have that $v$ is a subword of $\pi_\Gamma(w_\Sigma)^{|v|}$ by dropping in each iteration all but one letter. Therefore, all words in $\Gamma^*$ are subwords of $lin(\Gamma^*)$. If we combine those two insights, the desired statement follows. ◀

With Lemma 20 at hand, it remains to check $\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ for each product. To this end, we reduce this check to SUPPN. We first understand $lin(p)$ as a Petri net $N_{lin(p)}$.. We modify this Petri net by adding one place $p_\Gamma$ for each block $(\pi_\Gamma(w_\Sigma))^* = a_i \dots a_j$. Each transition that repeats or leaves the block (the ones labeled by $a_j$) is modified to generate a token in $p_\Gamma$. As a result, $p_\Gamma$ counts how often the word $\pi_\Gamma(w_\Sigma)$ has been executed.

The second step is to define an appropriate product of $N_{lin(p)}$ with the Petri net of interest. Intuitively, the product synchronizes with the downward closure of $N$.

▶ **Definition 21.** Consider two Petri nets $N_i = (\Sigma, P_i, T_i, F_i, \lambda)$, $i = 1, 2$, with $P_1 \cap P_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$. Their *right-synchronized product* $N_1 \rtimes N_2$ is the labeled Petri net

$$N_1 \rtimes N_2 = (\Sigma, P_1 \uplus P_2, T_1 \uplus T, F, \lambda) \ ,$$

where for the transitions $t_1 \in T_1$, $\lambda$ and $F$ remain unchanged. The new transitions are

$$T = \{merge(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2,\ \lambda_1(t_1) = \lambda_2(t_2)\} \quad \text{with}$$
$$\lambda(merge(t_1, t_2)) = \lambda_1(t_1) = \lambda_2(t_2) \ ,$$
$$F(p_i, merge(t_1, t_2)) = F_i(p_i, t_i), F(merge(t_1, t_2), p_i) = F_i(t_i, p_i) \text{ for } p_i \in P_i, i = 1, 2.$$

As indicated by the name *right-synchronized*, the transitions of $N_1$ can be fired without synchronization, while the transitions of $N_2$ can only be fired if a transition of $N_1$ with the same label is fired simultaneously.

Consider a Petri net $N$ with initial marking $M_0$. We compute the right-synchronized product $N' = N \rtimes N_{lin(p)}$, take the initial marking $M_0'$ that coincides with $M_0$ but puts a token on the initial place of $N_{lin(p)}$, and focus on the counting places $X = \{p_\Gamma \mid (\pi_\Gamma(w_\Sigma))^* \text{ is a block in } p\}$. The following correspondence holds.

▶ **Lemma 22.** *$\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N, M_0, M_\emptyset)\downarrow$ if and only if the places in $X$ are simultaneously unbounded in $N'$ from $M_0'$. Here $M_\emptyset$ is the zero marking, i.e. $M_\emptyset(p) = 0$ for all $p$.*

**Proof.** Let $lin(p) = a_1(\pi_{\Sigma_1}(w_\Sigma))^* a_2 \dots a_k(\pi_{\Sigma_k}(w_\Sigma))^* a_{k+1}$ and let $N' = N \rtimes N_{lin(p)}$ be the right-synchronized product as above.

We first assume that the places in $X$ are simultaneously unbounded in $N'$. Given a word $w \in \mathcal{L}(lin(p))$, we need to find $v$ such that $w \preceq v \in \mathcal{L}(N, M_0, M_\emptyset)$. The word is of the shape

$$w = a_1(\pi_{\Sigma_1}(w_\Sigma))^{n_1} a_2 \dots a_k(\pi_{\Sigma_k}(w_\Sigma))^{n_k} a_{k+1} \ .$$

Define $n = \max n_i$ and let $\sigma$ be the run that creates at least $n$ tokens in each place of $p_{\Sigma_i} \in X$. Since the run creates at least $n$ tokens in $p_i$, it has to fire the transition leaving the block

$\pi_{\Sigma_i}(w_\Sigma))$ $n$ times. This transition is a synchronized transition, i.e. of type $merge(t, t')$. The fact that it could be fired means that before it, we have actually seen the synchronized transitions corresponding to the rest of the block, and before the block the synchronization transition corresponding to $a_i$. Altogether, we obtain that

$$w' = a_1(\pi_{\Sigma_1}(w_\Sigma))^n a_2 \dots a_k(\pi_{\Sigma_k}(w_\Sigma))^n a_{k+1} \ .$$

is a subword of $\lambda(\sigma)$, and by the choice of $n$, $w$ is a subword of $w'$.

Towards a proof for the other direction, assume that $\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N', M_0, 0)\downarrow$ holds. Given any $n$, consider the word

$$w = a_1(\pi_{\Sigma_1}(w_\Sigma))^n \dots a_k(\pi_{\Sigma_k}(w_\Sigma))^n \in \mathcal{L}(lin(p)) \ .$$

Since $w \in \mathcal{L}(N', M_0, 0)\downarrow$, there is a valid firing sequence $\sigma$ with $w \preceq \lambda(\sigma)$. We consider the run $\sigma'$ of $N'$ that we construct as follows: For each $t$ in $\sigma$, whenever $\lambda(t)$ is present in $w$, we fire the synchronized transition $merge(t, t')$ (with suitable $t'$), whenever it is not present, we fire the non-synchronized transition $t$. If this run is a valid firing sequence, it is immediate that it generates $n$ tokens in each place $p_{\Sigma_i}$: Each block $\pi_{\Sigma_i}(w_\Sigma))$ is left $n$ times in $w$, so we trigger the synchronized transition that generates a token in $p_{\Sigma_i}$ $n$ times.

We have to argue why $\sigma'$ is a valid run. First note that on the places of $N$, firing the non-synchronized version $t$ or firing a synchronized version $merge(t, t')$ (for arbitrary suitable $t'$) has the same effect. This shows that the non-synchronized transitions occurring in $\sigma'$ can be fired, and the synchronized transitions satisfy the enabledness-condition on the places of $N$, since $\sigma$ was a valid firing sequence of $N$.

We still have to argue why the enabledness-condition on the places of $N_{lin(p)}$ for the synchronized transitions is also satisfied. This is since $N_{lin(p)}$ was constructed as net with language $\mathcal{L}(lin(p))$, and we only use the synchronized transitions for the subword $w \in \mathcal{L}(lin(p))$. ◀

The lemma does not yet involve the final marking $M_f$. We modify $N'$ and $X$ such that simultaneous unboundedness implies $\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$. The idea is to introduce a new place $p_f$ that can become unbounded only after $M_f$ has been covered. To this end, we also add a transition $t_f$ that consumes $M_f(p)$ tokens from each place $p$ of $N$ and produces one token in $p_f$. We add another transition $t_{pump}$ that consumes one token in $p_f$ and creates two tokens in $p_f$. Call the resulting net $N''$. The new initial marking $M_0''$ coincides with $M_0'$ and assigns no token to $p_f$.

Note that we do not enforce that $t_f$ is only fired after all the rest of the computation has taken place. We can rearrange the transitions in any valid firing sequence of $N''$ to obtain a sequence of the shape $\sigma.{t_f}^k.{t_{pump}}^{k'}$, where $\sigma$ contains neither $t_f$ nor $t_{pump}$.

▶ **Lemma 23.** *$\mathcal{L}(lin(p)) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ iff the places in $X \cup \{p_f\}$ are simultaneously unbounded in $N''$ from $M_0''$.*

To conclude the proof of Theorem 17, it remains to argue that the generated instance for SUPPN is polynomial in the input, i.e. in $(N, M_0, M_f)$ and $p$. The expression $lin(p)$ is certainly linear in $p$, and the net $N_{lin(p)}$ is polynomial in $lin(p)$. The blow-up caused by the right-synchronized product is at most quadratic, and adding the transitions and the places to deal with $M_f$ is polynomial. The size of $M_0''$ is polynomial in the size of $M_0$ and $p$. Altogether, the size of $N''$, $X \cup \{p_f\}$, and $M_0''$ (which together form the generated instance for SUPPN) is polynomial in the size of the original input.

## 5.2 BPP Nets

We show that the problem of deciding whether the language of an SRE is included in the downward closure of a BPP net language is NP-complete.

▶ **Theorem 24.** SRED *for BPP nets is* NP*-complete.*

Hardness can be shown using a reduction from BPP coverability, which is NP-complete, similar to Lemma 18. The hardness of BPP coverability itself can be easily shown by a reduction from SAT, similar to the proof of the NP-hardness of reachability in BPP nets [16].

For membership in NP, we give a reduction to satisfiability of an existential formula in *Presburger arithmetic*, the first-order theory of the natural numbers with addition, subtraction, and order.

▶ **Definition 25.** Let $\mathcal{V}$ be a set of variables with elements $x, y$. The set of terms $t$ in Presburger arithmetic and the set of formulas $\varphi$ are defined as follows:

$$t ::= 0 \mid 1 \mid x \mid t-t \mid t+t \qquad\qquad \varphi ::= t \leqslant t \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \colon \varphi\ .$$

An *existential* Presburger formula takes the form $\exists x_1 \ldots \exists x_n \colon \varphi$ where $\varphi$ is a quantifier-free formula. We shall also write positive Boolean combinations of existential formulas. By an appropriate renaming of the quantified variables, any such formula can be converted into an equivalent existential Presburger formula. We write $\varphi(\vec{x})$ to indicate that (at most) the variables $\vec{x} = x_1, \ldots, x_k$ occur free in $\varphi$. Given a function $M$ from $\vec{x}$ to $\mathbb{N}$, the meaning of $M$ *satisfies* $\varphi$ is as usual and we write $M \models \varphi$ to denote this. We rely on the following complexity result:

▶ **Theorem 26** ([42])**.** *Satisfiability in existential Presburger arithmetic is* NP*-complete.*

Note that $\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ iff the inclusion holds for every product $p$ in *sre*. Given such a product, we construct a new BPP net $N'$ and an existential Presburger formula $\psi(P')$ such that $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ iff there is a marking $M'$ reachable in $N'$ from a modified initial marking $M'_0$ with $M' \models \psi$. This concludes the proof with the help of the following characterization of reachability in BPP nets in terms of existential Presburger arithmetic.

▶ **Theorem 27** ([46, 16])**.** *Given a BPP net $N = (\Sigma, P, T, F, \lambda)$ and an initial marking $M_0$, one can compute in polynomial time an existential Presburger formula $\Psi(P)$ so that for all markings $M$: $M \models \Psi(P)$ if and only if $M_0[\sigma\rangle M$ for some $\sigma \in T^*$.*

After constructing $N'$ and $\psi(P')$, we may use Theorem 27 to construct formula $\Psi(P')$ that characterizes reachability in $N'$. We have that $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ if and only if $\psi(P') \wedge \Psi(P')$ is satisfiable, which can be checked in NP using Theorem 26.

Key to the construction of $N'$ is a characterization of the computations that need to be present in the BPP net for the inclusion $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\!\downarrow$ to hold. Wlog., in the following we will assume that the product takes the shape

$$(a_1 + \varepsilon)\Sigma_1^*(a_2 + \varepsilon) \ldots \Sigma_{n-1}^*(a_n + \varepsilon),$$

where $\Sigma_1, \ldots, \Sigma_{n-1} \subseteq \Sigma$ and $a_1, \ldots, a_n \in \Sigma$. For this language to be included in $\mathcal{L}(N, M_0, M_f)\!\downarrow$, the BPP should have a computation with parts $\sigma_i$ containing $a_i$ and parts $\rho_i$ between the $\sigma_i$ that contain all letters in $\Sigma_i$ and that can be repeated. To formalize the requirement, recall that we use $w_\Sigma$ for a total order on the alphabet and $\pi_{\Sigma_i}(w_\Sigma)$ for the projection to $\Sigma_i \subseteq \Sigma$.

Moreover, we define $M \leqslant^c M'$, with $c$ the constant defined in Lemma 16, if for all places $p \in P$ we have $M'(p) < c$ implies $M(p) \leqslant M'(p)$.

▶ **Definition 28.** Let $p$ be a product. The BPP net $N$ together with the markings $M_0, M_f$ admits a *$p$-witness* if there is a computation

$$M_0 = M_1[\sigma_0\rangle M_1'[\rho_1\rangle M_2[\sigma_1\rangle M_2'[\rho_2\rangle \dots M_{n-1}'[\rho_{n-1}\rangle M_n[\sigma_n\rangle M_n' \leqslant^c M_n' \ ,$$

i.e. there are markings markings $M_1, M_1', \dots, M_n, M_n'$ and firing sequences $\sigma_i$, $\rho_i$ that satisfy $M_i[\sigma_i\rangle M_i'$ for all $i \in [1..n]$, $M_i'[\rho_i\rangle M_{i+1}$ for all $i \in [1..n-1]$, and moreover:
(1) $a_i \preceq \lambda(\sigma_i)$, for all $i \in [1..n]$,
(2) $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$ for all $i \in [1..n-1]$,
(3) $M_i' \leqslant^c M_{i+1}$ for all $i \in [1..n-1]$, and
(4) $M_1 = M_0$ and $M_f \leqslant^c M_n'$.

In a $p$-witness, (1) enforces that the $a_i$ occur in the desired order, and the first part of (2) requires that $\pi_{\Sigma_i}(w_\Sigma)$ occurs in between. Property (3) means that each $\rho_i$ (and thus $\pi_{\Sigma_i}(w_\Sigma)$) can be repeated. Property (4) enforces that the computation still starts in the initial marking and can be extended to cover the final marking.

The following proposition reduces the problem SRED for BPP nets to checking whether the BPP admits a $p$-witness.

▶ **Proposition 29.** *$\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ holds iff $(N, M_0, M_f)$ admits a $p$-witness.*

**Proof.** Recall that we consider a product of the form

$$p = (a_1 + \varepsilon)\Sigma_1^*(a_2 + \varepsilon)\Sigma_2^* \cdots \Sigma_{n-1}^*(a_n + \varepsilon) \ .$$

Assume that $(N, M_0, M_f)$ admits a $p$-witness $(M_1, M_1', \cdots, M_n, M_n')$ such that

$$M_0 = M_1[\sigma_0\rangle M_1'[\rho_1\rangle M_2[\sigma_1\rangle M_2'[\rho_2\rangle \dots M_{n-1}'[\rho_{n-1}\rangle M_n[\sigma_n\rangle M_n' \leqslant^c M_n' \ ,$$

satisfying the required properties. We will show $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ by proving that for any word $w \in \mathcal{L}(p)$, there is a run $M_0[\sigma\rangle M''$ such that $w \preceq \lambda(\sigma)$ and $M_n' \leqslant^c M''$ (and hence also $M_f \leqslant M''$). Let $w = x_1 v_1 \cdots v_{n-1} x_n$, where $x_i \in \{a_i, \varepsilon\}$ and $v_i \in \Sigma_i^*$.

In sequel, we will prove that for every prefix $w' = x_1 v_1 \cdots x_i$, we have a run of the form $M_0[\gamma\rangle M_i'''$ such that $w' \preceq \lambda(\gamma)$ and $M_i' \leqslant^c M_i'''$. Similarly, we show that for any prefix of the form $w' = x_1 v_1 \cdots x_i v_i'$, where $v_i'$ is a prefix of $v_i$, we have a run of the form $M_0[\gamma\rangle M_i''$ such that $w' \preceq \lambda(\gamma)$ and $M_i \leqslant^c M_i''$.

We prove both statements simultaneously by induction. Consider the first statement for $i = 1$: We have $w' = x_1$, and $M_1[\sigma_1\rangle M_1'$ is the required run by Property (1).

Consider the second statement for some $i$ for which we assume that the first statement holds. Let

$$w'' = x_1 v_1 \dots x_i v_i' \ ,$$

where $v_i'.a$ is a prefix of $v_i$. To show the required statement, we consider an inner induction on the length of $v_i'$. In the base case, $v_i' = \varepsilon$, and the statement is immediate by the hypothesis of the outer induction. Assume that $v_i' = v_i''.b$, then by the inner induction, we have a run $M_1[\gamma\rangle M_i''$ such that $x_1 v_1 \cdots x_i v_i'' \preceq \lambda(\gamma)$ and $M_i \leqslant^c M_i''$. If for all places $p \in P$, $M_i''(p) < c$ holds, then we have $M_i \leqslant M_i''$. We prolong $\gamma$ by $\rho_i$, which is possible by Property (3), and get the required run by Property (2). Suppose the set of places $X$ to which more than $c$ tokens are assigned is non-empty. Using Lemma 16 repeatedly for each place in $X$, we pump the run to create enough tokens to be able to execute the rest of the run. We obtain a run $M_1[\gamma'\rangle M_i^*$ such that $\lambda(\gamma) \preceq \lambda(\gamma')$, for all $x \in X$ $M_i^*(x) - M_i''(x) \geqslant |\rho_i|$

and $M''_i \leqslant^c M^*_i$. This is can be prolonged by $\rho_i$ to obtain the desired computations. This concludes the inner induction and the proof of the second statement for $i$.

It remains to prove the first statement for $i+1$, assuming that the second statement holds for $i$. Consider

$$w' = x_1 v_1 x_2 \dots x_i v_i x_{i+1} \ .$$

By induction, we get a run $M_1[\gamma\rangle M''_i$ such that $x_1 v_1 x_2 \cdots x_i v_i \preceq \lambda(\gamma)$ and $M_i \leqslant^c M''_i$. Suppose for all places $p \in P$, $M''_i < c$, then we have $M_i \leqslant M''_i$. Hence we can easily extend the computation by $M''_i[\sigma_i\rangle M'''_i$, which gives us the required run. Otherwise, we proceed as for the second statement and pump up the values in these places to be greater than the size of $\sigma_i$. Afterwards, we can extend the computation by $\sigma_i$, obtaining the desired run.

For the other direction, consider the word

$$w = a_1.(\pi_{\Sigma_1}(w_\Sigma))^{\ell \cdot c+1}.a_2.(\pi_{\Sigma_2}(w_\Sigma))^{\ell \cdot c+1}.a_3 \cdots a_n \in L(p) \ .$$

Since we have $\mathcal{L}(p) \in \mathcal{L}(N, M_0, M_f)\downarrow\downarrow$, we have a run of the form

$$M_1[\alpha_1\rangle J'_1[\beta_1\rangle J_1[\alpha_2\rangle J'_2[\beta_2\rangle J_2 \cdots J'_n \ ,$$

such that $a_i \preceq \lambda(\alpha_i)$ and $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\beta_i)$. Since the length of $\beta_i$ is $\ell \cdot c + 1$, there have to be markings between $J'_i$ and $J_i$ such that

$$J'_i[\beta_i^1\rangle J_i^1[\beta_i^2\rangle J_i^2[\beta_i^3\rangle J_i \ ,$$

where $J_i^1 \leqslant^c J_i^2$. Now, we let $M'_i = J_i^1$, $M_i = J_i^2$, $\sigma_1 = \alpha_1.\beta_1^1$, $\sigma_i = \beta_{i-1}^3.\alpha_i.\beta_i^1$ and $\rho_i = \beta_i^2$. This gives us the required $p$-witness. ◀

We now reduce the problem of finding a $p$-witness to finding in a modified unlabeled BPP net $N' = (\emptyset, P', T', F', \lambda')$ a reachable marking that satisfies a Presburger formula $\Psi_{M_f}^{M_0}(P')$. The task is to identify $2n$ markings that are related by $2n-1$ computations as required by a $p$-witness. The idea is to create $2n-1$ replicas of the BPP net and run them independently to guess the corresponding computations $\sigma_i$ resp. $\rho_i$. The Presburger formula $\Psi_{M_f}^{M_0}$ will check that the target marking reached with $\sigma_i$ coincides with the initial marking for $\rho_i$, and the target marking reached with $\rho_i$ is the initial marking of $\sigma_{i+1}$. To this end, the net $N'$ remembers the initial marking that each replica started from in a full copy (per replica) of the set of places of the BPP net. Furthermore $\Psi_{M_f}^{M_0}$ checks that each $\rho^i$ can be repeated by ensuring that the final marking in the corresponding replica is larger than the initial marking. As initial marking for $N'$, we consider the marking $M_\emptyset$ with $M_\emptyset(p) = 0$ for all $p$.

Formally, the places of $N'$ are

$$P' = \bigcup_{i \in [1..2n-1]} B_i \cup E_i \cup L_i \ .$$

Here, $E_i = \{e_i^p \mid p \in P\}$ and $B_i = \{b_i^p \mid p \in P\}$ are $2n-1$ copies of the places of the given BPP net. The computation $\sigma_i$ or $\rho_i$ is executed on the places $E_i$, which will hold the target marking $M'_i$ or $M_{i+1}$ reached after the execution. The places $B_i$ remember the initial marking of the replica and there are no transitions that take tokens from them. The places $L_i$ record occurrences of $a_i$ and of symbols from $\Sigma_i$, depending on whether $i$ is odd or even. For all $i \in [1..n]$, we have $L_{2i-1} = \{l_{2i-1}\}$. For all $i \in [1..n-1]$, we set $L_{2i} = \{l_{2i}^a \mid a \in \Sigma_i\}$ otherwise. The transitions are

$$T' = \bigcup_{i \in [1..2n-1]} \mathrm{TC}_i \cup \mathrm{TE}_i \ .$$

The $\mathrm{TC}_i = \{\mathrm{tc}_i^p \mid p \in P\}$ populate $E_i$ and $B_i$. The transitions in $\mathrm{TE}_i = \left\{\mathrm{te}_i^t \;\middle|\; t \in T\right\}$ together with $E_i$ form a replica of the BPP net. The flow relation $F'$ is defined as follows, where numbers omitted are zero:

(1) For all $i \in [1..2n-1]$, $p \in P$, $F'(\mathrm{tc}_i^p, e_i^p) = F'(\mathrm{tc}_i^p, b_i^p) = 1$.
(2) For all $i \in [1..2n-1]$, $p \in P$, $t \in T$, $F'(\mathrm{te}_i^t, e_i^p) = F(t,p)$ and $F'(e_i^p, \mathrm{te}_i^t) = F(p,t)$.
(3) For all $i \in [1..n]$, $t \in T$ with $\lambda(t) = a_i$, $F'(\mathrm{te}_{2i-1}^t, l_{2i-1}) = 1$.
(4) For all $i \in [1..n]$, $t \in T$ with $\lambda(t) = a \in \Sigma_i$, $F'(\mathrm{te}_{2i}^t, l_{2i}^a) = 1$

The Presburger formula wrt. the initial and final markings $M_0$ and $M_f$ of $N$ has the places in $P'$ as variables. It takes the shape

$$\Psi_{M_f}^{M_0}(P') = \Psi_1(P') \wedge \Psi_2(P') \wedge \Psi_3(P') \wedge \Psi_4(P') \wedge \Psi_5^{M_0}(P') \wedge \Psi_6^{M_f}(P') \ ,$$

where

$$\begin{aligned}
\Psi_1(P') &= \bigwedge_{i\in[1..2n-2]} \bigwedge_{p\in P} e_i^p = b_{i+1}^p \\
\Psi_2(P') &= \bigwedge_{i\in[1..n-1]} \bigwedge_{p\in P} (e_{2i}^p < c \ \rightarrow \ b_{2i}^p \leqslant e_{2i}^p) \\
\Psi_3(P') &= \bigwedge_{i\in[1..n]} l_{2i-1} > 0 \\
\Psi_4(P') &= \bigwedge_{i\in[1..n-1]} \bigwedge_{a\in\Sigma_i} l_{2i}^a > 0 \\
\Psi_5^{M_0}(P') &= \bigwedge_{p\in P} b_1^p = M_0(p) \\
\Psi_6^{M_f}(P') &= \bigwedge_{p\in P} (e_{2n-1}^p < c \ \rightarrow \ e_{2n-1}^p \geqslant M_f(p)) \ .
\end{aligned}$$

Formula $\Psi_1$ states that $\sigma_i$ ends in the marking $M_i'$ that $\rho_i$ started from, and similarly $\rho_i$ ends in $M_{i+1}$ that $\sigma_{i+1}$ started from. Formula $\Psi_2$ states the required $\leqslant^c$ relation. To make sure we found letter $a_i$, we use $\Psi_3$. With $\Psi_4$, we express that all letters from $\Sigma_i$ have been seen. Conjunct $\Psi_5^I$ says that the places $b_1^p$ have been initialized to the value given by the initial marking $I$. Formula $\Psi_6^F$ states the condition on covering the final marking. The correctness of the construction is the next lemma. Note that the transitions $\mathrm{TC}_i$ are always enabled. Therefore, we can start in $N'$ from the initial marking $M_\emptyset$ that assigns zero to every place.

► **Proposition 30.** *There are $\sigma'$ and $M'$ so that $M_\emptyset[\sigma'\rangle M'$ in $N'$ and $M' \models \Psi_{M_f}^{M_0}$ if and only if $(N, M_0, M_f)$ admits a p-witness.*

**Proof.** Assume a $p$-witness $M_1, \ldots, M_{2n}$. We construct a run $M_\emptyset[\sigma'\rangle M'$ of $N'$ with $M' \models \Psi_{M_f}^{M_0}$ as follows:

$$\sigma' = \gamma_1\alpha_1\gamma_1'\beta_1\gamma_2\alpha_2\gamma_2'\beta_2 \ldots \gamma_n\alpha_n\gamma_n'\beta_n \ ,$$

where the $\alpha_i$ corresponds to $\sigma_i$ executed on the places $E_{2i-1}$ by using the transitions in $\mathrm{TE}_{2i-1}$ (i.e. if a transition $t \in T$ is used in $\sigma_i$, then the transition $\mathrm{te}_{2i-1}^t \in \mathrm{TE}_{2i-1}$ is used in $\alpha_i$). Similarly, the $\beta_i$ correspond to the $\rho_i$ executed on $E_{2n}$ using transitions in $\mathrm{TE}_{2n}$. The $\gamma_i$ and $\gamma_i'$ populate the set of places $E_i$ accordingly: $\gamma_1$ produces $M_1(p)$ many tokens on each place $e_1^p$ of $E_1$ using the transition in $\mathrm{TC}_1$. For $i > 1$, $\gamma_i$ produces $M_{2i-1}(p)$ many tokens on each place $e_{2i-1}^p$ of $E_{2i-1}$, and $\gamma_i'$ produces $M_{2i}(p)$ many tokens on each place $e_{2i}^p$

of $E_{2i}$. As a by-product, the places in each $B_i$ are also populated. It is easy to check that the marking $M'$ with $M_\emptyset[\sigma'\rangle M'$ indeed satisfies $\Psi_{M_f}^{M_0}$.

For the other direction, assume that a computation $M_\emptyset[\sigma'\rangle M'$ with $M' \models \Psi_{M_f}^{M_0}$ is given. First, observe that the transitions in $\mathrm{TC}_i$ and $\mathrm{TE}_i$ are not dependent on each other and hence can be independently fired. Furthermore, for $i \neq j$, the transitions in $\mathrm{TE}_i$ and $\mathrm{TE}_j$ are independent of each other. Therefore, we may assume that in $\sigma'$, the copy transitions in $\mathrm{TC} = \bigcup_{i\in[1..2n-1]} \mathrm{TC}_i$ are fired first, then the transition in $\mathrm{TE}_1$ followed by transition in $\mathrm{TE}_2$ and so on. All together, we may assume that the computation is of the form

$$M_\emptyset[\sigma''.\sigma'_1.\cdots.\sigma'_{2n-1}\rangle M' \ ,$$

where $\sigma'' = \pi_{\mathrm{TC}}(\sigma')$ and $\sigma'_i = \pi_{\mathrm{TE}_i}(\sigma')$ for all $i \in [1..2n-1]$ Note that for each $i \in [1..2n-1]$, the transition sequence $\sigma'_i$ in $N'$ induces a transition sequence $\alpha_i$ in the original net $N$ by using transition $t \in T$ instead of $\mathrm{te}_i^t \in \mathrm{TE}_i$.

The initial phase $\sigma''$ populates each $E_i$ with some initial marking. This initial marking is also copied to the places $B_i$, and these places are not touched during the rest of the computation. We may obtain a marking $J_i$ of $N$ for each $i \in [1..2n-1]$ by $J_i(p) = M'(b_i^p)$. For each $i \in [1..2n-1]$, we obtain a marking $K_i$ of $N$ by considering the assignment of tokens to the places of $E_i$ by $M'$, i.e. $K_i(p) = M'(e_i^p)$.

We claim that $M_0, K_1, K_2, \ldots K_{2n-1}$ is the required $p$-witness. To argue that they indeed satisfy the Properties (1) to (4), we use the fact that $J_i[\sigma'_i\rangle K_i$ as well as $M' \models \Psi_{M_f}^{M_0}$.

(a) Since $M' \models \Psi_5^{M_0}$, we have $J_1 = M_0$.
(b) Since $M' \models \Psi_6^{M_f}$, we have $K_{2n-1} \leqslant^c M_f$.
(c) Since $M' \models \Psi_1$, we have $J_{i+1} = K_i$.
(d) Since $M' \models \Psi_2$, we have $J_{2i} \leqslant^c K_{2i}$ for all $i \in [1..n[$.
(e) Since $M' \models \Psi_3$, we have for all $i \in [1..n]$, $a_i \preceq \lambda(\sigma'_{2i-1})$.
(f) Since $M' \models \Psi_4$, we have for all $i \in [1..n[$, for all $a \in \Sigma_i$, $a \preceq \lambda(\sigma_{2i})$.

We conclude that Property (4) holds using (a) and (b). We conclude Property (2), $a_i \preceq \lambda(\sigma_i)$, using (e) and Property (3), $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$, using (f). Finally, (b) and (d) yields the required Property (3). ◀

## 6 SRE Inclusion in Upward Closure

Rather than computing the upward closure of a Petri net language we now check whether a given SRE under-approximates it. Formally, the problem is defined as follows.

**SRE Inclusion in Upward Closure (SREU)**
**Given:** A Petri net instance $(N, M_0, M_f)$, an SRE *sre*.
**Decide:** $\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$?

### 6.1 Petri Nets

▶ **Theorem 31.** SREU *is* EXPSPACE-*complete for Petri nets.*

The EXPSPACE lower bound is immediate by hardness of coverability for Petri nets and can be proven similar to Lemma 18. The upper bound is due to the following fact: We only need to check whether the set of minimal words in the language of the given SRE is included in the upward closure of the Petri net language. Note that the minimal word of a product can be computed as follows:

$$\min(a) = a \qquad \min(p.p') = \min(p).\min(p')$$
$$\min(a + \varepsilon) = \varepsilon \qquad \min(\Gamma^*) = \varepsilon \ .$$

For an SRE $sre = p_1 + \ldots + p_n$, we have that the set of minimal words is a subset of $\{\min p_1, \ldots, \min p_n\}$. We have that

$$\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow \quad \text{iff} \quad \min p_i \in \mathcal{L}(N, M_0, M_f)\uparrow \text{ for all } i \in [1..n] \ .$$

For each product, the membership check $\min p_i \in \mathcal{L}(N, M_0, M_f)\uparrow$ can be reduced in polynomial time to coverability in Petri nets. Since the number of minimal words in the SRE language is less than the size of the SRE, and coverability is well-known to be in EXPSPACE [40], we obtain our EXPSPACE upper bound.

## 6.2 BPP Nets

▶ **Theorem 32.** SREU *is* NP*-complete for BPP nets.*

As before, the hardness is by a reduction of the coverability problem for BPP nets. For the upper bound, the algorithm is similar to the one for checking the inclusion of an SRE in the downward closure of a BPP language.

**Proof.** To check $\mathcal{L}(sre) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$, it is sufficient to check $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$ for each product in $sre$. Consider one such product $p$. The inclusion $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$ holds iff the minimal word of $\mathcal{L}(p)$, say $\min p = a_1 \ldots a_n$, belongs to $\mathcal{L}(N, M_0, M_f)\uparrow$. This in turn holds iff one of its subwords is in $\mathcal{L}(N, M_0, M_f)$. We check this by deciding whether a reachable marking $M$ in a different net $N'$ satisfies a Presburger formula $\Psi$.

We describe the BPP net $N'$ and the Presburger formula $\Psi$ that together characterize the subwords of $\min(p)$ included in the language of the BPP net. Net $N'$ is constructed similar to the net $N'$ from Section 5.2. We have two copies of the places for each $i \in [1..n]$, the places $B_i$ hold a copy of the guessed marking and $E_i$ provides a copy $e_i^p$ of the BPP net places $p$. Additionally for each $i$ we have a place $L_i = \{l_i\}$. The transitions $\mathrm{TC}_i$ populate the copy $E_i$ of the BPP net and store the same marking in $B_i$. The transitions $\mathrm{TE}_i$ contain a copy $\mathrm{te}_i^t$ of each BPP net transition $t$. To check for a subword of $a_1 \ldots a_n$, in each stage $i$ we only enable transitions $t$ that are either labeled by $\varepsilon$ or $a_i$, i.e. for all $p \in P$, $t \in T$, if $\lambda(t) = \varepsilon$ or $\lambda(t) = a_i$ we have $F'(\mathrm{te}_i^t, e_i^p) = F(t, p)$ and $F'(e_i^p, \mathrm{te}_i^t) = F(p, t)$. We also count the number of times a transition labeled $a_i$ is executed using place $l_i$, i.e. for all $t \in T$ such that $\lambda(t) = a_i$, we let $F'(\mathrm{te}_i^t, l_i) = 1$.

Now the required Presburger formula $\Psi$ — apart from checking that (1) the net starts with the initial marking, (2) covers the final marking, (3) the guessed marking in each stage is the same as the marking reached in the previous stage — also checks whether in each stage at most one non-epsilon transition is used, $\bigwedge_{i \in 1..n} l_i \leqslant 1$. This guarantees we have seen a subword of $a_1 \ldots a_n$. The initial marking $M_\emptyset$ is one that assigns zero to all places. It is easy to see that $\mathcal{L}(p) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$ iff there is a computation $M_\emptyset[\sigma\rangle M$ in $N'$ such that $M \models \Psi$. ◀

## 7 Being Upward/Downward Closed

We now study the problem of deciding whether a Petri net language actually is upward or downward closed, i.e. whether the closure that we can compute is actually a precise representation of the system's behavior. Formally, the problems BUC and BDC are defined as follows.n

**Being upward closed (BUC)**
**Given:** A Petri net instance $(N, M_0, M_f)$.
**Decide:** $\mathcal{L}(N, M_0, M_f) = \mathcal{L}(N, M_0, M_f)\uparrow$?

**Being downward closed (BDC)**
**Given:** A Petri net instance $(N, M_0, M_f)$.
**Decide:** $\mathcal{L}(N, M_0, M_f) = \mathcal{L}(N, M_0, M_f)\downarrow$?

▶ **Theorem 33.** BUC *and* BDC *are decidable for Petri nets.*

Note that $\mathcal{L}(N, M_0, M_f) \subseteq \mathcal{L}(N, M_0, M_f)\uparrow$ and $\mathcal{L}(N, M_0, M_f) \subseteq \mathcal{L}(N, M_0, M_f)\downarrow$ trivially hold. In both cases, it remains to decide the converse inclusion. Now note that $\mathcal{L}(N, M_0, M_f)\uparrow$ (resp. $\mathcal{L}(N, M_0, M_f)\downarrow$) is a regular language for which we can construct a generating FSA by Theorem 1 (resp. using [22]).

To prove Theorem 33 it is thus sufficient to show how to decide $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$ for any given FSA $A$. This regular inclusion should be a problem of independent interest.

**Containing a regular language**
**Given:** A Petri net instance $(N, M_0, M_f)$, FSA $A$.
**Decide:** $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$?

▶ **Theorem 34.** $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$ *is decidable.*

To prove this theorem, we rely on a result of Esparza et. al [26] that involves the *traces* of an FSA (resp. Petri net), labelings of computations that start from the initial state (resp. initial marking), regardless of whether they end in a final state (resp. covering marking). For a finite automaton $A$, we define

$$\mathcal{T}(A) = \{w \in \Sigma^* \mid q_{init} \xrightarrow{w} q \text{ for some } q \in Q\}.$$

Similarly, for a Petri net, we define

$$\mathcal{T}(N, M_0) = \{w \in \Sigma^* \mid \exists\, \sigma \in T^* \colon \lambda(\sigma) = w, M_0[\sigma\rangle M \text{ for some marking } M\}\ .$$

Note that both languages are necessarily *prefix closed*, e.g. if $w \in \mathcal{L}(A)$ for some FSA $A$, then for any prefix $v$ of $v$, we have $v \in \mathcal{L}(A)$.

▶ **Theorem 35** ([26]). *The inclusion* $\mathcal{T}(A) \subseteq \mathcal{T}(N, M_0)$ *is decidable.*

The algorithm constructs a computation tree of $A$ and $N$. This tree determinizes $N$ in that it tracks sets of incomparable markings reachable with the same trace. The construction terminates if either the set of markings becomes empty and the inclusion fails or (the automaton deadlocks or) we find a set of markings that covers a predecessor and the inclusion holds. The latter is guaranteed to happen due to the well-quasi ordering (wqo) of sets of markings. This dependence on wqos does not allow us to derive a complexity result.

We now show how to reduce checking the inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$ to deciding an inclusion among trace languages. Theorem 35 can be used to decide this inclusion. Let $(N, M_0, M_f)$ be the Petri net instance of interest, and let $A$ be the given FSA. As language $\mathcal{L}(N, M_0, M_f)$ is not prefix-closed in general, we consider the zero marking $M_\emptyset$ as the new final marking. This yields a prefix-closed language with $\mathcal{T}(N, M_0) = L(N, M_0, M_\emptyset)$, since now all valid firing sequences give a word in the language, and prefixes of valid firing sequences are again valid firing sequences. We still need to take the original final marking $M_f$ into account. To do so, we modify the net by adding a new transition that can only be fired after $M_f$ has been covered. Let $a \notin \Sigma$ be a fresh letter. Let $N.a$ be the Petri net that is obtained from $N$ and the given final marking $M_f$ by adding a new transition $t_{\mathit{final}}$ that consumes $M_f(p)$ many tokens from every place $p$ of $N$ and that is labeled by $a$. For the automaton, we use a similar trick. Let $A.a$ be an automaton for $\mathcal{L}(A).a$ that is reduced in the sense that the unique final state is reachable from every state.

▶ **Lemma 36.** *$\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$ holds iff $\mathcal{T}(A.a) \subseteq \mathcal{T}(N.a, M_0)$ holds.*

**Proof.** Assume the first inclusion holds and consider a word $v$ from $\mathcal{T}(A.a)$. We have to show membership of $v$ in $\mathcal{T}(N.a, M_0)$. As the unique final state of $A.a$ is reachable from every state, $v$ is a prefix of some word in the language $\mathcal{L}(A).a$, say $w.a$, where $w$ stems from $\mathcal{L}(A)$. The assumed first inclusion now yields $w \in \mathcal{L}(N, M_0, M_f)$. Thus, there is a $w$-labeled computation $M_0[\sigma\rangle M$ of $N$ with $M \geqslant M_f$. We obtain that $M_0[\sigma.t_{\mathit{final}}\rangle M'$ is a valid computation of $N.a$, thus, $w.a = \lambda(w.t_{\mathit{final}}) \in \mathcal{T}(N.a, M_0)$. Since trace languages are prefix closed and $v$ is a prefix of $w.a$, we obtain $v \in \mathcal{T}(N.a, M_0)$ as desired.

Assume the second inclusion holds and consider a word $w$ from $\mathcal{L}(A)$. The task is to prove membership of $w$ in $\mathcal{L}(N, M_0, M_f)$. To do so, note that $w.a \in \mathcal{L}(A).a \subseteq \mathcal{T}(A.a)$. By the assumption, we have $w.a \in \mathcal{T}(N.a, M_0)$. Thus, there is a valid computation $M_0[\sigma.t_{\mathit{final}}\rangle M'$ of $N.a$ with $\lambda(\sigma.t_{\mathit{final}}) = w.a$. (Here, we have used that $t_{\mathit{final}}$ is the only $a$-labeled transition). Since $w \in \mathcal{L}(A) \subseteq \Sigma^*$, and $a \notin \Sigma$, we have that $\sigma$ does not contain an occurrence of $t_{\mathit{final}}$, so $M_0[\sigma\rangle M$ is a valid computation of $N$. As $t_{\mathit{final}}$ could be fired in $M$, we have $M \geqslant M_f$ and $\sigma$ is indeed a covering computation in $N$. We conclude $\lambda(\sigma) = w \in \mathcal{L}(N, M_0, M_f)$ as desired. ◀

Combining Lemma 36 and Theorem 35 yields the proof of Theorem 34, which in turn proves Theorem 33: Given an FSA $A$ an a Petri net instance $(N, M_0, M_f)$ for which we should decide $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$, we construct $N.a$ and $A.a$, and apply Theorem 35 to decide $\mathcal{T}(A.a) \subseteq \mathcal{T}(N.a, M_0)$, which is equivalent to deciding $\mathcal{L}(A) \subseteq \mathcal{L}(N, M_0, M_f)$ by Lemma 36.

## 8 Conclusion

We considered the class of Petri net languages with coverability as the acceptance condition and studied the problem of computing representations for the upward and downward closure. For the upward closure of a Petri net language, we showed how to effectively obtain an optimal finite state representation of size at-most doubly exponential in the size of the input. In the case of downward closures, we showed an instance for which the minimum size of any finite state representation is at-least non-primitive recursive.

To tame the complexity, we considered two variants of the closure computation problem. The first restricts the input to BPP nets, which can be understood as compositions of unboundedly many finite automata. For BPPs, we showed how to effectively obtain an optimal finite state representation of size at-most exponential in the size of input, for both the upward and the downward closure of the language.

The second variant takes as input a simple regular expression (SRE), which is meant to under-approximate the upward or downward closure of a given language. For Petri net languages, we found an optimal algorithm that uses at-most exponential space to check whether a given SRE is included in the upward/downward closure. In the case of BPP nets, we showed that this problem is NP-complete.

Finally, we showed that, given a Petri net, deciding whether its language actually is upward or downward closed is decidable. If the check is successful, the finite state descriptions we compute are precise representations of the system behavior.

An interesting problem for future work is the complexity of checking separability by piecewise-testable languages (PTL) and the size of separators. A PTL is a Boolean combination of upward closures of single words. PTL-separability takes as input two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ and asks whether there is a PTL $\mathcal{S}$, called the separator, that includes $\mathcal{L}_1$ and has an empty intersection with $\mathcal{L}_1$. Taking a verification perspective, the separator is an over-approximation of the system behavior $\mathcal{L}_1$ that is safe wrt. the bad behaviors in $\mathcal{L}_1$. For deterministic finite state automata, PTL-separability was shown to be decidable in polynomial time by Almeida and Zeitoun [3], a result that was generalized to non-deterministic automata in [12]. Recently [13], Czerwiński, Martens, van Rooijen, Zeitoun, and Zetzsche have show that, for full trios, computing downward closures and deciding PTL-separability are recursively equivalent. A full trio is a class of languages that is closed under homomorphisms, inverse homomorphisms, and regular intersection. Petri net languages with coverability or reachability as the acceptance condition satisfy these requirements. Hence, we know that PTL-separability is decidable for them [22]. The aforementioned problems, however, remain open.

## References

1 P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *FMSD*, 25(1), 2004.

2 P. A. Abdulla, G. Delzanno, and L. V. Begin. Comparing the expressive power of well-structured transition systems. In *CSL*, LNCS. Springer, 2007.

3 J. Almeida and M. Zeitoun. The pseudovariety J is hyperdecidable. *RAIRO: ITA*, 31(5):457–482, 1997.

4 M. F. Atig, A. Bouajjani, K. Narayan Kumar, and P. Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS*, LIPIcs. Dagstuhl, 2014.

5 M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *LMCS*, 7(4), 2011.

6 M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, LNCS. Springer, 2008.

7 M. F. Atig, D. Chistikov, P. Hofman, K. N. Kumar, P. Saivasan, and G. Zetzsche. Complexity of regular abstractions of one-counter languages. In *LICS*, pages 207–216. ACM, 2016.

8 M. F. Atig, R. Meyer, S. Muskalla, and P. Saivasan. On the Upward/Downward Closures of Petri Nets. In *MFCS*, volume 83 of *LIPIcs*, pages 49:1–49:14. Dagstuhl, 2017. doi:10.4230/LIPIcs.MFCS.2017.49.

9 G. Bachmeier, M. Luttenberger, and M. Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In *LATA*, LNCS. Springer, 2015.

10 L. Clemente, P. Parys, S. Salvati, and I. Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *LICS*, pages 96–105. ACM, 2016.

11 B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 1991.

12 W. Czerwinski, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *ICALP*, volume 7966 of *LNCS*, pages 150–161. Springer, 2013.

13 W. Czerwinski, W. Martens, L. van Rooijen, M. Zeitoun, and G. Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

14 S. Demri. On selective unboundedness of VASS. *JCSS*, 79(5), 2013.

15 S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching vector addition systems. *Journal of Computer and System Sciences*, 79(1):23 – 38, 2013.

16 J. Esparza. Petri Nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31(1), 1997.

17 J. 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.

18 J. Esparza and K. Heljanko. *Unfoldings — A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008.

19 J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52, 1994.

20 A. Finkel, G. Geeraerts, J. F. Raskin, and L. V. Begin. On the omega-language expressive power of extended Petri nets. *ENTCS*, 2005.

21 H. Gruber, M. Holzer, and M. Kutrib. More on the size of Higman-Haines sets: Effective constructions. In *MCU*, LNCS. Springer, 2007.

22 P. Habermehl, R. Meyer, and H. Wimmel. The downward-closure of Petri net languages. In *ICALP*, LNCS. Springer, 2010.

23 M. Hague, J. Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL*. ACM, 2016.

24 L. H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1), 1969.

25 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.* (3), 2(7), 1952.

26 P. Jančar, J. Esparza, and F. Moller. Petri nets and regular processes. *J. Comput. Syst. Sci.*, 59(3):476–503, December 1999.

27 R. M. Karp and R. E. Miller. Parallel program schemata. *JCSS*, 3(2):147–195, 1969.

28 S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*. ACM, 1982.

29 S. La Torre, A. Muscholl, and I. Walukiewicz. Safety of parametrized asynchronous shared-memory systems is almost always decidable. In *CONCUR*, LIPIcs. Dagstuhl, 2015.

30 J. L. Lambert. A structure to decide reachability in Petri nets. *TCS*, 99(1), 1992.

31 J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*. ACM, 2011.

32 J. Leroux, V. Penelle, and G. Sutre. On the context-freeness problem for vector addition systems. In *LICS*. IEEE, 2013.

33 J. Leroux, M. Praveen, and G. Sutre. A relational trace logic for vector addition systems with application to context-freeness. In *CONCUR*, pages 137–151. Springer, 2013.

34 R. J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Department of Computer Science, 1976.

35 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, LNCS. Springer, 2012. `doi:10.1007/978-3-642-28872-2_25`.

36 E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comp.*, 13(3), 1984.

37 E. W. Mayr and A. R. Meyer. The complexity of the finite containment problem for Petri nets. *JACM*, 28(3), 1981.

38 R. Mayr. Undecidable problems in unreliable computations. *TCS*, 1-3(297), 2003.

39 R. Parikh. On context-free languages. *JACM*, 13(4), 1966.

40 C. Rackoff. The covering and boundedness problems for vector addition systems. *TCS*, 6(2), 1978.

41 W. Reisig. *Petri nets: An Introduction.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1985.

42 B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the AMS*, 284(1), 1984.

43 S. R. Schwer. The context-freeness of the languages associated with vector addition systems is decidable. *TCS*, 1992.

44 R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *JCSS*, 23(3), 1981.

45 J. van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3), 1978.

46 K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *CADE*, pages 337–352. Springer, 2005.

47 G. Zetzsche. An approach to computing downward closures. In *ICALP*, LNCS. Springer, 2015.

48 G. Zetzsche. Computing downward closures for stacked counter automata. In *STACS*, LIPIcs. Dagstuhl, 2015.

49 G. Zetzsche. The complexity of downward closure comparisons. In *ICALP*, volume 55 of *LIPIcs*, pages 123:1–123:14. Dagstuhl, 2016.