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-bs.de
3 **TU Braunschweig, Germany**
s.muskalla@tu-bs.de
4 **TU Braunschweig, Germany**
p.saivasan@tu-bs.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.

1998 ACM Subject Classification F.1.1 Models of Computation

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

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.49

1 Introduction

Petri nets are a popular model of concurrent systems [14]. 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) [31, 23, 25, 26], their regularity [39, 11], their context-freeness [38, 27], and many other decision problems (e.g. [17, 2, 15]). 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 L, denoted by $L{\downarrow}$, is the set of all subwords, all words that can be obtained from words in L by deleting letters. The upward closure of L, denoted by $L{\uparrow}$, is the set of all superwords, all words that can be obtained from words in

* The full version is available as technical report [7].
† A part of this work was carried out when the author was at Aalto University.

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 [33].

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: The languages contain Ackermann many words. The downward closure of Petri net languages has been shown to be effectively computable [17]. The algorithm is based on the Karp-Miller tree [22], 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 [11].

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. [21].

Finally, we consider *BPP*[1] *nets* [13], 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 (which is known to be NP-complete [37]). The hardness is by a reduction from SAT to the emptiness of BPP net languages, which in turn reduces to SRE inclusion.

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

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 [17]. 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 [40, 16, 10, 8]. 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 [6]. The effective computability of the downward closure has also been shown for stacked counter automata [43]. In [42], Zetzsche provides a characterization for a class of languages to have an effective downward closure. It has been used to prove the effective computability of downward closures of higher-order pushdown automata and higher-order recursion schemes [18, 9]. The downward closure of the languages of lossy channel systems is not computable [33].

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. [5, 4, 3, 24, 30, 44]). 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 $i, j \in \mathbb{N}$ with $i \leq j$, we use $[i..j]$ to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$ (resp. $[i..j[$ for $\{k \in \mathbb{N} \mid i \leq k < j\}$). Let Σ be a finite alphabet. We use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$. The length of a word u over Σ is denoted by $|u|$, where $|\varepsilon| = 0$. Let $k \in \mathbb{N}$ be a natural number, we use Σ^k (resp. $\Sigma^{\leq k}$) to denote the set of all words of length equal (resp. smaller or equal) to k. A language L over Σ is a (possibly infinite) set of finite words.

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

The *Parikh image* of a word [34] 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* $\preceq \subseteq \Sigma^* \times \Sigma^*$ [20] 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 L be a language over Σ. The *upward closure* of L consists of all words that have a subword in the language, $L{\uparrow} = \{v \in \Sigma^* \mid \exists u \in L : u \preceq v\}$. The *downward closure* of L contains all words that are dominated by a word in the language, $L{\downarrow} = \{u \in \Sigma^* \mid \exists v \in L : u \preceq v\}$. Higman showed that the subword relation is a well-quasi ordering [20], which means that every set of words $S \subseteq \Sigma^*$ has a finite *basis* — a finite set of *minimal elements* $v \in S$ such that $\nexists u \in S : u \neq v, u \preceq v$. With finite bases, $L{\uparrow}$ and $L{\downarrow}$ are guaranteed to be regular for every language $L \subseteq \Sigma^*$ [19]. 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, $sre ::= p \mid sre + sre$. Products interleave single letters a or $(a + \varepsilon)$ with iterations over letters from subsets $\Gamma \subseteq \Sigma$ of the alphabet: $p ::= a \mid (a + \varepsilon) \mid \Gamma^* \mid p.p$. The syntactic size of an SRE sre is denoted by $|sre|$ and defined as expected.

Finite State Automata. A *finite state automaton (FSA)* A is a tuple $(Q, \to, q_0, Q_f, \Sigma)$ where Q is a finite non-empty set of states, Σ is the finite input alphabet, $\to \subseteq Q \times \Sigma_\varepsilon \times Q$ is the non-deterministic transition relation, $q_0 \in Q$ 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 $q \xrightarrow{a}_A q'$ and generalize the relation to words in the expected way. The language of finite words accepted by A is denoted by $L(A)$. The size of A, denoted $|A|$, is defined by $|Q| + |\Sigma|$. An FSA is minimal for its language $L(A)$ if there is no FSA B with $L(A) = L(B)$ with a strictly smaller number of states.

Petri Nets. A *(labeled) Petri net* is a tuple $N = (\Sigma, P, T, F, \lambda)$ [36]. Here, Σ is a finite alphabet, P a finite set of *places*, T a finite set of *transitions*, $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ a *flow function*, and $\lambda : T \mapsto \Sigma_\varepsilon$ a labelling 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 $M(p) \geq F(p, t)$ 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 $F(p, t)$ many tokens and produces $F(t, p)$ many 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 $M \geq M_f$, if $M(p) \geq M_f(p)$ 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, $M_0[\sigma\rangle M \geq M_f$ for some $\sigma \in T^*$.

Given a Petri net N, an initial marking M_0, and a final marking M_f, the associated *covering language* is $L(N, M_0, M_f) = \{\lambda(\sigma) \mid \sigma \in T^*,\ M_0[\sigma\rangle M \geq M_f\}$, where the labeling function λ is extended to sequences of transitions in the straightforward manner. Given a natural number $k \in \mathbb{N}$, we define $L_k(N, M_0, M_f) = \left\{\lambda(\sigma) \;\middle|\; \sigma \in T^{\leq k},\ M_0[\sigma\rangle M \geq M_f\right\}$ 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 $|N| = |\Sigma| + \cdot |P| \cdot |T| \cdot (1 + \lceil log_2(1 + max(F)) \rceil)$, i.e. we assume that the flow function is encoded in binary. 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. 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.

A Petri net N is said to be a *BPP net* if every transition consumes at most one token from one place (i.e. $\Sigma_{p \in P} F(p, t) \leq 1$ 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 [30].

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 $N = (\Sigma, P, T, F, \lambda)$ and let M_0 and M_f be the initial and the final marking of interest. Define $n = |N| + |M_0| + |M_f|$.

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

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 $L_k(N, M_0, M_f)$ contains all minimal words of $L(N, M_0, M_f)$. Furthermore, $L_k(N, M_0, M_f) \subseteq L(N, M_0, M_f)$ and therefore the equality $L_k(N, M_0, M_f)\uparrow = L(N, M_0, M_f)\uparrow$ holds. Now we can use the following lemma to construct a finite automaton whose size is $O(2^{2^{\text{poly}(|n|)}})$ and that accepts $L_k(N, M_0, M_f)$. Without an increase in size, this automaton can be modified to accept $L_k(N, M_0, M_f)\uparrow$.

▶ **Lemma 2.** *Consider* N, M_0, *and* M_f. *For every* $k \in \mathbb{N}$, *we can construct an FSA of size* $O((k+2)^{\text{poly}(n)})$ *that accepts* $L_k(N, M_0, M_f)$, *where* $n = |N| + |M_0| + |M_f|$.

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* $M_0[\sigma\rangle M \geq M_f$, *there is* $M_0[\sigma'\rangle M' \geq M_f$ *with* $\lambda(\sigma') \preceq \lambda(\sigma)$ *and* $|\sigma'| \leq 2^{2^{cn \log n}}$, *where* c *is a constant.*

Our proof is an adaptation of Rackoff's technique to show that coverability can be solved in EXPSPACE [35]. 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 [12]. Lemma 5.3 in [28] claims that Rackoff's original proof already implies Proposition 3. This is not true as we provide a counterexample in the full version of this paper [7]. To handle labeled Petri nets, his 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 ℓ 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 *unrestricted marking* of N is a function $M : P \to \mathbb{Z}$. An unrestricted marking M *i-enables* a transition $t \in T$ if $M(j) \geq F(j,t)$ for all $j \in [1..i]$. Firing t yields a new unrestricted 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 $\pi = M_0[t_1\rangle_i M_1 \dots [t_m\rangle_i M_m$ is *i-bounded* with $i \in [1..\ell]$ if for each marking M_k with $k \in [0..m]$ and each place $j \in [1..i]$, we have $M_k(j) \geq 0$. We assume a fixed marking M_f to be covered. The computation π is *i-covering* (wrt. M_f) if $M_m(j) \geq M_f(j)$ 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 paths of all i-covering and i-bounded computations starting at M, i.e. $\mathit{Paths}(M,i) = \{\sigma \in T^* \mid \pi = M[\sigma\rangle_i M', \pi \text{ is } i\text{-bounded and } i\text{-covering}\}$. 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 $\mathit{Basis}(M,i)$,

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

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 SPath, i.e. $m(M,i)$ is well-defined. To remove the dependency on M, define $f(i) = \max\{m(M,i) \mid M \in \mathbb{Z}^\ell\}$ 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 $L(N, M_0, M_f)$. To derive the bound, we prove that $f(i+1) \leq (2^n f(i))^{i+1} + f(i)$ using Rackoff's famous case distinction [35].

▶ **Lemma 4.** *$f(0) = 1$ and $f(i+1) \leq (2^n f(i))^{i+1} + f(i)$ for all $i \in [0..\ell[$.*

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 [29] that shows how to calculate in a precise way (including zero tests) with values up to 2^{2^n} in Petri nets.

▶ **Lemma 5.** *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 $L(N(n), M_0, M_f) = \{a^{2^{2^n}}\}$.*

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. Fix the BPP net $N = (\Sigma, P, T, F, \lambda)$ and assume M_0 and M_f to be the initial and the final marking of interest. Let $n = |N| + |M_0| + |M_f|$.

▶ **Theorem 6.** *One can construct an FSA of size $O(2^{\mathrm{poly}(n)})$ for $L(N, M_0, M_f)\uparrow$.*

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 $O(2^{\mathrm{poly}(n)})$ that accepts $L_k(N, M_0, M_f)\uparrow$.

▶ **Proposition 7.** *Consider a BPP net N. For every computation $M_0[\sigma\rangle M \geq M_f$ there is $M_0[\sigma'\rangle M' \geq M_f$ with $\lambda(\sigma') \preceq \lambda(\sigma)$ and $|\sigma'| \leq tc(M_f)^2|T|$.*

The key to proving the lemma 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.

Lower Bound. We present a family of BPP net languages for which the minimal finite state automata representing the upward closure are exponential in the size of the input. The idea is to rely on the final marking, which is encoded in binary and hence can require 2^n tokens.

▶ **Lemma 8.** *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 $L(N(n), M_0, M_f) = \{a^{2^n}\}$.*

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.

Petri Nets. The downward closure of Petri net languages has been shown to be effectively computable in [17]. The algorithm is based on the Karp-Miller tree [22], 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 [32]. It gives a family of Petri nets whose computations all terminate but, upon halting, may have produced Ackermann many tokens on a distinguished place.

▶ **Lemma 9.** *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 $L(N(n), M_0^{(x)}, M_f) = \{a^k \mid k \leq Acker_n(x)\}$.*

Our lower bound is an immediate consequence of this lemma.

▶ **Theorem 10.** *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.

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. Fix the BPP net $N = (\Sigma, P, T, F, \lambda)$ and let M_0 and M_f be the initial and the final marking of interest. Let $n = |N| + |M_0| + |M_f|$.

▶ **Theorem 11.** *We can construct a finite automaton of size* $O(2^{\text{poly}(n)})$ *for* $L(N, M_0, M_f)\downarrow$.

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 12.** *Let* $M_0[\sigma\rangle M$ *such that for some place* $p \in P$*, we have* $M(p) > c$ *with* $c = tc(M_0)(|P| \cdot m)^{(|T|+1)}$ *Then for each* $j \in \mathbb{N}$*, there is* $M_0[\sigma_j\rangle M_j$ *such that (1)* $\sigma \preceq \sigma_j$*, (2)* $M \leq M_j$*, and (3)* $M_j(p) > j$*.*

The automaton for $L(N, M_0, M_f)\downarrow$ is the state space of N with token values beyond c set to ω. For every transition, we also have an ε-variant to obtain the downward closure. The language of this automaton is the downward closure of the language of the given BPP net.

Lower Bound. Consider the family of BPP net languages from the Lemma 8: $L(N(n), M_0, M_f) = \{a^{2^n}\}$, for all $n \in \mathbb{N}$. Its downward closure is $\{a^i \mid i \leq 2^n\}$. The minimal finite state automata recognising the downward closure have 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. Assumev 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 (N, M_0, M_f), an SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\downarrow$?

Petri Nets.

▶ **Theorem 13.** SRED *is* EXPSPACE*-complete for Petri nets.*

Hardness is due to the hardness of coverability [29]. Indeed, marking M_f is coverable from M_0 in N iff $L(N, M_0, M_f) \neq \emptyset$ iff $\{\varepsilon\} \subseteq L(N, M_0, M_f)\downarrow$. Note that $\{\varepsilon\} = L(\emptyset^*)$ and therefore is a simple regular language.

For the upper bound, we take inspiration from a recent result of Zetzsche [42]. 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 σ_n such that $M_0[\sigma_n\rangle M_{\sigma_n}$ with $M_{\sigma_n}(p) \geq n$ for all places $p \in X$. In [11], Demri has shown that this problem is EXPSPACE-complete.

▶ **Theorem 14** ([11]). 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 $L(sre) \subseteq L(N, M_0, M_f)\downarrow$ holds iff $L(p) \subseteq L(N, M_0, M_f)\downarrow$ holds for all products p in *sre*. Since the Petri net language is downward closed, we can further simplify the products by removing choices. Fix a total ordering on the alphabet Σ. Such an ordering can be represented by a word w_Σ. We define the *linearization operation* that takes a product and returns a regular expression:

$$lin(a + \varepsilon) = a \qquad lin(a) = a$$
$$lin(\Gamma^*) = (\pi_\Gamma(w_\Sigma))^* \qquad lin(p_1 p_2) = lin(p_1)lin(p_2)\ .$$

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 15.** *$L(sre) \subseteq L(N, M_0, M_f)\downarrow$ if and only if for all products p in sre we have $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$.*

We will reduce $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$ to SUPPN. To this end, we first understand $lin(p)$ as a Petri net $N_{lin(p)}$. We modify this Petri net by adding one place p_Γ for each block $(\pi_\Gamma(w_\Sigma))^* = a_i \dots a_j$. Each transition that repeats or leaves the block is modified to generate a token in p_Γ. As a result, p_Γ 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 16.** 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 $N_1 \rtimes N_2 = (\Sigma, P_1 \cup P_2, T_1 \cup T, F, \lambda)$, where for the transitions $t_1 \in T_1$, λ and F remain unchanged. The new transitions are $T = \{merge(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, \lambda_1(t_1) = \lambda_2(t_2)\}$ with $\lambda(merge(t_1, t_2)) = \lambda_1(t_1) = \lambda_2(t_2)$ and similarly $F(p_i, merge(t_1, t_2)) = F_i(p_i, t_i)$, $F(merge(t_1, t_2), p_i) = F_i(t_i, p_i)$ 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 $X = \{p_\Gamma \mid (\pi_\Gamma(w_\Sigma))^*\}$ is a block in p. The following correspondence holds.

▶ **Lemma 17.** *$L(lin(p)) \subseteq L(N, M_0, M_\emptyset)\downarrow$ 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.*

The lemma does not yet involve the final marking M_f. We modify N' and X such that simultaneous unboundedness implies $L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$. The idea is to introduce a new place p_f that can become unbounded only after M_f has been covered. We furthermore 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 σ contains neither t_f nor t_{pump}.

▶ **Lemma 18.** *$L(lin(p)) \subseteq L(N, M_0, M_f)\downarrow$ iff the places in $X \cup \{p_f\}$ are simultaneously unbounded in N'' from M_0''.*

To conclude the proof of Theorem 13, 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.

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 19.** SRED *for BPP nets is* NP*-complete.*

Hardness is by a reduction from SAT to BPP coverability, which in turn reduces to deciding whether the language of an SRE is included in the downward closure of a BPP language. For the reverse direction, 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. An *existential* Presburger formula takes the form $\exists x_1 \dots \exists x_n . \varphi$ where φ 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 $\vec{x} = x_1 \dots x_k$ occur free in φ. Given a function M from $\vec{x}$ to $\mathbb{N}$, the meaning of M *satisfies* φ is as usual and we write $M \models \varphi$ to denote this. We rely on the following complexity result:

▶ **Theorem 20** ([37])**.** *Satisfiability in existential Presburger arithmetic is* NP*-complete.*

Note that $L(sre) \subseteq L(N, M_0, M_f)\downarrow$ 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 $L(p) \subseteq L(N, M_0, M_f)\downarrow$ 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 21** ([41, 13])**.** *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 $\sigma \in T^*$.*

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

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

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

Moreover, we define $M \leq^c M'$, with c the constant defined in Lemma 12, if for all places $p \in P$ we have $M'(p) < c$ implies $M(p) \leq M'(p)$.

▶ **Definition 22.** Let p be a product. The BPP net N together with the markings M_0, M_f admits a *p-witness* if there are markings $M_1, M_1', \ldots, M_n, M_n'$ and computations σ_i, ρ_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 $i \in [1..n[$, and moreover: (1) $a_i \preceq \lambda(\sigma_i)$, for all $i \in [1..n]$, (2) $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$ and $M_i' \leq^c M_{i+1}$, for all $i \in [1..n-1]$, and (3) $M_1 = M_0$ and $M_f \leq^c M_n'$.

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. The second part of (2) means that each ρ_i (and thus $\pi_{\Sigma_i}(w_\Sigma)$) can be repeated. Property (3) 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 23.** *$L(p) \subseteq L(N, M_0, M_f)\downarrow$ holds iff (N, M_0, M_f) admits a p-witness.*

We now reduce the problem of finding such a p-witness to finding in another 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 σ_i resp. ρ_i. The Presburger formula $\Psi_{M_f}^{M_0}$ will check that the target marking reached with σ_i coincides with the initial marking for ρ_i, and the target marking reached with ρ_i is the initial marking of σ_{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 ρ^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.

▶ **Proposition 24.** *There are σ' 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.*

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 (N, M_0, M_f), SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\uparrow$?

▶ **Theorem 25.** SREU *is* EXPSPACE*-complete for Petri nets.*

The EXPSPACE lower bound is immediate by hardness of coverability for Petri nets [29]. 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. Since the number of minimal words in the SRE language is less than the size of the SRE, and since checking whether a word is included in the upward closure of the language of the Petri net N can be reduced in polynomial time to coverability in Petri nets (which is well-known to be in EXPSPACE [35]), we obtain our EXPSPACE upper bound.

▶ **Theorem 26.** SREU *is* NP*-complete for BPP nets.*

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. Consider a product p of the given SRE. The inclusion $L(p) \subseteq L(N, M_0, M_f)\uparrow$ holds iff the minimal subwords $\min(p) = a_1 \dots a_n$ belong to $L(N, M_0, M_f)\uparrow$. Word $a_1 \dots a_n$ belongs to the upward closure iff one of its subwords is in $L(N, M_0, M_f)$. We reduce this check for an accepted subword to checking whether a reachable marking M in a different net N' satisfies a Presburger formula Ψ.

7 Being Upward/Downward Closed

We now study the problem to decide 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 problem BUC is defined as follows:

Being upward closed (BUC)	
Given:	A Petri net (N, M_0, M_f).
Question:	$L(N, M_0, M_f) = L(N, M_0, M_f)\uparrow$?

The problem of being downward closed (BDC) replaces $\uparrow$ by $\downarrow$ in the above definition.

▶ **Theorem 27.** BUC *and* BDC *are decidable for Petri nets.*

Note that $L(N, M_0, M_f) \subseteq L(N, M_0, M_f)\uparrow$ trivially holds. It remains to decide the converse. First, we show how to decide $L(A) \subseteq L(N, M_0, M_f)$ for any given FSA A. This regular inclusion should be a problem of independent interest. Then, we can use the automaton for the upward closure constructed by Theorem 1 (resp. the automaton for the downward closure that can be constructed by [17]) to decide BUC (resp. BDC).

We rely on a result of Esparza et. al [21] 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 $\mathcal{T}(N, M_0) = \{w \in \Sigma^* \mid \exists\, \sigma \in T^* \colon \lambda(\sigma) = w, M_0[\sigma\rangle M \text{ for some marking } M\}$.

▶ **Theorem 28** ([21]). *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 $L(A) \subseteq L(N, M_0, M_f)$ to deciding an inclusion among trace languages. Theorem 28 can be used to decide this inclusion. Let (N, M_0, M_f) be the Petri net of interest together with its initial and final marking, and let A be the given FSA. As language $L(N, M_0, M_f)$ 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_{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 $L(A).a$ that is reduced so that the unique final state is reachable from every state.

▶ **Lemma 29.** *$L(A) \subseteq L(N, M_0, M_f)$ holds iff $\mathcal{T}(A.a) \subseteq \mathcal{T}(N.a, M_0)$ holds.*

The second inclusion is decidable using Theorem 28. This yields the desired result.

▶ **Theorem 30.** *$L(A) \subseteq L(N, M_0, M_f)$ is decidable.*

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. M. F. Atig, A. Bouajjani, K. Narayan Kumar, and P. Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS*, LIPIcs. Dagstuhl, 2014.
4. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *LMCS*, 7(4), 2011.
5. M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, LNCS. Springer, 2008.
6. 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.
7. M. F. Atig, R. Meyer, S. Muskalla, and P. Saivasan. On the upward/downward closures of petri nets. *CoRR*, abs/1701.02927, 2017. URL: `http://arxiv.org/abs/1701.02927`.
8. 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.
9. 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.
10. B. Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 1991.
11. S. Demri. On selective unboundedness of VASS. *JCSS*, 79(5), 2013.
12. 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.
13. J. Esparza. Petri Nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31(1), 1997.
14. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52, 1994.
15. A. Finkel, G. Geeraerts, J. F. Raskin, and L. V. Begin. On the omega-language expressive power of extended Petri nets. *ENTCS*, 2005.
16. H. Gruber, M. Holzer, and M. Kutrib. More on the size of Higman-Haines sets: Effective constructions. In *MCU*, LNCS. Springer, 2007.
17. P. Habermehl, R. Meyer, and H. Wimmel. The downward-closure of Petri net languages. In *ICALP*, LNCS. Springer, 2010.
18. M. Hague, J. Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *POPL*. ACM, 2016.

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

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

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

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

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

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

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

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

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

28 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.

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

30 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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