On the Upward/Downward Closures of Petri Nets

Mohamed Faouzi Atig
Uppsala University
mohamed_faouzi.atig@it.uu.se

Roland Meyer, Sebastian Muskalla, and Prakash Saivasan
TU Braunschweig
{roland.meyer,s.muskalla,p.saivasan}@tu-bs.de

arXiv:1701.02927v1 [cs.FL] 11 Jan 2017

***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.**

I. Introduction

Petri nets are a popular model of concurrent systems [17]. 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) [33], [26], [28], [29], their regularity [41], [13], their context-freeness [40], [30], and many other decision problems (e.g. [20], [2], [18]). 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 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 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. For example, it is not possible to compute the downward closure of languages recognized by lossy channel systems [35].

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 for the upward closure in doubly-exponential time. The size of the constructed FSA is also doubly exponential in the size of the input. 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 provide 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 in [20]. The algorithm is based on the Karp-Miller tree [25], which also has a 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 compute 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 of the Petri net language. In the case of downward closures, we reduce the problem to the simultaneous unboundedness problem for Petri nets, which is EXPSPACE-complete [13].

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 [24].

Finally, we consider *BPP*[1] *nets* [14], 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

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

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 [39]). 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:

Problem \ Net type	Petri net	BPP net
Upward closure	Doubly Exponential (Size & Time, Optimal)	Exponential (Size & Time, Optimal)
Downward closure	Non-primitive recursive (Size & Time, Optimal)	Exponential (Size & Time, Optimal)
SRE in upward closure	EXPSPACE-complete	NP-complete
SRE in downward closure	EXPSPACE-complete	NP-complete
Being upward/downward closed	Decidable	Decidable

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 [20]. 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 [42], [19], [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 [7]. The effective computability of the downward closure has also been shown for stacked counter automata [44]. In [43], 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 [21], [9]. The downward closure of the languages of lossy channel systems is not computable [35].

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], [27], [32], [45]). Throughout the paper, we will give hints to applications in verification.

II. Preliminaries

In this section, we fix some basic definitions and notations that will be used throughout the paper.

Let $\mathbb{N}$ denote the set of natural numbers. 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^* (resp. Σ^+) to denote the set of all finite words (resp. non-empty words) over Σ and ε to denote the empty word. We use Σ_ε to denote $\Sigma \cup \{\varepsilon\}$. Let u be a word over Σ. The length of u 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 [36] counts the number of occurrences of all letters while forgetting about their relative 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* $\preceq \subseteq \Sigma^* \times \Sigma^*$ [23] 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 [23], 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 $\nexists u \in S : u \preceq v$. With finite bases, $L\!\uparrow$ and $L\!\downarrow$ are guaranteed to be regular for every language $L \subseteq \Sigma^*$ [22]. 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, which in turn interleave single letters a or $(a + \varepsilon)$ with iterations over letters from subsets $\Gamma \subseteq \Sigma$ of the alphabet:

$$sre ::= p \mid sre + sre \qquad p ::= a \mid (a + \varepsilon) \mid \Gamma^* \mid p.p\ .$$

The syntactic size of an SRE sre is denoted by $|sre|$. The definition is as expected, every piece of syntax contributes to it.

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 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 $(q, a, q') \in \to$ 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 $L(A)$. Finally, the size of A, denoted $|A|$, is defined by $|Q| + |\Sigma|$.

III. Petri Nets

In the following, we recall the basic notions concerning Petri nets following standard textbooks [38]. A *(labeled) Petri net* is a tuple $N = (\Sigma, P, T, F, \lambda)$. 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 $\lambda : T \mapsto \Sigma \cup \{\varepsilon\}$ is a labelling function with ε being the empty word. When convenient, we will assume that the places are ordered, i.e. $P = [1..\ell]$ for some $\ell \in \mathbb{N}$. For any place or transition $x \in P \cup T$, we define the *preset* to consist of the elements that have an arc to the corresponding place or transition, ${}^{\bullet}x = \{y \in P \cup T \mid F(y, x) > 0\}$. 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* $M : P \to \mathbb{N}$, runtime configurations, 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 the flow function F. We define the size of the Petri net N to be

$$|N| = |\Sigma| + (2 \cdot |P| \cdot |T| \cdot (1 + \lceil log_2(1 + max(F))\rceil)) \ .$$

For a marking M, we define its size as

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

where $max(M)$ denotes the maximum of the range of M.

We define the *token count* $tc(M)$ of a marking M to be the sum of all tokens assigned by M, $tc(M) = \Sigma_{p \in P} M(p)$.

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

IV. 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 [32].

A. 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 a finite state automaton of size $O(2^{2^{\mathrm{poly}(n)}})$ 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 $O(2^{2^{\mathrm{poly}(n)}})$. 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 $O(2^{2^{\mathrm{poly}(\lceil n \rceil)}})$ 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 a finite state automaton of size $O((k+2)^{\mathrm{poly}(n)})$ 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 [37]. Rackoff derives a bound (similar to ours) on the length of the shortest computations that cover a given marking. In our setting where we deal with 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 then 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 $\pi = M_0[t_1\rangle_i M_1 \ldots [t_m\rangle_i M_m$ 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 $M_0[t_1\rangle_i \cdots [t_k\rangle_i M_k[t'_1\rangle_i M''_{k+1} \cdots [t'_s\rangle_i M''_{k+s}$. Note that M''_{k+1} coincides with M'_1 on the first i places but may have a different token count on the remaining places, and similar for the other new markings.

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 consider 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 associated with i-covering and i-bounded computations starting at M, i.e.

$$\mathit{Paths}(M,i) = \left\{ \sigma \in T^* \;\middle|\; \begin{array}{l} \pi = M[\sigma\rangle_i M', \\ \pi \text{ is } i\text{-bounded and } i\text{-covering} \end{array} \right\}.$$

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 the minimal elements in this set. The following is the central definition. It gives the set of shortest paths that yield (via their labelings) the minimal words in $\mathit{Basis}(M,i)$:

$$\begin{aligned} \mathit{SPath}(M,i) = \{\sigma \in \mathit{Paths}(M,i) \mid \lambda(\sigma) \in \mathit{Basis}(M,i) \text{ and} \\ \nexists \sigma' \in \mathit{Paths}(M,i) : |\sigma'| < |\sigma| \text{ and } \lambda(\sigma') = \lambda(\sigma)\}\,. \end{aligned}$$

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 from the definition, we comment on this in a moment. 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 $f(i+1) \leq (2^n f(i))^{i+1} + f(i)$. Together with $f(0) = 1$, this also shows that $f(i)$ is well-defined. The proof is by Rackoff's famous case distinction [37].

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 [31] 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 the language is*

$$L(N(n), M_0, M_f) = \left\{a^{2^{2^n}}\right\}\,.$$

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

B. 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 a finite state automaton of size $O(2^{\mathrm{poly}(n)})$ for $L(N, M_0, M_f)\uparrow$.*

We will show (Proposition 7) 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.

To make this proof sketch precise, we use (and adapt to our purposes) unfoldings, a true concurrency semantics for Petri nets [16]. 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 unlabelled 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: $\sum_{t' \in T'} F(t', p') \leq 1$ for every $p' \in P'$. Two elements $x, y \in P' \cup T'$ are *causally related*, $x \preceq y$, if there is a path from x to y. We use $\lfloor x \rfloor = \{y \in P' \cup T' \mid y \preceq x\}$ to denote the predecessors of $x \in P' \cup T'$. The $\preceq$-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 for all $p \in P$, $(h(t'^\bullet))(p) = F(h(t'), p)$.
(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 $t'_1 = t'_2$.

Here, $h(P'_1)$ with $P'_1 \subseteq P'$ is a function from P to $\mathbb{N}$ with $(h(P'_1))(p) = |\{p' \in P'_1 \mid h(p') = p\}|$. 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 above statement.

Proof of Proposition 7: Consider a computation $M_0[\sigma\rangle M$ with $M \geq M_f$ 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 τ in O with $h(\tau) = \sigma$ and $\mathit{Min}(O)[\tau\rangle P'_1$ with $h(P'_1) = M$. Since $M \geq M_f$, 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 τ induces a forest in O that consists of all places that contain a token after firing τ and their predecessors. We now construct a subcomputation by restricting τ 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 τ onto $\lfloor X \rfloor$. In τ_1, we mark all $\preceq$-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 $t'' \succeq t'$ with $t'' \in \lfloor x \rfloor \cap \lfloor y \rfloor$, 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 either is no join transition or a unique one leading to these two places, so t' and t'' have to lead to different places of X. Let $t't^1 \dots t^m t''$ 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 τ_1, all t^i also have to occur in τ_1. If there are indices $j < k$ such that $t^j = t^k$, we may delete $t^{j+1} \dots t^k$ from τ_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 τ_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 τ_2 is bounded by $tc(M_f)^2 \cdot |T|$. ■

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 key 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 the language is*

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

Proof: The BPP net $N(n)$ consists of places p_1 to p_n connected by a series of transitions that double the tokens from place to place. As a result, on the last place 2^n many tokens can be produced. These tokens will be used to create 2^n letters a, and the occurrence of the letters is checked by the final marking.

Formally, the places are $P = \{p_i \mid i \in [1..n]\} \cup \{p_{\mathit{final}}\}$. The transitions are $T = \{t_i \mid i \in [1..n-1]\} \cup \{t_{\mathit{final}}\}$ with $F(p_i, t_i) = 1$, $F(t_i, p_{i+1}) = 2$ for all $i \in [1..n-1]$, $F(p_n, t_{\mathit{final}}) = 1$, $F(t_{\mathit{final}}, p_{\mathit{final}}) = 1$, and F is 0 for all other arguments. The label of all t_i is ε, the label of t_{final} is a. The initial marking places one token on p_1 and no tokens elsewhere, the final marking requires 2^n tokens on p_{final} and no tokens elsewhere. Note that 2^n in binary is polynomial. ■

V. 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. Different from the upward closure, 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. Indeed, 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.

A. Petri Nets

The downward closure of Petri net languages has been shown to be effectively computable in [20]. The algorithm is based on the Karp-Miller tree [25], 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 are of non-primitive recursive size in the size of the input. Our result relies on a construction due to Mayr and Meyer [34]. 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 numbers $n, x \in \mathbb{N}$, we can construct 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*

$$L(N(n), M_0^{(x)}, M_f) = \{a^k \mid k \leq \mathit{Acker}_n(x)\}.$$

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.

B. 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 $O(2^{\mathrm{poly}(n)})$ 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 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 idea is that if more than c tokens are created in some place p during a firing sequence σ, then there has to be a subsequence $\sigma' \preceq \sigma$ so that (1) the first and the last transition of σ' coincide, (2) each transition in σ' (but the first one) consumes the token produced by its predecessor, and (3) at least one of the transitions produces a token on place p that is not consumed by any of the other transitions in σ'. In this case, σ' can be repeatedly inserted after its first occurrence to create arbitrarily many tokens in p. As in Proposition 7, this subsequence is identified via join transitions in the unfolding. The bound c is chosen such that there exists a branch (Condition (2)) of the computation in the unfolding, in which there have to be two occurrences of the same transition (Condition (1)), between which another branch spawns that creates a token in p that is not consumed later (Condition (3)).

Lemma 12 allows us to set place p to ω, meaning that it may receive arbitrarily many tokens. We let $\omega > k$ for every $k \in \mathbb{N}$ and define $\oplus$ and $\ominus$ as variants of $+$ and $-$ that treat ω as infinity: $x \oplus y = x + y$ if $x + y < c$, $x \oplus y = \omega$ otherwise. Similarly, $x \ominus y = x - y$ if $x \neq \omega$, and $x \ominus y = \omega$ otherwise.

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. Formally, the functions that assign to each place a number of tokens up to c or ω form the set of states, $S = P \to ([0..c] \cup \{\omega\})$. For each transition t labeled by $x \in \Sigma_\varepsilon$ and from all states q where t is enabled in the sense that $q(p) \geq F(p,t)$ for all $p \in P$, we add the transitions $q \xrightarrow{x} q'$ and $q \xrightarrow{\varepsilon} q'$. Here, $q'(p) = q(p) \ominus F(p,t) \oplus F(t,p)$ for all $p \in P$.

The initial state of the automaton is given by q_0, where $q_0(p) = M_0(p)$ for all $p \in P$. The set of final states is given by $F = \{q_f \mid \forall p \in P, q_f(p) \geq M_f(p)\}$. It is easy to see that 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 Lemma 8: $L(N(n), M_0, M_f) = \{a^{2^n}\}$, for all $n \in \mathbb{N}$. The downward closure is $L(N(n), M_0, M_f)\!\downarrow = \{a^i \mid i \leq 2^n\}$. The minimal finite state automata recognising the downward closure have at least 2^n states.

VI. 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 (N, M_0, M_f) and an SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\!\downarrow$?

A. Petri Nets

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

Hardness is due to the hardness of coverability [31]. 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 [43]. 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 the unboundedness problem which 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 $M_0[\sigma_n\rangle M_\sigma$ with $M_{\sigma_n}(p) \geq n$ for all places $p \in X$. In [13], Demri has shown that this problem is EXPSPACE-complete.

Theorem 14 ([13])**.** 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:

$$\begin{aligned} &\mathit{lin}(a+\varepsilon) = a && \mathit{lin}(a) = a \\ &\mathit{lin}(\Gamma^*) = (\pi_\Gamma(w_\Sigma))^* && \mathit{lin}(p_1p_2) = \mathit{lin}(p_1)\mathit{lin}(p_2)\ . \end{aligned}$$

For example, if $\Sigma = \{a, b, c\}$ and we take $w_\Sigma = abc$, then $p = (a+c)^*(a+\varepsilon)(b+c)^*$ is turned into $\mathit{lin}(p) = (ac)^*a(bc)^*$. 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 $L(\mathit{lin}(p)) \subseteq L(N, M_0, M_f)\!\downarrow$.*

We will reduce $L(\mathit{lin}(p)) \subseteq L(N, M_0, M_f)\!\downarrow$ to SUPPN. To this end, we first understand $\mathit{lin}(p)$ as a Petri net $N_{\mathit{lin}(p)}$. The Petri net is essentially the finite automaton for the language with states turned into places and transitions of the automaton turned into transitions of the Petri net. 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 (the ones labeled by a_j) 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 $N_{\mathit{lin}(p)}$ 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 $N_1 \triangleright N_2$ *is the labeled Petri net $N_1 \triangleright N_2 = (\Sigma, P_1 \uplus P_2, T_1 \uplus T, F, \lambda)$, where for the transitions $t_1 \in T_1$, λ and F remain unchanged. The new transitions are*

$$T = \{\mathit{merge}(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, \lambda_1(t_1) = \lambda_2(t_2)\}$$

with $\lambda(\mathit{merge}(t_1, t_2)) = \lambda_1(t_1) = \lambda_2(t_2)$ and similarly

$$\begin{aligned} F(p_i, \mathit{merge}(t_1, t_2)) &= F_i(p_i, t_i), \\ F(\mathit{merge}(t_1, t_2), p_i) &= F_i(t_i, p_i) \end{aligned}$$

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 $N' = N \triangleright N_{\mathit{lin}(p)}$, take the initial marking M_0' that coincides with M_0 but puts a token on the initial place of $N_{\mathit{lin}(p)}$, 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 17. $L(\mathit{lin}(p)) \subseteq L(N, M_0, M_\emptyset)\!\downarrow$ *if and only if the places in X are simultaneously unbounded in N' from M_0'.*

The lemma does not yet involve the final marking M_f. We modify N' and X such that simultaneous unboundedness even implies $L(\mathit{lin}(p)) \subseteq L(N, M_0, M_f)\!\downarrow$. The idea is to introduce a new place that can become unbounded only after M_f has been covered. We add a place p_f and 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) are polynomial in the size of the original input.

B. 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 in 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. Let $\mathcal{V}$ be a set of variables with elements x, y. The set of terms in Presburger arithmetic is defined as follows:

$$t ::= 0 \mid 1 \mid x \mid t - t \mid t + t\ .$$

The set of formulas is defined by

$$\varphi ::= t \leq t \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x. \varphi\ .$$

An *existential* Presburger formula takes the form $\exists x_1 \ldots \exists x_n.\varphi$ 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 $\vec{x} = x_1 \ldots x_k$ 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 ([39]). *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 ([14]). *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

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

where $\Sigma_1, \ldots, \Sigma_{n-1} \subseteq \Sigma$ and $a_1, \ldots, a_n \in \Sigma$. For this language to be included in $L(N, M_0, M_f)\downarrow$, 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 $i \in [1..n-1]$.
(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 $\pi_{\Sigma_i}(w_\Sigma)$ occur 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 if and only if (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 $N' = (\{\varepsilon\}, P', T', F', \lambda')$ a reachable marking that satisfies a Presburger formula $\Psi(P')$. 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 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.

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 σ_i or ρ_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 Σ_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 $i \in [1..n[$, we set $L_{2i} = \{l^a_{2i} \mid a \in \Sigma_i\}$ otherwise. The transitions are $T' = \bigcup_{i\in[1..2n-1]} \mathrm{TC}_i \cup \mathrm{TE}_i$. The $\mathrm{TC}_i = \{\mathrm{tc}^p_i \mid p \in P\}$ populate E_i and B_i. The transitions in $\mathrm{TE}_i = \{\mathrm{te}^t_i \mid t \in T\}$ together with E_i form a replica of the BPP net. The flow relation F' is defined as follows (numbers omitted are zero):

(1) For all $i \in [1..2n-1]$, $p \in P$, $F'(\mathrm{tc}^p_i, e^p_i) = F'(\mathrm{tc}^p_i, b^p_i) = 1$.
(2) For all $i \in [1..2n-1]$, $p \in P$, $t \in T$, $F'(\mathrm{te}^t_i, e^p_i) = F(t,p)$ and $F'(e^p_i, \mathrm{te}^t_i) = F(p,t)$.
(3) For all $i \in [1..n]$, $t \in T$ with $\lambda(t) = a_i$, $F'(\mathrm{te}^t_{2i-1}, l_{2i-1}) = 1$.
(4) For all $i \in [1..n[$, $t \in T$ with $\lambda(t) = a \in \Sigma_i$, $F'(\mathrm{te}^t_{2i}, l^a_{2i}) = 1$.

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

$$\begin{aligned}\Psi^I_F(P') =& \Psi_1(P') \wedge \Psi_2(P') \wedge \Psi_3(P')\\ &\wedge \Psi_4(P') \wedge \Psi^I_5(P') \wedge \Psi^F_6(P')\ ,\end{aligned}$$

where

$$\begin{aligned}
\Psi_1(P') &= \bigwedge_{i\in[1..2n-2]} \bigwedge_{p\in P} e^p_i = b^p_{i+1}\\
\Psi_2(P') &= \bigwedge_{i\in[1..n[} \bigwedge_{p\in P} (e^p_{2i} < c \ \rightarrow\ b^p_{2i} \le e^p_{2i})\\
\Psi_3(P') &= \bigwedge_{i\in[1..n]} l_{2i-1} > 0\\
\Psi_4(P') &= \bigwedge_{i\in[1..n[} \bigwedge_{a\in\Sigma_i} l^a_{2i} > 0\\
\Psi^I_5(P') &= \bigwedge_{p\in P} b^p_1 = I(p)\\
\Psi^F_6(P') &= \bigwedge_{p\in P} (e^p_{2n-1} < c \ \rightarrow\ e^p_{2n-1} \ge F(p))\ .
\end{aligned}$$

Formula Ψ_1 states that σ_i ends in the marking M'_i that ρ_i started from, and similarly ρ_i ends in M_{i+1} that σ_{i+1} started from. Formula Ψ_2 states the required $\le^c$ relation. To make sure we found letter a_i, we use Ψ_3. With Ψ_4, we express that all letters from Σ_i have been seen. Conjunct Ψ^I_5 says that the places b^p_1 have been initialized to the value given by the initial marking I. Formula Ψ^F_6 states the condition on covering the final marking. The correctness of the construction is the next lemma. Note that the transitions TC_i are always enabled. Therefore, we can start in N' from the initial marking $M_\emptyset$ that assigns zero to every place.

Lemma 24. *There are σ' and M' so that $M_\emptyset[\sigma'\rangle M'$ in N' and $M' \models \Psi^{M_0}_{M_f}$ if and only if (N, M_0, M_f) admits a p-witness.*

VII. SRE Inclusion in Upward Closure

Like in the previous section, 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) and an SRE sre.
Question:	$L(sre) \subseteq L(N, M_0, M_f)\!\uparrow$?

A. Petri Nets

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

The EXPSPACE lower bound is immediate by hardness of coverability for Petri nets [31]. 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 [37]), we obtain our EXPSPACE upper bound.

B. BPP Nets

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. The construction of N' and ψ is as follows.

As in the case of downward closures, 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^p_i of the BPP net places p. Additionally for each i we have a place $L_i = \{l_i\}$. The transitions TC_i again populate the copy of the BPP net and store the same marking in B_i. The transitions TE_i contain a copy te^t_i of each BPP net transition t. To check for a subword of $a_1 \dots a_n$, in each stage i we only enable transitions t that are either labeled by ε or a_i, i.e. for all $p \in P$, $t \in T$, if $\lambda(t) = \varepsilon$ or $\lambda(t) = a_i$ we have $F'(\mathrm{te}^t_i, e^p_i) = F(t,p)$ and $F'(e^p_i, \mathrm{te}^t_i) = F(p,t)$. 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 $F'(\mathrm{te}^t_i, l_i) = 1$. Now the required Presburger formula Ψ — 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, $\bigwedge_{i\in 1..n} l_i \le 1$. This guarantees we have seen a subword of $a_1 \dots a_n$. The initial marking $M_\emptyset$ is one that assigns zero to all places. It is easy to see that $L(p) \subseteq L(N, M_0, M_f)\uparrow$ iff there is a computation $M_\emptyset[\sigma\rangle M$ in N' such that $M \models \Psi$.

VIII. Being Upward/Downward Closed

We now study the problem to decide whether a Petri net language actually is upward or downward closed, respectively. This is interesting as it means that an automaton for the closure, which we can compute with the previously introduced methods, is actually a precise representation of the system's behavior. Formally, 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. Then, we can use the automaton for the upward closure constructed given by Theorem 1 (resp. the automaton for the downward closure that can be constructed by [20]) to decide BUC (resp. BDC).

The regular inclusion problem, i.e. given a regular language, is it included in the coverability language of a Petri net, is a problem of independent interest. We will show its decidability in the following. To this end, we rely on a result of Esparza et. al [24]. To state it, we define the *traces* of a finite automaton (resp. Petri net) as the computations that start from the initial state (resp. initial marking), regardless of whether they end in a final state (resp. in a marking that covers the final marking). Formally, the set of traces are the words that occur as the labelings of such computations. For a finite automaton $A = (Q, \rightarrow, q_{init}, Q_f, \Sigma)$, we define

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

Similarly, the *traces* of a Petri net are all words that occur as labelings of valid firing sequences from the initial marking:

$$\mathcal{T}(N, M_0) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} \sigma \in T^*, \lambda(\sigma) = w, M_0[\sigma\rangle M \\ \text{for some marking } M \end{array} \right\}.$$

Using this notation, the result from [24] shows that it is decidable whether the traces of a given FSA A are included in the traces of a given Petri net N with initial marking M_0.

Theorem 28 ([24])**.** *The inclusion $\mathcal{T}(A) \subseteq \mathcal{T}(N, M_0)$ is decidable.*

The result is surprising, given that the complements of covering languages are not known to belong to any class that affords closure under regular intersection and effective emptiness checking. Instead, 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. At the same time, this dependence on wqos does not allow us to derive a complexity result.

We split the proof of the decidability of regular inclusion into two parts. Firstly, we show how to reduce checking the inclusion $L(A) \subseteq L(N, M_0, M_f)$ to deciding an inclusion among *prefix-closed* languages. Secondly, we explain how to apply Theorem 28 to decide this inclusion. Here, we use $PfC(\mathcal{L})$ to denote the *prefix closure* of a language $\mathcal{L}$, the set of all words that can be prolonged to a word in the language:

$$PfC(\mathcal{L}) = \{ w \in \Sigma \mid \exists v \in \Sigma^* \colon wv \in \mathcal{L} \}.$$

From now on, 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. Note that the language $L(N, M_0, M_f)$ is not prefix-closed in general. To obtain a prefix-closed language, we consider the zero marking $M_\emptyset$ (with $M_\emptyset(p) = 0$ for all p) as the new final marking. This yields a prefix-closed language, 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 and define $\Sigma' = \Sigma \cup \{a\}$. 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. Formally, $N.a = (\Sigma', P, T', F', \lambda')$ with $T' = T \cup \{t_{\mathit{final}}\}$, $\lambda'(t_{\mathit{final}}) = a$, $F'(p, t_{\mathit{final}}) = M_f(p)$, $F'(t_{\mathit{final}}, p) = 0$ for all $p \in P$. On the existing transitions, λ' and F' behave as λ resp. F do. With this definition, the reduction of upward closedness to checking an inclusion among prefix-closed languages is this.

Lemma 29. *$L(A) \subseteq L(N, M_0, M_f)$ holds if and only if $PfC(L(A).a) \subseteq PfC(L(N.a, M_0, M_\emptyset))$.*

Proof: Assume the first inclusion holds and consider a word v from $PfC(L(A).a)$. We have to show membership of v in $PfC(L(N.a, M_0, M_\emptyset))$. By definition, v is a prefix of some word in the language $L(A).a$, say wa, where w stems from $L(A)$. The assumed first inclusion now yields $w \in L(N, M_0, M_f)$. From this, we can conclude $wa \in L(N.a, M_0, M_\emptyset)$ as follows. Since $w \in L(N, M_0, M_f)$, there is a computation $M_0[\sigma\rangle M$ of N with $M \geq M_f$ for some $\sigma \in T^*$ with $\lambda(\sigma) = w$. We now append t_{final} to σ, which means we consider the computation $M_0[\sigma\rangle M[t_{\mathit{final}}\rangle M'$ of $N.a$. We can indeed fire t_{final} since $M \geq M_f$ and t_{final} consumes $M_f(p)$ many tokens from every place p. We have $\lambda'(\sigma.t_{\mathit{final}}) = wa$, which proves $wa \in L(N.a, M_0, M_\emptyset)$ and thus $v \in PfC(L(N.a, M_0, M_\emptyset))$.

Assume the second inclusion holds and consider a word w from $L(A)$. The task is to prove membership of w in $L(N, M_0, M_f)$. To do so, note that $wa \in L(A).a$ and thus also $wa \in PfC(L(A).a)$. By the second inclusion, we have $wa \in PfC(L(N.a, M_0, M_\emptyset))$. This means wa is a prefix of some word, say wav, in $L(N.a, M_0, M_\emptyset)$. We consider a corresponding firing sequence $\sigma.t_{\mathit{final}}.\sigma'$ of $N.a$ with $\lambda'(\sigma) = w$ and $\lambda'(\sigma') = v$. Since $w \in L(N, M_0, M_f)\!\uparrow \subseteq \Sigma^*$ does not contain a, σ is a firing sequence of the original net N. Furthermore, it was possible to fire t_{final} in $N.a$ after σ, meaning that the marking M reached by $M_0[\sigma\rangle M$ covers M_f. This proves $w \in L(N, M_0, M_f)$. ■

The second inclusion from Lemma 29 is decidable using Theorem 28.

Lemma 30. *The inclusion*

$$\mathit{PfC}(L(A).a) \subseteq \mathit{PfC}(L(N.a, M_0, M_\emptyset))$$

is decidable.

Proof: To apply Theorem 28, we construct an FSA A' with $\mathcal{T}(A') = \mathit{PfC}(L(A).a)$ and a Petri net N' with initial marking M_0' so that $\mathcal{T}(N', M_0') = \mathit{PfC}(L(N.a, M_0, M_\emptyset))$.

The second part is easy, Petri net $N.a$ with the initial marking of N is already as desired. Indeed, since the zero marking is to be covered, every valid firing sequence yields a word in $L(N.a, M_0, M_\emptyset)$ and vice versa, $\mathcal{T}(N', M_0') = L(N.a, M_0, M_\emptyset)$. Furthermore, every prefix of a valid firing sequence is again a valid firing sequence, so $\mathcal{T}(N', M_0')$ is prefix-closed and thus $\mathit{PfC}(L(N.a, M_0, M_\emptyset)) = L(N.a, M_0, M_\emptyset) = \mathcal{T}(N', M_0')$.

It may be that $L(A) = \emptyset$, which is, given the automaton, easy to check. If $L(A)$ is empty, we also have $L(A).a = \emptyset$ and the inclusion trivially holds. In the following, we assume $L(A) \neq \emptyset$.

We modify A to an automaton accepting $L(A).a$ as follows. We add a new state q_{final}, an a-labeled transition from every final state of A to q_{final}, and then make q_{final} the unique final state. We refer to the result as $A.a$.

To obtain A', we remove from $A.a$ all states from which the unique final state cannot be reached, as well as all transition from and to such states. Identifying these states is easy: Saturate the set of states from which q_{final} can be reached, and then take the complement. This will not lead to an automaton without states, because $L(A) \neq \emptyset$. Furthermore, we have $L(A') = L(A.a) = L(A).a$, since every accepting run in $A.a$ is still an accepting run in A'.

We claim that $\mathcal{T}(A') = \mathit{PfC}(L(A).a)$ and so the automaton is as desired. The inclusion from right to left is immediate. Every prefix $w \in \mathit{PfC}(L(A).a)$ corresponds to the prefix of an accepting run of A', to a trace. For the other inclusion, note that every trace of A' can be prolonged to obtain an accepting run, since the final state is reachable from every state of A'. Formally, for any $w \in \mathcal{T}(A')$ there is a $v \in \Sigma'^*$ such that $wv \in L(A') = L(A).a$. Since w is a prefix of wv, we have $w \in \mathit{PfC}(L(A).a)$. ■

Combining the Lemmas 29 and 30 yields the desired result that can be used to prove Theorem 27.

Theorem 31. $L(A) \subseteq L(N, M_0, M_f)$ *is decidable.*

IX. 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 L_1 and L_2 and asks whethere there is a PTL S, called the separator, that includes L_1 and has an empty intersection with L_2. Taking a verification perspective, the separator is an over-approximation of the system behavior L_1 that is safe wrt. the bad behaviors in L_2. 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 [11]. Recently [12], Czerwiński, Martens, van Rooijen, Zeitoun, and Zetzsche have show that, for full trios, computing downward closures and deciding PTL-separability are 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 [20]. 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] 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] 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.

[12] W. Czerwiński, W. Martens, L. van Rooijen, M. Zeitoun, and G. Zetzsche. A characterization for decidable separability by piecewise testable languages, 2017. To appear in Discrete Mathematics and Theoretical Computer Science.

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

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

[15] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[32] Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, LNCS. Springer, 2012.

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

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

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

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

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

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

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

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

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

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

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

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

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

Appendix A
Proofs for Section IV.

A. Petri Net Languages

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

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

The set of states is $Q = (P \to [0..(k+1) \cdot 2^n]) \times |[0..k]$. 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 $M(p) \geq F(p,t)$ for all p and $i < k$, there is a transition from (M, i) to $(M', i+1)$ in $\to$, 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 $(M_0, 0) \xrightarrow{w}_A (M', j)$ with (M', j) final iff there is a $\sigma \in T^{\leq k}$ such that $M_0[\sigma\rangle M'$ with $M' \geq M_f$. Hence we have $L(A) = L_k(N, M_0, M_f)$.

We estimate the size of Q by

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

Since $|\Sigma| \leq n$, we have that $|A| = |Q| + |\Sigma|$ is in $\mathcal{O}\big((k+2)^{\mathrm{poly}(n)}\big)$. ■

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

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 ε.

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 $|\sigma'| \geq (2^n f(i))^{i+1} + f(i)$. We distinguish two cases.

1st Case: Suppose σ' is the sequence of transitions inducing an $(i+1)$-bounded, $(i+1)$-covering computation π', 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 π' an $(i+1)$-bounded, $(i+1)$-covering computation π where no two markings agree on the first $(i+1)$ places: Whenever such a repetition occurs in π', 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 σ. The computation σ satisfies

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

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

2nd Case: Otherwise, σ' is the path of an $(i+1)$-bounded, $(i+1)$-covering computation π', 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 π' as follows:

$$\pi' = M[\sigma'_1\rangle_{i+1} M_1 [\sigma'_2\rangle_{i+1} M'$$

so that M_1 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 σ'_1 by a repetition-free sequence σ_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 $\pi'_2 = M_1[\sigma'_2\rangle_{i+1} M'$ is also an i-bounded, i-covering computation. By the definition of $f(i)$, there is an i-bounded, i-covering computation π_2 starting from M_1 such that the corresponding path σ_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 π_2 is also an $(i+1)$-bounded, $(i+1)$-covering computation starting in M_1: 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 $\sigma'_1 \cdot_i \sigma'_2$ is then an $(i+1)$-bounded, $(i+1)$-covering run starting in M of length at most $((2^n f(i))^{i+1} - 1) + 1 + (f(i) - 1) < (2^n f(i))^{i+1} + f(i)$. ■

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.*

Proof: As in [37], 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 4, we can conclude $f(i) \leq g(i)$ for all $i \in [0..\ell]$. Furthermore,

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

for some suitable constant c.

Let $M_0[\sigma\rangle M \geq M_f$ 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 $|\sigma'| < m(M_0, \ell) \leq f(\ell) \leq 2^{2^{cn \log n}}$. ■

Lemma 5. *For any $n \in N$, 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 the language is $L(N(n), M_0, M_f) = \{a^{2^{2^n}}\}$.*

Proof: We rely on Lipton's proof [31] 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 [15].

Lipton defines *net programs* (called *parallel programs* in [31]) 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 [31], [15].

We will use the following commands in the net program.

$l : x := x - 1$	decrement a variable x
$l : \texttt{gosub}\ s$	call the subroutine s

Furthermore, we will use the following established subroutines from [15].

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

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. ■

Appendix B
Proofs for Section V.

A. Petri Net Languages

Definition 32 (Ackermann function). *We define the Ackermann function inductively as follows:*

$$\begin{aligned} Acker_0 \quad (x) &= x + 1 \ , \\ Acker_{n+1}(0) &= Acker_n(1) \ , \\ Acker_{n+1}(x+1) &= Acker_n(Acker_{n+1}(x)) \ . \end{aligned}$$

Definition 33. *We define the Petri net AN_0 to be*

$$AN_0 = (\{a\}, P^0, T^0, F^0, \lambda^0)$$

with

$$\begin{aligned} P^0 &= \{in^0, out^0, strt^0, stp^0, cp^0\}, \\ T^0 &= \{tstrt^0, tstp^0, tcp^0\}, \\ \lambda^0(t) &= \varepsilon \text{ for all } t \in T^0 \ . \end{aligned}$$

The flow relation is given as in Figure 1, where each edge carries a weight of 1.

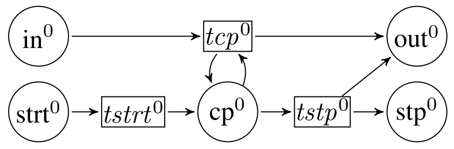

Fig. 1. The flow relation of the Petri net AN_0

For $n \in \mathbb{N}$, we define AN_{n+1} inductively by

$$AN_{n+1} = (\{a\}, P^{n+1}, T^{n+1}, F^{n+1}, \lambda^{n+1})$$

with

$$\begin{aligned} P^{n+1} &= P^n \cup \{in^{n+1}, strt^{n+1}, cp^{n+1}, out^{n+1}, stp^{n+1}, \\ &\qquad swp^{n+1}, tmp^{n+1}, \} \\ T^{n+1} &= T^n \cup \{tstrt^{n+1}, tcp^{n+1}, tstp^{n+1}, \\ &\qquad trestart^{n+1}, tin^{n+1}, tswp^{n+1}, ttmp^{n+1}\}, \\ \lambda^{n+1}(t) &= \varepsilon \text{ for all } t \in T^{n+1}. \end{aligned}$$

The flow relation is given as in Figure 2, where again each edge carries a weight of 1.

We prove that AN_n indeed can weakly compute $Acker_n(m)$.

Lemma 34. *For all $n, x \in \mathbb{N}$, let M_0 be the marking for AN_n that places one token on $strt^n$, x tokens on in^n and no token elsewhere. (1) There is a computation $M_0[\sigma\rangle M$ of AN_n such that $M(out^n) = Acker_n(x)$, $M(stp^n) = 1$. (2) There is no computation starting in M_0 that creates more than $Acker_n(x)$ tokens on out^n.*

Proof: We proceed 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 $tstrt^0$. Firing it leads to one token on the copy place cp^0. Now we can fire the copy transition tcp^0 x times, leading to x tokens on out^0. Finally, we fire the stopping transition $tstp^0$, which leads to one token on stp^0 and in total $x + 1$ tokens on out^0. This is the computation maximizing the number of tokens on out. Firing tcp^0 less than x times or not firing $tstp^0$ leads to less tokens on out^0.
Step n to $n+1$: Initially, we can only fire the starting transition $tstrt^{n+1}$, creating one token on in^n and one token on strt^n. We can now execute the computation of AN_n that creates $Acker_n(1)$ tokens on out^n and one token on stp^n, which exists

by induction. We consume one token from in^{n+1} and the token on stp^n to create a token on swp^{n+1} using tin^{n+1}. This token allows us to swap all $Acker_n(1)$ tokens from out^n to in^n using $tswp^{n+1}$. After doing this, we move the token from swp^{n+1} to $tstrt^n$ using the restart transition $trestart^{n+1}$. 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 $tswp$ as often as possible, say we leave y' out of y tokens on out^n, we end up with $Acker_n(y - y') + y' \leq Acker_n(y)$ tokens on out^n, since

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

for all $k, k' \in \mathbb{N}$. Finally, we move the token on $tstp^{n+1}$ to tmp^{n+1} using $ttmp^{n+1}$, and then move all $Acker_n(x)$ tokens to out^{n+1} using tcp^{n+1}. We end the computation by firing $tstp^{n+1}$ to move the token on tmp^{n+1} to stp^{n+1}. We now have one token on stp^{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 tcp^{n+1} less than $Acker_n(x)$ times will lead to less tokens on out^{n+1}. ■

Lemma 9. *For all $n, x \in \mathbb{N}$, we can construct 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 $L(N(n), M_0^{(x)}, M_f) = \{a^k \mid k \leq Acker_n(x)\}$.*

Proof: 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 $tfinal$ that is labeled by a and moves one token from out^n to final. The initial marking $M_0^{(x)}$ places x tokens on in^n, one token on strt^n and no tokens elsewhere. The final marking M_f is zero on all places.

By Lemma 34, 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 as in the process. This proves $L(N(n), M_0^{(x)}, M_f) = \{a^k \mid k \leq Acker_n(x)\}$.

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 and even logarithmic in x. ■

B. BPP Net Languages

Lemma 12. *Let $M_0[\sigma\rangle M$ where $M(p) > c$ with*

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

for some $p \in P$. Here, m is the maximal multiplicity of an edge. 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) $M_j(p) > j$.

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

Executing σ' 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 $r' \in Min(O)$ 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 tree $\mathcal{T}_p$ 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 the ancestor of a p-leaf. This tree has the following properties:

(1) Its leaves are exactly the p-leaves in $\mathcal{T}$.
(2) 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 $\mathcal{T}_p$.
(3) Each transition has out-degree at most $m \cdot \ell$. (In the worst case, each transition creates m tokens in each of the ℓ 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-leave. Our goal is to show that there is a branch of $\mathcal{T}_p$ in which at least $|T| + 1$ many join-transitions occur.

We prove the following:

Subclaim: 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.

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 of this tree is k^h, but since $h < \log_k x$, this is less than x: $k^h < k^{\log_k x} = x$. This finishes the proof of the Subclaim.

Instantiating the Subclaim for $x = c_1$ and $k = m \cdot |M_0|$, we obtain 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 τ and τ' in the same branch with $h(\tau) = h(\tau') = t$ for some transition t of the original BPP net.

Since τ was a join-transition, it has at least two child branches b_1 and b_2 that lead to a p-leaf. We assume now that b_1 is the branch on which τ' occurs, and b_2 is another branch.

We consider the sequence of transitions σ_p^{pump} that occur on b_1 when going from τ to τ' (including τ, not including τ'). We also consider the sequence of transitions σ_p^{gen} that occur on b_2 when going from τ to a p-leave (not including τ). Let σ^{pump} and σ^{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 σ 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 τ in $\mathcal{T}_p$. We extend the run to

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

Obviously, σ is a subsequence of σ_j, and therefore satisfies the required Property (1). We have to argue why σ_j is a valid firing sequence, why firing σ_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: σ^{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 σ.

Note that up to σ_1, σ and σ_j coincide. Since t could be fired after σ_1, σ^{pump} can be fired after σ_1: t corresponds to transition τ in O, and so does the first transition in σ^{pump}. Since σ^{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 τ', but $h(\tau) = t = h(\tau')$, so after firing σ^{pump}, we can fire t again. (Either as the first transition of the next σ^{pump}, or after all pumps have been fired.) Furthermore, t corresponds to the join-transition τ, i.e. it does create a token in the place that is the starting point of σ^{gen}. This token will not be consumed by any other transition in the sequence $t.\sigma_2$, so after firing σ^{pump} j times, we can indeed fire σ^{gen} j times.

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

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

Proof: We will restate the construction of the automaton given in Section V-B, prove its soundness and that the size of the automaton is as required.

The required finite state automaton $A = (Q, \to_A, q_0, F, \Sigma)$ is defined as follows:

Its set of states is $Q = P \mapsto ([0..c] \cup \{\omega\})$, where $c = tc(M_0)(|P| \cdot m)^{(|T|+1)}$ as in Lemma 12. This means each state is a marking that will assign to each place a number of tokens up to c or ω.

For each transition t of the BPP net N and each state $q \in Q$ such that $q(p) \geq F(p,t)$ (where we define $\omega > k$ to be true for all $k \in \mathbb{N}$), $\to_A$ contains two transitions $(q, \lambda(t), q')$ and (q, ε, q'). Here, q' is defined by

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

where $\oplus$ and $\ominus$ are variants of $+$ and $-$ that treat ω 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 ε, the two transitions coincide.

For the initial state, $\forall p \in P, q_0(p) = M_0(p)$. A state $q \in F$ is final if it covers the final marking M_f of N, i.e. $q(p) \geq M_f(p)$ for all places p. Again, we assume $\omega > k$ to hold for all $k \in \mathbb{N}$.

We prove that indeed $L(A) = L(N, M_0, M_f)\downarrow$.

For the one inclusion, assume $w \in L(N, M_0, M_f)\downarrow$. Then there is a computation $\pi = M_0[\sigma\rangle M$ of N such that $M \geq M_f$ and $w \preceq \lambda(\sigma)$. We can delete transitions in σ to obtain a sequence of transitions τ with $\lambda(\tau) = w$. (One may not be able to fire τ.) We construct a run ρ of the automaton A starting in q_0 by replacing transitions in σ by a corresponding transition of the automaton. For the transitions present in τ, we pick the variant of the transitions labeled by $\lambda(t)$ (where t is the transition of σ), for the ones not present in τ, we pick the ε-labeled variant. Note that ρ is a valid run of A because π was a computation of N. The run ρ ends in a state q_ρ such that for each place p, either $q_\rho(p) = M(p)$ holds, or $q(p) = \omega$. Since $M \geq M_f$, this means that q_ρ is final. We have constructed an accepting run of A that produces the word w.

Now assume that $w \in L(A)$ is a word accepted by the automaton. Let ρ be an accepting run and let $q_0, q_1, \ldots, q_s$ be the states occurring during ρ. We prove that there is a computation $\pi = M_0[\sigma\rangle M$ of N such that $M \geq M_f$ and $w \preceq \lambda(\sigma)$.

Assume the final state q_s does not assign ω 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 σ corresponding to ρ: For each transition in ρ, we take the corresponding transition of N. Note that the transition in ρ can be labeled by ε while the transition of N is not labeled by ε. Still, $w \preceq \lambda(\sigma)$ will hold. Furthermore, q_s is a marking for N (since ω does not occur), and since q_s was final, $q_s \geq M_f$ 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 ρ_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 σ_1 and σ_2 that correspond to ρ_1 and ρ_2. In particular, we have $w \preceq \lambda(\sigma_1).\lambda(\sigma_2)$. The first sequence σ_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 $M_1(p) > c$.

Still, it might not be possible to execute σ_2 from M_1, because σ_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 $\sigma_2 = t_1 \cdot t_2 \cdots t_{|\sigma_2|}$. The number d is certainly an upper bound for the number of tokens needed in place p to be able to fire σ_2 and end up in a marking M_2 such that $M_2(p) \geq M_f(p)$. We apply Lemma 12 to obtain a supersequence σ_1' of σ_1 with $M_0[\sigma_1'\rangle M_1'$ where $M_1' \geq M_1$ and $M_1(p) \geq d$.

Now consider the concatenation $\sigma = \sigma_1'.\sigma_2$. Since the marking M_1' has enough tokens in place p, σ is executable and $M_0[\sigma\rangle M$, where $M \geq M_f$.

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

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

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

■

Appendix C
Proofs for Section VI.

A. Petri Net Languages

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

Proof: $L(lin(p)) \subseteq L(sre)$ holds, so one direction is clear. For the other direction, we show that every word in $L(sre)$ is a subword of a word in $L(lin(p))$. From this, if $L(lin(p))$ is included in the downward closure, then all its subwords will be contained in the downward closure. In particular, all words in $L(sre)$ will be contained in the downward closure.

Towards proving that every word in $L(sre)$ is a subword of a word in $L(lin(p))$, note that for an arbitrary word in $L(sre)$, the letter a_i may or may not occur, while in $lin(p)$, they are forced to occur. If they do not occur, the resulting word is a subword of the word in which they occur. Furthermore, given $v \in \Sigma_i^s$, we have that v is a subword of $\pi_{\Sigma_i}(w_\Sigma)^s$ by dropping in each iteration all but one letter. Therefore, all words in Σ_i^* are subwords of $lin(\Sigma_i^*)$. If we combine those two insights, the desired statement follows. ■

Lemma 17. *$L(lin(p)) \subseteq L(N, M_0, 0)\downarrow$ if and only if the places in X are simultaneously unbounded in N' from M_0'.*

Proof: Assume the places in X are simultaneously unbounded in N'. Given a word $w \in L(lin(p))$, we need to find v such that $w \preceq v \in L(N, M_0, 0)$. Assume

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

Define $n = \max n_i$ and let σ be the run that creates at least n tokens in each place of $p_i \in X$. 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 $L(lin(p)) \subseteq L(N', M_0, 0)\downarrow$. Given any n, consider the word

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

Since $w \in L(N', M_0, 0)\downarrow$, there is a valid firing sequence σ with $w \preceq \lambda(\sigma)$. We consider the run σ' 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_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_i n times.

We have to argue why σ' 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 σ' can be fired, and the synchronized transitions satisfy the enabledness-condition on the places of N, since σ 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 clear since $L(N_{lin(p)}, M_0, M_f) = L(lin(p))$, and we only use the synchronized transitions for the subword $w \in L(lin(p))$. ∎

B. BPP Net Languages

For the sake of completeness, we give the standard construction that reduces SAT to the coverability of (ε-labeled) BPP nets. We can again rephrase the coverability query as checking whether the simple regular language $\{\varepsilon\}$ is contained in the downward closure of the coverability language of the net. This proves one half of Theorem 19.

Lemma 35. *Coverability in BPP nets is* NP*-hard.*

Proof: Given a Boolean formula φ in conjunctive normal form, one can construct a BPP net N and markings M_0, M_f polynomial in the size of φ such that φ is satisfiable if and only if M_f is coverable in N from M_0. This proves the lemma since SAT is NP-hard.

Let $x_1, \ldots, x_n$ be the variables occurring in φ.

The net N has three places x_i, x_i^+ and x_i^- for each a variable, and two transitions t_i^+ and t_i^- that both consume a token in x_i each and produce one in x_i^+ respectively x_i^-. If initially one token is present on x_i, either t_i^+ or t_i^- can be fired. This corresponds to setting x_i to one (and thus making positive occurrences of x_i true) or setting it to zero (and making negative occurrences true).

Furthermore, the net contains one place for each clause K of φ, and one place l_j for each literal l_j of K. For each literal occurrence l_j, there are two transitions $sat(l_j)$ and $get(l_j)$.

The first transition $sat(l_j)$ checks l_j for having a token (by consuming and producing it) and it additionally produces one token in the place K representing the clause in which l_j is contained. This transition represents that the clause is satisfied (carries at least one token) as soon as at least one of its literals is satisfied (carries at least one token).

The second transition $get(l_j)$ produces a token in l_j after checking $x_i^\pm$ for having a token, where x_i is the variable occurring in l_j, and we check x_i^+ if $l_j = x_i$ and we check x_i^- if $l_j = \neg x_i$. This transition represents that a literal can be set to true if the variable is assigned in the corresponding way.

The initial marking M_0 places one token on each place x_i and no token elsewhere, thus allowing each variable to be assigned to one value. The final marking M_f that has to be covered enforces that each clause is satisfied by requiring one token on each place K.

There is an immediate correspondence between covering computations of N and satisfying assignment for φ.

The final marking reached by an covering computation will assign one token to either x_i^+ and x_i^-, thus representing a variable assignment (or the token is left at x_i, in this case, x_i can be assigned arbitrarily). The fact that the computation is covering is a witness for the assignment being satisfying.

A satisfying assignment can be turned into a covering computation by first firing t_i^+ or t_i^- for each variable, depending on whether the variable is set to one or zero. We can then propagate the values of the variables to the literals that are made true by the assignment by using $get(l_j)$ exhaustively. Since the assignment is satisfying, we can fire $sat(l_j)$ for at least one literal per clause, thus ending up in a marking that covers M_f. ∎

Proposition 23. $L(p) \subseteq L(N, M_0, M_f)\downarrow$ *holds if and only if* (N, M_0, M_f) *admits a* p*-witness.*

Proof:

We will wlog. fix our product to be 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

$$M_1[\sigma_1\rangle M_1'[\rho_1\rangle M_2 \cdots [\rho_{n-1}\rangle M_n[\sigma_n\rangle M_n' \ ,$$

such that $M_n' \geq M_f$, $a_i \preceq \lambda(\sigma_i)$ and $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$, we will show that $L(p) \subseteq L(N, M_0, M_f)$. For this, we will show that for any word $w \in L(p)$, there is a run $M_0[\sigma\rangle M''$ such that $w \preceq \lambda(\sigma)$ and $M_n' \leq^c M''$ (and hence also $M_f \leq M''$). 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 $M_i' \leq^c M_i'''$, 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 $M_i \leq^c M_i''$.

Let $w' = x_1 v_1 \cdots x_i$. If $i = 1$ then the run $M_1[\sigma_1\rangle M_1'$ is the required run. Otherwise, inductively we get a run $M_1[\gamma\rangle M_i''$ such that $x_1 v_1 \cdots v_{i-1} \preceq \lambda(\gamma)$ and $M_i \leq^c M_i''$. Suppose for all places $p \in P$, $M_i'' < c$, then we have $M_i \leq M_i''$. Hence we can easily extend the computation by $M_i''[\sigma_i\rangle M_i'''$, which gives us the required run. Suppose the set of places X to which more than c tokens are assigned is non-empty. Using Lemma 12 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 $M_0[\gamma'\rangle M_i^*$ such that $\lambda(\gamma) \preceq \lambda(\gamma')$, for all $x \in X$ $M_i^*(x) - M_i''(x) \geq |\sigma_i|$ and $M_i'' \leq^c M_i^*$. Now the required extension of the run is $M_i^*[\sigma_i\rangle M_i'''$. Clearly $M_i \leq^c M_i'''$ since $M_i \leq^c M_i''$ and $M_i'' \leq^c M_i^*$ (also note that M_i^* is greater than $|\sigma_i|$ in places where the value is greater than c) .

For the second case, let

$$w' = (w''.a = x_1 v_1 \cdots x_i v_i'.a) \ ,$$

where $v_i'.a$ is a prefix of v_i. By induction, we have a run $M_1[\gamma\rangle M_i''$ such that $w'' \preceq \lambda(\gamma)$ and $M_i' \leq^c M_i''$. As in the previous case, if all for places $p \in P$, $M_i'' < c$ holds, then we have $M_i' \leq M_i''$. We now simply execute ρ_i and get the required run. Suppose we have some places with tokens more than c. As in the previous case, we pump up the values in these places to be greater than the size of ρ_i and then execute ρ_i to get the required 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 $L(p) \in L(N, M_0, M_f) \downarrow$, 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 $a_i \preceq \alpha_i$ and $\pi_{\Sigma_i}(w_\Sigma) \preceq \beta_i$. Since the length of β_i is $\ell c + 1$, 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 $J_i^1 \leq^c J_i^2$. 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. ■

Lemma 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.*

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 α_i corresponds to σ_i executed on the places E_{2i-1} by using the transitions in TE_{2i-1} (i.e. if a transition $t \in T$ is used in σ_i, then the transition $\text{te}_{2i-1}^t \in \text{TE}_{2i-1}$ is used in α_i). Similarly, the β_i correspond to the ρ_i executed on E_{2n} using transitions in TE_{2n}. The γ_i and γ_i' populate the set of places E_i accordingly: γ_1 produces $M_1(p)$ many tokens on each place e_1^p of E_1 using the transition in TC_1. For $i > 1$, γ_i produces $M_{2i-1}(p)$ many tokens on each place e_{2i-1}^p of E_{2i-1}, and γ_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 TC_i and TE_i are not dependent on each other and hence can be independently fired. Furthermore, for $i \neq j$, the transitions in TE_i and TE_j are independent of each other. Therefore, we may assume that in σ', the copy transitions in $\text{TC} = \bigcup_{i\in[1..2n[} \text{TC}_i$ are fired first, then the transition in TE_1 followed by transition in 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 $\sigma'' = \pi_{\text{TC}}(\sigma')$ and $\sigma_i' = \pi_{\text{TE}_i}(\sigma')$ for all $i \in [1..2n[$

Note that for each $i \in [1..2n-1]$, the transition sequence σ_i' in N' induces a transition sequence α_i in the original net N by using transition $t \in T$ instead of $\text{te}_i^t \in \text{TE}_i$.

The initial phase σ'' 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 (3), 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 $K_{2n-1} \leq^c M_f$.
(c) Since $M' \models \Psi_1$, we have $J_{i+1} = K_i$.
(d) Since $M' \models \Psi_2$, we have $J_{2i} \leq^c K_{2i}$ 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 (3) holds using (a) and (b). We conclude $a_i \preceq \lambda(\sigma_i)$ using (e) and $\pi_{\Sigma_i}(w_\Sigma) \preceq \lambda(\rho_i)$ using (f). Finally, (b) and (c) yield the rest of the required Properties (1) and (2). ■

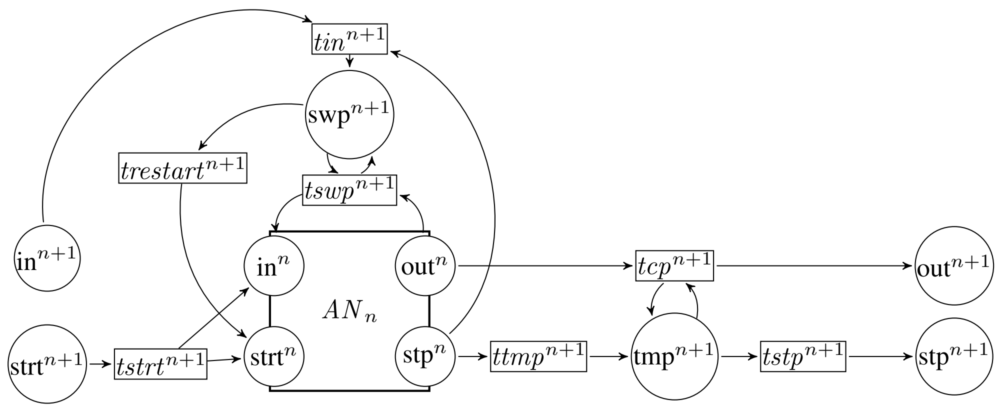

Fig. 2. The flow relation of the Petri net AN_{n+1}