ELSEVIER

Information Processing Letters 56 (1995) 237-243

Information Processing Letters

A note on fine covers and iterable factors of VAS languages

Hsu-Chun Yen [1]

Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC

Received 4 September 1995; revised 11 October 1995
Communicated by L. Boasson

Keywords: Complexity; Finite automata; Petri net languages; Vector addition system languages

1. Introduction

Vector addition systems (VASs, for short), or equivalently, *Petri nets*, represent a formalism useful for modeling concurrent systems. Once modeled by a VAS, the behavior of a system can be characterized by the set of all executable sequences, which in turn can be viewed as a language over an alphabet of symbols corresponding to the addition rules of the underlying VAS. It is known that all VAS languages are context-sensitive (assuming that a transition's symbol cannot be λ), and are incomparable with regular and context-free languages.

A useful tool for analyzing VAS problems is based on the *Karp–Miller coverability graph analysis* [1]. A coverability graph is a generalized reachability graph in which each potentially unbounded position of the VAS is represented by a special symbol "ω". It has been shown in [1] that for every VAS, its coverability graph is finite. As a result, a VAS is unbounded iff an ω occurs in its coverability graph. Aside from their direct application to the analysis of VASs, Karp and Miller's coverability graphs are also of interest to the language aspect of VASs. The finiteness of coverability graphs suggests a way to approximate VAS languages by finite automata. To be more precise, the coverability graph of a given VAS, say $\mathcal{P}$, can be regarded as a finite automaton which accepts a superset of the VAS language defined by $\mathcal{P}$. Such an observation gives rise to a natural question: how accurate is the above approximation? In an attempt to answer the above question, the notion of a *fine cover* was proposed by Schwer in [2,3] to formalize the concept of a good approximation of irregular languages by finite automata. The work of [2] involves, given a VAS $\mathcal{P}$, modifying the coverability graph of $\mathcal{P}$ so as to yield a finite automaton whose language is a fine cover of the VAS language defined by $\mathcal{P}$. The notion of an *iterable factor* arises in the study of fine covers of VAS languages. For a language L (over an alphabet A) and a string $w \in A^+$, w is said to be an *iterable factor* (of L) iff $\forall n \geqslant 0$, $A^*w^nA^* \cap L \neq \emptyset$. It has been shown in [2,3] that, given a VAS $\mathcal{P}$ and a string w, w is an iterable factor with respect to the language defined by $\mathcal{P}$ iff there exists a so-called "strong loop" labeled w in the coverability graph of $\mathcal{P}$.

The contributions of this paper include the following:

(1) We provide a simpler and faster way of constructing a fine cover, despite the fact that our construction may result in a fine cover which is bigger than the Schwer's one. Instead of doing as in [2] (i.e.,

[1] Email: yen@cc.ee.ntu.edu.tw.

Elsevier Science B.V.
SSDI 0020-0190(95)00168-9

for each loop, verifying whether it is a strong loop, and if not, suppressing the loop by substituting it with a finite-length path), what we do here is adding what is missing in order to make the loop a strong loop. The simplicity of our approach comes from the fact that our construction can be done simultaneously with the execution of the Karp–Miller procedure. In contrast, the approach used in [2] requires that the Karp–Miller coverability graph be built first.

(2) In this paper, we show the problem of, given a VAS $\mathcal{P}$ and a string w, determining whether w is an iterable factor of the language defined by $\mathcal{P}$ is EXPSPACE-complete.

2. Definitions

Let $\mathbb{Z}$ ($\mathbb{N}$) denote the set of (nonnegative) integers, and $\mathbb{Z}^k$ ($\mathbb{N}^k$) the set of vectors of k (nonnegative) integers. In this paper, the definition of a vector is generalized by allowing its components to bear a special symbol ω, i.e., a vector is now an element in $(\mathbb{Z} \cup \{\omega\})^k$ (or $(\mathbb{N} \cup \{\omega\})^k$). Furthermore, the following rules are used for additions involving ω:

$$\omega + c = c + \omega = \omega, \quad \text{for all } c \in \mathbb{Z}.$$

For a k-dimensional vector v, let $v[i]$, $\forall 1 \leqslant i \leqslant k$, denote the ith component of v. For a given value of k, let $\mathbf{0}$ denote the vector of k zeros (i.e., $\mathbf{0}[i] = 0$, $\forall 1 \leqslant i \leqslant k$). Given an alphabet A, we write A^* to denote the set of all finite-length strings (including the empty string λ) using symbols from A. We write A^+ to denote $A^* - \{\lambda\}$. Given a string u, let $u^+ = \{u^n \mid n \geqslant 1\}$.

A *k-dimensional vector addition system* (*k-VAS*, for short) is a 3-tuple (T, φ, v_0), where T is a finite set of symbols representing the *addition rules* of the VAS, $\varphi : T \to \mathbb{Z}^k$ is a mapping that associates each addition rule with a vector (called *addition vector*), and v_0 ($\in \mathbb{N}^k$) is the *start vector*. A string $w = v_1 \cdots v_j$ (for some j) in T^* is said to be *legal* in a vector v ($\in \mathbb{N}^k$) iff for every l, $1 \leqslant l \leqslant j$, $v + \sum_{i=1}^{l} \varphi(v_i) \geqslant \mathbf{0}$. We denote by $L(T, \varphi, v_0)$ the set of legal strings in v_0. We write $\varphi(w)$ to denote $\sum_{i=1}^{j} \varphi(v_i)$. We also write $v \xrightarrow{w} v + \varphi(w)$ to denote that w is legal in v and the *firing* of w results in vector $v + \varphi(w)$. (The word "firing" is borrowed from the analogous definition in Petri nets.) The *reachability set* of VAS (T, φ, v_0), denoted by $R(T, \varphi, v_0)$, is the set $\{v \in \mathbb{N}^k \mid \exists w \in L(T, \varphi, v_0),\ v = v_0 + \varphi(w)\}$. Throughout the rest of this paper, k is reserved for the dimension of the VASs.

Given a string $w \in T^*$, the following notations (adopted from [2,3]) will be used throughout the rest of this paper:

- $\|w\|^+ = \{i \mid 1 \leqslant i \leqslant k,\ \varphi(w)[i] > 0\}$ is the *positive support* of w.
- $\|w\|^- = \{i \mid 1 \leqslant i \leqslant k,\ \varphi(w)[i] < 0\}$ is the *negative support* of w.

For a language L (over an alphabet A) and a string $w \in A^+$, w is said to be an *iterable factor* (of L) iff $\forall n \geqslant 0$, $A^* w^n A^* \cap L \neq \emptyset$. For a *prefix-closed* language L, w is an *iterable factor* (of L) iff $\forall n \geqslant 0$, $A^* w^n \cap L \neq \emptyset$. (A language L is prefix-closed if $u \in L$ implies every prefix of u is also in L.) Given a language L, a rational (regular) language R is a *rational cover* of L if $L \subseteq R$. Suppose R is a rational cover of a language L, a rational language C is a *refinement* of R iff $L \subseteq C \subseteq R$. Consider a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the input alphabet, δ defines the transition function, q_0 ($\in Q$) is the initial state, and F ($\subseteq Q$) is the set of final states. A 2-tuple (q, w), $q \in Q$ and $w \in \Sigma^+$, is a *loop* iff $q \in \delta^*(q, w)$ (δ^*, the reflexive and transitive closure of δ, is defined in the conventional sense). A loop (q, w) is said to be a *strong loop* related to a language L iff $\forall n \in \mathbb{N}$, $\exists u, v \in \Sigma^*$ such that

$$q_0 \xrightarrow{u} q \xrightarrow{w^n} q \xrightarrow{v} q',$$

where $q' \in F$, and $uw^n v \in L$. Given a finite automaton M and a language L, M is said to define a *fine cover* of L iff $L(M)$ (i.e., the language accepted by M) is a rational cover of L and every loop of M is a strong loop related to L. See [2,3] for more details regarding the above definitions.

In [1], an effective algorithm known as the *Karp–Miller* procedure was designed to construct from a VAS $\mathcal{P}$ the so-called *coverability graph*, denoted by $\mathrm{CG}(\mathcal{P})$, in which each node of $\mathrm{CG}(\mathcal{P})$ is labeled by an element in $(\mathbb{N} \cup \{\omega\})^k$, and each directed edge of $\mathrm{CG}(\mathcal{P})$ is labeled by an addition rule in T. It is well known that $\mathrm{CG}(\mathcal{P})$ is always finite [1]. Given a coverability graph $\mathrm{CG}(\mathcal{P})$ of a VAS $\mathcal{P} = (T, \varphi, v_0)$ and a loop (q, w), an *iterating system of length p* related to (q, w) is a path (in $\mathrm{CG}(\mathcal{P})$ with q_0 the

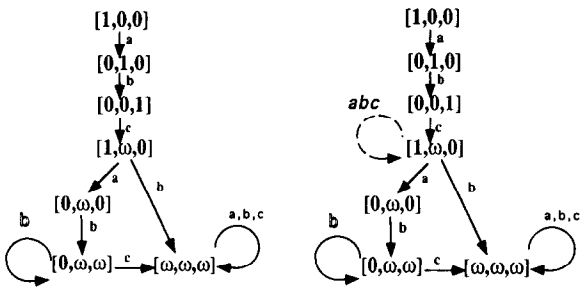

Fig. 1. Coverability graphs. (a) Karp–Miller coverability graph. (b) A modified coverability graph.

initial state)

$$q_0 \overset{w_1}{\longmapsto} q_1 \overset{\sigma_1}{\longmapsto} q_1 \overset{w_2}{\longmapsto} q_2 \overset{\sigma_2}{\longmapsto} q_2 \cdots \overset{w_p}{\longmapsto} q_p \overset{\sigma_p}{\longmapsto} q_p \overset{w_{p+1}}{\longmapsto} q \overset{w}{\longmapsto} q,$$

where $w_1, w_2, \ldots, w_{p+1}, \sigma_1, \sigma_2, \ldots, \sigma_p \in T^*$, such that

(1) $w_1\sigma_1^+ w_2\sigma_2^+ \ldots w_p\sigma_p^+ w_{p+1} w \cap L(\mathcal{P}) \neq \emptyset$,
(2) $\|\sigma_1\|^- = \emptyset$,
(3) $\|\sigma_i\|^- \subseteq \bigcup_{1 \leqslant j < i} \|\sigma_j\|^+$, $\forall 2 \leqslant i \leqslant p$, and
(4) $\|w\|^- \subseteq \bigcup_{1 \leqslant j \leqslant p} \|\sigma_j\|^+$.

Intuitively, the existence of an iterating system related to loop (q, w) ensures that $\forall n \geqslant 0$, w^n can be fired in the end provided that sufficient numbers of σ_1s, σ_2s, ..., and σ_ps are fired in the given order in advance. This, in turn, implies that w is an iterable factor with respect to $L(\mathcal{P})$. (See [2,3] for more about iterating systems and their related properties.)

3. Fine cover of a VAS language

The notion of a *fine cover* was defined by Schwer in [2,3] as a means for capturing the concept of "good" approximation of irregular VAS languages. The work of [2] involves, given a VAS $\mathcal{P}$, altering $\mathrm{CG}(\mathcal{P})$ so as to yield a finite automaton defining a fine cover of $L(\mathcal{P})$. (Notice that in $\mathrm{CG}(\mathcal{P})$, all states are accepting states.) In this section, we provide an alternative construction which, we feel, is simpler and can be done simultaneously with the execution of the Karp–Miller procedure, as opposed to that of [2] in which the construction is done in a more complex fashion. Before proceeding, we require the following results from [2]:

Theorem 1 (from Proposition 2.2.5 in [2]). *Let $\mathcal{P}$ be a VAS. If w is an iterable factor of $L(\mathcal{P})$, then there is a strong loop (q, w) in $\mathrm{CG}(\mathcal{P})$.*

Theorem 2 (from Proposition 3.5 in [2]). *Let $\mathcal{P}$ be a VAS and $\mathrm{CG}(\mathcal{P})$ be its coverability graph (automaton). The following two are equivalent:*
(1) *(q, w) is a strong loop,*
(2) *there exists an iterating system (of length less than or equal to k) related to (q, w).*

Theorem 2, coupled with the fact that detecting whether an iterating system exists is decidable [2], yields the decidability result of checking a strong loop, which, in turn, facilitates the effective construction of the corresponding fine cover of a VAS. For more details, the interested reader is referred to [2]. In what follows, we provide an alternative to the construction of a fine cover. Instead of doing as in [2] (i.e., for each loop, verifying whether it is a strong loop, and if not, suppressing the loop by substituting it with a finite-length path), what we do here is adding what is missing in order to make the loop a strong loop, knowing that for every loop (q, w) there is a strong loop (q', w), for some state q' (Lemma 2.2.4 in [2]).

We begin with a simple example using which the key idea behind our approach is illustrated. Consider a VAS

$$\mathcal{P} = (\{a, b, c\}, \{\varphi(a) = (-1, 1, 0), \varphi(b) = (0, -1, 1), \varphi(c) = (1, 1, -1)\}, (1, 0, 0)).$$

Fig. 1(a) shows the associated coverability graph of $\mathcal{P}$. Clearly $([0, \omega, \omega], b)$ is not a strong loop. Notice that the second coordinate actually decreases as the result of firing b in $[0, \omega, \omega] \overset{b}{\longmapsto} [0, \omega, \omega]$. However, the conventional definition of the ω arithmetic (in particular, $\omega - d = \omega$, for any constant d) fails to report the phenomenon of such a decrease in token. The idea behind the use of ω in the Karp–Miller procedure is to mark a potentially unbounded place. For instance,

$$[1, 0, 0] \overset{a}{\longmapsto} [0, 1, 0] \overset{b}{\longmapsto} [0, 0, 1] \overset{c}{\longmapsto} [1, 1, 0]$$

and $[1, 1, 0] > [1, 0, 0]$ result in the use of an ω in $[1, \omega, 0]$. Clearly, transition sequence abc can be fired an arbitrary number of times in $[1, 1, 0]$ to "blow up" the second place. In view of the above, it is safe to

attach an arc label abc to $[1,\omega,0]$ without introducing any infeasible sequence, for an arbitrary number of abcs can be fired in $[1,\omega,0]$. By doing so, the subsequent loop $([0,\omega,\omega],b)$ becomes a strong loop (see Fig. 1(b)). $([1,\omega,0],abc)$ acts as a pump to supply necessary tokens for the subsequent transition sequence b^n in $[0,\omega,\omega]$ to fire, for any $n \in \mathbb{N}$.

Based upon the above idea, we have the procedure in Fig. 2 for constructing a fine cover of a given VAS. Our design will relate each state in the resulting automaton to a vector ($\in (\mathbb{N} \cup \{\omega\})^k$) of the original VAS. For convenience, we associate a labeling function l to such a mapping.

Algorithm *Cover* is a slight modification of the original Karp–Miller procedure, the termination of *Cover* follows. Hence, we have

Theorem 3. *Algorithm Cover terminates.*

Given a VAS $\mathcal{P}$, let $\mathrm{CG}(\mathcal{P})$, $C(\mathcal{P})$, and $Cover(\mathcal{P})$ be the finite automata generated by the *Karp–Miller* procedure, the procedure defined by *Schwer* in [2], and our *Cover* procedure, respectively. Then we have

Theorem 4.

$$L(\mathcal{P}) \subseteq L(C(\mathcal{P})) \subseteq L(\mathrm{CG}(\mathcal{P})) \subseteq L(Cover(\mathcal{P})).$$

Proof. $L(\mathcal{P}) \subseteq L(C(\mathcal{P})) \subseteq L(\mathrm{CG}(\mathcal{P}))$ follows from Proposition 4.2 in [2], whereas $L(\mathrm{CG}(\mathcal{P})) \subseteq L(Cover(\mathcal{P}))$ is trivial. □

According to Theorem 4, finite automaton $Cover(\mathcal{P})$ defines a rational cover of VAS $\mathcal{P}$. In what follows, we prove that $Cover(\mathcal{P})$ is a fine cover. It suffices to prove the following theorem:

Theorem 5. *Given a VAS $\mathcal{P}$, every loop in the automaton constructed by algorithm $Cover(\mathcal{P})$ is a strong loop.*

Proof. We let $\omega(x)$ be the set of ω places of a vector x. Consider a loop u emanating from a state with label v (which is a vector in $(\mathbb{N} \cup \{\omega\})^k$). Let

$$v_0 \xrightarrow{w_1} v_1 \xrightarrow{w_2} v_2 \cdots \xrightarrow{w_m} v_m \xrightarrow{w_{m+1}} v$$

be a computation path such that $v_1, v_2, \ldots, v_m$ mark the labels of those states at which some coordinates become ω for the first time along the path. A careful examination of procedure *Cover* reveals that there exist loops $\sigma_1, \sigma_2, \ldots, \sigma_m$ attached to $v_1, v_2, \ldots, v_m$, respectively, such that

$$\|\sigma_1\|^+ = \omega(v_1),$$
$$\|\sigma_1\|^- = \emptyset,$$
$$\|\sigma_2\|^- \subseteq \omega(v_1) \quad (= \|\sigma_1\|^+),$$
$$\vdots$$
$$\|\sigma_j\|^- \subseteq \omega(v_{j-1}) \quad \left(= \bigcup_{1 \leqslant i \leqslant j-1} \|\sigma_i\|^+\right)$$
$$\vdots$$
$$\|u\|^- \subseteq \omega(v) \quad \left(= \bigcup_{1 \leqslant i \leqslant m} \|\sigma_i\|^+\right).$$

(See lines 10–15 of procedure *Cover*.) Hence, we have an iterating system. Clearly, u is a strong loop. □

In view of the above, we have:

Corollary 6. *Given a VAS $\mathcal{P}$, algorithm $Cover(\mathcal{P})$ generates a fine cover of $\mathcal{P}$.*

In comparison with Schwer's procedure [2], our Algorithm *Cover* provides a simpler and faster way of constructing a fine cover, despite the fact that our construction may result in a fine cover which is bigger than Schwer's one (Theorem 4).

4. Complexity of deciding iterable factors

Recall that for a prefix-closed language L (over an alphabet A), a string w is said to be an *iterable factor* (of L) iff $\forall n \geqslant 0$, $A^* w^n \cap L \neq \emptyset$. Intuitively, w is an iterable factor if it can be "pumped" an arbitrary number of times, provided that an appropriate prefix computation is executed. As mentioned in the previous section, the concept of an iterable factor plays an important role in the study of fine covers of VAS languages. Now consider the following problem:

- *Input instance*: A VAS $\mathcal{P}$ and a string w.
- *Question*: Is w an iterable factor of $L(\mathcal{P})$?

In what follows, we show the above problem to be EXPSPACE-complete. Our analysis takes advantage

Procedure *Cover*$((T, \varphi, v_0))$
{The procedure is to construct a finite automaton M, in which all states are *accepting*}
1. **begin**
2. mark the initial state q_0 ($l(q_0) = v_0$) as "*active*"
3. **while** M contains some active node, say p, **do**
4. **for** every $t \in T$
5. which is enabled in $l(p)$, and
6. no edge emanating from p is labeled by t
7. **do**
8. **if** there is a node q' from q_0 to p with label $l(p) + \varphi(t)$
9. **then** add transition $p \xrightarrow{t} q'$ to M
10. **else if** there is a node q'' from q_0 to p such that $l(q'') < l(p) + \varphi(t)$
11. **then do**
12. add an *active* state q with the following label and a transition $p \xrightarrow{t} q$
13. (a) $l(q)_i = \omega$, if $l(q'')_i < (l(p) + \varphi(t))_i$
14. (b) $l(q)_i = l(q'')_i$, otherwise
15. add a cycle (loop) in q with the same edge sequence from q'' to q
16. **od**
17. **else** add an *active* state q ($l(q) = l(p) + \varphi(t)$) and transition $p \xrightarrow{t} q$ to M
18. **fi**
19. **fi**
20. **od**
21. mark p as *inactive*

end procedure.

Fig. 2.

of the connection between strong loops and iterating systems for the coverability automata of VAS languages [2], as well as an algorithm derived in [4] for testing the existence of certain paths in Petri nets (or equivalently, VASs).

It has been shown in [2] that given a VAS $\mathcal{P}$, if w is an iterable factor of $L(\mathcal{P})$, then there is a strong loop (q, w) in the coverability automaton of $\mathcal{P}$ (see Theorem 1). The converse trivially holds as well. Also shown in [2] is that (q, w) is a strong loop with respect to the coverability automaton of a VAS $\mathcal{P}$ iff there exists an iterating system related to (q, w) (see Theorem 2). The connection between iterable factors and iterating systems is interesting; however, such a result is useful for checking iterable factors only if the associated coverability automaton of the considered VAS is in presence (for iterating systems are defined on coverability automata, rather than on VASs directly). The following theorem, describing necessary and sufficient conditions for a string to be an iterable factor, allows us to bypass coverability automata in the process of checking iterable factors.

Theorem 7. *In a VAS $\mathcal{P} = (T, \varphi, v_0)$, a string w is an iterable factor of $L(\mathcal{P})$ iff there exists a VAS computation*

$$v_0 \xrightarrow{w_1} v_1 \xrightarrow{\delta_1} v_1' \xrightarrow{w_2} v_2 \xrightarrow{\delta_2} v_2' \cdots \xrightarrow{w_p} v_p \xrightarrow{\delta_p} v_p' \xrightarrow{w_{p+1}} v \xrightarrow{w} v',$$

for some $p \leqslant k$, $w_1, w_2, \ldots, w_{p+1}, \delta_1, \delta_2, \ldots, \delta_p \in T^*$, $v_1, \ldots, v_p, v_1', \ldots, v_p', v, v' \in \mathbb{N}^k$ such that
(i) $\|\delta_1\|^- = \emptyset$,
(ii) $\|\delta_i\|^- \subseteq \bigcup_{1 \leqslant j < i} \|\delta_j\|^+$, $\forall 2 \leqslant i \leqslant p$, *and*
(iii) $\|w\|^- \subseteq \bigcup_{1 \leqslant j \leqslant p} \|\delta_j\|^+$.
(*In words, w is an iterable factor iff it is a right factor of an iterating system.*)

Proof. We first show the *only if* part. Let

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{\sigma_1} q_1 \xrightarrow{w_2} q_2 \xrightarrow{\sigma_2} q_2 \cdots \xrightarrow{w_p} q_p \xrightarrow{\sigma_p} q_p \xrightarrow{w_{p+1}} q \xrightarrow{w} q$$

be the iterating system guaranteed by Theorems 1 and 2. According to Condition (1) of the definition of iterating systems (i.e., $w_1\sigma_1^+ w_2\sigma_2^+ \ldots w_p\sigma_p^+ w_{p+1} w \cap L(\mathcal{P}) \neq \emptyset$), there must exist δ_i ($\in \sigma_i^+$), $1 \leqslant i \leqslant p$, and $v_1, \ldots, v_p, v_1', \ldots, v_p', v, v' \in N^k$ satisfying the statement of the theorem. Conditions (i), (ii), and (iii) follow immediately from Conditions (2), (3),

and (4), respectively, of the definition of iterating systems. (Notice that $\|\delta_i\|^- = \|\sigma_i\|^-$, for all i.)

The *if* part is obvious. □

In [4], a class of path formulas for general Petri nets (equivalently, VASs) has been defined for which the satisfiability problem has been shown to be solvable in EXPSPACE. As it turns out, testing Conditions (i), (ii), and (iii) of Theorem 7 can be expressed as a formula allowed in [4]. Hence, the EXPSPACE upper bound for testing iterable factors follows. For the sake of completeness, we now briefly define the class of path formulas discussed in [4]. (Notice that in [4], the definition is given in the context of Petri nets.) Let (T, φ, μ_0) be a k-dimensional VAS with m addition rules. Each path formula consists of the following elements:

(1) *Variables*: There are two types of variables, namely, *marking variables* $\mu_1, \mu_2, \ldots$ and *variables for transition sequences* $\sigma_1, \sigma_2, \ldots$, where each μ_i denotes a vector in $\mathbb{N}^k$ and each σ_i denotes a finite sequence of addition rules.

(2) *Terms*: Terms are defined recursively as follows.

(a) $\forall$ constant $c \in \mathbb{N}^k$, c is a term.

(b) $\forall\, j > i$, $\mu_j - \mu_i$ is a term, where μ_i and μ_j are marking variables.

(c) $T_1 + T_2$ and $T_1 - T_2$ are terms if T_1 and T_2 are terms.

(3) *Atomic predicates*: Let $\odot$ denote the inner product operator of vectors, and $\#_\sigma$ be a mapping from T to $\mathbb{N}$ such that $\#_\sigma(t)$ represents the number of occurrences of addition rule t in sequence σ. ($\#_\sigma$ can be viewed as a vector in $\mathbb{N}^m$.) There are two types of atomic predicates, namely, *transition predicates* and *marking predicates*.

(a) *Transition predicates*:

- $y \odot \#_{\sigma_i} < c$, $y \odot \#_{\sigma_i} = c$ and $y \odot \#_{\sigma_i} > c$ are predicates, where $i > 1$, y (a constant) $\in \mathbb{Z}^m$, $c \in \mathbb{N}$.
- $\#_{\sigma_1}(t_j) \leqslant c$ and $\#_{\sigma_1}(t_j) \geqslant c$ are predicates, where $c \in \mathbb{N}$ and $t_j \in T$.

(b) *Marking predicates*:

- *Type* 1: $\mu(i) \geqslant c$ and $\mu(i) > c$ are predicates, where μ is a marking variable and c ($\in \mathbb{Z}$) is a constant.
- *Type* 2: $T_1(i) = T_2(j)$, $T_1(i) < T_2(j)$ and $T_1(i) > T_2(j)$ are predicates, where T_1, T_2 are terms and $1 \leqslant i, j \leqslant k$.

$F_1 \vee F_2$ and $F_1 \wedge F_2$ are predicates if F_1 and F_2 are predicates.

In [4], the *satisfiability problem* (i.e., the problem of determining whether there exists a path satisfying the given formula) for the following class of formulas has been shown to be solvable in EXPSPACE:

$$\exists \mu_1, \ldots, \mu_m \ \exists \sigma_1, \ldots, \sigma_m \ ((\mu_0 \xmapsto{\sigma_1} \mu_1 \xmapsto{\sigma_2} \cdots \mu_{m-1} \xmapsto{\sigma_m} \mu_m) \wedge F(\mu_1, \ldots, \mu_m, \sigma_1, \ldots, \sigma_m))$$

Theorem 8. *Given a VAS $\mathcal{P} = (T, \varphi, v_0)$ and a string w, determining whether w is an iterable factor of $L(\mathcal{P})$ is EXPSPACE-complete.*

Proof. We first consider the upper bound. Suppose $w = t_1 t_2 \cdots t_n$, $t_i \in T$, $\forall 1 \leqslant i \leqslant n$. According to Theorem 7, string w is an iterable factor of $\mathcal{P}$ iff there exists a computation

$$v_0 \xmapsto{w_1} v_1 \xmapsto{\delta_1} v_1' \xmapsto{w_2} v_2 \xmapsto{\delta_2} v_2' \cdots \xmapsto{w_p} v_p \xmapsto{\delta_p} v_p' \xmapsto{w_{p+1}} v \xmapsto{w} v',$$

for some $p \leqslant k$, $w_1, w_2, \ldots, w_{p+1}, \delta_1, \delta_2, \ldots, \delta_p \in T^*$, $v_1, \ldots, v_p, v_1', \ldots, v_p', v, v' \in \mathbb{N}^k$, satisfying Conditions (i), (ii), and (iii) of the statement of Theorem 7.

Our algorithm begins with guessing p ($\leqslant k$) and sets Δ_i^- ($\subseteq \{1, \ldots, k\}$, for $2 \leqslant i \leqslant p$) to capture $\|\delta_i\|^-$ so that Conditions (i), (ii), and (iii) are met. Then w is an iterable factor iff the following path formula is satisfiable:

$$\exists v_1, v_1', \ldots, v_p, v_p', v \ \exists w_1, \ldots, w_{p+1}, \delta_1, \ldots, \delta_p, (v_0 \xmapsto{w_1} v_1 \xmapsto{\delta_1} v_1' \xmapsto{w_2} v_2 \xmapsto{\delta_2} v_2' \cdots \xmapsto{w_p} v_p \xmapsto{\delta_p} v_p' \xmapsto{w_{p+1}} v)$$

and

(I) $(v_1' \geqslant v_1)$

(expressing Condition (i) of Theorem 7, i.e., $\|\delta_1\|^- = \emptyset$)

$$\text{(II)}\quad \bigwedge_{2\leqslant i\leqslant p}\Big(\Big(\bigwedge_{l\notin\Delta_i^-}(v_i'(l)-v_i(l)\geqslant 0)\Big)\wedge\Big(\bigwedge_{l\in\Delta_i^-}\Big(\bigvee_{1\leqslant j<i}(v_j'(l)-v_j(l)>0)\Big)\Big)\Big)$$

(expressing Condition (ii) of Theorem 7, i.e., $\|\delta_i\|^-\subseteq\bigcup_{1\leqslant j<i}\|\delta_j\|^+$),

$$\text{(III)}\quad \bigwedge_{l\in\|w\|^-}\Big(\bigvee_{1\leqslant j\leqslant p}(v_j'(l)-v_j(l)>0)\Big)$$

(expressing Condition (iii) of Theorem 7, i.e., $\|w\|^-\subseteq\bigcup_{1\leqslant j\leqslant p}\|\delta_j\|^+$),

$$\text{(IV)}\quad \bigwedge_{1\leqslant j\leqslant n}\Big(v+\sum_{i=1}^{j}\varphi(t_i)\geqslant\mathbf{0}\Big)$$

(ensuring that w is legal in v).

Hence, testing iterable factors can be done in EXPSPACE. The lower bound can easily be proved by reducing the boundedness problem for VASs (which is well-known to require EXPSPACE) to our problem. The details are left to the reader. □

Acknowledgment

The author thanks the anonymous referee for suggestions that improved the correctness as well as the presentation of this paper.

References

[1] R. Karp and R. Miller, Parallel program schemata, *J. Comput. System Sci.* **3** (1969) 167–195.

[2] S. Schwer, Fine covers of a VAS language, *Theoret. Comput. Sci.* **95** (1992) 159–168.

[3] S. Schwer, The context-freeness of the languages associated with vector addition systems is decidable, *Theoret. Comput. Sci.* **98** (1992) 199–247.

[4] H. Yen, A unified approach for deciding the existence of certain Petri net paths, *Inform. and Comput.* **96** (1992) 119–137.