Expand, Enlarge, and Check for Branching Vector Addition Systems

Rupak Majumdar and Zilong Wang

MPI-SWS, Germany

Abstract. Expand, enlarge, and check (EEC) is a successful heuristic for the coverability problem of well-structured transition systems. EEC constructs a sequence of under- and over-approximations with the property that the presence of a bug is eventually exhibited by some under-approximation and the absence of a bug is eventually exhibited by some over-approximation.

In this paper, we consider the application of EEC to the coverability problem for branching vector addition systems (BVAS), an expressive model that subsumes Petri nets. We describe an EEC algorithm for BVAS, and prove its termination and correctness. We prove an upper bound on the number of iterations for our EEC algorithm, both for BVAS and, as a special case, vector addition systems (or Petri nets). We show that in addition to practical effectiveness, the EEC heuristic is asymptotically optimal. For BVAS, it requires at most doubly-exponentially many iterations, thus matching the optimal 2EXPTIME upper bound. For Petri nets, it can be implemented in EXPSPACE, again matching the optimal bound. We have implemented our algorithm and used it to verify safety properties of concurrent programs with asynchronous tasks.

1 Introduction

Branching vector addition systems (BVAS) are an expressive model that generalize vector addition systems (VAS, or Petri nets) with branching structures. Intuitively, one can consider a VAS as producing a linear sequence of vectors using unary rewrite rules, where a rewrite rule takes a vector v and adds a constant δ to it, as long as the sum $v + \delta$ remains non-negative on all co-ordinates. A branching VAS adds a second, binary rewrite rule that takes two vectors v_1 and v_2 and rewrites them to $v_1 + v_2 + \delta$ for a constant δ, again provided the sum is non-negative on all co-ordinates. Thus, a BVAS generates a derivation tree of vectors, starting with a multiset of initial vectors, or axioms, at the leaves and generating a vector at the root of a derivation, where each internal node in the tree applies a unary or a binary rewrite rule. The reachability problem for BVAS is to check if a given vector can be derived, and the coverability problem asks, given a vector v, if a vector $v' \geq v$ can be derived. These generalize the corresponding problems for VAS. Several verification problems, such as the analysis of recursively parallel programs [1] and the analysis of some cryptographic protocols [17], have been shown to reduce to the coverability problem for BVAS.

Coverability for BVAS is known to be decidable, both through a generalized Karp-Miller construction [16] as well as through a bounding argument [5]. Further, the bounding argument characterizes the complexity of the problem: coverability is 2EXPTIME-complete [5] (contrast with the EXPSPACE-completeness for VAS [15]). The Karp-Miller construction is non-primitive recursive, since BVAS subsume VAS [11].

Despite potential applications, the study of BVAS has so far remained in the domain of theoretical results, and to the best of our knowledge, there have not been any attempts to build analysis tools for coverability. In contrast, tools for VAS coverability have made steady progress and can now handle quite large benchmarks derived from the analysis of multi-threaded programs [10,13]. In our view, one reason is that a direct implementation of the algorithms from [16,5] are unlikely to perform well: Karp-Miller trees for VAS do not perform well in practice, and Demri et al.'s complexity-theoretically optimal algorithm performs a non-deterministic guess and enumeration by an alternating Turing machine.

In this paper, we apply the *expand, enlarge, and check* paradigm (EEC) [7] to the analysis of BVAS. EEC is a successful heuristic for checking coverability of well-structured transition systems such as Petri nets. It constructs a sequence of under- and over-approximations of the state space of a system such that, for a target state t, (1) if t is coverable, then a witness is found by an under-approximation, (2) if t is not coverable, then a witness for un-coverability is found by an over-approximation, and (3) eventually, one of the two outcomes occur and the algorithm terminates.

EEC offers several nice features for implementation. First, each approximation it considers is finite-state, thus opening the possibility of applying model checkers for finite-state systems. Second, EEC is goal-directed: it computes abstractions that are precise enough to prove or disprove coverability of a target, unlike a Karp-Miller procedure that computes the exact coverability set independent of the target. Third, it allows a forward abstract exploration of the state space, which is often more effective in practice.

Our first contribution is to port the EEC paradigm to the coverability analysis of BVAS. We show how to construct a sequence of under- and over-approximations of derivations such that if a target is coverable, an under-approximation derives a witness for coverability, and if a target is not coverable, an over-approximation derives a witness for un-coverability. We generalize the proof of correctness of EEC for well-structured systems. Since there is no BVAS analogue of a backward-reachability algorithm for VAS, our proofs instead use induction on derivations and the Karp-Miller construction of [16].

A natural question is how well EEC performs in the worst case compared to asymptotically optimal algorithms. For example, even for VAS, it is unknown if the EEC algorithm can match the known EXPSPACE upper bound for coverability, or if it matches the non-primitive recursive lower bound for Karp-Miller trees. Our second contribution is to bound the number of iterations of the EEC algorithm in the worst case. We show that we can compute a constant c of size doubly exponential in the size of the BVAS and the target vector such that the

EEC algorithm is guaranteed to terminate in c iterations. In each iteration, the algorithm explores approximate state spaces of derivations, that correspond to exploring AND-OR trees of size doubly exponential in the input. In other words, if each exploration is performed optimally, we get an optimal asymptotic upper bound for EEC. Specifically, for VAS, we get an EXPSPACE upper bound, since there are doubly exponential iterations and each iteration checks two reachability problems over doubly-exponential state spaces. (In practice though, model checkers do not implement space-optimal reachability procedures.) While our proof uses Rackoff-style bounds [15,5], our implementation does not require any knowledge of these bounds. A similar argument was used in [2] to show a doubly exponential bound on the backward reachability algorithm for VAS.

We have implemented the EEC-based procedure for BVAS coverability. Our motivation for analyzing BVAS came from the analysis of recursively parallel programs [6,1]. It is known that the analysis of asynchronous programs, a co-operatively scheduled concurrency model, can be reduced to coverability of VAS [6], and there have been EEC-based tools for these programs [9]. However, some asynchronous programs use features such as posting a set of tasks in a handler and waiting on the first task to return, that are not reducible to asynchronous programs. Bouajjani and Emmi [1] define a class of recursively parallel programs that can express such constructs, and show that the safety verification problem for this class is equivalent to coverability of BVAS. We applied this reduction in our implementation, and used our tool to model check safety properties of recursively parallel programs. We coded the control flow of tasks in a simple web server [4] and showed that our tool can successfully check for safety properties and find bugs. On our examples, the EEC algorithm terminates in one iteration, that is, with a $\{0, 1, \infty\}$ abstraction. While our evaluations are preliminary, we believe there is a potential for model checking tools for complex concurrent programs based on BVAS coverability.

2 Preliminaries

Well Quasi Ordering. A *quasi ordering* $(X, \preceq)$ is a reflexive and transitive binary relation on X. A quasi ordering $(X, \preceq)$ is a *well quasi ordering* iff for every infinite sequence $x_0, x_1, \ldots$ of elements from X, there exists $i < j$ with $x_i \preceq x_j$. A subset X' of X is *upward closed* if for each $x \in X$, if there is an $x' \in X'$ with $x' \preceq x$ then $x \in X'$. A subset X' of X is *downward closed* if for each $x \in X$, if there is an $x' \in X'$ with $x \preceq x'$ then $x \in X'$. Given $x \in X$, we write $x\!\downarrow$ and $x\!\uparrow$ for the *downward closure* $\{x' \in X \mid x' \preceq x\}$ and *upward closure* $\{x' \in X \mid x \preceq x'\}$ of x respectively. Downward and upward closures are naturally extended to sets, i.e., $X\!\downarrow = \bigcup_{x \in X} x\!\downarrow$ and $X\!\uparrow = \bigcup_{x \in X} x\!\uparrow$. A subset $S \subseteq X$ is *minimal* iff for every two elements $x, x' \in S$, we have $x \not\preceq x'$.

Numbers and Vectors. We write $\mathbb{N}$, $\mathbb{N}^+$ and $\mathbb{Z}$ for the set of non-negative, positive and arbitrary integers, respectively. Given two integers a and b, we write $[a, b]$ for $\{n \in \mathbb{Z} \mid a \leq n \leq b\}$.

For a vector $v \in \mathbb{Z}^k$ and $i \in [1, k]$, we write $v[i]$ for the ith component of v. Given two vectors $v, v' \in \mathbb{Z}^k$, $v \leq v'$ iff for all $i \in [1, k]$, $v[i] \leq v'[i]$. Moreover,

$v < v'$ iff $v \leq v'$ and $v' \not\leq v$. It is well-known that $(\mathbb{N}^k, \leq)$ is a well quasi ordering. We write $\mathbf{0}$ for the zero vector.

Given a finite set $S \subseteq \mathbb{Z}$ of integers, we write $\max(S)$ for the greatest integer in the set. We define $\max(\emptyset) = 0$. Given a vector $v \in \mathbb{Z}^k$, let $\max(v) = \max(\{v[1], \ldots, v[k]\})$. When $k = 0$, we have $\max(\langle\rangle) = 0$. We define $\min(S)$ analogously. We write $\min(0, v)$ for the vector $\langle \min(\{0, v[1]\}), \ldots, \min(\{0, v[k]\})\rangle$. The vector $\max(0, v)$ is defined analogously. For simplicity, we write v^- for the vector $-\min(0, v)$ and v^+ for the vector $\max(0, v)$. Given a finite set of vectors $R \subseteq \mathbb{Z}^k$, let $R^{-/+}$ be the set $\{v^{-/+} \mid v \in R\}$ respectively. We define $\max(R) = \max(\{\max(v^+) \mid v \in R\})$. The size of a vector is the number of bits required to encode it, all numbers being encoded in binary.

Trees. A *finite binary tree* $\mathcal{T}$, which may contain nodes with one child, is a non-empty finite subset of $\{1, 2\}^*$ such that, for all $n \in \{1, 2\}^*$ and $i \in \{1, 2\}$, $n \cdot 2 \in \mathcal{T}$ implies $n \cdot 1 \in \mathcal{T}$, and $n \cdot i \in \mathcal{T}$ implies $n \in \mathcal{T}$. The nodes of $\mathcal{T}$ are its elements. The root of $\mathcal{T}$ is ε, the empty word. All notions such as parent, child, subtree and leaf, have their standard meanings. The height of $\mathcal{T}$ is the number of nodes in the longest path from the root to a leaf.

BVAS, Derivations, and Coverability. A *branching vector addition system* (BVAS) [16,5] is a tuple $\mathcal{B} = \langle k, A, R_1, R_2\rangle$, where $k \in \mathbb{N}$ is the *dimension*, $A \subseteq \mathbb{N}^k$ is a non-empty finite set of *axioms*, and $R_1, R_2 \subseteq \mathbb{Z}^k$ are finite sets of *unary and binary rules*, respectively. The size of a BVAS, *size*$(\mathcal{B})$ is the number of bits required to encode a BVAS, where numbers are encoded in binary.

The semantics of a BVAS $\mathcal{B}$ is captured using derivations. Intuitively, a derivation starts with a number of axioms from A, proceeds by applying rules from $R_1 \cup R_2$, and ends with a single vector. Applying a unary rule means adding it to a derived vector, and applying a binary rule means adding it to the sum of two derived vectors. While applying rules, all derived vectors are required to be non-negative. Formally, a *derivation* $\mathcal{D}$ of $\mathcal{B}$ is defined inductively as follows.

D1: If $v \in A$, then $\overline{v}$ is a derivation.

D2: If $\mathcal{D}_1$ is a derivation with a derived vector $v_1 \in \mathbb{N}^k$, then for each unary rule $\delta_1 \in R_1$ with $\mathbf{0} \leq v_1 + \delta_1$,

$$\begin{array}{c} \vdots\, \mathcal{D}_1 \\ \mathcal{D} : \dfrac{v_1}{v}\, \delta_1 \end{array}$$

is a derivation, where $v = v_1 + \delta_1$.

D3: If $\mathcal{D}_1$ and $\mathcal{D}_2$ are derivations with derived vectors $v_1, v_2 \in \mathbb{N}^k$ respectively, then for each binary rule $\delta_2 \in R_2$ with $\mathbf{0} \leq v_1 + v_2 + \delta_2$,

$$\begin{array}{c} \vdots\, \mathcal{D}_1 \qquad\quad \vdots\, \mathcal{D}_2 \\ \mathcal{D} : \dfrac{v_1 \qquad\qquad v_2}{v}\, \delta_2 \end{array}$$

is a derivation, where $v = v_1 + v_2 + \delta_2$.

A derivation $\mathcal{D}$ can be represented as a finite binary tree whose nodes are labelled by non-negative vectors. Therefore, all notions of trees can be naturally applied to derivations. For a derivation $\mathcal{D}$ and its node n, we write $\mathcal{D}(n)$ for the non-negative vector labelled at n. We say $\mathcal{D}$ *derives a vector* v iff $\mathcal{D}(\varepsilon) = v$.

A derivation $\mathcal{D}$ is *compact* iff for each node n and for each its ancestor n', we have $\mathcal{D}(n) \neq \mathcal{D}(n')$. Given a derivation $\mathcal{D}$ with a node n and an ancestor n' of n with $\mathcal{D}(n) = \mathcal{D}(n')$, a *contraction* $\mathcal{D}[n' \leftarrow n]$ over $\mathcal{D}$ is obtained by replacing the subtree rooted at n' with the subtree rooted at n in $\mathcal{D}$. We write $\mathsf{compact}(\mathcal{D})$ for the compact derivation computed by a finite sequence of contractions over $\mathcal{D}$.

Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2\rangle$, we say a vector v is *reachable* in $\mathcal{B}$ iff there is a derivation $\mathcal{D}$ with $\mathcal{D}(\varepsilon) = v$. We write $\mathit{Reach}(\mathcal{B}) = \{v \mid \exists \mathcal{D}.\ \mathcal{D}(\varepsilon) = v\}$ for the set of reachable vectors in $\mathcal{B}$. We say a vector v is *coverable* in $\mathcal{B}$ iff there is a derivation $\mathcal{D}$ with $v \leq \mathcal{D}(\varepsilon)$. We call a derivation $\mathcal{D}$ a *covering witness* of v iff $v \leq \mathcal{D}(\varepsilon)$. The *coverability problem* asks, given a BVAS $\mathcal{B}$ and a vector $t \in \mathbb{N}^k$, whether t is coverable in $\mathcal{B}$. Equivalently, t is coverable iff $t \in \mathit{Reach}(\mathcal{B})\!\downarrow$.

3 Under-and Over-Approximation

We give two approximate analyses for BVAS: an under-approximation that fixes a finite set of vectors and only considers those vectors in that finite set, and an over-approximation that introduces limit elements. The under-approximation can show that a vector is coverable and the over-approximation can prove that a vector is not coverable.

3.1 Underapproximation

Truncated Derivations. Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2\rangle$ and an $i \in \mathbb{N}$, define $C_i \subseteq \mathbb{N}^k$ as $A \cup \{0, \ldots, i\}^k$. Given a vector $v \in \mathbb{N}^k$ and an $i \in \mathbb{N}$, We write $\mathsf{under}(v, i)$ for a *truncated vector* such that for all $j \in [1, k]$, $\mathsf{under}(v, i)[j] = v[j]$ if $v[j] \leq i$, $\mathsf{under}(v, i)[j] = i$ otherwise. For all vector $v \in \mathbb{N}^k$ and for all $i \in \mathbb{N}$, $\mathsf{under}(v, i) \leq v$. A *truncated derivation* $\mathcal{F}$ w.r.t. i is defined inductively as follows.

T1: If $v \in A$, then $\overline{v}$ is a truncated derivation.

T2: If $\mathcal{F}_1$ is a truncated derivation with a derived truncated vector $v_1 \in \mathbb{N}^k$, then for each unary rule $\delta_1 \in R_1$ with $\mathbf{0} \leq v_1 + \delta_1$,

$$\mathcal{F} : \frac{\begin{matrix}\vdots\, \mathcal{F}_1 \\ v_1\end{matrix}}{v}\ \delta_1$$

is a truncated derivation, where $v = \mathsf{under}(v_1 + \delta_1, i)$.

T3: If $\mathcal{F}_1$ and $\mathcal{F}_2$ are truncated derivations with derived truncated vectors $v_1, v_2 \in \mathbb{N}^k$ respectively, then for each binary rule $\delta_2 \in R_2$ with $\mathbf{0} \leq v_1 + v_2 + \delta_2$,

$$\mathcal{F} : \frac{\begin{matrix}\vdots\, \mathcal{F}_1 \\ v_1\end{matrix} \qquad \begin{matrix}\vdots\, \mathcal{F}_2 \\ v_2\end{matrix}}{v}\ \delta_2$$

is a truncated derivation, where $v = \mathsf{under}(v_1 + v_2 + \delta_2, i)$.

Analogously to derivations, a truncated derivation $\mathcal{F}$ is a finite binary tree whose nodes are labelled by truncated vectors. We say $\mathcal{F}$ *derives a truncated vector* v iff $\mathcal{F}(\varepsilon) = v$. We naturally extend the notions of *compactness*, *covering witness*, and *coverability* to truncated derivations w.r.t. $\leq$.

Lemma 1. *Let $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ be a BVAS and $i \in \mathbb{N}$. For any $h \in \mathbb{N}^+$, there are finitely many truncated derivations of a BVAS of height h.*

Given a BVAS $\mathcal{B}$, we define a total ordering $\sqsubseteq$ on truncated derivations according to their heights as follows. Since for each $h \in \mathbb{N}^+$ there are only finitely many, say k_h, truncated derivations of height h, we can enumerate them without repetition, arbitrarily as $\mathcal{F}_{h1}, \ldots, \mathcal{F}_{hk_h}$. We define $\mathcal{F}_{mi} \sqsubseteq \mathcal{F}_{nj}$ iff $m < n$, or $m = n$ and $i \leq j$.

The Forest $\mathsf{Under}(\mathcal{B}, C_i)$. Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and $i \in \mathbb{N}$, we construct a forest $\mathsf{Under}(\mathcal{B}, C_i)$ whose nodes are compact truncated derivations by the following rules:

U1: For each axiom $v \in A$, the truncated derivation $\overline{v}$ is a root.

U2: Let $\mathcal{F}_1$ be a compact truncated derivation in the forest. Let $\mathcal{F}$ be a truncated derivation obtained by applying a unary rule $\delta_1 \in R_1$ to $\mathcal{F}_1$ (as in rule T2). If $\mathsf{compact}(\mathcal{F})$ has not been added to the forest then add $\mathsf{compact}(\mathcal{F})$ as a child of $\mathcal{F}_1$ in the forest.

U3: Suppose compact truncated derivations $\mathcal{F}_1, \mathcal{F}_2$ are in the forest. Let $\mathcal{F}$ be a truncated derivation obtained by applying a binary rule $\delta_2 \in R_2$ to $\mathcal{F}_1$ and $\mathcal{F}_2$ (as in rule T3). If $\mathsf{compact}(\mathcal{F})$ has not been added to the forest then we add $\mathsf{compact}(\mathcal{F})$ to the forest as a child of $\mathcal{F}'$ where $\mathcal{F}'$ is the greater one between $\mathcal{F}_1$ and $\mathcal{F}_2$ w.r.t. the total order $\sqsubseteq$.

The following lemma shows that the construction of $\mathsf{Under}(\mathcal{B}, C_i)$ eventually terminates, and that it can be used to prove coverability.

Theorem 1 (Underapproximation). *Let $\mathcal{B}$ be a BVAS.*

1. *For any $i \in \mathbb{N}$, the forest $\mathsf{Under}(\mathcal{B}, C_i)$ is finite.*
2. *Given an $i \in \mathbb{N}$, for any truncated derivation $\mathcal{F}$, there is a derivation $\mathcal{D}$ in $\mathcal{B}$ such that $\mathcal{F}(\varepsilon) \leq \mathcal{D}(\varepsilon)$.*
3. *For any vector $v \in \mathbb{N}^k$, we have $v \in Reach(\mathcal{B})\downarrow$ iff there exists $i \in \mathbb{N}$ such that there is a truncated derivation $\mathcal{F}$ in $\mathsf{Under}(\mathcal{B}, C_i)$ with $v \leq \mathcal{F}(\varepsilon)$.*

Proof. Part (1). Fix i. It is easy to see that there are finitely many trees in the forest and each tree is finitely branching, since there are at most finitely many trees of a given height. If the forest is not finite, then by König's lemma, there is an infinite simple path of compact truncated derivations $\mathcal{F}_1, \mathcal{F}_2, \ldots$ in the forest such that for every $i \geq 1$, $\mathcal{F}_i$ is a sub-compact truncated derivation of $\mathcal{F}_{i+1}$. This induces an infinite sequence of truncated vectors $\mathcal{F}_1(\varepsilon), \mathcal{F}_2(\varepsilon) \ldots$ such that for every $i \neq j$, $\mathcal{F}_i(\varepsilon) \neq \mathcal{F}_j(\varepsilon)$. However, since for all $\mathcal{F}$ in the forest, $\mathcal{F}(\varepsilon) \in C_i$ and C_i is finite, such infinite sequence of truncated vectors does not exist.

Part (2). By induction on the height of $\mathcal{F}$.

Part (3). $\Rightarrow$: Since $\mathit{Reach}(\mathcal{B}) \cap v\uparrow \neq \emptyset$, there is a derivation $\mathcal{D}$ in $\mathcal{B}$ such that $v \leq \mathcal{D}(\varepsilon)$. Let S be the union of the set of axioms A and the set of all vectors in $\mathsf{compact}(\mathcal{D})$. Because both sets are finite, let i be $\max(S)$. Then $\mathsf{compact}(\mathcal{D})$ is in $\mathsf{Under}(\mathcal{B}, C_i)$ and $v \leq \mathcal{D}(\varepsilon) = \mathsf{compact}(\mathcal{D})(\varepsilon)$.

$\Leftarrow$: By Part (2), there is a derivation $\mathcal{D}$ in $\mathcal{B}$ such that $\mathcal{F}(\varepsilon) \leq \mathcal{D}(\varepsilon)$. Since $\mathcal{D}(\varepsilon) \in \mathit{Reach}(\mathcal{B})$ and $v \leq \mathcal{F}(\varepsilon)$, $v \in \mathit{Reach}(\mathcal{B})\downarrow$. ∎

3.2 Overapproximation

To define over-approximation of derivations, we introduce *extended derivations* which consider vectors over $\mathbb{N} \cup \{\infty\}$. We then present an algorithm that builds a forest overapproximating the downward closure of reachable vectors of a given BVAS and prove termination and correctness.

Let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ be the extension of the natural numbers with infinity. An *extended vector* is an element of $\mathbb{N}_\infty^k$. For extended vectors $u, u' \in \mathbb{N}_\infty^k$, we write $u \leq_e u'$ iff for all $i \in [1, k]$, we have $u[i] \leq u'[i]$ or $u'[i] = \infty$. We write $u <_e u'$ iff $u \leq_e u'$ and $u' \not\leq_e u$. We always use words starting with the letter u to denote an extended vector (e.g. u, u', u_1 etc.) and words starting with the letter v to denote a vector in $\mathbb{Z}^k$ (e.g. v, v', v_1 etc.). Extended vectors describe sets of vectors: we define $\gamma : \mathbb{N}_\infty^k \to 2^{\mathbb{N}^k}$ as $\gamma(u) = \{v \in \mathbb{N}^k \mid v \leq_e u\}$, and naturally extend γ to sets of extended vectors.

Proposition 1. *[7] (1) Given an extended vector $u \in \mathbb{N}_\infty^k$ and a finite set of extended vectors $S \subseteq \mathbb{N}_\infty^k$, $\gamma(u) \subseteq \gamma(S)$ iff there is $u' \in S$ such that $u \leq_e u'$. (2) Given two finite and minimal sets $S_1, S_2 \subseteq \mathbb{N}_\infty^k$, $S_1 = S_2$ if and only if $\gamma(S_1) = \gamma(S_2)$.*

Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$, there exists a finite and minimal subset $\mathsf{CS}(\mathcal{B}) \subseteq \mathbb{N}_\infty^k$ such that $\gamma(\mathsf{CS}(\mathcal{B})) = \mathit{Reach}(\mathcal{B})\downarrow$. We shall call $\mathsf{CS}(\mathcal{B})$ the *finite representation* of $\mathit{Reach}(\mathcal{B})\downarrow$.

Extended Derivations. Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and an $i \in \mathbb{N}$, let $C_i = \{0, \ldots, i\}^k \cup A$ and $L_i = \{0, \ldots, i, \infty\}^k \setminus \{0, \ldots, i\}^k$. Given two sets $S_1 \subseteq \mathbb{N}^k$ and $S_2 \subseteq \mathbb{N}_\infty^k$, we say that S_2 is an *overapproximation* of S_1 iff $S_1 \subseteq \gamma(S_2)$. Moreover, we say that S_2 is *the most precise overapproximation* of S_1 in $L_i \cup C_i$ iff there is no finite and minimal subset $S \subseteq L_i \cup C_i$ such that $S_1 \subseteq \gamma(S) \subset \gamma(S_2)$. In the following, in case S_2 is a singleton set $\{u\}$, we write that u is (the most precise) overapproximation of S_1 for simplicity.

Given an extended vector $u \in \mathbb{N}_\infty^k$ and an $i \in \mathbb{N}$, We write $\mathsf{over}(u, i)$ for the extended vector such that for all $j \in [1, k]$, $\mathsf{over}(u, i)[j] = u[j]$ if $u[j] \leq i$, $\mathsf{over}(u, i)[j] = \infty$ otherwise. Note that $\mathsf{over}(u, i)$ is an overapproximation of $\gamma(u)$, and interestingly, is the most precise overapproximation of $\gamma(u)$ in $L_i \cup C_i$ [7].

We can naturally extend the addition of vectors to the addition of extended vectors by assuming that $\infty + \infty = \infty$ and $\infty + c = \infty$ for all $c \in \mathbb{Z}$.

Given a BVAS $\mathcal{B} = (k, A, R_1, R_2)$ and $i \in \mathbb{N}$, an *extended derivation* $\mathcal{E}$ is defined inductively as follows.

E1: If $v \in A$, then $\overline{v}$ is an extended derivation.
E2: If $\mathcal{E}_1$ is an extended derivation with a derived extended vector $u_1 \in \mathbb{N}_\infty^k$, then for each unary rule $\delta_1 \in R_1$ with $\mathbf{0} \leq_e u_1 + \delta_1$,

$$\mathcal{E} : \cfrac{\begin{matrix}\vdots\, \mathcal{E}_1 \\ u_1\end{matrix}}{u}\, \delta_1$$

is an extended derivation, where $u = \mathsf{over}(u_1 + \delta_1, i)$.
E3: If $\mathcal{E}_1$ and $\mathcal{E}_2$ are extended derivations with derived extended vectors $u_1, u_2 \in \mathbb{N}_\infty^k$ respectively, then for each binary rule $\delta_2 \in R_2$ with $\mathbf{0} \leq_e u_1 + u_2 + \delta_2$,

$$\mathcal{E} : \cfrac{\begin{matrix}\vdots\, \mathcal{E}_1 \\ u_1\end{matrix} \qquad \begin{matrix}\vdots\, \mathcal{E}_2 \\ u_2\end{matrix}}{u}\, \delta_2$$

is an extended derivation, where $u = \mathsf{over}(u_1 + u_2 + \delta_2, i)$.

Analogously to derivations, an extended derivation $\mathcal{E}$ is a finite binary tree whose nodes are labelled by extended vectors. For an extended derivation $\mathcal{E}$ and its node n, we write $\mathcal{E}(n)$ for the extended vector labelled at n. We say $\mathcal{E}$ *derives an extended vector* u iff $\mathcal{E}(\varepsilon) = u$. We naturally extend the notions of *compactness*, *covering witness*, and *coverability* to extended derivations w.r.t. $\leq_e$. Similar to derivations, the following lemma shows that there are finitely many extended derivations of a given height.

Lemma 2. *Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and $i \in \mathbb{N}$, for each $h \in \mathbb{N}^+$, there are finitely many extended derivations of height h.*

Given a BVAS $\mathcal{B}$, we define a total ordering $\sqsubseteq_e$ on extended derivations according to their heights. Since for each $h \in \mathbb{N}^+$ there are only finitely many, say k_h, extended derivations of height h, we can enumerate them without repetition, arbitrarily as $\mathcal{E}_{h1}, \ldots, \mathcal{E}_{hk_h}$. We define $\mathcal{E}_{mi} \sqsubseteq_e \mathcal{E}_{nj}$ iff $m < n$, or $m = n$ and $i \leq j$.

The Forest $\mathsf{Over}(\mathcal{B}, L_i, C_i)$. Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and an $i \in \mathbb{N}$, we construct a forest $\mathsf{Over}(\mathcal{B}, L_i, C_i)$ whose nodes are compact extended derivations by following the rules below.

O1: For each axiom $v \in A$, the extended derivation $\overline{v}$ is a root.
O2: If a compact extended derivation $\mathcal{E}_1$ is already in the forest and $\mathsf{compact}(\mathcal{E})$ has not been added in the forest where $\mathcal{E}$ is computed by applying a unary rule to $\mathcal{E}_1$ as in Rule E2, then add $\mathsf{compact}(\mathcal{E})$ as a child of $\mathcal{E}_1$ in the forest.
O3: If compact extended derivations $\mathcal{E}_1, \mathcal{E}_2$ are already in the forest and $\mathsf{compact}(\mathcal{E})$ has not been added in the forest where $\mathcal{E}$ is computed by applying a binary rule to $\mathcal{E}_1$ and $\mathcal{E}_2$ as in Rule E3, then we add $\mathsf{compact}(\mathcal{E})$ to the forest as a child of $\mathcal{E}'$ where $\mathcal{E}'$ is the greater one between $\mathcal{E}_1$ and $\mathcal{E}_2$ w.r.t. the total order $\sqsubseteq_e$.

Algorithm 1. EEC Algorithm to decide the coverability problem of BVAS.

```
Input: A BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and a vector $t \in \mathbb{N}^k$.
Output: “Cover” if $t$ is coverable in $\mathcal{B}$, “Uncover” otherwise.
begin
    $i \longleftarrow 0$
    while true do
        Compute $\mathsf{Under}(\mathcal{B}, C_i)$                // Expand
        Compute $\mathsf{Over}(\mathcal{B}, L_i, C_i)$            // Enlarge
        // Check
        if $\exists \mathcal{F} \in \mathsf{Under}(\mathcal{B}, C_i).\ t \leq \mathcal{F}(\varepsilon)$ then
            return “Cover”
        else if $\forall \mathcal{E} \in \mathsf{Over}(\mathcal{B}, L_i, C_i).\ t \not\leq_e \mathcal{E}(\varepsilon)$ then
            return “Uncover”
        $i \longleftarrow i + 1$
```

Theorem 2 (Overapproximation). *Let $\mathcal{B}$ be a BVAS.*

1. *For each $i \in \mathbb{N}$, the forest $\mathsf{Over}(\mathcal{B}, L_i, C_i)$ is finite.*
2. *Given $i \in \mathbb{N}$, for any derivation $\mathcal{D}$, there is a compact extended derivation $\mathcal{E}$ in $\mathsf{Over}(\mathcal{B}, L_i, C_i)$ with $\mathcal{D}(\varepsilon) \leq_e \mathcal{E}(\varepsilon)$.*
3. *For $v \in \mathbb{N}^k$, $Reach(\mathcal{B}) \cap v\uparrow = \emptyset$ iff there exists an $i \in \mathbb{N}$ such that for any compact extended derivation $\mathcal{E}$ in $\mathsf{Over}(\mathcal{B}, L_i, C_i)$, we have $\gamma(\mathcal{E}(\varepsilon)) \cap v\uparrow = \emptyset$.*

Proof. The proof of Part (1) is similar to the proof of Theorem 1(1), because $L_i \cup C_i$ is finite.

The proof of Part (2) is by induction on the height of $\mathcal{D}$.

Part (3). $\Leftarrow$: Suppose $Reach(\mathcal{B}) \cap v\uparrow \neq \emptyset$. Then there is a derivation $\mathcal{D}$ in $\mathcal{B}$ such that $\mathcal{D}(\varepsilon) \in v\uparrow$. Using Part (2), we can find $\mathcal{E}$ in $\mathsf{Over}(\mathcal{B}, L_i, C_i)$ such that $\mathcal{D}(\varepsilon) \leq_e \mathcal{E}(\varepsilon)$. For $\mathcal{E}$, we have $\mathcal{D}(\varepsilon) \in \gamma(\mathcal{E}(\varepsilon))$ and thus $\gamma(\mathcal{E}(\varepsilon)) \cap v\uparrow \neq \emptyset$.

$\Rightarrow$: Since $Reach(\mathcal{B}) \cap v\uparrow = \emptyset$ iff $Reach(\mathcal{B})\downarrow \cap v\uparrow = \emptyset$, $\gamma(\mathsf{CS}(\mathcal{B})) \cap v\uparrow = \emptyset$. Take $i \in \mathbb{N}$ such that $\mathsf{CS}(\mathcal{B}) \subseteq L_i \cup C_i$. For every extended derivation $\mathcal{E}$ in $\mathcal{B}$, we have $\gamma(\mathcal{E}(\varepsilon)) \subseteq \gamma(\mathsf{CS}(\mathcal{B}))$. This can be proved by induction on the height of $\mathcal{E}$.

For every compact extended derivation $\mathcal{E}$ in $\mathsf{Over}(\mathcal{B}, L_i, C_i)$, we therefore have that $\gamma(\mathcal{E}(\varepsilon)) \subseteq \gamma(\mathsf{CS}(\mathcal{B}))$. Hence $\gamma(\mathcal{E}(\varepsilon)) \cap v\uparrow = \emptyset$. ∎

3.3 EEC Algorithm

Algorithm 1 shows the schematic of the EEC algorithm. It takes as input a BVAS $\mathcal{B}$ and a target vector t. It uses an abstraction parameter i, initially 0, and defines the family of abstractions C_i and L_i. It iteratively computes the under-approximation Under and over-approximation Over w.r.t. i. If the under-approximation covers t, it returns “Cover”; if the over-approximation shows t cannot be covered, it returns “Uncover.” Otherwise, it increments i and loops again. From Theorems 1 and 2, we conclude that this algorithm eventually terminates with the correct result.

We briefly remark on two optimizations. First, instead of explicitly keeping forests of derivations in Over and Under, we can only maintain the vectors that label the roots of the derivations. The structure of the forest was required to prove termination in [16], but can be reconstructed using only the vectors and the timestamps at which the vectors were added. Second, in Under (resp. Over), we can only keep maximal vectors (resp. extended vectors): if two vectors $v_1 \le v_2$ (resp. extended vectors $u_1 \le_e u_2$), we can omit v_1 (resp. u_1) and only keep v_2 (resp. u_2). Indeed, if $t \le v_1$ in Under, we also have $t \le v_2$, and so the cover check succeeds in the EEC algorithm. Further, if $t \not\le_e u_2$ in Over, we have $t \not\le_e u_1$, and so the uncover check succeeds as well. We thank Sylvain Schmitz for these observations.

4 Complexity Analysis

We now give an upper bound on the number of iterations of the EEC algorithm. Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2\rangle$ and a derivation $\mathcal{D}$, for each internal node n, we write $\delta(n) \in \mathbb{Z}^k$ for the rule $\delta \in R_1 \cup R_2$ that is applied to derive $\mathcal{D}(n)$. We extend this notation to truncated and extended derivations as well. Given a derivation $\mathcal{D}$ and an $i \in \mathbb{N}^k$, we define a truncated derivation $\mathsf{under}(\mathcal{D}, i)$ inductively as follows:

1. If n is a leaf, then $\mathsf{under}(\mathcal{D}, i)(n) = \mathcal{D}(n)$.
2. If n has a child n' and $\mathcal{D}(n) = \mathcal{D}(n') + \delta(n)$, then $\mathsf{under}(\mathcal{D}, i)(n) = \mathsf{under}(\mathsf{under}(\mathcal{D}, i)(n') + \delta(n), i)$.
3. If n has two children n', n'' and $\mathcal{D}(n) = \mathcal{D}(n') + \mathcal{D}(n'') + \delta(n)$, then $\mathsf{under}(\mathcal{D}, i)(n) = \mathsf{under}(\mathsf{under}(\mathcal{D}, i)(n') + \mathsf{under}(\mathcal{D}, i)(n'') + \delta(n), i)$.

We can also define an extended derivation $\mathsf{over}(\mathcal{D}, i)$ inductively by following the above rules except that we replace all $\mathsf{under}(\square, i)$ by $\mathsf{over}(\square, i)$.

We start with some intuition in the special case of vector addition systems. A *vector addition system* (VAS) $\mathcal{V}$ is a BVAS $\langle k, \{a\}, R, \emptyset\rangle$. For simplicity, we write a VAS as just $\langle k, a, R\rangle$. Note that a derivation $\mathcal{D}$ of a VAS $\mathcal{V}$ is degenerated to a sequence of non-negative vectors. In the following, we say the *length* of $\mathcal{D}$ instead of the height of $\mathcal{D}$ for convenience in the VAS context. For VAS, Rackoff [15] proved the coverability problem is EXPSPACE-complete by showing that if a covering witness (derivation) exists, then there must exist one whose length h is at most doubly exponential in the size of the VAS $\mathcal{V}$ and the target vector t. Further, there is a derivation of length at most h in which the maximum constant is bounded by $i := h \cdot \mathit{size}(\mathcal{V}) + \max(t)$. This is because in h steps, a vector can decrease at most $h \cdot \mathit{size}(\mathcal{V})$, so if any co-ordinate goes over i, it remains higher than $\max(t)$ after executing the path. By the same argument, if there is an extended derivation of length at most h and constant i covering t, then we can find a derivation for t.

If t is coverable, using the above argument and Theorem 1, we see that $\mathsf{Under}(\mathcal{V}, C_i)$ will contain a covering witness of t. If t is not coverable, then the above argument shows that all extended derivations of $\mathsf{Over}(\mathcal{V}, L_i, C_i)$ of length at most h will not cover t. However, there may be longer extended derivations in $\mathsf{Over}(\mathcal{V}, L_i, C_i)$. For these, we can show that $\mathsf{Over}(\mathcal{V}, L_i, C_i)$ also contains a contraction of that extended derivation of length at most h. In both cases, EEC terminates in i iterations, which is doubly exponential in the size of the input.

We now show the bound for BVAS. The following lemma is the key observation in the optimal algorithm of [5].

Lemma 3. *[5] Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and a vector $t \in \mathbb{N}^k$, if t is coverable in $\mathcal{B}$, then there is a covering witness (derivation) $\mathcal{D}$ whose height is at most* $(\max((R_1 \cup R_2)^-) + \max(t) + 2)^{(3k)!}$.

Moreover, the following lemma shows that the maximum constant appearing in a height-bounded derivation can remain polynomial in the height.

Lemma 4. *Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$, a vector $t \in \mathbb{N}^k$ and a derivation $\mathcal{D}$ whose height is at most h, for any bound $i \geq h \cdot \max((R_1 \cup R_2)^-) + \max(t)$, $\mathcal{D}$ is a covering witness of t iff $\mathsf{under}(\mathcal{D}, i)$ is a covering witness of t.*

Proof. Fix an i such that $i \geq h \cdot \max((R_1 \cup R_2)^-) + \max(t)$.

$\Leftarrow$: It holds by Theorem 1.

$\Rightarrow$: Given a derivation $\mathcal{D}$, we say that an index j is *marked* iff during the construction of $\mathsf{under}(\mathcal{D}, i)$, there is a vector v, which is computed after applying a rule and before comparing to i, such that $v[j] > i$.

Given a derivation $\mathcal{D}$, during the construction of $\mathsf{under}(\mathcal{D}, i)$, for each index $j \in [1, k]$, we check the following: If j is marked, then there is a node n such that $\mathsf{under}(\mathcal{D}, i)(n)[j] = i$. Since $\mathit{height}(\mathsf{under}(\mathcal{D}, i)) = \mathit{height}(\mathcal{D}) \leq h$, we know that the length of the path from n to the root ε is at most h. Hence $\mathsf{under}(\mathcal{D}, i)(\varepsilon)[j] \geq \mathsf{under}(\mathcal{D}, i)(n)[j] - h \cdot \max((R_1 \cup R_2)^-) = i - h \cdot \max((R_1 \cup R_2)^-) \geq \max(t) \geq t[j]$. On the other hand, if j is not marked, we have that for all node n, $\mathsf{under}(\mathcal{D}, i)(n)[j] = \mathcal{D}(n)[j]$. Hence $\mathsf{under}(\mathcal{D}, i)(\varepsilon)[j] = \mathcal{D}(\varepsilon)[j] \geq t[j]$. Hence $\mathsf{under}(\mathcal{D}, i)$ is a covering witness of t. ∎

We now prove the case where the target vector t is coverable. We show that $\mathsf{Under}(\mathcal{B}, C_i)$ contains a truncated derivation covering t, where i is bounded by a doubly exponential function of the input.

Lemma 5. *Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2 \rangle$ and a vector $t \in \mathbb{N}^k$, if t is coverable in $\mathcal{B}$, then there exists $\mathcal{F} \in \mathsf{Under}(\mathcal{B}, C_i)$ such that $t \leq \mathcal{F}(\varepsilon)$ for some $i = 2^{2^{O(n \log n)}}$, where $n = \mathit{size}(\mathcal{B}) + \mathit{size}(t)$.*

Proof. Let h be the bound from Lemma 3. Clearly, $h = 2^{2^{O(nlogn)}}$. Pick $i = h^2$. By Lemma 3, there is a derivation $\mathcal{D}$ that covers t and whose height is at most h. Since $i = h^2 \geq h \cdot \max((R_1 \cup R_2)^-) + \max(t)$, by Lemma 4, there is a truncated derivation $\mathsf{under}(\mathcal{D}, i)$ that covers t. Moreover, $\mathsf{compact}(\mathsf{under}(\mathcal{D}, i))$ is in $\mathsf{Under}(\mathcal{B}, C_i)$. ∎

Assume now that the target vector $t \in \mathbb{N}^k$ is not coverable. Lemma 6, from [5], connects derivations of "small" height to extended derivations for high enough constants. Lemma 7 shows that extended derivations of "large" height can be contracted. The proof of this lemma mimicks the proof for (ordinary) derivations.

Lemma 6. *[5] Given a BVAS $\langle k, A, R_1, R_2\rangle$, a vector $t \in \mathbb{N}^k$, and a derivation $\mathcal{D}$ whose height is at most h, for any bound $i \geq h \cdot \max((R_1 \cup R_2)^-) + \max(t)$, $\mathcal{D}$ is a covering witness of t iff $\mathsf{over}(\mathcal{D}, i)$ is a covering witness of t.*

Lemma 7. *Let $\mathcal{B} = \langle k, A, R_1, R_2\rangle$ be a BVAS and $i \in \mathbb{N}$. If there is an extended derivation $\mathcal{E}$ that covers $t \in \mathbb{N}^k$, then there is a contraction of $\mathcal{E}$ whose height is at most $(\max((R_1 \cup R_2)^-) + \max(t) + 2)^{(3k)!}$.*

Finally, we prove that if t is not coverable, then $\mathsf{Over}(\mathcal{B}, L_i, C_i)$ does not find an extended derivation covering t, for i as above.

Lemma 8. *Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2\rangle$ and $t \in \mathbb{N}^k$, there is an $i = 2^{2^{O(n \log n)}}$, where $n = size(\mathcal{B}) + size(t)$, such that if t is not coverable in $\mathcal{B}$, then for all extended derivations $\mathcal{E} \in \mathsf{Over}(\mathcal{B}, L_i, C_i)$, we have $\mathcal{E}$ does not cover t.*

Proof. Suppose not. Then there is an $\mathcal{E} \in \mathsf{Over}(\mathcal{B}, L_i, C_i)$ so that $\mathcal{E}$ covers t. Let h be the bound from Lemma 7, and let $i = h^2$. We consider two cases: (1) The height of $\mathcal{E}$ is at most h. Then since $i = h^2 \geq h \cdot \max((R_1 \cup R_2)^-) + \max(t)$, by Lemma 6, t is coverable in $\mathcal{B}$. Contradiction. (2) The height of $\mathcal{E}$ is greater than h. By Lemma 7, there is a contraction of $\mathcal{E}$ that covers t and whose height is at most h. Following the arguments in case (1), we again get a contradiction. ∎

Our main theorem follows from Lemmas 5 and 8.

Theorem 3. *Given a BVAS $\mathcal{B} = \langle k, A, R_1, R_2\rangle$ and a vector $t \in \mathbb{N}^k$, the EEC algorithm terminates in $2^{2^{O(n \log n)}}$ iterations, where $n = size(\mathcal{B}) + size(t)$.*

The bound on the number of iterations also provides a bound on the overall asymptotic complexity of the algorithm. For BVAS, each iteration of the EEC algorithm performs two instances of AND-OR reachability to perform the cover and the uncover checks. Moreover, the size of the graph is at most doubly exponential in the size of the BVAS, since the finite component of each vector is bounded by a doubly exponential function of the input. Since AND-OR reachability can be performed in time linear in the size of the graph, this gives a 2EXPTIME algorithm. For VAS, each iteration of the EEC algorithm performs two instances of reachability to perform the checks. Thus, if reachability is implemented in a space optimal (NLOGSPACE) way, we get an EXPSPACE upper bound. (In practice, reachability is implemented using a linear time algorithm, which leads to a 2EXPTIME upper bound.)

5 Implementation and Evaluation

We have implemented the EEC algorithm for BVAS and used our implementation to model check safety properties of single-wait recursively parallel programs [1]. Our programs are written in the syntax of [1], and we assume all program variables range over finite domains. In the following, we briefly recall recursively parallel programs with a *wait* construct. We then describe the performance of our tool on a web server example.

Recursively Parallel Programs. Single-wait recursively parallel programs (see [1] for details) are a concurrent programming model in which computations are hierarchically organized into isolated parallelly executing tasks. Each task executes sequentially, and maintains *regions* of *handles* to other tasks. A task t can *post* subtasks and store their handles in one of its regions. A subtask posted by t executes in parallel with the task t. A task can create unboundedly many subtasks in a single region and each subtask may also recursively create additional parallel tasks, storing their handles in its own regions. At some later point, when the task t requires the results computed by its subtasks, t has to wait until a posted subtask completes. A wait is implemented with an "ewait" construct: when task t executes $ewait(r)$ for a region r, its execution is suspended until *some* task whose handle is stored in r completes execution. The return value of the completed subtask is combined with the current state of t via a programmer-supplied *return-value handler*, and t continues executing from this point with the updated state.

The *state reachability problem* for a recursively parallel program P, is to determine, given an initial valuation l_0 of variables of the program P and a valuation l, if there is an execution of P such that the valuation l is reachable.

Single-wait programs capture many constructs in modern structural parallel programming languages, such as pruning in Orc [12], futures in Multilisp [8], task-parallel libraries [3,14], asynchronous programs [9,6], etc. The state reachability problem for single-wait programs is equivalent to the coverability problem for BVAS [1].[1]

Web Server Case Study. We implemented the EEC algorithm for BVAS and a reduction from recursively parallel programs with ewait to BVAS. We used our implementation to verify safety properties of a model of a web server taken from [4]. Our model executes asynchronous calls to serve requests, and waits on multiple concurrent requests to implement DNS lookup.

We model a server main that helps clients upload files. Given a provider's domain name and files, the server main simulates arbitrarily many client requests by posting unboundedly many dns requests. The server waits on the first DNS server to return. The DNS server returns the provider's real IP address, which is stored in a variable x.

Each task dns waits for DNS lookup requests, and returns either an *ip* if the lookup succeeds, or not_found if the provided domain name is not valid. If the domain name is valid, dns then either returns *ip* if it already has an IP address for the requested domain name by looking up its local database, or posts a task server_i to ask other servers to resolve an IP address. Before asking a remote server server_i, dns first allocates a buffer buf that is used for the communication between itself and server_i, in particular, used for storing an *ip* returned from server_i. When dns receives an *ip* stored in the buffer buf from some remote server, it returns the contents of buf. dns may ask multiple remote servers at the same time, and accept

[1] The authors of [1] state that the 2EXPTIME algorithm of Demri et al. will be hard to implement. They propose an alternate algorithm, without any upper bound on the running time. However, that algorithm has not been implemented either.

Table 1. Results of the DNS example

#server	#dimension	#axiom	#urule	#brule	#iter	result	time
2	19	20	9153	6950	1	Uncover	31.25s
3	22	23	14832	11664	1	Uncover	79.16s
4	25	26	22640	18326	1	Uncover	151.46s
5	28	29	33070	27392	1	Uncover	279.56s
6	31	32	46638	39366	1	Uncover	463.71s
6 (buggy)	31	32	40077	32805	1	Cover	63.58s

the very first *ip* returned from some server. We model this scenario precisely by posting all server tasks into a single region and waiting for the first one to return its result (using ewait). A task server_i can either return a *timeout* to model the cases where server_i encounters a problem, or an *ip* when server_i successfully completes a lookup. Since server_i can produce *timeout*, the task dns contains error handling mechanism: if it receives a *timeout*, it de-allocates the buffer buf.

The first property we check is that the value stored in x when task dns returns always equals either an *ip* or a not_found.

If the server main gets the provider's IP address successfully, then it uploads the file on the client's behalf by allocating and using a buffer for the transmission. Since errors may occur during the transmission, the server provides a block of code for error handling: whenever an error happens, the server does some cleaning work such as de-allocating the buffer. Once the transmission is over, the server main receives an acknowledgement in the buffer and returns this buffer to the client, informing the client that the file has been successfully uploaded. The second property we check is if the buffer is always de-allocated properly after the error handling block completes.

Since any task can create subtasks and moreover the task main creates unboundedly many subtasks, by the classification from [1], this example falls into the general case of single-wait programs which are equivalent to BVAS.

Results. We have run our tool to verify the two properties above. All experiments were performed on a 2 core Intel Xeon X5650 CPU machine with 64GB memory and 64bit Linux (Debian/Lenny). Table 1 lists the analysis results of both cases. We report: (1) the size of the generated BVAS (the dimension, the number of axioms, unary rules, and binary rules), (2) the number of iterations i for EEC, and (3) the answer, "Cover" or "Uncover," and the execution time.

We modeled a bug found in [4] where the task main can receive a value in the variable x that is neither *ip* nor not_found from dns. In the buggy version, in the error handling code of the task dns, the programmer forgot to return to its caller on an error after de-allocating a buffer. Therefore, after the de-allocation of the buffer, the dns lookup continued to execute and returned the value of the buffer (in the normal case, the buffer contains an *ip*). After we added an immediate return to dns, our tool proved the model is correct.

To explore the scalability of our implementation, we increased the number of remote servers (as shown in the first column of the table) which a task dns posts to a single region in parallel. The "Uncover" instances in Table 1 use the corrected version of dns and increase the number of servers from 2 to 6. As the size of the BVASs becomes larger, the EEC algorithm takes more time to verify instances. However, the largest example, with 31 dimensions, still finishes within a few minutes. We present the run time of the unsafe case when the number of servers is six. For fewer servers, the bug is found quicker. Additionally, for these examples, one iteration of the EEC algorithm (i.e., the counter value is in $\{0, 1, \infty\}$) is sufficient to prove or disprove the property.

References

1. Bouajjani, A., Emmi, M.: Analysis of recursively parallel programs. In: POPL, pp. 203–214 (2012)
2. Bozzelli, L., Ganty, P.: Complexity analysis of the backward coverability algorithm for VASS. In: Delzanno, G., Potapov, I. (eds.) RP 2011. LNCS, vol. 6945, pp. 96–109. Springer, Heidelberg (2011)
3. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. SIGPLAN Not. 40(10), 519–538 (2005)
4. Cunningham, R.: Eel: Tools for debugging, visualization, and verification of event-driven software. Master's thesis, UCLA (2005)
5. Demri, S., Jurdzinski, M., Lachish, O., Lazic, R.: The covering and boundedness problems for branching vector addition systems. J. Comput. Syst. Sci. 79(1), 23–38 (2013)
6. Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. In: ACM Transactions on Programming Languages and Systems (2012)
7. Geeraerts, G., Raskin, J.-F., Begin, L.V.: Expand, enlarge and check: New algorithms for the coverability problem of wsts. J. Comput. Syst. Sci. 72(1), 180–203 (2006)
8. Halstead, R.H.: Multilisp: a language for concurrent symbolic computation. ACM Transactions on Programming Languages and Systems 7, 501–538 (1985)
9. Jhala, R., Majumdar, R.: Interprocedural analysis of asynchronous programs. In: POPL 2007, pp. 339–350. ACM Press (2007)
10. Kaiser, A., Kroening, D., Wahl, T.: Efficient coverability analysis by proof minimization. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 500–515. Springer, Heidelberg (2012)
11. Karp, R., Miller, R.: Parallel program schemata. Journal of Comput. Syst. Sci. 3(2), 147–195 (1969)
12. Kitchin, D., Quark, A., Cook, W., Misra, J.: The Orc programming language. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS 2009. LNCS, vol. 5522, pp. 1–25. Springer, Heidelberg (2009)
13. Kloos, J., Majumdar, R., Niksic, F., Piskac, R.: Incremental, inductive coverability. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 158–173. Springer, Heidelberg (2013)
14. Leijen, D., Schulte, W., Burckhardt, S.: The design of a task parallel library. In: OOPSLA 2009, pp. 227–242. ACM (2009)
15. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theoretical Computer Science 6(2), 223–231 (1978)
16. Verma, K., Goubault-Larrecq, J.: Karp-Miller trees for a branching extension of VASS. Discrete Mathematics & Theoretical Computer Science 7(1), 217–230 (2005)
17. Verma, K.N., Goubault-Larrecq, J.: Alternating two-way AC-tree automata. Inf. Comput. 205(6), 817–869 (2007)