# Incremental, Inductive Coverability

Johannes Kloos, Rupak Majumdar, Filip Niksic, and Ruzica Piskac

MPI-SWS, Kaiserslautern and Saarbrücken

**Abstract.** We give an incremental, inductive (IC3) procedure to check coverability of well-structured transition systems. Our procedure generalizes the IC3 procedure for safety verification that has been successfully applied in finite-state hardware verification to infinite-state well-structured transition systems. We show that our procedure is sound, complete, and terminating for *downward-finite* well-structured transition systems —where each state has a finite number of states below it— a class that contains extensions of Petri nets, broadcast protocols, and lossy channel systems.

We have implemented our algorithm for checking coverability of Petri nets. We describe how the algorithm can be efficiently implemented without the use of SMT solvers. Our experiments on standard Petri net benchmarks show that IC3 is competitive with state-of-the-art implementations for coverability based on symbolic backward analysis or expand-enlarge-and-check algorithms both in time and space usage.

## 1 Introduction

The IC3 algorithm [3] was recently introduced as an efficient technique for safety verification of hardware. It computes an inductive invariant by maintaining a sequence of over-approximations of forward-reachable states, and incrementally strengthening them based on counter-examples to inductiveness. The counter-examples are obtained using a backward exploration from error states. Efficient implementations of the procedure show remarkably good performance on hardware benchmarks [8].

A natural direction is to extend the IC3 algorithm to classes of systems beyond finite-state hardware circuits. Indeed, an IC3-like technique was recently proposed for interpolation-based software verification [5], and the technique was generalized to finite-data pushdown systems, as well as systems using linear real arithmetic, such as timed pushdown automata [15]. It is natural to ask for what other classes of infinite-state systems does IC3 form a decision procedure for safety verification.

In this paper, we consider well-structured transition systems (WSTS) [1,12]. WSTS are infinite-state transition systems whose states have a well-quasi-ordering, and whose transitions satisfy a monotonicity property w.r.t. the quasi-ordering. WSTS capture many important infinite-state models, such as Petri nets and their monotonic extensions [11,4,7,13], broadcast protocols [9,10], and lossy channel systems [2]. A general decidability result shows that the coverability problem (reachability in an upward-closed set) is decidable for WSTS [1]. The decidability result performs a backward reachability analysis, and shows, using properties of well-quasi-orderings, that the reachability procedure must terminate. In many verification problems, techniques based on computing inductive invariants outperform methods based on backward or forward reachability analysis; indeed, IC3 for hardware circuits is a prime example. Thus, it is natural to ask if there is an IC3-style decision procedure for the coverability problem.

We answer this question positively. We give a generalization of IC3 for WSTS, and show that it terminates on the class of *downward-finite* WSTS, in which each state has a finite number of states lower than itself in the well-quasi-ordering. The class of downward-finite WSTS contains most important classes of WSTS used in verification, including Petri nets and their extensions, broadcast protocols, and lossy channel systems. Hence, our results show that IC3 is a decision procedure for the coverability problem for these classes of systems. While termination is trivial in the finite-state case, our technical contribution is to show, using the termination of the backward reachability procedure, that the sequence of (downward-closed) invariants produced by IC3 is guaranteed to converge. We also show that the assumption of downward-finiteness is necessary: we give an example of a general WSTS on which the algorithm does not terminate.

We have implemented our algorithm in a tool called IIC to check coverability in Petri nets. Using combinatorial properties of Petri nets, we derive an optimized implementation of the algorithm that does not use an SMT solver. Our implementation outperforms, both in space and time usage, several other implementations of coverability, such as EEC [13] or backward reachability, on a set of standard Petri net benchmarks.

A full version, including proofs of theorems, is available on arXiv [17].

## 2 Preliminaries

*Well-quasi-orderings.* For a set $X$, a relation $\preceq \subseteq X \times X$ is a *well-quasi-ordering (wqo)* if it is reflexive, transitive, and if for every infinite sequence $x_0, x_1, \ldots$ of elements from $X$, there exists $i < j$ such that $x_i \preceq x_j$. A set $Y \subseteq X$ is *upward-closed* if for every $y \in Y$ and $x \in X$, $y \preceq x$ implies $x \in Y$. Similarly, a set $Y \subseteq X$ is *downward-closed* if for every $y \in Y$ and $x \in X$, $x \preceq y$ implies $x \in Y$. For a set $Y$, we define its upward closure $Y\!\uparrow = \{x \mid \exists y \in Y, y \preceq x\}$. For a singleton $\{x\}$, we simply write $x\!\uparrow$ instead of $\{x\}\!\uparrow$. Similarly, we define $Y\!\downarrow = \{x \mid \exists y \in Y, x \preceq y\}$ for the downward closure of a set $Y$. Clearly, $Y\!\uparrow$ (resp., $Y\!\downarrow$) is an upward-closed set (resp. downward-closed) for each $Y$. The union and intersection of upward-closed sets are upward-closed, and the union and intersection of downward-closed sets are downward-closed. Furthermore, the complement of an upward-closed set is downward-closed, and vice versa. For the convenience of the reader, we will mark upward-closed sets with a small up-arrow superscript, like this: $U^{\uparrow}$, and downward-closed sets with a small down-arrow superscript, like this: $D^{\downarrow}$.

A basis of an upward-closed set $Y$ is a set $Y_b \subseteq Y$ such that $Y = \bigcup_{y \in Y_b} y\!\uparrow$. It is known [14,1,12] that any upward-closed set $Y$ in a wqo has a finite basis: the set of minimal elements of $Y$ has finitely many equivalence classes under the equivalence relation $\preceq \cap \succeq$, so take any system of representatives. We write $\min Y$ for such a system of representatives. Moreover, it is known that any non-decreasing sequence $I_0 \subseteq I_1 \subseteq \dots$ of upward-closed sets eventually stabilizes, i.e., there exists $k \in \mathbb{N}$ such that $I_k = I_{k+1} = I_{k+2} = \dots$.

A wqo $(X, \preceq)$ is *downward-finite* if for each $x \in X$, the downward closure $x\downarrow$ is a finite set.

*Examples.* Let $\mathbb{N}^k$ be the set of $k$-tuples of natural numbers, and let $\preceq$ be point-wise comparison: $v \preceq v'$ if $v_i \leqslant v'_i$ for $i = 1, \dots, k$. Then, $(\mathbb{N}^k, \preceq)$ is a downward-finite wqo [6].

Let $A$ be a finite alphabet, and consider the subword ordering $\preceq$ on words over $A$, given by $w \preceq w'$ for $w, w' \in A^*$ if $w$ results from $w'$ by deleting some occurrences of symbols. Then $(A^*, \preceq)$ is a downward-finite wqo [14].

*Well-structured Transition Systems.* A well-structured transition system (WSTS for short) is a tuple $(\Sigma, I, \rightarrow, \preceq)$ consisting of a set $\Sigma$ of states, a finite set $I \subseteq \Sigma$ of initial states, a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$, and a well-quasi-ordering $\preceq \subseteq \Sigma \times \Sigma$ with the monotonicity property: for all $s_1, s_2, t_1 \in \Sigma$ such that $s_1 \rightarrow s_2$ and $s_1 \preceq t_1$, there exists $t_2$ such that $t_1 \rightarrow t_2$ and $s_2 \preceq t_2$. A WSTS is downward-finite if $(\Sigma, \preceq)$ is downward-finite.

Let $x, y \in \Sigma$. If $x \rightarrow y$, we call $x$ a *predecessor* of $y$, and $y$ a *successor* of $x$. We write $\mathrm{pre}(x) := \{y \mid y \rightarrow x\}$ for the *set of predecessors* of $x$, and $\mathrm{post}(x) := \{y \mid x \rightarrow y\}$ for the *set of successors* of $x$. For $X \subseteq \Sigma$, $\mathrm{pre}(X)$ and $\mathrm{post}(X)$ are defined as natural extensions, i.e., $\mathrm{pre}(X) = \bigcup_{x \in X} \mathrm{pre}(x)$ and $\mathrm{post}(X) = \bigcup_{x \in X} \mathrm{post}(x)$.

We write $x \rightarrow^k y$ if there are states $x_0, \dots, x_k \in \Sigma$ such that $x_0 = x$, $x_k = y$ and $x_i \rightarrow x_{i+1}$ for $0 \leq i < k$. Furthermore, $x \rightarrow^* y$ iff there exists a $k \geqslant 0$ such that $x \rightarrow^k y$, i.e., $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$. We say that there is a *path of length $k$ from $x$ to $y$* if $x \rightarrow^k y$, and that there is a *path from $x$ to $y$* if $x \rightarrow^* y$.

The set of *$k$-reachable* states $\mathsf{Reach}_k$ is the set of states reachable in at most $k$ steps, formally, $\mathsf{Reach}_k := \{y \in \Sigma \mid \exists k' \leqslant k, \exists x \in I, x \rightarrow^{k'} y\}$. The set of *reachable* states $\mathsf{Reach} := \bigcup_{k \geq 0} \mathsf{Reach}_k = \{y \mid \exists x \in I, x \rightarrow^* y\}$. Using downward closure, we can define the *$k$-th cover* $\mathsf{Cover}_k$ and the *cover* $\mathsf{Cover}$ of the WSTS as $\mathsf{Cover}_k := \mathsf{Reach}_k\downarrow$ and $\mathsf{Cover} := \mathsf{Reach}\downarrow$. The *coverability problem for WSTS* asks, given a WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, if every reachable state is contained in $P^{\downarrow}$, i.e., if $\mathsf{Reach} \subseteq P^{\downarrow}$. It is easy to see that this question is equivalent to checking if $\mathsf{Cover} \subseteq P^{\downarrow}$.

In the following, we make some standard effectiveness assumptions on WSTS [1,12]. We assume that $\preceq$ is decidable, and that for any state $x \in \Sigma$, there is a computable procedure that returns a finite basis for $\mathrm{pre}(x\uparrow)$. These assumptions are met by most classes of WSTS considered in verification [12].

Under the preceding effectiveness assumptions, one can show that the coverability problem is decidable for WSTS by a backward-search algorithm [1]. The main construction is the following sequence of upward-closed sets:

$$\mathsf{U}_0^{\uparrow} := \Sigma \setminus P^{\downarrow}, \qquad \mathsf{U}_{i+1}^{\uparrow} := \mathsf{U}_i^{\uparrow} \cup \mathrm{pre}(\mathsf{U}_i^{\uparrow}). \qquad \text{(BackwardReach)}$$

The sequence of sets $\mathsf{U}_i^{\uparrow}$ forms an increasing chain of upward-closed sets, therefore it eventually stabilizes: there is some $L$ such that $\mathsf{U}_L^{\uparrow} = \mathsf{U}_{L+i}^{\uparrow}$ for all $i \geq 0$. Then, $\mathsf{Cover} \subseteq P^{\downarrow}$ iff $I \cap \mathsf{U}_L^{\uparrow} = \emptyset$. Moreover, if $I \cap \mathsf{U}_L^{\uparrow} = \emptyset$, then $\Sigma \setminus \mathsf{U}_L^{\uparrow}$ contains $I$, is contained in $P^{\downarrow}$ and satisfies $\mathrm{post}(\Sigma \setminus \mathsf{U}_L^{\uparrow}) \subseteq \Sigma \setminus \mathsf{U}_L^{\uparrow}$.

We generalize from $\Sigma \setminus \mathsf{U}_L^{\uparrow}$ to the notion of an (inductive) *covering set*. A downward-closed set $C^{\downarrow}$ is called a *covering set* for $P^{\downarrow}$ iff (a) $I \subseteq C^{\downarrow}$, (b) $C^{\downarrow} \subseteq P^{\downarrow}$, and (c) $\mathrm{post}(C^{\downarrow}) \subseteq C^{\downarrow}$. By induction, we have $\mathsf{Cover} \subseteq C^{\downarrow} \subseteq P^{\downarrow}$ for any covering set $C^{\downarrow}$. Therefore, to solve the coverability problem, it is sufficient to exhibit any covering set.

## 3 IC3 for Coverability

We now describe an algorithm for the coverability problem that takes as input a WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, and constructs either a path from some state in $I$ to a state not in $P^{\downarrow}$ (if $\mathsf{Cover} \not\subseteq P^{\downarrow}$), or a covering set for $P^{\downarrow}$. In the algorithm we consider sets that are not necessarily inductive by themselves, but they are *inductive relative to* some other sets. Relative inductivity is a weakening of the regular notion of inductivity. Formally, for a set $R^{\downarrow}$ such that $I \subseteq R^{\downarrow}$, a downward-closed set $S^{\downarrow}$ is inductive relative to $R^{\downarrow}$ if $I \subseteq S^{\downarrow}$ and $\mathrm{post}(R^{\downarrow} \cap S^{\downarrow}) \subseteq S^{\downarrow}$. An upward-closed set $U^{\uparrow}$ is inductive relative to $R^{\downarrow}$ if its downward-closed complement $\Sigma \setminus U^{\uparrow}$ is inductive relative to $R^{\downarrow}$, i.e. if $I \cap U^{\uparrow} = \emptyset$ and $\mathrm{post}(R^{\downarrow} \setminus U^{\uparrow}) \subseteq \Sigma \setminus U^{\uparrow}$.

The condition $\mathrm{post}(R^{\downarrow} \cap S^{\downarrow}) \subseteq S^{\downarrow}$ is equivalent to $\mathrm{pre}(\Sigma \setminus S^{\downarrow}) \cap R^{\downarrow} \cap S^{\downarrow} = \emptyset$. Stated in terms of an upward-closed set $U^{\uparrow}$, the equivalent condition is $\mathrm{pre}(U^{\uparrow}) \cap R^{\downarrow} \setminus U^{\uparrow} = \emptyset$. Since all these conditions are equivalent, we will use them interchangeably.

### 3.1 Algorithm

The core idea of the algorithm is to build a vector $\mathbf{R} = (R_0^{\downarrow}, \ldots, R_N^{\downarrow})$, consisting of sets $R_i^{\downarrow}$ which over-approximate $\mathsf{Cover}_i$. The algorithm ensures that $R_{i+1}^{\downarrow}$ is inductive relative to $R_i^{\downarrow}$ in each step, and that $R_i^{\downarrow} \subseteq P^{\downarrow}$ for $i < N$. This allows us to prove that if the vector stabilizes, we have found an inductive covering set.

The algorithm alternates between trying to find a counter-example and refining the over-approximations. A backward search looks for counter-examples. Whenever a candidate counter-example is revealed to be spurious, the vector $\mathbf{R}$ is refined to exclude this counter-example. In particular, counter-example traces are constructed such that if $a_0 \cdots a_N$ is a counter-example, then $a_i \in R_i^{\downarrow}$. If it can be shown that $a_i \in R_i^{\downarrow}$, but the counter-example cannot be extended backwards beyond $R_i^{\downarrow}$, the set $R_i^{\downarrow}$ is refined. In particular, some state $b$ is found such that removing $b\uparrow$ from $R_0^{\downarrow}, \ldots, R_i^{\downarrow}$ gives a new over-approximation vector that still satisfies all invariants, and $a \in b\uparrow$.

When no progress can be made in refining the current vector or in finding counter-examples, $N$ is increased, and a second strengthening step is carried

$$[\mathsf{Initialize}]\,\frac{}{\mathsf{init} \mapsto I\!\downarrow \mid \varnothing} \qquad [\mathsf{Valid}]\,\frac{R_i^{\downarrow} = R_{i+1}^{\downarrow} \text{ for some } i < N}{\mathbf{R} \mid Q \mapsto \mathsf{valid}}$$

$$[\mathsf{CandidateNondet}]\,\frac{a \in R_N^{\downarrow} \setminus P^{\downarrow}}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \mid \langle a, N\rangle} \qquad [\mathsf{Model}]\,\frac{\min Q = \langle a, 0\rangle}{\mathbf{R} \mid Q \mapsto \mathsf{invalid}}$$

$$[\mathsf{DecideNondet}]\,\frac{\min Q = \langle a, i\rangle \quad i > 0 \quad b \in \mathrm{pre}(a\!\uparrow) \cap R_{i-1}^{\downarrow} \setminus a\!\uparrow}{\mathbf{R} \mid Q \mapsto \mathbf{R} \mid Q.\textsc{Push}(\langle b, i-1\rangle)}$$

$$[\mathsf{Conflict}]\,\frac{\min Q = \langle a, i\rangle \quad i > 0 \quad \mathrm{pre}(a\!\uparrow) \cap R_{i-1}^{\downarrow} \setminus a\!\uparrow = \emptyset \quad b \in \mathrm{Gen}_{i-1}(a)}{\mathbf{R} \mid Q \mapsto \mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus b\!\uparrow]_{k=1}^{i} \mid Q.\textsc{PopMin}}$$

$$[\mathsf{Induction}]\,\frac{R_i^{\downarrow} = \Sigma \setminus \{r_{i,1}, \ldots, r_{i,m}\}\!\uparrow \quad b \in \mathrm{Gen}_i(r_{i,j}) \text{ for some } 1 \le j \le m}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R}[R_k^{\downarrow} \leftarrow R_k^{\downarrow} \setminus b\!\uparrow]_{k=1}^{i+1} \mid \varnothing}$$

$$[\mathsf{Unfold}]\,\frac{R_N^{\downarrow} \subseteq P^{\downarrow}}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \cdot \Sigma \mid \varnothing}$$

**Fig. 1.** The rule system for a IC3-style algorithm for WSTS – generic version. The map $\mathrm{Gen}_i$ is defined in equation (1).

out, in which states in $R_i^{\downarrow}$ that can be proven to be unreachable from $R_{i-1}^{\downarrow}$ are removed. This strengthening step is known as induction.

Figure 1 shows the algorithm as a set of non-deterministic state transition rules, similar to [15]. A state of the computation is either the initial state $\mathsf{init}$, the special terminating states $\mathsf{valid}$ and $\mathsf{invalid}$, or a pair $\mathbf{R} \mid Q$ defined as follows.

The first component of the pair is a vector $\mathbf{R}$ of downward-closed sets, indexed starting from 0. The elements of $\mathbf{R}$ are denoted $R_i^{\downarrow}$. In particular, we denote by $R_0^{\downarrow}$ the downward closure of $I$, i.e., $R_0^{\downarrow} = I\!\downarrow$. These sets contain successive approximations to a potential covering set. The function length gives the length of the vector, disregarding $R_0^{\downarrow}$, i.e., $\mathrm{length}(R_0^{\downarrow}, \ldots, R_N^{\downarrow}) = N$. If it is clear from the context which vector is meant, we often abbreviate $\mathrm{length}(\mathbf{R})$ simply with $N$. We write $\mathbf{R} \cdot X$ for the concatenation of the vector $\mathbf{R}$ with the downward-closed set $X$: $(R_0^{\downarrow}, \ldots, R_N^{\downarrow}) \cdot X = (R_0^{\downarrow}, \ldots, R_N^{\downarrow}, X)$.

The second component of the pair is a priority queue $Q$, containing elements of the form $\langle a, i\rangle$, where $a \in \Sigma$ is a state and $i \in \mathbb{N}$ is a natural number. The priority of the element is given by $i$, and is called the level of the element. The statement $\langle a, i\rangle \in Q$ means that the priority queue contains an element $\langle a, i\rangle$, while with $\min Q$ we denote the minimal element of the priority queue. Furthermore, $Q.\textsc{PopMin}$ yields $Q$ after removal of its minimal element, and $Q.\textsc{Push}(x)$ yields $Q$ after adding element $x$.

The elements of $Q$ are states that lead outside of $P^{\downarrow}$. Let $\langle a, i\rangle$ be an element of $Q$. Either $a$ is a state that is in $R_i$ and outside of $P^{\downarrow}$, or there is a state $b$ leading outside of $P^{\downarrow}$ such that $a \in \mathrm{pre}(b\!\uparrow)$. Our goal is to try to discard those states and show that they are not reachable from the initial states, as $R_i$ denotes an over-approximation of the states reachable in $i$ or less steps. If an element of $Q$ is reachable from the initial states, then $\mathsf{Cover} \not\subseteq P^{\downarrow}$.

The state valid signifies that the search has terminated with $\mathsf{Cover} \subseteq P^{\downarrow}$, while invalid signifies that the algorithm has terminated with $\mathsf{Cover} \not\subseteq P^{\downarrow}$. In the description of the algorithm, we will omit the actual construction of certificates and instead just state that the algorithm terminates with invalid or valid; the calculation of certificates is straightforward.

The transition rules of the algorithm are of the form

$$[\text{Name}]\frac{C_1 \ \cdots \ C_k}{\sigma \mapsto \sigma'} \tag{Rule}$$

and can be read thus: whenever the algorithm is in state $\sigma$ and conditions $C_1$–$C_k$ are fulfilled, the algorithm can apply rule [Name] and transition to state $\sigma'$. We write $\sigma \mapsto \sigma'$ if there is some rule which the algorithm applies to go from $\sigma$ to $\sigma'$. We write $\mapsto^*$ for the reflexive and transitive closure of $\mapsto$.

Let Inv be a predicate on states. We say that a rule *preserves the invariant* Inv if whenever $\sigma$ satisfies Inv and conditions $C_1$ to $C_k$ are met, it also holds that $\sigma'$ satisfies Inv.

Two of the rules use the map $\mathrm{Gen}_i : \Sigma \to 2^{\Sigma}$. It yields those states that are valid generalizations of $a$ relative to some set $R_i^{\downarrow}$. A state $b$ is a generalization of the state $a$ relative to the set $R_i^{\downarrow}$, if $b \preceq a$ and $b\!\uparrow$ is inductive relative to $R_i^{\downarrow}$. Formally,

$$\mathrm{Gen}_i(a) := \{b \mid b \preceq a \wedge b\!\uparrow \cap I = \emptyset \wedge \mathrm{pre}(b\!\uparrow) \cap R_i^{\downarrow} \setminus b\!\uparrow = \emptyset\}\,. \tag{1}$$

Finally, the notation $\mathbf{R}[R_k^{\downarrow} \leftarrow R'^{\downarrow}_k]_{k=1}^{i}$ means that $\mathbf{R}$ is transformed by replacing $R_k^{\downarrow}$ by $R'^{\downarrow}_k$ for each $k = 1, \ldots, i$, i.e.,

$$\mathbf{R}[R_k^{\downarrow} \leftarrow R'^{\downarrow}_k]_{k=1}^{i} = (R_0^{\downarrow}, R'^{\downarrow}_1, \ldots, R'^{\downarrow}_i, R_{i+1}^{\downarrow}, \ldots, R_N^{\downarrow}).$$

We provide an overview of each rule of the calculus.

[Initialize] The algorithm starts by defining the first downward-closed set $R_0^{\downarrow}$ to be the downward closure of the initial state.

[CandidateNondet] If there is a state $a$ such that $a \in R_N^{\downarrow}$, but at the same time it is not an element of $P^{\downarrow}$, we add $\langle a, N\rangle$ to the priority queue $Q$.

[DecideNondet] To check if the elements of $Q$ are spurious counter-examples, we start by processing an element $a$ with the lowest level $i$. If there is an element $b$ in $R_{i-1}^{\downarrow}$ such that $b \in \mathrm{pre}(a\!\uparrow)$, then we add $\langle b, i-1\rangle$ to the priority queue.

[Model] If $Q$ contains a state $a$ from the level 0, then we have found a counter-example trace and the algorithm terminates in the state invalid.

[Conflict] If none of predecessors of a state $a$ from the level $i$ is contained in $R_{i-1}^{\downarrow} \setminus a\!\uparrow$, then $a$ belongs to a spurious counter-example trace. Therefore, we update the downward-closed sets $R_1^{\downarrow}, \ldots, R_i^{\downarrow}$ as follows: since the states in $a\!\uparrow$ are not reachable in $i$ steps, they can be safely removed from all the sets $R_1^{\downarrow}, \ldots, R_i^{\downarrow}$. Moreover, instead of $a\!\uparrow$ we can remove even a bigger set

$b\!\uparrow$, for any state $b$ which is a generalization of the state $a$ relative to $R^{\downarrow}_{i-1}$, as defined in (1). After the update, we remove $\langle a, i\rangle$ from the priority queue. In practice, if $i < N$, we also add $\langle a, i+1\rangle$ to the priority queue. This allows the construction of paths that are longer than $N$ in a given search, which speeds up search significantly (see [3]). It also justifies the use of a priority queue for $Q$, instead of a stack. We omit this modification in our rules to simplify proofs.

[Induction] If for some state $r_{i,j}$ that was previously removed from $R^{\downarrow}_i$, a set $r_{i,j}\!\uparrow$ becomes inductive relative to $R^{\downarrow}_i$ (i.e. $\mathrm{post}(R^{\downarrow}_i \setminus r_{i,j}\!\uparrow) \subseteq \Sigma \setminus r_{i,j}\!\uparrow$), none of the states in $r_{i,j}\!\uparrow$ can be reached in at most $i+1$ steps. Thus, we can safely remove $r_{i,j}\!\uparrow$ from $R^{\downarrow}_{i+1}$ as well. Similarly as in [Conflict], we can even remove $b\!\uparrow$ for any generalization $b \in \mathrm{Gen}_i(r_{i,j})$.

[Valid] If there is a downward-closed set $R^{\downarrow}_i$ such that $R^{\downarrow}_i = R^{\downarrow}_{i+1}$, the algorithm terminates in the state valid.

[Unfold] If the priority queue is empty and all elements of $R^{\downarrow}_N$ are in $P^{\downarrow}$, we start with a construction of the next set $R^{\downarrow}_{N+1}$. Initially, $R^{\downarrow}_{N+1}$ contains all the states, $R^{\downarrow}_{N+1} = \Sigma$, and we append $R^{\downarrow}_{N+1}$ to the vector $\mathbf{R}$.

In an implementation, these rules are usually applied in a specific way. The algorithm generally proceeds in rounds consisting of two phases: backward search and inductive strengthening.

In the backward search phase, the algorithm tries to construct a path $a_0 \cdots a_N$ from $I$ to $\Sigma \setminus P^{\downarrow}$, with $a_i \in R^{\downarrow}_i$. The path is built backwards from $R^{\downarrow}_N$, by first finding a possible end-point using [CandidateNondet]. Next, [DecideNondet] is applied to prolong the path, until $R^{\downarrow}_0$ is reached (as witnessed by [Model]). If the path cannot be prolonged at $a_i$, [Conflict] is applied to refine the sets $R^{\downarrow}_1, \ldots, R^{\downarrow}_i$, removing known-unreachable states (including $a_i$), and the search backtracks one step. If the backward search phase ends unsuccessfully, i.e. no more paths can be constructed, [Unfold] is applied.

In the inductive strengthening phase, [Induction] is repeatedly applied until its pre-conditions fail to hold. After that, it is checked whether [Valid] applies. If not, the algorithm proceeds to the next round.

### 3.2 Soundness

We first show that the algorithm is sound: if it terminates, it produces the right answer. If it terminates in the state invalid, there is a path from an initial state to a state outside of $P^{\downarrow}$, and if it terminates in the state valid, then $\mathsf{Cover} \subseteq P^{\downarrow}$.

We prove soundness by showing that on each state $\mathbf{R} \mid Q$ the following invariants are preserved by the transition rules:

$$I \subseteq R^{\downarrow}_i \qquad \text{for all } 0 \le i \le N \tag{I1}$$

$$\mathrm{post}(R^{\downarrow}_i) \subseteq R^{\downarrow}_{i+1} \qquad \text{for all } 0 \le i < N \tag{I2}$$

$$R^{\downarrow}_i \subseteq R^{\downarrow}_{i+1} \qquad \text{for all } 0 \le i < N \tag{I3}$$

$$R^{\downarrow}_i \subseteq P^{\downarrow} \qquad \text{for all } 0 \le i < N \tag{I4}$$

These properties imply $R_i^{\downarrow} \supseteq \mathsf{Cover}_i$, that is, the region $R_i$ provides an over-approximation of the $i$-cover.

The first step of the algorithm (rule [$\mathsf{Initialize}$]) results with the state $I\!\downarrow \mid \varnothing$, which satisfies (I2)–(I4) trivially, and $I \subseteq I\!\downarrow$ establishes (I1). The following lemma states that the invariants are preserved by rules that do not result in $\mathsf{valid}$ or $\mathsf{invalid}$.

**Lemma 1.** *The rules* [$\mathsf{Unfold}$]*,* [$\mathsf{Induction}$]*,* [$\mathsf{Conflict}$]*,* [$\mathsf{CandidateNondet}$]*, and* [$\mathsf{DecideNondet}$] *preserve* (I1) – (I4)*,*

By induction on the length of the trace, it can be shown that if $\mathsf{init} \mapsto^* \mathbf{R} \mid Q$, then $\mathbf{R} \mid Q$ satisfies (I1) – (I4). When $\mathsf{init} \mapsto^* \mathsf{valid}$, there is a state $\mathbf{R} \mid Q$ such that $\mathsf{init} \mapsto^* \mathbf{R} \mid Q \mapsto \mathsf{valid}$, and the last applied rule is [$\mathsf{Valid}$]. To be able to apply [$\mathsf{Valid}$], there must be an $i$ such that $R_i^{\downarrow} = R_{i+1}^{\downarrow}$.

We claim that $R_i^{\downarrow}$ is a covering set. This claim follows from (1) $R_i^{\downarrow} \subseteq P^{\downarrow}$ by invariant (I4), (2) $I \subseteq R_i^{\downarrow}$ by invariant (I1), and (3) $\mathrm{post}(R_i^{\downarrow}) \subseteq R_{i+1}^{\downarrow} = R_i^{\downarrow}$ by invariant (I2). This proves the correctness of the algorithm in case $\mathsf{Cover} \subseteq P^{\downarrow}$:

**Theorem 1 (Soundness of uncoverability).** *Given a WSTS* $(\Sigma, I, \rightarrow, \preceq)$ *and a downward-closed set* $P^{\downarrow}$*, if* $\mathsf{init} \mapsto^* \mathsf{valid}$*, then* $\mathsf{Cover} \subseteq P^{\downarrow}$*.*

We next consider the case when $\mathsf{Cover} \not\subseteq P^{\downarrow}$. The following lemma describes an invariant of the priority queue.

**Lemma 2.** *Let* $\mathsf{init} \mapsto^* \mathbf{R} \mid Q$*. For every* $\langle a, i\rangle \in Q$*, there is a path from* $a$ *to some* $b \in \Sigma \setminus P^{\downarrow}$*.*

**Theorem 2 (Soundness of coverability).** *Given a WSTS* $(\Sigma, I, \rightarrow, \preceq)$ *and a downward-closed set* $P^{\downarrow}$*, if* $\mathsf{init} \mapsto^* \mathsf{invalid}$*, then* $\mathsf{Cover} \not\subseteq P^{\downarrow}$*.*

*Proof.* The assumption $\mathsf{init} \mapsto^* \mathsf{invalid}$ implies that there is some state $\mathbf{R} \mid Q$ such that $\mathsf{init} \mapsto^* \mathbf{R} \mid Q \mapsto \mathsf{invalid}$, and the last applied rule was [$\mathsf{Model}$]. Therefore, there is an $a$ such that $\langle a, 0\rangle \in Q$. By Lemma 2, there is a path from $a$ to some $b \in \Sigma \setminus P^{\downarrow}$. Since $a \in I\!\downarrow$, we have $b \in \mathsf{Cover}$. □

## 3.3 Termination

While the above non-deterministic rules guarantee soundness for any WSTS, termination requires some additional choices. We modify the [$\mathsf{DecideNondet}$] and [$\mathsf{CandidateNondet}$] rules into more restricted rules [$\mathsf{Decide}$] and [$\mathsf{Candidate}$], shown in Figure 2. All other rules are unchanged.

The restricted rules still preserve the invariants (I1) – (I4). Thus, the modified algorithm is still sound. To show termination, we first note that the system can make progress until it reaches either $\mathsf{valid}$ or $\mathsf{invalid}$.

**Proposition 1 (Progress).** *If* $\mathsf{init} \mapsto^* \mathbf{R} \mid Q$*, then one of the rules* [$\mathsf{Candidate}$]*,* [$\mathsf{Decide}$]*,* [$\mathsf{Conflict}$]*,* [$\mathsf{Induction}$]*,* [$\mathsf{Unfold}$]*,* [$\mathsf{Model}$] *and* [$\mathsf{Valid}$] *is applicable.*

$$\text{[Candidate]}\frac{a \in R_N^{\downarrow} \cap \min(\Sigma \setminus P^{\downarrow})}{\mathbf{R} \mid \varnothing \mapsto \mathbf{R} \mid \langle a, N\rangle}$$

$$\text{[Decide]}\frac{\min Q = \langle a, i\rangle \quad i > 0 \quad b \in \min(\mathrm{pre}(a\uparrow)) \cap R_{i-1}^{\downarrow} \setminus a\uparrow}{\mathbf{R} \mid Q \mapsto \mathbf{R} \mid Q.\textsc{Push}(\langle b, i-1\rangle)}$$

**Fig. 2.** Rules replacing [CandidateNondet] and [DecideNondet] in Fig. 1.

Next, we define an ordering on states $\mathbf{R} \mid Q$.

**Definition 1.** *Let $\mathbf{A}^{\downarrow} = (A_1^{\downarrow}, \dots, A_N^{\downarrow})$ and $\mathbf{B}^{\downarrow} = (B_1^{\downarrow}, \dots, B_N^{\downarrow})$ be two finite sequences of downward-closed sets of the equal length $N$. Define $\mathbf{A}^{\downarrow} \sqsubseteq \mathbf{B}^{\downarrow}$ iff $A_i^{\downarrow} \subseteq B_i^{\downarrow}$ for all $i = 1, \dots, N$. Let $Q$ be a priority queue whose elements are tuples $\langle a, i\rangle \in \Sigma \times \{0, \dots, N\}$. Define $\ell_N(Q) := \min(\{i \mid \langle a, i\rangle \in Q\} \cup \{N+1\})$, to be the smallest priority in $Q$, or $N+1$ if $Q$ is empty.*

*For two states $\mathbf{R} \mid Q$ and $\mathbf{R}' \mid Q'$ such that* $\mathrm{length}(\mathbf{R}) = \mathrm{length}(\mathbf{R}') = N$, *we define the ordering $\leq_s$ as:*

$$\mathbf{R} \mid Q \leq_s \mathbf{R}' \mid Q' :\iff \mathbf{R} \sqsubseteq \mathbf{R}' \wedge (\mathbf{R} = \mathbf{R}' \rightarrow \ell_N(Q) \leq \ell_N(Q'))\,.$$

*We write $\mathbf{R} \mid Q <_s \mathbf{R}' \mid Q'$ if $\mathbf{R} \mid Q \leq_s \mathbf{R}' \mid Q'$ and $\mathbf{R}' \mid Q' \not\leq_s \mathbf{R} \mid Q$.*

**Lemma 3.** *Given a natural number $N$, the relation $\leq_s$ is a well-quasi-ordering on the set $\mathcal{D}^N \times \mathcal{Q}_N$, where $\mathcal{D}$ is a set of downward-closed subsets of $\Sigma$, and $\mathcal{Q}_N$ denotes the set of priority queues over $\Sigma \times \{0, \dots, N\}$.*

Using the well-quasi-ordering $\leq_s$, we can prove a lemma which characterizes infinite runs of the algorithm. The proof follows from the observation that if $\mathbf{R} \mid Q \mapsto \mathbf{R}' \mid Q'$ as a result of applying the [Candidate], [Decide], [Conflict], or [Induction] rules, then $\mathbf{R}' \mid Q' <_s \mathbf{R} \mid Q$.

**Lemma 4 (Infinite sequence condition).** *Any infinite sequence $\mathsf{init} \mapsto \sigma_1 \mapsto \sigma_2 \mapsto \cdots$, must contain infinitely many $i$ such that $\sigma_i \mapsto \sigma_{i+1}$ by applying the rule* [Unfold].

Note that applying the rule [Unfold] increases the length of the sequence $\mathbf{R}$. Therefore, in any infinite run of the algorithm, $\mathrm{length}(\mathbf{R})$ increases unboundedly. On the other hand, only a finite number of different sets $R_i^{\downarrow}$ can be generated for a downward-finite WSTS. To show that, we define a sequence of sets $D_i$, which provide a finite representation of states backward reachable from $\Sigma \setminus P^{\downarrow}$.

Recall the sequence $\mathsf{U}_i^{\uparrow}$ of backward reachable states from (BackwardReach). The set $D_i$ captures all new elements that are introduced in $\mathsf{U}_i^{\uparrow}$ and that were not present in the previous iterations. Formally, we define sets $D_i$ as follows:

$$D_0 := \min(\Sigma \setminus P^{\downarrow})\,, \qquad D_{i+1} := \bigcup_{a \in D_i} \min(\mathrm{pre}(a\uparrow)) \setminus \mathsf{U}_i^{\uparrow}\,. \tag{2}$$

By induction and the finiteness of the set of minimal elements, it follows that $D_i$ is finite for all $i \geqslant 0$. Furthermore, there is an $L$ such that $\mathsf{U}_L^{\uparrow} = \mathsf{U}_{L+j}^{\uparrow}$, and therefore $D_{L+j} = \emptyset$, for all $j > 0$. As a consequence, the set $\bigcup_{i \geqslant 0} D_i$ is finite.

**Lemma 5.** *Let $\mathit{init} \mapsto^* \mathbf{R} \mid Q$. For every $\langle a, i \rangle \in Q$, it holds that $a \in D_{N-i}$.*

**Lemma 6.** *Given a downward-finite WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, let $D := \bigcup_{i \geqslant 0} D_i$. Then there is $N_0$ such that for any sequence $\mathbf{R}$ of* $\text{length}(\mathbf{R}) = N > N_0$*, generated by applying the restricted rules, there is $i < N$ such that $R_i^{\downarrow} = R_{i+1}^{\downarrow}$.*

*Proof.* From Lemma 5 and the definition (1) of $\text{Gen}_i$, it follows that the restricted rules generate sequences $\mathbf{R}$ such that $R_i^{\downarrow} = \Sigma \setminus B_i\!\uparrow$ , $B_i \supseteq B_{i+1}$ and $B_i \subseteq D\!\downarrow$, for $i > 0$. Since $D$ is finite, $D\!\downarrow$ is also finite by downward-finiteness. Hence, there is only a finite number of possible sets of the form $\Sigma \setminus B\!\uparrow$, for $B \subseteq D\!\downarrow$. Therefore, sequences $\mathbf{R}$ of sufficient length contain equal adjacent sets. $\square$

Combining Lemmas 4 and 6, we see that in any infinite run of the algorithm, the rule [Valid] becomes applicable from some point onward. If we impose a fair usage of that rule (i.e. the rule is eventually used after it becomes applicable), we get termination.

**Theorem 3 (Termination).** *Given a downward-finite WSTS $(\Sigma, I, \rightarrow, \preceq)$ and a downward-closed set $P^{\downarrow}$, if the rule* [Valid] *is used fairly, the algorithm reaches either valid or invalid.*

Note that Theorem 3 is the only result that requires downward-finiteness of the WSTS. We show that the downward-finiteness condition is necessary for the termination of the abstract IC3 algorithm, using arbitrary generalization schemes and the full leeway in applying the rules. Consider a WSTS $(\mathbb{N} \cup \{\omega\}, \{0\}, \rightarrow, \leqslant)$, where $x \rightarrow x + 1$ for each $x \in \mathbb{N}$ and $\omega \rightarrow \omega$, and $\leqslant$ is the natural order on $\mathbb{N}$ extended with $x \leqslant \omega$ for all $x \in \mathbb{N}$. Consider the downward-closed set $\mathbb{N}$. The backward analysis terminates in one step, since $\text{pre}(\omega) = \{\omega\}$. However, the IC3 algorithm need not terminate. After unfolding, we find a conflict since $\text{pre}(\omega) = \{\omega\}$, which is not initial. Generalizing, we get $R_1^{\downarrow} = \{0, 1\}$. At this point, we unfold again. We find another conflict, and generalize to $R_2^{\downarrow} = \{0, 1, 2\}$. We continue this way to generate an infinite sequence of steps without terminating.

## 4 Coverability for Petri Nets

We now describe an implementation of our algorithm for the coverability problem for Petri nets, a widely used model for concurrent systems.

## 4.1 Petri Nets

A Petri net (PN, for short) is a tuple $(S, T, W)$, where $S$ is a finite set of *places*, $T$ is a finite set of *transitions* disjoint from $S$, and $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is the arc multiplicity function.

The semantics of a PN is given using *markings*. A marking is a function from $S$ to $\mathbb{N}$. For a marking $m$ and place $s \in S$, we say $s$ has $m(s)$ tokens. A transition $t \in T$ is *enabled* at marking $m$, written $m|t\rangle$, if $m(s) \geqslant W(s,t)$ for all $s \in S$. A transition $t$ that is enabled at $m$ can fire, yielding a new marking $m'$ such that $m'(s) = m(s) - W(s,t) + W(t,s)$. We write $m|t\rangle m'$ to denote the transition from $m$ to $m'$ on firing $t$.

A PN $(S, T, W)$ and an initial marking $m_0$ give rise to a WSTS $(\Sigma, \{m_0\}, \to, \preceq)$, where $\Sigma$ is the set of markings, and $m_0$ is a single initial state. The transition relation is defined as follows: there is an edge $m \to m'$ if and only if there is some transition $t \in T$ such that $m|t\rangle m'$. The well-quasi-ordering satisfies the following property: $m \preceq m'$ if and only if for each $s \in S$ we have $m(s) \leqslant m'(s)$. The compatibility condition holds: if $m_1|t\rangle m_2$ and $m_1 \preceq m_1'$, then there is a marking $m_2'$ such that $m_1'|t\rangle m_2'$ and $m_2 \preceq m_2'$. Moreover, since markings can be only non-negative integers, the wqo is downward-finite. The coverability problem for PNs is defined as the coverability problem on this WSTS.

We represent Petri nets as follows. Let $S = \{s_1, \ldots, s_n\}$ be the set of places. A marking $m$ is represented as the tuple of natural numbers $(m(s_1), \ldots, m(s_n))$. A transition $t$ is represented as a pair $(\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$, where $\mathbf{g}$ represents the enabling condition, and $\mathbf{d}$ represents the difference between the number of tokens in a place if the transition fires, and the current number of tokens. Formally, $\mathbf{g} = (W(s_1,t), \ldots, W(s_n,t))$ and $\mathbf{d} = (W(t,s_1) - W(s_1,t), \ldots, W(t,s_n) - W(s_n,t))$.

We represent upward-closed sets with their minimal bases, which are finite sets of $n$-tuples of natural numbers. A downward-closed set is represented as its complement (which is an upward-closed set). The sets $R_i^{\downarrow}$, which are constructed during the algorithm run, are therefore represented as their complements. Such a representation comes naturally as the algorithm executes. Originally each set $R_i^{\downarrow}$ is initialized to contain all the states. The algorithm removes sets of states of the form $\mathbf{b}\!\uparrow$ from $R_i^{\downarrow}$, for some $\mathbf{b} \in \mathbb{N}^n$. If a set $\mathbf{b}\!\uparrow$ was removed from $R_i^{\downarrow}$, we say that states in $\mathbf{b}\!\uparrow$ are *blocked* by $\mathbf{b}$ at level $i$. At the end every $R_i^{\downarrow}$ becomes to a set of the form $\Sigma \setminus \{\mathbf{b}_1, \ldots, \mathbf{b}_l\}\!\uparrow$ and we conceptually represent $R_i^{\downarrow}$ with $\{\mathbf{b}_1, \ldots, \mathbf{b}_l\}$.

The implementation uses a succinct representation of $\mathbf{R}$, so called *delta-encoding* [8]. Let $R_i^{\downarrow} = \Sigma \setminus B_i\!\uparrow$ and $R_{i+1}^{\downarrow} = \Sigma \setminus B_{i+1}\!\uparrow$ for some finite sets $B_i$ and $B_{i+1}$. Applying the invariant (I3) yields $B_{i+1} \subseteq B_i$. Therefore we only need to maintain a vector $\mathbf{F} = (F_0, \ldots, F_N, F_\infty)$ such that $\mathbf{b} \in F_i$ if $i$ is the highest level where $\mathbf{b}$ was blocked. This is sufficient because $\mathbf{b}$ is also blocked on all lower levels. As an illustration, for $(R_0^{\downarrow}, R_1^{\downarrow}, R_2^{\downarrow}) = (\{\mathbf{i}_1, \mathbf{i}_2\}, \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}, \{\mathbf{b}_2, \mathbf{b}_3\})$, the matching vector $\mathbf{F}$ might be $(F_0, F_1, F_2, F_\infty) = (\{\mathbf{i}_1, \mathbf{i}_2\}, \{\mathbf{b}_1, \mathbf{b}_4\}, \{\mathbf{b}_2, \mathbf{b}_3\}, \emptyset)$. The set $F_\infty$ represents states that can never be reached.

## 4.2 Implementation Details and Optimizations

Our implementation follows the rules given in Figures 1 and 2. In addition, we use optimizations from [8]. The main difference between our implementation and [8] is in the interpretation of sets being blocked: in [8] those are cubes identified with partial assignments to boolean variables, whereas in our case those are upward-closed sets generated by a single state. Also, a straightforward adaptation of the implementation [8] would replace a SAT solver with a solver for integer difference logic, a fragment of linear integer arithmetic which allows the most natural encoding of Petri nets. However, we observed that Petri nets allow an easy and efficient way of computing predecessors and deciding relative inductiveness directly. Thus we were able to eliminate the overhead of calling the SMT solver.

*Testing Membership in $R_i^{\downarrow}$.* Many of the rules given in Figures 1 and 2 depend on testing whether some state $\mathbf{a}$ is contained in a set $R_k^{\downarrow}$. Using the delta-encoded vector $\mathbf{F}$ this can be done by iterating over $F_i$ for $k \leqslant i \leqslant N+1$ and checking if any of them contains a state $\mathbf{c}$ such that $\mathbf{c} \preceq \mathbf{a}$. If there is such a state, it blocks $\mathbf{a}$, otherwise $\mathbf{a} \in R_k^{\downarrow}$. If $k = 0$, we search for $\mathbf{c}$ only in $F_0$.

*Implementation of the Rules.* The delta-encoded representation $\mathbf{F}$ also makes [Valid] easy to implement. Checking if $R_i^{\downarrow} = R_{i+1}^{\downarrow}$ reduces to checking if $F_i$ is empty for some $i < N$. [Unfold] is applied when [Candidate] can no longer yield a bad state contained in $R_N^{\downarrow}$. It increases $N$ and inserts an empty set to position $N$ in the vector $\mathbf{F}$, thus pushing $F_\infty$ from position $N$ to $N+1$. We implemented rules [Initialize], [Candidate] and [Model] in a straightforward manner.

*Computing Predecessors.* In the rest of the rules we need to find predecessors $\mathrm{pre}(\mathbf{a}\uparrow)$ in $R_i^{\downarrow} \setminus \mathbf{a}\uparrow$, or conclude relative inductiveness if no such predecessors exist. The implementation in [8] achieves this by using a function *solveRelative()* which invokes the SAT solver. But *solveRelative()* also does two important improvements. In case the SAT solver finds a cube of predecessors, it applies *ternary simulation* to expand it further. If the SAT solver concludes relative inductiveness, it extracts information to conclude a generalized clause is inductive relative to some level $k \geqslant i$. We succeeded to achieve analogous effects in case of Petri nets by the following observations. While it is unclear what ternary simulation would correspond to for Petri nets, the following lemma shows how to compute the most general predecessor along a fixed transition directly.

**Lemma 7.** *Let $\mathbf{a} \in \mathbb{N}^n$ be a state and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition. Then $\mathbf{b} \in \mathrm{pre}(\mathbf{a}\uparrow)$ is a predecessor along $t$ if and only if $\mathbf{b} \succeq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$.*

Therefore, to find an element of $\mathrm{pre}(\mathbf{a}\uparrow)$ and $R_i^{\downarrow} \setminus \mathbf{a}\uparrow$, we iterate through all transitions $t = (\mathbf{g}, \mathbf{d})$ and find the one for which $\max(\mathbf{a} - \mathbf{d}, \mathbf{g}) \in R_i^{\downarrow} \setminus \mathbf{a}\uparrow$.

If there are no such transitions, then $\mathbf{a}\uparrow$ is inductive relative to $R_i^{\downarrow}$. In that case, for each transition $t = (\mathbf{g}, \mathbf{d})$ the predecessor $\max(\mathbf{a}-\mathbf{d}, \mathbf{g})$ is either blocked by $\mathbf{a}$ itself, or there is $i_t \geqslant i$ and a state $\mathbf{c}_t \in F_{i_t}$ such that $\mathbf{c}_t \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. We define

$$i' := \min\{i_t \mid t \text{ is a transition}\},$$

where $i_t := N + 1$ for $t = (\mathbf{g}, \mathbf{d})$ if $\max(\mathbf{a} - \mathbf{d}, \mathbf{g})$ is blocked by $\mathbf{a}$ itself. Then $i' \geqslant i$ and $\mathbf{a}\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$.

*Computing Generalizations.* The following lemma shows that we can also significantly generalize $\mathbf{a}$, i.e. there is a simple way to compute a state $\mathbf{a}' \preceq \mathbf{a}$ such that for all transitions $t = (\mathbf{g}, \mathbf{d})$, $\max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$ remains blocked either by $\mathbf{a}'$ itself, or by $\mathbf{c}_t$.

**Lemma 8.** *Let $\mathbf{a}, \mathbf{c} \in \mathbb{N}^n$ be states and $t = (\mathbf{g}, \mathbf{d}) \in \mathbb{N}^n \times \mathbb{Z}^n$ be a transition.*

1. *Let $\mathbf{c} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$. Define $\mathbf{a}'' \in \mathbb{N}^n$ by $a''_j := c_j + d_j$ if $g_j < c_j$ and $a''_j := 0$ if $g_j \geqslant c_j$, for $j = 1, \ldots, n$. Then $\mathbf{a}'' \preceq \mathbf{a}$. Additionally, for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}'' \preceq \mathbf{a}' \preceq \mathbf{a}$, we have $\mathbf{c} \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*
2. *If $\mathbf{a} \preceq \max(\mathbf{a} - \mathbf{d}, \mathbf{g})$, then for each $\mathbf{a}' \in \mathbb{N}^n$ such that $\mathbf{a}' \preceq \mathbf{a}$, it holds that $\mathbf{a}' \preceq \max(\mathbf{a}' - \mathbf{d}, \mathbf{g})$.*

To continue with the case when the predecessor $\max(\mathbf{a} - \mathbf{d}, \mathbf{g})$ is blocked for each transition $t = (\mathbf{g}, \mathbf{d})$, we define $\mathbf{a}''_t$ as in Lemma 8 (1) if the predecessor is blocked by some state $\mathbf{c}_t \in F_{i_t}$ and $\mathbf{a}''_t := (0, \ldots, 0)$ if it is blocked by $\mathbf{a}$ itself. The state $\mathbf{a}''$ is defined to be the pointwise maximum of all states $\mathbf{a}''_t$. By Lemma 8, predecessors of $\mathbf{a}''$ remain blocked by the same states $\mathbf{c}_t$ or by $\mathbf{a}''$ itself.

However, $\mathbf{a}''$ still does not have to be a valid generalization, because it might be in $R^{\downarrow}_0$. If that is the case, we take any state $\mathbf{c} \in F_0$ which blocks $\mathbf{a}$ (such a state exists because $\mathbf{a} \notin R^{\downarrow}_0$). Then $\mathbf{a}' := \max(\mathbf{a}'', \mathbf{c})$ is a valid generalization: $\mathbf{a}' \preceq \mathbf{a}$ and $\mathbf{a}'\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$.

Using this technique, rules [Decide], [Conflict] and [Induction] become easy to implement. Note that some additional handling is needed in rules [Conflict] and [Induction] when blocking a generalized upward-closed set $\mathbf{a}'\!\uparrow$. If $\mathbf{a}'\!\uparrow$ is inductive relative to $R^{\downarrow}_{i'}$ for $i' < N$, we update the vector $\mathbf{F}$ by adding $\mathbf{a}'$ to $F_{i'+1}$. However, if $i' = N$ or $i' = N + 1$, we add $\mathbf{a}'$ to $F_{i'}$. Additionaly, for $1 \leqslant k \leqslant i' + 1$ (or $1 \leqslant k \leqslant i'$) we remove all states $\mathbf{c} \in F_k$ such that $\mathbf{a}' \preceq \mathbf{c}$.

## 5 Experimental Evaluation

We have implemented the IC3 algorithm in a tool called IIC. Our tool is written in C++ and uses the input format of mist2[1]. We evaluated the efficiency of the algorithm on a collection of Petri net examples. The goal of the evaluation was to compare the performance —both time and space usage— of IIC against other implementations of Petri net coverability.

We compare the performance of IIC, using our implementation described above, to the following algorithms: EEC [13] and backward search [1], as implemented by the tool mist2, and the MCOV algorithm [16] for parameterized multithreaded programs as implemented by bfc[2]. All experiments were performed on identical machines, each having Intel Xeon 2.67 GHz CPUs and 48 GB of memory, running Linux 3.2.21 in 64 bit mode. Execution time was limited to 1 hour, and memory to five gigabytes.

[1] See `http://software.imdea.org/~pierreganty/ist.html`
[2] See `http://www.cprover.org/bfc/`

| Problem | IIC | | Backward | | EEC | | MCOV | |
|---|---|---|---|---|---|---|---|---|
| Instance | Time | Mem | Time | Mem | Time | Mem | Time | Mem |
| Uncoverable instances | | | | | | | | |
| Bingham ($h = 150$) | **0.1** | **3.5** | 970.3 | 146.3 | 1.8 | 19.0 | **0.1** | $7.6^{2c}$ |
| Bingham ($h = 250$) | **0.2** | **6.7** | Timeout | | 9.6 | 45.4 | **0.2** | $19.6^{2c}$ |
| Ext. ReadWrite (small consts) | **0.0** | **1.3** | 0.1 | 3.7 | Timeout | | Timeout/OOM | |
| Ext. ReadWrite | **0.3** | **1.5** | 216.3 | 34.1 | Timeout | | 0.6 | $4.1^{2b}$ |
| FMS (old) | < **0.1** | **1.3** | 1.3 | 5.5 | Timeout | | 0.1 | $5.8^{2c}$ |
| Mesh2x2 | < **0.1** | **1.3** | 0.3 | 3.9 | 266.9 | 24.3 | < **0.1** | $4.2^{1c}$ |
| Mesh3x2 | < **0.1** | **1.5** | 4.1 | 7.0 | Timeout | | < **0.1** | $2.0^{2b}$ |
| Multipoll | 1.5 | **1.6** | 0.5 | 4.3 | 21.8 | 7.1 | < **0.1** | $1.7^{2b}$ |
| MedAA1 | **0.5** | **173.3** | 8.8 | 598.8 | | | 3.7 | $210.4^{2b}$ |
| MedAA2 | Timeout | | Timeout | | | | Timeout/OOM | |
| MedAA5 | Timeout | | Timeout | | | | Timeout/OOM | |
| MedAR1 | **0.8** | **173.3** | 8.77 | 598.8 | | | 3.7 | $210.4^{2b}$ |
| MedAR2 | 33.2 | **173.3** | 15.7 | 599.4 | | | **13.7** | $210.4^{2b}$ |
| MedAR5 | 128.1 | **173.3** | 26.6 | 600 | | | **12.9** | $210.4^{2b}$ |
| MedHA1 | **0.8** | **173.3** | 8.9 | 598.7 | | | $5.5^{2c}$ | $210.4^{2b}$ |
| MedHA2 | 33.2 | **173.3** | 14.7 | 599.5 | | | **12.6** | $210.4^{2b}$ |
| MedHA5 | Timeout | | 3219.7 | 647.3 | | | 12.5 | $210.4^{2b}$ |
| MedHQ1 | **0.7** | **173.3** | 8.8 | 598.8 | | | 12.2 | $210.4^{2b}$ |
| MedHQ2 | 33.8 | **173.3** | 16.6 | 596.9 | | | **13.2** | $210.4^{2b}$ |
| MedHQ5 | 125.8 | **173.3** | 26.6 | 600 | | | **12.6** | $210.4^{2b}$ |
| Coverable instances | | | | | | | | |
| Kanban | < **0.1** | **1.4** | 804.7 | 55.1 | Timeout | | 0.1 | $6.0^{2c}$ |
| pncsacover | 2.8 | **2.2** | 7.9 | 11.2 | 36.5 | 8.8 | **1.0** | $23.0^{1c}$ |
| pncsasemiliv | 0.1 | **1.5** | 0.2 | 3.9 | 32.1 | 8.8 | < **0.1** | $3.7^{2c}$ |
| MedAA1-bug | **0.8** | **172.7** | 1.0 | 596.9 | 56.5 | 658.0 | 3.6 | $210.4^{2b}$ |
| MedHR2-bug | **0.6** | **172.7** | 0.6 | 596.9 | 57.2 | 658.0 | 12.8 | $210.4^{2b}$ |
| MedHQ2-bug | 0.4 | **172.7** | **0.3** | 596.9 | 56.8 | 658.0 | 12.9 | $210.4^{2b}$ |

**Table 1.** Experimental results: comparison of running time and memory consumption for different coverability algorithms on Petri net benchmarks. Memory consumption is in MB, and running time in seconds. In the MCOV column, the superscripts indicate the version of bfc used ($^1$ means the version Jan 2012 version, $^2$ the Feb 2013 version), and the analysis mode ($^c$: combined, $^b$: backward only, $^f$: forward only). We list the best result for all the version/parameter combinations that were tried. EEC timed out on all MedXXX examples.

*Mist2 and MedXXX Benchmarks.* We used 29 Petri net examples from the mist2 distribution and 12 examples from checking security properties of message-passing programs communicating through unbounded and unordered channels (MedXXX examples [18]). We focus on examples that took longer than 2 seconds for at least one algorithm. Table 1 show run times and memory usage on the mist2 and message-passing program benchmarks. For each row, the column in bold shows the winner (time or space) for each instance. IIC performs well on these benchmarks, both in time and in memory usage. To account for mist2's use of a pooled memory, we estimated its baseline usage to 2.5 MB by averaging over all examples that ran in less than 1 second. The memory statistics includes the size of the program binary. We created statically-linked versions of all binaries, and the binaries were within 1 MB of each other.

*Multithreaded Program Benchmarks.* Table 2 shows comparisons of IIC with MCOV on a set of multithreaded programs distributed with MCOV. While IIC

| Problem Instance | IIC Time | IIC Mem | MCOV Time | MCOV Mem |
|---|---|---|---|---|
| Coverable instances | | | | |
| Boop 2 | 82.0 | 287.9 | **0.1** | **12.1**$^{1c}$ |
| FuncPtr3 1 | $<$ 0.1 | **1.5** | $<$ 0.1 | 3.4$^{2c}$ |
| FuncPtr3 2 | 0.2 | **12.3** | 0.1 | 7.9$^{2c}$ |
| FuncPtr3 3 | 28.5 | 939.1 | **3.6** | **303.8**$^{1c}$ |
| DoubleLock1 2 | Timeout | | **0.8** | **56.7**$^{2c}$ |
| DoubleLock3 2 | 8.0 | 41.3 | $<$ **0.1** | **4.8**$^{2c}$ |
| Lu-fig2 3 | Timeout | | **0.1** | **10.4**$^{2c}$ |
| Peterson 2 | Timeout | | **0.2** | **23.0**$^{1c}$ |
| Pthread5 3 | 132428 | 468.8 | **0.1** | **17.0**$^{1c}$ |
| Pthread5 3 | | | 0.2 | **49.6**$^{2c}$ |
| SimpleLoop 2 | 7.9 | 6.0 | $<$ **0.1** | **4.8**$^{2c}$ |
| Spin2003 2 | 4852.2 | 54.4 | $<$ **0.1** | **2.7**$^{2c}$ |
| StackCAS 2 | 2.5 | **1.6** | $<$ **0.1** | 3.7$^{2c}$ |
| StackCAS 3 | 5.5 | 21.7 | $<$ **0.1** | **4.4**$^{2c}$ |
| Szymanski 2 | Timeout | | **0.4** | **26.7**$^{2c}$ |

| Problem Instance | IIC Time | IIC Mem | MCOV Time | MCOV Mem |
|---|---|---|---|---|
| Uncoverable instances | | | | |
| Conditionals 2 | 0.1 | **3.6** | $<$ **0.1** | 5.7$^{2c}$ |
| RandCAS 2 | $<$ **0.1** | **2.0** | $<$ **0.1** | 3.9$^{2c}$ |

**Table 2.** Experimental results: comparison between MCOV and IIC on examples derived from parameterized multithreaded programs. The superscripts for MCOV are as in Table 1.

is competitive on the uncoverable examples, MCOV performs much better on the coverable ones. We have identified two reasons for MCOV's better performance.

The first reason is an encoding problem. MCOV represents examples as thread transition systems (TTS) [16] that have 1-bounded "global" states and potentially unbounded "local" states. A state of a TTS is a pair consisting of a single global state and a multiset of local states. While [16] defines multiple kinds of transitions, we only need to consider what happens to the global state. In TTS, each transition is of the form: given a global state $g_1$ and a condition on the local states, go to global state $g_2$ and modify the local states in a specific way. For example, the SimpleLoop-2 example is a TTS with 65 global states and 28 local states. In the TTS, we find that for each global state, there are at most 32 transitions that have any given global state as final global state. In particular, if the IIC algorithm were to be directly applied on this representation, the [Decide] rule could be applied at most 32 times to a given state generated by [Candidate], enumerating all pre-images induced by the corresponding transitions. Since our implementation works on Petri nets, we translate TTS to PN. The translation maps each global state to a Petri net place, but IIC does not know the invariant that exactly one of these places contains a token. Thus, IIC generates pre-images in which two or more places corresponding to global states contain tokens, and rules them out later through a conflict. On the translation of SimpleLoop-2, we found that 165 pre-images were generated for the target state in $R_1^{\downarrow}$. This causes a significant enlargement of the search space. We believe the use of PN invariants can improve the performance of IIC.

The second reason is the use of a combined forward and backward search in MCOV versus a backward search in IIC. It has been observed before that forward search performs better on software examples [13]. For comparison, we ran MCOV both in combined-search mode and in backward-search mode. In

80% of the cases (12 out of 16), the combined search was faster by at least a factor of 10, while no measurable difference was observed in the other four cases. Nevertheless, MCOV using backward search still outperforms IIC in these examples because of the better encoding.

In conclusion, based on experimental results, we believe that IIC, an IC3-based algorithm, is a practical coverability checker.

**Acknowledgements.** We thank Alexander Kaiser for help with the bfc tool. We thank Aaron Bradley for helpful and detailed comments and suggestions.

## References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS '96*, pages 313–321. IEEE, 1996.
2. P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *CAV'98*, LNCS 1427, pages 305–318. Springer, 1998.
3. A.R. Bradley. SAT-based model checking without unrolling. In *VMCAI'11*, LNCS, pages 70–87. Springer, 2011.
4. G. Ciardo. Petri nets with marking-dependent arc multiplicity: properties and analysis. In *ICATPN '94*, volume 815 of *LNCS*, pages 179–198. Springer, 1994.
5. A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV'12: Computer-Aided Verification*, LNCS 7358, pages 277–293. Springer, 2012.
6. L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.
7. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP '98*, LNCS 1443, pages 103–115. Springer, 1998.
8. N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *FMCAD'11*, pages 125–134. FMCAD Inc, 2011.
9. E.A. Emerson and K.S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS '98*, pages 70–80. IEEE, 1998.
10. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS '99*, pages 352–359. IEEE Computer Society, 1999.
11. J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
12. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
13. G. Geeraerts, J.-F. Raskin, and L. Van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *J. Comput. Syst. Sci.*, 72(1):180–203, 2006.
14. G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, s3-2(1):326–336, 1952.
15. K. Hoder and N. Bjørner. Generalized property directed reachability. In *SAT'12*, pages 157–171. Springer, 2012.
16. A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *CONCUR 2012*, LNCS 7454, pages 500–515. Springer, 2012.
17. J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. Technical Report 1301.7321, CoRR, 2013.
18. R. Majumdar, R. Meyer, and Z. Wang. Static provenance verification for message-passing programs. In *SAS 13*, 2013.