Ideal Abstractions for Well-Structured Transition Systems

Damien Zufferey[1], Thomas Wies[2], and Thomas A. Henzinger[1]

[1] IST Austria*
[2] New York University

Abstract. Many infinite state systems can be seen as well-structured transition systems (WSTS), i.e., systems equipped with a well-quasi-ordering on states that is also a simulation relation. WSTS are an attractive target for formal analysis because there exist generic algorithms that decide interesting verification problems for this class. Among the most popular algorithms are acceleration-based forward analyses for computing the covering set. Termination of these algorithms can only be guaranteed for flattable WSTS. Yet, many WSTS of practical interest are not flattable and the question whether any given WSTS is flattable is itself undecidable. We therefore propose an analysis that computes the covering set and captures the essence of acceleration-based algorithms, but sacrifices precision for guaranteed termination. Our analysis is an abstract interpretation whose abstract domain builds on the ideal completion of the well-quasi-ordered state space, and a widening operator that mimics acceleration and controls the loss of precision of the analysis. We present instances of our framework for various classes of WSTS. Our experience with a prototype implementation indicates that, despite the inherent precision loss, our analysis often computes the precise covering set of the analyzed system.

1 Introduction

One of the great successes in applying model checking techniques to the analysis of infinite state systems has been achieved by studying the class of *well-structured transition systems* (WSTS) [1, 12–16, 19, 20]. A WSTS is a transition system equipped with a well-quasi-ordering $\leq$ on its states that satisfies the following monotonicity property: for all states s, s', and t if $s \leq t$ and $s \to s'$ then there exists a state t' such that $t \to t'$ and $s' \leq t'$. In other words, $\leq$ is a simulation relation for the system. Interesting classes of WSTS include Petri nets [25] and their monotonic extensions [10], lossy channel systems [3], and dynamic process networks such as depth-bounded processes [22, 28].

Many interesting verification problems are decidable for WSTS. In particular, the verification of a large class of safety properties can be reduced to the *coverability problem*, which is decidable for WSTS that satisfy only a few additional mild assumptions [1]. The coverability problem asks whether, given a *bad state* s, there exists a reachable state s' of the system that covers the bad state, i.e., $s_0 \to^* s'$ and $s \leq s'$

* This research was supported in part by the European Research Council (ERC) Advanced Investigator Grant QUAREM and by the Austrian Science Fund (FWF) project S11402-N23.

where s_0 is an initial state $s_0 \in S_0$. In this paper, we are not just interested in solving the coverability problem, but in the more general problem of computing the *covering set* of a WSTS T. The covering set $Cover(T)$ is defined as the downward-closure of the reachable states of the system $Cover(T) = {\downarrow}\mathsf{post}^*({\downarrow}S_0)$. With the help of the covering set one can decide the coverability problem, but also answer other questions of interest such as boundedness (which asks whether $Cover(T)$ is finite) and U-boundedness (which asks whether $Cover(T) \cap U$ is finite for some upward-closed set U). While coverability is decidable for most WSTS, boundedness is not [10], i.e., the covering set is not always computable. Therefore, our goal is to compute precise over-approximations of the covering set, instead of computing this set exactly. In this paper, we present a new analysis based on abstract interpretation [7, 8] that accomplishes this goal.

One might question the rational of using an approximate analysis for solving decidable problems such as coverability. However, in practice one often uses coverability to give approximate answers to verification problems that are undecidable even for WSTS (such as general reachability). Thus, completeness is not always a primary concern. Also, one should bear in mind that even though coverability is decidable, its complexity is non-primitive recursive for many classes of WSTS [27], i.e., from a practical point of view the problem might as well be undecidable. Nevertheless, the techniques that have been developed for solving the coverability problem provide important algorithmic insights for the design of good approximate analyses.

Among the best understood algorithms for computing the exact covering set of a WSTS are acceleration-based algorithms such as the Karp-Miller tree construction for Petri nets [20] or the more general clover algorithm [13]. These algorithms exploit the fact that every downward-closed subset of a well-quasi-ordering can be effectively represented as a finite union of order ideals [12, 17]. The covering set is then computed by identifying sequences of transitions in the system that correspond to loops leading from smaller to larger states in the ordering, and then computing the exact set of ideals covering the states reachable by arbitrary many iterations of these loops. This process is referred to as ω- or lub-acceleration. Since acceleration is exact, these algorithm compute the exact covering set of a WSTS, whenever they terminate. Since the covering set is not always computable, termination is only guaranteed for so-called *flattable* systems [13]. In a flattable WSTS the covering set can be obtained by a finite sequence of lub-accelerations of finite sequences of transitions. In particular, this means that every nested loop of transitions can be decomposed into a finite sequence of simple loops. Many WSTS of practical interest do not satisfy this property. We provide an example of such a system in the next section.

Contributions. We are the first to propose an abstract interpretation framework that computes precise approximation of covering sets for WSTS, captures the key insights of acceleration-based algorithms, yet is guaranteed to terminate even on non-flattable WSTS. The abstract domain of our analysis is based on the ideal completion of the well-quasi-ordering of the analyzed WSTS and an accompanying widening operator. The widening operator mimics the effect of acceleration, but loses enough precision to guarantee termination. Instead of accelerating loops that lead from sets of smaller to sets of larger states, our widening operator only accelerates the difference between these sets of states, independently of the actual sequence of transitions that produced

Equations:

$$\begin{aligned}
\text{client}(C,S) &= C().\text{client}(C,S) \oplus (\overline{S}(C).0 \mid \text{client}(C,S))\\
\text{server}(S) &= S(C).(\overline{C}().0 \mid \text{server}(S))\\
\text{env}(S) &= \text{env}(S) \mid (\nu\, C)\text{client}(C,S)
\end{aligned}$$

Initial state: $(\nu\, S)(\text{server}(S) \mid \text{env}(S))$

Fig. 1. A π-calculus process implementing a client-server protocol.

them. We present instances of our framework for the WSTS classes of Petri nets, lossy channel systems, and depth-bounded process networks. Our experience with a prototype implementation indicates that, despite its inherent incompleteness, our analysis often computes the precise covering set of the analyzed system.

Further Related Work. We have already explained, in detail, the connection of our work with acceleration-based algorithms for computing the covering set. We discuss further connections with algorithms for solving the related coverability problem. The simplest algorithm for this problem is a backward analysis described in [1]. In practice, backward algorithms tend to be less efficient than forward algorithms, especially for dynamic process networks where the pre operator is expensive to compute [28]. Therefore, many attempts have been made at deriving complete forward algorithms for this problem. The most general solutions are described in [16] and [15]. The expand, enlarge, and check algorithm [16] decides the covering problem using a combination of an under-approximating and an over-approximating forward analysis. The over-approximating analysis relies on a so-called adequate domain of limits for the representation of downward-closed sets, which is actually the ideal completion of the underlying well-quasi ordering [12]. Ganty et al. propose an alternative algorithm [15] based on abstract interpretation. Unlike our approach, this algorithm uses a finite abstract domain that represents downward-closed sets by complements of upward-closed sets. The algorithm then relies on a complete refinement scheme to refine the abstraction for a specific coverability goal. Both algorithms [12, 15] compute an over-approximation of the covering set as a byproduct of the analysis, namely an invariant whose complement contains the coverability goal. To ensure completeness, the precision of this computed invariant is geared towards proving the specific instance of the coverability problem. Instead, our analysis computes a precise approximation of the covering set that is independent of any specific coverability instance.

An extended version of this paper with additional material (including proofs) is available as a technical report [30].

2 Motivating Example

We start with an example of a non-flattable system and illustrate how our analysis computes its covering set. Our example is given by the π-calculus process shown in Figure 1. The process models a concurrent system that implements a client-server protocol using asynchronous message passing. The process consists of one single server thread, an environment thread, and an unbounded number of client threads. Each type of threads is

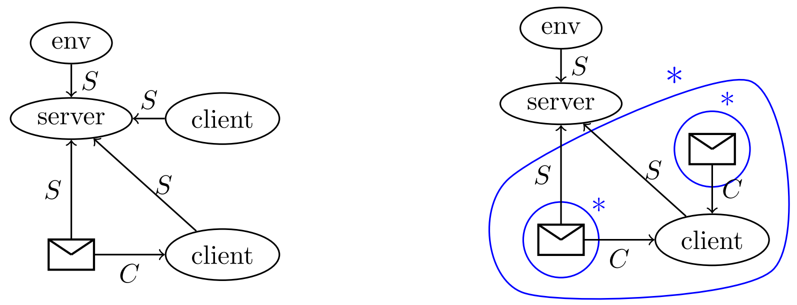

Fig. 2. Communication graph of the system in Figure 1 and the symbolic representation of the covering set of this system

defined by a recursive π-calculus equation. In each loop iteration of a client, the client non-deterministically chooses to either wait for a response from the server on its own dedicated channel C, or to send a new request to the server. Requests are sent asynchronously and modeled as threads that wait for the server to receive the client's channel name over the server's dedicated channel S and then terminate immediately. In each iteration of the server loop, the server waits for incoming requests on its own channel S and then asynchronously sends a response back to the client using the client's channel name C received in the request. The environment thread models the fact that new clients can enter the system at anytime. In each iteration of the environment thread, it spawns a new client thread with its own dedicated fresh channel name. The initial state of the system consists only of the server and the environment thread.

The states of a π-calculus process can be represented as a *communication graph* with nodes corresponding to threads (labeled by their id) and edges corresponding to channels (labeled by channel names). The left hand side of Figure 2 shows the communication graph representing the process:

$$\mathrm{server}(S) \mid \mathrm{client}(C_1, S) \mid \overline{S}(C_1).0 \mid \mathrm{client}(C_2, S) \mid \mathrm{env}(S)$$

The transition relation on processes is monotone with respect to the ordering on processes that is induced by subgraph isomorphism between their communication graphs, i.e., a process represented by a communication graph G can take all transitions of processes represented by the subgraphs of G. We call a set of graphs *depth-bounded*, if there exists a bound on the length of all simple paths in all graphs in the set. A *depth-bounded process* [22] is a process whose set of reachable communication graphs is depth-bounded. The subgraph isomorphism ordering is a well-quasi-ordering on sets of depth-bounded graphs, i.e., depth-bounded processes are WSTS. The process defined in Figure 1 is depth-bounded because the longest simple path in any of its reachable communication graphs has length at most 2. We now explain our analysis through this example.

Our analysis computes an over-approximation of the covering set of the analyzed WSTS, i.e., the downward-closure (with respect to the well-quasi-ordering) of its reachable set of states. The elements of the abstract domain of the analysis are the downward-closed sets. In our example, these are sets of communication graphs that are downward-closed with respect to the subgraph ordering. A finite downward-closed set of graphs can be represented by the maximal graphs in the set. The downward-closure of a single graph is an ideal of the subgraph ordering. Thus, any finite downward-closed set is a finite union of ideals. For well-quasi-orderings this is true for arbitrary downward-closed sets, including infinite ones. We symbolically represent the infinite ideals of the subgraph ordering by graphs where some subgraphs are marked with the symbol '*'. These markings of subgraphs can be nested. Such a symbolic graph represents the downward-closure of all graphs that result from (recursively) unfolding the marked subgraphs arbitrarily often. The right hand side of Figure 2 shows such a symbolic graph. It represents a downward-closed set of communication graphs of our example system that is also the covering set of the system. The covering set consists of all graphs that contain one server thread, one environment thread, and arbitrarily many clients with arbitrarily many unprocessed request and response messages each.

Our analysis works as follows: it starts with a set of symbolic communication graphs that represents the downward-closure of the initial states of the system. Then it iterates a fixed point functional that is composed of the following two steps: (1) compute the set of symbolic communication graphs that represent the downward-closure of the post states of the states represented by the current set of symbolic graphs, and (2) widen the resulting set of symbolic graphs with respect to the sequence of iterates that have been computed in the previous steps. The widening step compares the symbolic graphs in the new iterate pairwise to the symbolic graphs obtained in the previous iterates. If a symbolic graph in the new iterate is larger than some symbolic graph in a previous iterate then the larger graph must contain a subgraph that is not contained in the smaller one. This subgraph in the larger graph is then marked with a '*'. The intuition behind the widening is that, because of monotonicity of the transition relation, the sequence of transitions that lead from the smaller to the larger graph can be repeated arbitrarily often, which results in graphs with arbitrarily many copies of the new subgraph. Figure 3 shows a sequence of symbolic graphs obtained during the analysis of the client-server example. The final symbolic graph in the sequence represents the covering set of the system. This symbolic graph is also the fixed point obtained by our analysis, i.e., in this example the analysis does not lose precision.

Note that the covering set of our example system cannot be computed by a finite number of accelerations of finite sequences of transitions, i.e., the system is not flattable. This is reflected by the nesting of marked subgraphs in the symbolic graph that represents the covering set. To obtain this covering set via acceleration, one would need to compute the set of states reachable by a transfinite sequence of transitions resulting from ω-acceleration of a sequence of transition that is already infinite. The infinite sequence of transition that is to be accelerated corresponds to the creation of a client by the environment thread, followed by infinitely many exchanges of request and response messages between this client and the server. Since acceleration-based algorithms such as the clover algorithm [13] cannot accelerate infinite sequences of transitions, they do not terminate on our example system.

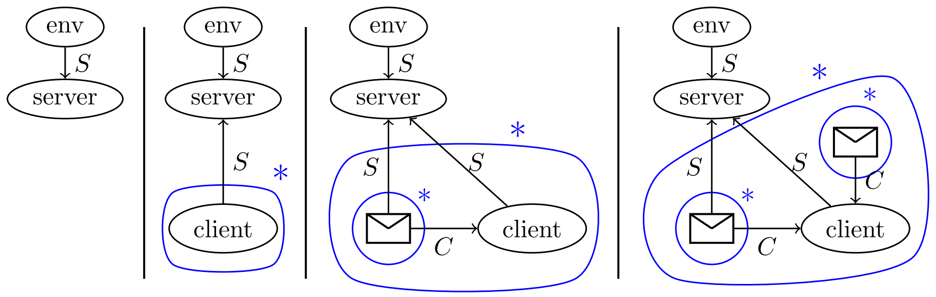

Fig. 3. Sequence of symbolic communication graphs produced by the analysis of the system in Figure 1

3 Preliminaries

Posets, lattices, wqos, and bqos. A *quasi-ordering* $\leq$ is a reflexive and transitive relation $\leq$ on a set X. In the following $X(\leq)$ is a quasi-ordered set. The *upward closure* $\uparrow Y$ of a set $Y \subseteq X$ is $\uparrow Y = \{\, x \in X \mid \exists y \in Y.\, y \leq x \,\}$. The *downward closure* $\downarrow Y$ of Y is $\downarrow Y = \{\, x \in X \mid \exists y \in Y.\, x \leq y \,\}$. A set $Y \subseteq X$ is *upward-closed* if $Y = \uparrow Y$ and *downward-closed* if $Y = \downarrow Y$. An *upper bound* $x \in X$ of a set $Y \subseteq X$ is such that for all $y \in Y$, $y \leq x$. The notion of *lower bound* is defined dually. A nonempty set $D \subseteq X$ is called *directed* if any two elements in D have a common upper bound in D. A set $I \subseteq X$ is an *ideal* of X if I is downward-closed and directed. We denote by $\mathit{Idl}(X)$ the set of all ideals of X and call $\mathit{Idl}(X)$ the *ideal completion* of X.

If a quasi-ordering $\leq$ on a set X is antisymmetric it is called a *partial ordering* and $X(\leq)$ a *poset*. A poset $L(\leq)$ is called a *complete lattice* if every subset $X \subseteq L$ has a least upper bound $\sqcup X$ and a greatest lower bound $\sqcap X$ in L. In particular, L has a least element $\bot = \sqcap L$ and a greatest element $\top = \sqcup L$. This lattice will be denoted by $L(\leq, \top, \bot, \sqcup, \sqcap)$. For a function $f : X \to Y$ and $X' \subseteq X$ we denote by $f(X')$ the set $\{\, f(x) \mid x \in X' \,\}$. A monotone function $f : L \to L$ on a complete lattice $L(\leq, \top, \bot, \sqcup, \sqcap)$ is called *continuous* if for every directed subset D of L, $\sqcup f(D) = f(\sqcup D)$. Recall Kleene's fixed point theorem which states that if $f : L \to L$ is continuous then its least fixed point $\mathit{lfp}^{\leq}(f) \in L$ exists and is given by $\sqcup \{\, f^i(\bot) \mid i \in \mathbb{N} \,\}$.

Let $L_1(\leq_1)$ and $L_2(\leq_2)$ be posets. A *Galois connection* between $L_1(\leq_1)$ and $L_2(\leq_2)$ is a pair of functions $\alpha : L_1 \to L_2$ and $\gamma : L_2 \to L_1$ that satisfy for all $x \in L_1, y \in L_2$, $\alpha(x) \leq_2 y$ iff $x \leq_1 \gamma(y)$. If γ is also injective then (α, γ) is called *Galois insertion*.

A quasi-ordering $\leq$ on a set X is called *well-quasi-ordering* (wqo) if any infinite sequence $x_0, x_1, x_2, \ldots$ of elements from X contains an increasing pair $x_i \leq x_j$ with $i < j$. We extend the ordering $\leq$ to an ordering $\leq$ on subsets of X as expected: for $Y_1, Y_2 \subseteq X$, we have $Y_1 \leq Y_2$ iff for all $y_1 \in Y_1$ there exists $y_2 \in Y_2$ such that $y_1 \leq y_2$. We will also refer to the notion of *better-quasi-ordering*. For all intents and purposes in this paper, it suffices to know that better-quasi-orderings are well-quasi-orderings that are closed under powerset construction, i.e., if $X(\leq)$ is a bqo then $\mathcal{P}(X)(\leq)$ is also a bqo. We refer to [23] for the precise (but rather technical) definition of bqos.

Well-structured transition system. A *transition system* is a tuple $T = (S, S_0, \to)$ where S is a set of states, $S_0 \subseteq S$ a set of initial states, and $\to \subseteq S \times S$ is a transition relation. We denote by $\mathsf{post} : \mathcal{P}(S) \to \mathcal{P}(S)$ the *post operator* of T defined by $\mathsf{post}(X) = \{ x' \in S \mid \exists x \in X.\, x \to x' \}$. Note that post is continuous on the complete lattice $\mathcal{P}(S)(\subseteq, S, \emptyset, \cup, \cap)$.

A *well-structured transition system* (WSTS) is a tuple $T = (S, S_0, \to, \leq)$ where $(S, S_0, \to)$ is a transition system and $\leq \subseteq S \times S$ a wqo that is upward-compatible with respect to $\to$, i.e., for all s_1, s_2, t_1 such that $s_1 \leq t_1$ and $s_1 \to s_2$, there exists t_2 such that $t_1 \to t_2$ and $s_2 \leq t_2$. The *covering set* of a well-structured transition system T, denoted $Cover(T)$, is defined by $Cover(T) = {\downarrow} \mathit{lfp}^{\subseteq}(\lambda X. {\downarrow} S_0 \cup \mathsf{post}(X))$.

4 Ideal Abstraction

We next describe our abstract interpretation framework for computing over-approximations of the covering sets of WSTS. For this purpose we fix a WSTS $T = (S, S_0, \to$ $, \leq)$ throughout the rest of this section.

4.1 Concrete and Abstract Domain

Following the framework of abstract interpretation [7, 8], a static analysis is defined by lattice-theoretic domains and by fixed point iteration over the domains. The concrete domain $\mathcal{D}$ of our analysis is the powerset domain over the states S of WSTS T:

$$\mathcal{D} \stackrel{\text{def}}{=} \mathcal{P}(S)(\subseteq, \emptyset, S, \cup, \cap)$$

Since our analysis is to compute an over-approximation of the covering set of T, which is a downward-closed set, we define the abstract domain $\mathcal{D}_{\downarrow}$ as the set of all downward-closed subsets of S, again ordered by subset inclusion:

$$\mathcal{D}_{\downarrow} \stackrel{\text{def}}{=} \{ {\downarrow} X \mid X \subseteq S \} \, (\subseteq, \emptyset, S, \cup, \cap)$$

One can easily verify that $\mathcal{D}_{\downarrow}$ is a complete lattice. This choice of the abstract domain suggests the following abstraction function $\alpha_{\downarrow} : \mathcal{D} \to \mathcal{D}_{\downarrow}$ and concretization function $\gamma_{\downarrow} : \mathcal{D}_{\downarrow} \to \mathcal{D}$ defined as $\alpha_{\downarrow}(X) \stackrel{\text{def}}{=} {\downarrow} X$ and $\gamma_{\downarrow}(Y) \stackrel{\text{def}}{=} Y$.

Proposition 1. *The pair $(\alpha_{\downarrow}, \gamma_{\downarrow})$ forms a Galois insertion between domains $\mathcal{D}$ and $\mathcal{D}_{\downarrow}$.*

According to [8], the Galois insertion $(\alpha_{\downarrow}, \gamma_{\downarrow})$ defines the *best abstract post operator* $\mathsf{post}_{\downarrow}$ on the abstract domain $\mathcal{D}_{\downarrow}$:

$$\mathsf{post}_{\downarrow} \stackrel{\text{def}}{=} \alpha_{\downarrow} \circ \mathsf{post} \circ \gamma_{\downarrow}$$

We next show that we can effectively represent the elements of $\mathcal{D}_{\downarrow}$ and, for all practical purposes, effectively compute $\mathsf{post}_{\downarrow}$ on this representation. To obtain this representation, we exploit the fact that any downward-closed subset of a wqo-set $S(\leq)$ is a finite union of ideals of $S(\leq)$.

Denote by $\mathcal{P}_{\text{fin}}(\mathit{Idl}(S))$ the finite sets of ideals of $S(\leq)$ and define the quasi-ordering $\sqsubseteq$ on $\mathcal{P}_{\text{fin}}(\mathit{Idl}(S))$ as the point-wise extension of $\subseteq$ from the ideal completion $\mathit{Idl}(S)$ of $S(\leq)$ to $\mathcal{P}_{\text{fin}}(\mathit{Idl}(S))$:

$$L_1 \sqsubseteq L_2 \overset{\text{def}}{\iff} \forall I_1 \in L_1.\, \exists I_2 \in L_2.\, I_1 \subseteq I_2$$

Let $\mathcal{D}_{\mathit{Idl}}$ be the quotient of $\mathcal{P}_{\text{fin}}(\mathit{Idl}(S))$ with respect to the equivalence relation $\sqsubseteq \cap \sqsubseteq^{-1}$. For notational convenience we use the same symbol $\sqsubseteq$ for the quasi-ordering on $\mathcal{P}_{\text{fin}}(\mathit{Idl}(S))$ and the partial ordering that it defines on the quotient $\mathcal{D}_{\mathit{Idl}}$. We further identify the elements of $\mathcal{D}_{\mathit{Idl}}$ with the finite sets of maximal ideals, i.e., for all $L \in \mathcal{D}_{\mathit{Idl}}$ and $I_1, I_2 \in L$, if $I_1 \subseteq I_2$ then $I_1 = I_2$.

Now, define the function $\gamma_{\mathit{Idl}} : \mathcal{D}_{\mathit{Idl}} \to \mathcal{D}_{\downarrow}$ as $\gamma_{\mathit{Idl}}(L) \overset{\text{def}}{=} \bigcup L$.

Proposition 2. *The function γ_{Idl} is an order-isomorphism.*

Let $\sqcup$ and $\sqcap$ be the least upper bound and greatest lower bound operators on the poset $\mathcal{D}_{\mathit{Idl}}(\sqsubseteq)$. These operators exist because $\mathcal{D}_{\downarrow}$ is a complete lattice and $\mathcal{D}_{\downarrow}$ and $\mathcal{D}_{\mathit{Idl}}$ are order-isomorphic according to Proposition 2. The following proposition then follows immediately.

Proposition 3. *$\mathcal{D}_{\mathit{Idl}}(\sqsubseteq, \emptyset, \{S\}, \sqcup, \sqcap)$ is a complete lattice.*

Let $\alpha_{\mathit{Idl}} : \mathcal{D}_{\downarrow} \to \mathcal{D}_{\mathit{Idl}}$ be the inverse of γ_{Idl}. Since γ_{Idl} is an order-isomorphism, the pair $(\alpha_{\mathit{Idl}}, \gamma_{\mathit{Idl}})$ forms a Galois insertion between $\mathcal{D}_{\downarrow}$ and $\mathcal{D}_{\mathit{Idl}}$.

Let $\alpha = \alpha_{\mathit{Idl}} \circ \alpha_{\downarrow}$ and $\gamma = \gamma_{\downarrow} \circ \gamma_{\mathit{Idl}}$. Then (α, γ) forms a Galois insertion between concrete domain $\mathcal{D}$ and abstract domain $\mathcal{D}_{\mathit{Idl}}$. Let $\mathsf{post}_{\mathit{Idl}} = \alpha \circ \mathsf{post} \circ \gamma$ be the induced best abstract post operator on $\mathcal{D}_{\mathit{Idl}}$ and let F_{Idl} be the function $F_{\mathit{Idl}} = \lambda L.\, \alpha(S_0) \sqcup \mathsf{post}_{\mathit{Idl}}(L)$. The following proposition is then a simple consequence of Proposition 2.

Proposition 4. *The least fixed point of F_{Idl} on $\mathcal{D}_{\mathit{Idl}}$ is the covering set of T:*

$$\gamma(\mathit{lfp}^{\sqsubseteq}(F_{\mathit{Idl}})) = \mathit{Cover}(T) \ .$$

Can we compute $\mathit{lfp}^{\sqsubseteq}(F_{\mathit{Idl}})$? In general the answer is "no" for various reasons. First, we may not be able to compute the iterates of the abstract functional F_{Idl}, respectively, decide the fixed point test on the abstract domain. However, for the classes of WSTS that are of practical interest, this is not a problem: We say that the ideal completion $\mathit{Idl}(S)$ of a WSTS $T = (S, S_0, \to, \leq)$ is *effective* if (i) for all $I_1, I_2 \in \mathit{Idl}(S)$, checking $I_1 \subseteq I_2$ is decidable, and (ii) for all $I \in \mathit{Idl}(S)$, $\mathsf{post}_{\mathit{Idl}}(\{I\})$ is computable. It follows from [12, Theorem 3.4] that all WSTS with a so called *effective adequate domain of limits* [16] also have an effective ideal completion. Classes of WSTS with this property include, e.g., Petri nets and their monotone extensions [16], lossy channel systems [12], and depth-bounded processes [28].

Thus, assume that T has an effective ideal completion. Then, for any $L \in \mathcal{D}_{\mathit{Idl}}$ we can compute $F_{\mathit{Idl}}(L)$ and decide $F_{\mathit{Idl}}(L) \sqsubseteq L$. However, this is not yet sufficient for guaranteeing termination. In general, the covering set of a WSTS is not computable, i.e., we cannot expect that the sequence of iterates $(\bigsqcup_{i \leq n} F^i_{\mathit{Idl}}(\emptyset))_{n \in \mathbb{N}}$ stabilizes. In fact, even if the exact covering set $\mathit{Cover}(T)$ is computable for a particular WSTS, the sequence of fixed point iterates might not stabilize because the abstract domain $\mathcal{D}_{\mathit{Idl}}$ has (typically) infinite height. To ensure termination of our analysis, we next define an appropriate widening operator for the abstract domain $\mathcal{D}_{\mathit{Idl}}$.

4.2 Widening

Let us first recall the notion of set-widening operators [9]. A *set-widening operator* for a poset $X(\leq)$ is a partial function $\nabla : \mathcal{P}(X) \rightharpoonup X$ that satisfies the following two conditions:

– *Covering*: For all $Y \subseteq X$, if $\nabla(Y)$ is defined then for all $y \in Y$, $y \leq \nabla(Y)$.
– *Termination*: For every ascending chain $\{x_i\}_{i\in\mathbb{N}}$ in $X(\leq)$, the sequence $y_0 = x_0$, $y_i = \nabla(\{x_0, \dots, x_i\})$, for all $i > 0$, is well-defined and an ascending stabilizing chain.

In the following, we define a general set-widening operator for the abstract domain $\mathcal{D}_{Idl}$. The reason for using a set-widening operator instead of the more popular pair widening operator is that we want to enable the widening operator to take into account the whole history of the previous iterates of the fixed point computation. This allows us to use widening to mimic the effect of acceleration for computing the exact covering set of flattable WSTS.

The set-widening operator on the abstract domain $\mathcal{D}_{Idl}$ is obtained by lifting a given set-widening operator for the ideal completion $Idl(S)$. This underlying widening operator on ideals is a parameter of the analysis because it is domain-specific for each class of WSTS. In the next section, we will describe several such widening operators for common classes of WSTS.

In general, extending a widening operator from a base domain to its finite powerset is non-trivial [5]. We can simplify this task by making a stronger assumption about the ordering $\leq$ on the base set S: we assume that $S(\leq)$ is not just a wqo, but a bqo. This ensures that the ideal completion $Idl(S)$ is itself a bqo with respect to the subset inclusion ordering. Using this fact we can then lift the set-widening operator on ideals to sets of ideals. From a practical point of view, requiring a bqo is not a real restriction, since all wqos of WSTS occurring in practice are actually bqos.

Assume that ∇_S is a set-widening operator on the poset $Idl(S)(\subseteq)$. Then define the operator $\nabla : \mathcal{P}(\mathcal{D}_{Idl}) \rightharpoonup \mathcal{D}_{Idl}$ as follows: for $C \subseteq \mathcal{D}_{Idl}$, if C is a finite ascending chain $C = \{L_i\}_{0\leq i\leq n}$ in $\mathcal{D}_{Idl}(\sqsubseteq)$ let $\nabla(C)$ be defined recursively by

$$\begin{aligned} \nabla(\{L_0\}) &= L_0 \\ \nabla(\{L_0, \dots, L_i\}) &= \nabla(\{L_0, \dots, L_{i-1}\}) \sqcup \\ &\quad \{\, \nabla_S(\mathcal{I}) \mid \mathcal{I} \text{ maximal ascending chain in } \nabla(\{L_0, \dots, L_{i-1}\}) \,\} \end{aligned}$$

for all $0 < i \leq n$. In all other cases let $\nabla(C)$ be undefined.

Proposition 5. *If $S(\leq)$ is a bqo then ∇ is a set-widening operator for $\mathcal{D}_{Idl}(\sqsubseteq)$.*

We now define our analysis in terms of the widening sequence $\{W_i\}_{i\in\mathbb{N}}$ as follows:

$$W_0 = \emptyset \qquad \textit{and} \qquad W_{i+1} = \nabla(\{W_0, \dots, W_i, F_{Idl}(W_i) \sqcup W_i\})$$

Note that for computing the image of ∇ in step $i+1$ we can reuse W_i. The properties of set-widening operators, Proposition 4, and Proposition 5 imply the soundness and termination of the analysis.

Theorem 6. *If $S(\leq)$ is a bqo then the sequence $\{W_i\}_{i\in\mathbb{N}}$ stabilizes and its least upper bound approximates the covering set of T, i.e., $Cover(T) \subseteq \gamma(\bigcup \{W_i\}_{i\in\mathbb{N}})$.*

Trace Partitioning. Note that, unlike acceleration, the widening operator ∇ does not take into account whether each widened chain of ideals is actually correlated by some sequence of transition in the system. This incurs an additional loss of precision that is not needed to ensure termination of the analysis. To avoid this loss of precision, we can refine the above analysis via combination with an appropriate trace partitioning domain [26]. The resulting analysis is a generalized Karp-Miller tree construction where acceleration has been replaced by widening.

5 Set-Widening Operators for Ideal Completions

We now discuss several instantiations of our analysis for different classes of WSTS by presenting the corresponding ideal completions and set-widening operators on ideals. We discuss, in turn, Petri nets, lossy channel systems, and depth-bounded processes.

5.1 Petri Nets

A *Petri net* is a tuple (S, T, W) where S is a finite set of places, T is a finite set of transitions, and $W : (S, T) \cup (T, S) \to \mathbb{N}$ is a (multi)set of arcs. A marking M is a map: $S \to \mathbb{N}$. We denote by $\mathcal{M}(S)$ the set of all markings over S. A transition $t \in T$ is fireable at M iff for all $s \in S$, $M(s) \geq W(s,t)$. Firing t at M gives M' defined as $M'(s) = M(s) - W(s,t) + W(t,s)$. The point-wise ordering of markings is a bqo [23]. The ideal completion $Idl(\mathcal{M}(S))$ of the markings of a Petri net can be represented by extended markings, which are functions $S \to \mathbb{N} \cup \{\omega\}$ [17]. The ordering on extended markings is given by $M \leq M'$ iff for all $s \in S$, $M'(s) = \omega$ or $M(s) \in \mathbb{N}$ and $M(s) \leq M'(s)$.

Widening for Petri Nets. The set-widening operator ∇_{PN} for a Petri Net corresponds to the usual acceleration used in the Karp-Miller tree construction for Petri nets. For a finite ascending chain $\{M_i\}_{0\leq i\leq n}$ we define $\nabla_{\mathsf{PN}}(\{M_i\}_{0\leq i\leq n}) = M$ where $M(s) = \omega$ if $M_n(s) > M_0(s)$ and $M_n(s)$ otherwise. Clearly this set-widening operator satisfies the covering condition. It also satisfies termination, since the set of places S is finite.

Precision of the Widening and Monotonic Extensions of Petri Nets. For standard Petri nets the above widening operator corresponds to the acceleration used in the Karp-Miller tree construction. In fact, for this class of WSTS our analysis does not lose precision. The reason is that in Petri nets sequences of firing transitions σ that increase the value of a marking M by some δ, $\sigma(M) = M + \delta$, do the same for all larger markings $M' \geq M$, i.e., $\sigma(M') = M' + \delta$.

For monotonic extensions of Petri nets, such as transfer nets and reset nets, the situation is more complicated. In a transfer net a transition can transfer all the tokens from one place to another place in a single step. In both cases we can use the same widening as for standard Petri nets, but the analysis may lose precision because neither transfer nets nor reset nets are flattable, in general. However, for a concrete net the loss of precision does not depend on the flattability of the net in consideration, i.e., there are non-flattable nets where the result of the analysis is exact and flat nets were the analysis over-approximates the actual covering set.

5.2 Lossy Channel Systems

A *lossy channel system* (LCS) [3] is a tuple (S, s_0, C, M, δ) where S is a finite set of control locations, s_0 is the initial location, C is a finite set of channels, M is a finite set of messages, and δ is a set of transitions. A state of an LCS is a tuple (s, w) where $s \in S$ and w is a mapping $C \to M^*$ denoting the content of the channels. A transition t is a tuple (s_1, Op, s_2) where $s_1, s_2 \in S$ and Op is of the form $c\,!/?\,m$ ($c \in C$,$m \in M$). The system can go from state (s_1, w_1) to (s_2, w_2) by firing transition t iff $Op = c!m \,\wedge\, w_2(c) \leq w_1(c)m$ or $Op = c?m \,\wedge\, mw_2(c) \leq w_1(c)$, the remaining channels are unchanged. The systems are called *lossy* because messages can be dropped from channels before and after performing a send or receive operation. The ordering on states $\leq$ is defined as $(s, w) \leq (s', w')$ iff $s = s'$ and for all $c \in C$, $w(c)$ is a subword of $w'(c)$. The subword ordering is a bqo [23] and thus so is the ordering $\leq$ on states. In the following we describe a widening on the content of individual channels. Its extension to states is defined as expected.

The downward-closed sets of the subword ordering are exactly the languages of simple regular expressions (SRE) [2], which are defined by the following grammar:

$$\begin{aligned} \mathit{atom} &::= (m + \epsilon) \mid (m_1 + \ldots + m_n)^* \\ \mathit{product} &::= \epsilon \mid \mathit{atom}\;\mathit{product} \\ \mathit{SRE} &::= \mathit{product}\;[\;+\;\mathit{SRE}] \end{aligned}$$

The ideals of the subword ordering are the languages denoted by the products in SRE. The ordering on the ideals is language inclusion.

Widening for LCS. The first step in defining the widening operator on channel contents is to define a notion of difference on the corresponding ideals. For a product p we denote by $|p|$ the number of atoms appearing in p and for $1 \leq i \leq |p|$ we denote by $p[i]$ the ith atom of p.

Let p, q be products. If $p \leq q$ then we can find a mapping $\iota : [1, |p|] \to [1, |q|]$ such that (i) ι is monotone, i.e., for all $i, j \in [1, |p|]$ if $i \leq j$ then $\iota(i) \leq \iota(j)$, (ii) for all $i \in [1, |p|]$ the language of $p[i]$ is included in the language of $q[\iota(i)]$, and (iii) for all $i, j \in [1, |p|]$ if $\iota(i) = \iota(j)$ and $q[\iota(i)]$ is of the form $(a + \epsilon)$ then $i = j$. We call ι an *inclusion mapping* for $p \leq q$. Note that we consider an interval $[l, r]$ to be empty if $l > r$, i.e., if $p = \epsilon$ then the inclusion mapping exists trivially.

Let p and q be atoms such that $p \leq q$ and let ι be an inclusion mapping for $p \leq q$. We define an extrapolation operator χ_{LCS} for p, q and ι as follows. If $p = \epsilon$ then $\chi_{\mathsf{LCS}}(p, q, \iota) = (\sum_i q[i])^*$. Otherwise, let $i_1, \ldots, i_n$ be the increasing sequence of indices in the range of ι. For each $j \in [1, n-1]$ define the interval $d_j = [i_j + 1, i_{j+1} - 1]$. Furthermore, define $d_0 = [1, i_1 - 1]$ and $d_n = [i_n, |q|]$. For all $j \in [0, n]$, define $s_j = \big(\sum_{i \in d_j} q[i]\big)^*$. Note that s_j is equivalent to ϵ if d_j is empty and, otherwise, s_j is equivalent to an atom of the form $\big(\sum_k m_k\big)^*$ where the m_k are the messages appearing in the atoms $q[i]$ for $i \in d_j$. Then define $\chi_{\mathsf{LCS}}(p, q, \iota) = s_0\, q[i_1] \ldots s_{k-1}\, q[i_k]\, s_k$.

Inclusion mappings are not necessarily unique. We therefore fix for each ascending sequence of products $p_1 \leq p_2 \dots$ a corresponding sequence $\iota_1, \iota_2, \dots$ such that (1) for all i, ι_i is an inclusion mapping for $p_i \leq p_{i+1}$, and (2) for every two ascending chains of products that share a common prefix, the corresponding sequences of inclusion mappings agree on this prefix.

Let $\pi = \{p_i\}_{0 \leq i \leq n}$ be an ascending chain of products with $n > 0$. The set-widening of π is then defined as $\nabla_{\mathsf{LCS}}(\pi) = \chi_{\mathsf{LCS}}(p_0, p_n, \iota_{0,n})$ where $\iota_{0,n}$ is the composition of the fixed sequence of inclusion mappings for π, $\iota_{0,n} = \iota_{n-1} \circ \cdots \circ \iota_0$.

Note that one cannot use the operator χ_{LCS} to define a standard pair widening operator ∇ on ideals of the subword ordering: $\nabla(p, q) = \chi_{\mathsf{LCS}}(p, q, \iota)$ where ι is an inclusion mapping for $p \leq q$. As a counterexample for termination of this operator consider the following sequence of ideals: $x_0 = \epsilon$, $x_1 = (a + \epsilon)$, $x_2 = a^*(b + \epsilon)$, $x_3 = a^*b^*(a + \epsilon)$, etc. Applying ∇ pairwise on consecutive elements of the sequence leads to the following diverging sequence: $y_0 = x_0 = \epsilon$, $y_1 = \nabla(y_0, x_1) = a^*$, $y_2 = \nabla(y_1, x_2) = a^*b^*$, $y_3 = \nabla(a^*b^*, x_3) = a^*b^*a^*$, etc. On the other hand, the set-widening operator ∇_{LCS} produces the stabilizing sequence: $y_0 = x_0 = \epsilon$, $y_1 = \nabla_{\mathsf{LCS}}(\{x_0, x_1\}) = a^*$, $y_2 = \nabla_{\mathsf{LCS}}(\{x_0, x_1, x_2\}) = (a + b)^*$, $y_3 = \nabla_{\mathsf{LCS}}(\{x_0, x_1, x_2, x_3\}) = (a + b)^*$, etc. For termination, it is crucial that the maximal length of the products provided as first argument of χ_{LCS} is bounded throughout all widening steps. This is for instance ensured by fixing the first argument of χ_{LCS} to one particular element of the widened sequence (e.g., the first element as in the definition of ∇_{LCS}). For a more detailed discussion and the proof of termination for the operator ∇_{LCS} we refer to the technical report [30].

5.3 Depth-Bounded Processes

Depth bounded processes (DBP) [22] form the largest known fragment of the π-calculus for which non-trivial verification problems are still decidable. In particular, we proved in [28] that the covering problem is decidable for this class. As for many other classes of WSTS, the coverability problem has non-primitive recursive complexity. This makes DBP a particularly interesting target for approximate analysis. We have already informally introduced DBP in Section 2 and explained how our analysis works for this class of WSTS. In the following, we explain the analysis of DBP in more detail. We outline an analysis that operates directly on process terms, instead of communication graphs.

We assume basic knowledge of the syntax and semantics of the π-calculus and refer the reader to [24] for a detailed introduction. We consider π-calculus processes that are described by finite systems of recursive π-calculus equations together with a process term denoting the initial state. We denote by $\equiv$ the usual syntactic congruence relation on π-calculus process terms.

The *nesting of restrictions* nest_ν of a process term is measured recursively as follows $\mathit{nest}_\nu(0) = \mathit{nest}_\nu(A(\boldsymbol{x})) = 0$, $\mathit{nest}_\nu((\nu x)P) = 1 + \mathit{nest}_\nu(P)$, and $\mathit{nest}_\nu(P_1 \mid P_2) = \max\{\mathit{nest}_\nu(P_1), \mathit{nest}_\nu(P_2)\}$. The *depth* of a process term P is the minimal nesting of restrictions of process terms in the congruence class of P: $\mathit{depth}(P) = \min\{\, \mathit{nest}_\nu(Q) \mid Q \equiv P \,\}$. A set of process terms $\mathcal{P}$ is called *depth-bounded* if there is $k_D \in \mathbb{N}$ such that $\mathit{depth}(P) \leq k_D$ for all $P \in \mathcal{P}$. A process is called *depth-bounded* if its set of reachable process terms is depth-bounded. As shown in [22], this definition is equivalent to the definition of depth-bounded processes that is given in Section 2 and refers to communication graphs.

We define the following natural quasi-ordering $\leq$ on process terms: let $P \equiv (\nu \boldsymbol{x})P'$ and Q be process terms then $P \leq Q$ if and only if $Q \equiv (\nu \boldsymbol{x})(P' \mid F)$ for some process term F. The ordering $\leq$ defines a bqo on sets of depth-bounded process terms. We have shown in [28] that the ideals of this bqo can be represented by extending process terms with a replication operator ! to encode that certain subprocesses may be repeated arbitrarily often. We call these terms *limit process terms*. For instance the covering set of the example discussed in Section 2 is denoted by the following limit process term:

$$(\nu S)(\text{server}(S) \mid \text{env}(S) \mid !(\nu C)(\text{client}(C, S) \mid !(\overline{S}(C).0) \mid !(C().0)))$$

The ordering $\leq$ is extended to limit process terms by extending the congruence relation $\equiv$ with additional axioms for replication. The resulting congruence relation (which we also denote by $\equiv$) corresponds to the *extended congruence relation* studied in [11], where it is also shown to be decidable.

Widening for Depth-Bounded Processes. We first describe an extrapolation operator χ_{DBP} on pairs of limit process terms, which is then lifted to a set-widening operator ∇_{DBP}. The extrapolation operator relies on a set of inference rules for checking validity of clauses of the form $P \leq Q$ where P, Q are limit process terms. The inference rules do not just prove $P \leq Q$ but do a bit more: given P and Q, the rules derive judgments of the form $\boldsymbol{x}, R, F \vdash P \leq Q$. The semantics of these judgments is that if $\boldsymbol{x}, R, F \triangleright P \parallel Q \equiv$ can be derived then $(\nu \boldsymbol{x})R \equiv P$ and $(\nu \boldsymbol{x})(R \mid F) \equiv Q$. We call F an *anti-frame*[3] of $P \leq Q$. The anti-frame captures the difference between process terms P and Q. The basic idea of extrapolation is that if $\boldsymbol{x}, R, F \vdash P \leq Q$ can be derived then $\chi_{\mathsf{DBP}}(P, Q)$ is given by $(\nu \boldsymbol{x})(R \mid !F)$. The set-widening operator ∇_{DBP} then applies extrapolation recursively on the input chain. A detailed description of the operators χ_{DBP} and ∇_{DBP} can be found in the technical report [30].

The intuition behind the termination argument for the operator ∇_{DBP} is that for an infinite ascending chain of limit processes, ∇_{DBP} gradually saturates the finitely many nesting levels of restrictions in the elements of the chain. It is important to realize that the extrapolation operator χ_{DBP} is not a pair-widening operator for limit process terms. The recursion built into the set-widening operator ∇_{DBP} ensures that a sufficiently high nesting depth of the replication operator is achieved. Intuitively, this recursion approximates the acceleration of infinite traces that correspond to unfoldings of inner loops within nested loops of the analyzed system. This is crucial for the termination of the analysis on non-flattable WSTS, such as the example presented in Section 2.

6 Implementation and Evaluation

We have implemented a prototype tool called PICASSO and applied it to a set of example programs. PICASSO combines our ideal abstraction domain with a trace partitioning domain [26]. The resulting analysis is a generalized Karp-Miller tree construction with widening instead of acceleration. The implementation is parameterized by the concrete

[3] The term "anti-frame" refers to abduction in entailment provers for separation logic [6].

ideal completion and the widening operator on ideals that are used in the analysis. The tool PICASSO and the example programs are available on-line [29].

For the analysis of our examples we have implemented a generalization of the ideal abstraction domain and widening operator for depth-bounded processes that we described in Sec. 5.3. The representation of ideals used in the implementation more closely resembles the communication graphs with nested repeated substructures described in Sec. 2. This representation admits process nodes in communication graphs with arbitrarily many outgoing edges. Such nodes correspond to process identifiers in π-calculus process terms with unbounded (but unordered) parameter lists. To represent the limit elements we annotate the nodes in the graph with natural numbers indicating the nesting depth of the nodes. Testing the ordering on states is done by computing morphisms between the corresponding graphs. The morphisms take into account the nesting structure by allowing mappings to nodes of higher nesting depth to be non-injective. The actual test is encoded into a set of Boolean constraints and passed to a SAT solver. The morphisms are then reconstructed from the obtained satisfying assignments. The algorithm constructs a Karp-Miller tree using a depth-first search. When the tree is extended with a new node, widening is applied to the chains on the path to the root of the tree that contain the new node. Among the smaller ancestors of a node, not all are used for the widening. Instead, nodes are selected using an exponential back-off strategy. When the depth of the constructed tree becomes too large, the algorithm tends to slow down significantly. For such cases, we have implemented a restart policy. When a restart occurs, the leaves of the current tree are used as roots to construct new trees. The restart policy ensures that, for larger examples, the analysis terminates within reasonable time. The current implementation uses restart intervals of 5 minutes. The implementation exploits parallelism and makes use of multiple cores when possible.

We ran our experiments on a machine with two AMD Opteron 2431 processors and a total of 12 cores. We found that memory consumption was not an issue for the analysis of our examples. The examples that we have considered are depth-bounded processes, which are inspired by Scala programs. These Scala programs use the Scala actor library [18] for the implementation of dynamic process networks. Table 1 summarizes the results of our experiments. The `ping-pong` example is the "*Hello World*" of actor programming and is taken from the tutorial for the Scala actor library. All remaining examples follow a client-server type of communication with an unbounded number of clients. These examples cover common patterns that arise in message passing programs. The second and third program are variations of the example presented in Section 2. In the third program, we added a timeout to the receive operations of clients. We model the timeout by letting the clients send `Timeout` messages to themselves. This pattern is often used in programs based on the Scala actor library. The `genericComputeServer` example is the message passing skeleton of a tutorial for remote actors [4]. The example implements a compute server that accepts computation tasks from clients and then executes them. The second version uses actors to model the closures that are sent to the server. This model is obtained using the usual reduction of high-order π-calculus to the standard π-calculus. The `liftChatLike` example is the message-passing skeleton extracted from a chat application based on the lift web framework [21]. Since our implementation does not yet support collections, the broad-

Name	tree size	cov. set size	time
`ping-pong`	17	14	0.6 s
`client-server`	25	2	1.9 s
`client-server-with-TO`	184	5	12.8 s
`genericComputeServer`	57	4	4.6 s
`genericComputeServer-fctAsActor`	98	8	14.8 s
`liftChatLike`	1846	21	1830.9 s
`round_robin_2`	830	63	48.8 s
`round_robin_3`	3775	259	737.8 s

Table 1. Experimental results: the columns indicate the number of nodes in the Karp-Miller tree, the number of ideals in the covering set, and the running time.

cast pattern that is used in the original implementation has been changed into a polling pattern. The `round_robin_`k example is a load balancer that routes requests to a pool of k workers. Increasing the value of k greatly increase the number of interleavings that the analysis has to consider. With added support for collections, we can analyze a generic `round_robin_`k, which should also reduce the symmetry in the model.

Our experiments indicate that our analysis produces sufficiently precise approximations of the covering set to be useful for program verification and program understanding. The main bottle neck of our analysis is the explosion caused by interleavings of the transitions of individual processes. We did not yet explore techniques such as partial order reduction to tackle this problem.

7 Conclusion

We proposed a novel abstract interpretation framework to compute precise approximations of the covering set of WSTS. Our analysis captures the essence of acceleration-based algorithms that compute the exact covering set but only terminate on flattable systems. By replacing acceleration with widening we ensure that our analysis always terminates. We discussed several concrete instances of our framework including the application to depth-bounded process networks, which are typically non-flattable. Our experience with a prototype implementation shows that the analysis is often precise, which makes it a useful tool for verification and program analysis.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
2. P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *FMSD*, 25(1):39–65, 2004.
3. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *LICS*, pages 160–170, 1993.
4. T. Azzopardi. Generic compute server in Scala using remote actors. `http://tiny.cc/yjzva`, 2008. Accessed Nov 2011.

5. R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *Software Tools for Technology Transfer*, 8(4/5):449–466, 2006.
6. C. Calcagno, D. Distefano, P. W. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. In *POPL*, pages 289–300, 2009.
7. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282. ACM, 1979.
9. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
10. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, pages 103–115, 1998.
11. J. Engelfriet and T. Gelsema. Multisets and structural congruence of the pi-calculus with replication. *Theor. Comput. Sci.*, 211(1-2):311–337, 1999.
12. A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part I: Completions. In *STACS*, volume 09001 of *Dagstuhl Sem. Proc.*, pages 433–444, 2009.
13. A. Finkel and J. Goubault-Larrecq. Forward Analysis for WSTS, Part II: Complete WSTS. In *ICALP (2)*, pages 188–199, 2009.
14. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
15. P. Ganty, J.-F. Raskin, and L. V. Begin. A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In *VMCAI*, pages 49–64, 2006.
16. 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.
17. J. Goubault-Larrecq. On noetherian spaces. In *LICS*, pages 453–462. IEEE Computer Society, 2007.
18. P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theor. Comput. Sci.*, 410(2-3):202–220, 2009.
19. S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV*, pages 214–226, 2008.
20. R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.
21. Lift. Lift web framework. `http://liftweb.net/`.
22. R. Meyer. On boundedness in depth in the pi-calculus. In *IFIP TCS*, volume 273 of *IFIP*, pages 477–489. Springer, 2008.
23. E. C. Milner. Basic wqo- and bqo-theory. *Graphs and order*, 1985.
24. R. Milner. The polyadic pi-calculus: A tutorial. In *Logic and Algebra of Specification*, Computer and Systems Sciences. Springer, 1993.
25. C. A. Petri and W. Reisig. Scholarpedia, 3(4):6477. `http://www.scholarpedia.org/article/Petri_net`, 2008.
26. X. Rival and L. Mauborgne. The trace partitioning abstract domain. *ACM Trans. Program. Lang. Syst.*, 29(5), 2007.
27. P. Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In *MFCS*, pages 616–628, 2010.
28. T. Wies, D. Zufferey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *FoSSaCS 2010*, volume 4349 of *LNCS*, pages 94–108. Springer, 2010.
29. D. Zufferey and T. Wies. Picasso Analyzer. `http://ist.ac.at/~zufferey/picasso/`.
30. D. Zufferey, T. Wies, and T. A. Henzinger. On ideal abstractions for well-structured transition systems. Technical Report IST-2011-10, IST Austria, November 2011.