Is Lazy Abstraction a Decision Procedure for Broadcast Protocols?

Rayna Dimitrova[1] and Andreas Podelski[2]

[1] Universität des Saarlandes
[2] University of Freiburg

Abstract. Lazy abstraction builds up an abstract reachability tree by locally refining abstractions in order to eliminate spurious counterexamples in smaller and smaller subtrees. The method has proven useful to verify systems code. It is still open how good the method is as a decision procedure, i.e., whether the method terminates for already known decidable verification problems. In this paper, we answer the question positively for broadcast protocols and other infinite-state models in the class of so-called well-structured systems. This extends an existing result on systems with a finite bisimulation quotient.

1 Introduction

Lazy abstraction [1] is an interesting verification method that deserves a study on its own right. It defines the de-facto standard for verification based on the scheme coined *counterexample-guided abstraction refinement* (CEGAR) in [2]. While lazy abstraction has demonstrated its practical usefulness [3, 4], it is still open whether its practical performance is matched by its theoretical qualities.

In this paper, we investigate the suitability of lazy abstraction as a decision procedure. The general question is for what (already known) decidable verification problems the method is guaranteed to terminate. We give a positive answer for the case of so-called *well-structured systems*. This class, which contains broadcast protocols and other interesting infinite-state models, is well-investigated [5–9]. The corresponding verification problem (called *coverability*) is known to be decidable [10]. We prove that lazy abstraction is guaranteed to terminate for every well-structured system; i.e., lazy abstraction is a decision procedure for coverability.

The high-level formulation of lazy abstraction given in [1] specifies no control for building up the abstract reachability tree. Strictly speaking, our positive answer refers to a version of lazy abstraction with *control*. The control (for building up the abstract reachability tree) implements a breadth-first strategy. This corresponds to the default choice in the implementation of lazy abstraction, e.g., in the tool BLAST [1, 3].

We also give a negative answer. If the lazy abstraction is implemented by a non-deterministic algorithm (the control for choosing the branches for building up the abstract reachability tree is non-deterministic choice), then there exists an example of a well-structured system, an instance of the coverability problem and a sequence of non-deterministic choices that results in non-termination.

In summary, the contribution of the paper is the comprehensive answer to the question whether lazy abstraction is a decision procedure for broadcast protocols.

Related Work. Our result relates to two lines of work: the investigation of various notions of completeness for lazy abstraction and other CEGAR methods, and, respectively, the design of specific CEGAR methods as complete tests for well-structured systems.

Completeness of CEGAR for finite-state systems is established in [2]. Systems with finite trace equivalence include finite-state systems but also timed automata; lazy abstraction was shown complete for this class in [1]. The completeness proof exploits the termination guarantee of the finite-quotient construction in [11]. Weaker notions of completeness are investigated in [12] (for general CEGAR schemes) and in [13] (for lazy abstraction). The proofs employ combinatorial arguments which, again, are unrelated to the proof methods employed in this paper.

The design of two specific CEGAR methods as complete tests for well-structured systems is presented in [6]. The two methods differ from lazy abstraction. They lack its main characteristics, the incremental construction of an abstract reachability tree with localized abstraction refinement for subtrees. The proofs of the termination guarantee given in [6] and, respectively, in this paper, both rely on the property defining well-structured systems. In [6], the proofs are based on the saturation of the set of possible refinements. This is not sufficient here, already because refinements refer to the (a priori unbounded) number of subtrees in the abstract reachability tree.

To our knowledge, we are the first to investigate the logical intersection of the two lines of work described above: the question of whether lazy abstraction is already a complete test for well-structured systems.

2 Preliminaries

2.1 Well-Structured Systems and Coverability

Here we introduce the class of *well-structured transition systems*(WSS) following the conceptual frameworks from [5] and [10]. The preorder $\preceq$ on a set S is called a *well-quasi order* if for every infinite sequence $s_0, s_1, s_2, \ldots$ of elements of S, there exist indices i and j such that i is strictly less than j and $s_i \preceq s_j$. Let $(S, \preceq)$ be a well-quasi ordered set. A subset A of S is *upward-closed* if for every element s of A and for every element t of S such that $s \preceq t$, it holds that t is an element of A. The existence of such an order on the set of states of an infinite-state system, combined with some compatibility property of the transition relation with respect to this order, guarantees the termination of certain fixpoint computations.

A *labeled transition system* $\mathcal{S}$ is a tuple $\langle S, I, C, \delta \rangle$ where S is a possibly infinite set of states, $I \subseteq S$ is a set of initial states, C is a finite set of labels and

$\delta \subseteq S \times C \times S$ is a labeled transition relation. Transition systems are usually represented symbolically using formulas over some set of atoms. Let $\mathcal{V} = \mathcal{X} \cup \mathcal{X}'$ be a set of variables. The set $\mathcal{X}'$ consists of the primed versions of the variables in $\mathcal{X}$. Let $\mathcal{AP}$ be a fixed infinite set of atomic formulas over the variables from $\mathcal{V}$. The language $\mathcal{L}$ is the closure of $\mathcal{AP}$ under Boolean connectives. For a finite subset P of $\mathcal{AP}$, we write $\mathcal{L}(P)$ for the closure of P under Boolean connectives. The set of atomic formulas that appear in a formula φ we denote with $\mathsf{atoms}(\varphi)$. A program is specified by a tuple $\langle X, \mathsf{init}, D\rangle$ where $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ is a finite set of program variables (including program counters), each of which is associated with a domain, $\mathsf{init}(X)$ is a formula that denotes the set of initial states, and D is a set of guarded commands that describes the transition relation. Each guarded command is of the form $c_i : g_i(X) \wedge x'_1 = e^i_1(X) \wedge \ldots \wedge x'_n = e^i_n(X)$ where c_i is the label of the command, g_i is the guard and the other conjuncts are the updates of the variables (the primed variables stand for the next-state program variables). With each program $\mathcal{S} = \langle X, \mathsf{init}, D\rangle$ we associate a transition system $\mathcal{S} = \langle S, I, C, \delta\rangle$, which describes its semantics, in the usual way. Each formula φ over the variables in X denotes a set of states: the states in which φ evaluates to true. From now on, we identify formulas over the variables in X and the sets of states denoted by them. The symbolic operators post and pre that map a label of a guarded command and a set of states to a set of states are defined in the usual way.

Consider a transition system $\mathcal{S}$ equipped with some well-quasi order $\preceq$ on the set of states S. Let the transition relation satisfy the following notion of *strong compatibility w.r.t. the labeled transitions*, which is the *strong compatibility* notion from [5]. That is: for every three states s_1, s_2 and t_1 such that $s_1 \preceq t_1$ and $(s_1, c, s_2) \in \delta$ for some label $c \in C$, there exists a state t_2 such that $s_2 \preceq t_2$ and $(t_1, c, t_2) \in \delta$. Such labeled transition systems we call *well-structured systems.*

Let $\mathcal{S}$ be a WSS and the formula unsafe denote a set of error states which is upward-closed w.r.t. the corresponding order $\preceq$. The *coverability problem* is to check whether the upward-closed set of error states is reachable in $\mathcal{S}$. This problem is known to be decidable for WSS [10].

2.2 Predicate Abstraction

We define abstraction and concretization functions w.r.t. some finite set of predicates P in the usual way. Let $\models$ denote entailment modulo some fixed theory. The abstraction function α_P is parameterized by the finite set of predicates P. It maps a formula φ to the smallest w.r.t. $\models$ formula over P that is greater than φ w.r.t. $\models$, formally, $\alpha_P(\varphi) = \mu_{\models}\psi \in \mathcal{L}(P).\varphi \models \psi$. The concretization function γ_P is defined to be the identity. The functions α_P and γ_P form a Galois connection. Hence, they are monotone and $\varphi \models \gamma_P(\alpha_P(\varphi))$. If $\mathcal{L}(P)$ contains a formula that is equivalent to a formula φ, then the abstraction of φ is equivalent to φ. If P is a subset of the finite set of predicates Q then for every formula φ it holds that $\gamma_Q(\alpha_Q(\varphi)) \models \gamma_P(\alpha_P(\varphi))$.

2.3 Lazy Abstraction

We recall the *lazy abstraction* algorithm(LA) from [1]. The algorithm iteratively explores the abstract state space by constructing an abstract reachability tree. If it terminates, it returns either a *genuine counterexample* (a path in the concrete system from an initial to an error state) or an overapproximation of the set of reachable states whose intersection with the set of error states is empty. In order to simplify the presentation, in this section we present the general scheme of lazy abstraction. In the next section we present in more detail a particular instantiation of that scheme and then state our contribution.

We describe an algorithm scheme LA[$\star$] parameterized by: (1) the strategy for exploring the abstract state space, namely the operator $\mathsf{choose\text{-}element}$ used to select the node of the abstract reachability tree that is going to be processed, (2) the predicate $\mathsf{covered}$ that determines whether the subtree below a node can be discarded, (3) the abstract operator $\widehat{\mathsf{post}}$ used to compute the regions of the nodes in the tree and (4) the operator Φ that is used to select the refinement predicates.

Each edge in the abstract reachability tree is labeled by a label of a guarded command. The finite sequences of labels of guarded commands we call *traces*. We characterize a node $\mathbf{n}$ by the trace σ that labels the path from the root to $\mathbf{n}$. Each node $\mathbf{n}$ in the tree is labeled by a pair (φ, P) called *region*, written $\mathbf{n}{:}(\varphi, P)$. P is a finite set of predicates over the variables in X and φ is a Boolean formula over P. The formula φ, which we call the *reachable region* of $\mathbf{n}$, denotes an overapproximation of the set of states reachable via the corresponding trace σ. If the conjunction of φ and the formula unsafe is satisfiable then $\mathbf{n}$ is an *error node*, otherwise we call it a *safe node*.

The procedure constructs a sequence of trees. We denote the current tree at iteration k with T_k. The initial tree T_0 consists of a single node $\mathbf{r}$ labeled with the region (init, P_0), where P_0 consists of the atoms that appear in the formula init and the atoms that appear in the formula unsafe. Each node in the current tree has a mark that can be one of the following: *unprocessed*, *covered* or *uncovered*. The list L consists of all nodes in the current tree that are marked as unprocessed. These are nodes that have been added to the tree but have not been processed yet. At each step the algorithm chooses a node $\mathbf{n}$ from L, unless L is empty, and deletes it from the list. Then it processes the node $\mathbf{n}$ and constructs the next tree T_{k+1} as explained below or returns a counterexample. If L is empty, the algorithm terminates with the formula *Reach* as a result, where the formula *Reach* is defined to be the disjunction of the reachable regions of all nodes in the current tree that are marked as uncovered.

When $\mathbf{n}$ is a safe node, the algorithm LA[$\star$] proceeds as follows. If $\mathbf{n}$ should be marked as covered, i.e., $\mathsf{covered}(\varphi)$ is true for the reachable region φ of $\mathbf{n}$ (this holds if φ is contained in the disjunction of the reachable regions of some of the nodes in the current tree that are marked as uncovered), then $\mathbf{n}$ is marked as covered and its children are not generated. Otherwise, it is marked as uncovered and for each command c, the algorithm does the following. If $\mathsf{post}(c, \varphi)$ is not empty, it adds a new node as a child of $\mathbf{n}$ and labels it with the region $\widehat{\mathsf{post}}(c, (\varphi, P))$.

Algorithm 1: LA[$\star$]

```
Input: a program $\mathcal{S}$ , a formula unsafe
Output: either "CORRECT" and a formula $\theta$ or
"NOT CORRECT" and counterexample $\sigma$
$P_0 := \mathsf{atoms}(\mathsf{init}) \cup \mathsf{atoms}(\mathsf{unsafe})$;
$T$ consists of a single node $\mathbf{r}$:$(\mathsf{init}, P_0)$;
$L := \{\mathbf{r}\}$; $Reach = \mathsf{false}$;
repeat
    $\mathbf{n}$:$(\varphi, P)$ :=choose–element$(L)$;
    remove $\mathbf{n}$:$(\varphi, P)$ from $L$;
    if $\varphi \models \neg\mathsf{unsafe}$ then
        if covered$(\varphi)$ then
            mark $\mathbf{n}$ as covered
        else
            mark $\mathbf{n}$ as uncovered;
            $Reach := Reach \vee \varphi$;
            forall $c \in C$ do
                if $\mathsf{post}(c, \varphi) \neq \emptyset$ then
                    add $\mathbf{m}$:$\widehat{\mathsf{post}}(c, (\varphi, P))$ as a child of $\mathbf{n}$ in $T$;
                    label the edge from $\mathbf{n}$ to $\mathbf{m}$ with $c$;
                    mark $\mathbf{m}$ as unprocessed and add $\mathbf{m}$ to $L$
                end
            end
        end
    else
        $\mathbf{m}$:$(\psi, Q)$ is the pivot node for $\mathbf{n}$;
        if $m = \perp$ then
            return ("NOT CORRECT",the trace from the root to $\mathbf{n}$)
        else
            $\tau$ is the trace from $\mathbf{m}$ to $\mathbf{n}$;
            relabel $\mathbf{m}$ with $(\psi, Q \cup \Phi(\psi, \tau, \mathsf{unsafe}))$;
            mark $\mathbf{m}$ as unprocessed and add it to $L$;
            delete the subtrees that start from the children of $\mathbf{m}$;
            all nodes that were marked as covered after the last time $\mathbf{m}$ was
            processed are marked as unprocessed and added to $L$;
            $Reach := \bigvee_{\mathbf{n}':(\varphi', P'):\text{uncovered}} \varphi'$
        end
    end
until $L = \emptyset$ ;
return ("CORRECT", $Reach$)
```

The edge from $\mathbf{n}$ to the new node is labeled with c. All the children of $\mathbf{n}$ are marked as unprocessed and added to the list L.

When the processed node $\mathbf{n}{:}(\varphi, P)$ is an error node, the procedure analyzes the abstract counterexample backwards. The *error region* for a trace τ is defined as $\mathsf{pre}(\tau, \mathsf{unsafe})$. For each node $\mathbf{n}'$ on the path, LA[$\star$] computes the error region for the trace from $\mathbf{n}'$ to $\mathbf{n}$ until it finds the first (in backward direction) node on the path, for which the conjunction of the corresponding reachable and error regions is unsatisfiable. This is the *pivot node* $\mathbf{m}{:}(\psi, Q)$. Then, $\mathbf{m}$ *is refined w.r.t. the trace* τ, where τ is the *error trace* – the sequence of the labels of the edges on the path from $\mathbf{m}$ to $\mathbf{n}$. The set of predicates of the pivot node is enhanced with the predicates in $\Phi(\psi, \tau, \mathsf{unsafe})$. The subtrees that start at the children of the pivot node $\mathbf{m}$ are deleted, $\mathbf{m}$ is marked as unprocessed and so are all nodes marked as covered after $\mathbf{m}$ was last processed.

Provided that the operator $\widehat{\mathsf{post}}$ fulfills the requirement: if $\widehat{\mathsf{post}}(c, (\varphi, P)) = (\varphi', P')$, then $P' = P$ and $\mathsf{post}(c, \varphi) \models \varphi'$, the following two properties of the labels of the nodes are direct consequences of the construction of the sequence of trees.

Property 1. Let $\mathbf{n}{:}(\varphi, P)$ be a node in the tree T_i. Let j be an index greater or equal to i such that for every $i \leq k \leq j$, the node $\mathbf{n}$ is not deleted from the tree T_k. Let $\mathbf{m}{:}(\psi, Q)$ be a node in T_j that is in the subtree rooted at $\mathbf{n}$. Then, it holds that $P \subseteq Q$.

Property 2. Let $\mathbf{n}{:}(\varphi, P)$ and $\mathbf{m}{:}(\psi, Q)$ be two nodes in some tree T_i such that $\mathbf{m}$ is in the subtree rooted at $\mathbf{n}$. Let σ be the sequence of labels on the path from $\mathbf{n}$ to $\mathbf{m}$. The set denoted by ψ is an overapproximation of the set of states that can be reached from a state in φ by executing the sequence of commands σ, formally, $\mathsf{post}(\sigma, \varphi) \models \psi$.

3 Lazy Abstraction with Breadth-First Strategy: LA[BF]

We obtain the procedure LA[BF] by instantiating the algorithm scheme LA[$\star$]. In particular, we impose more control on the abstract state-space exploration by fixing the search strategy. We restrict the non-deterministic choice, which node to be processed at the current iteration, to the set $\mathsf{min\text{-}depth}(L)$ of unprocessed nodes with minimal depth. We instantiate $\mathsf{choose\text{-}element}$ with the operator $\mathsf{pick\text{-}min\text{-}element}$, which selects non-deterministically an element of $\mathsf{min\text{-}depth}(L)$. This amounts to a breadth-first exploration of the abstract reachability tree. We mark a safe node $\mathbf{n}$ with reachable region φ as covered, i.e., $\mathsf{covered}(\varphi)$ is true, exactly when $\varphi \models \mathit{Reach}_{\mathbf{n}}$, where the formula $\mathit{Reach}_{\mathbf{n}}$ is defined to be the disjunction of the reachable regions of the nodes in the current tree with depth less than the depth of $\mathbf{n}$ that are marked as uncovered. The parameter $\widehat{\mathsf{post}}$ is instantiated with the abstract post operator $\mathsf{post}^{\#}$, which is defined as $\mathsf{post}^{\#}(c, (\varphi, P)) = (\alpha_P(\mathsf{post}(c, \varphi)), P)$.

Let $\mathbf{m}{:}(\psi, Q)$ be the pivot node that is to be refined by the procedure and τ be the corresponding error trace. We define the focus operator Φ, which determines the refinement predicates for the error trace τ, as follows. For a trace τ and indices $1 \leq i \leq j \leq |\tau| + 1$, we denote with $\tau[i, j)$ the subword of τ that starts at position i and ends at position $j - 1$ (including the $j - 1$-th element). Then $\Phi(\psi, \tau, \mathsf{unsafe}) = \bigcup_{i=1}^{|\tau|+1} \mathsf{atoms}(\mathsf{pre}(\tau[i, |\tau| + 1), \mathsf{unsafe}))$. The refinement consists in enhancing the set of predicates of the pivot node $\mathbf{m}$ and deleting the subtrees that originate from its children.

At each iteration LA[BF] processes an unprocessed node with minimal depth. Therefore, the refinement satisfies the following property.

Lemma 1. *Let $\boldsymbol{n}$ be the node that is processed by the procedure in the tree T_i at some iteration i. Let σ be a trace and $\boldsymbol{m}$:(φ, P) be a node in T_i such that the sum of the length of σ and the depth of $\boldsymbol{m}$ is strictly less than the depth of the node $\boldsymbol{n}$. Then, $\varphi \wedge \mathsf{pre}(\sigma, \mathsf{unsafe})$ is not satisfiable.*

Proof. The proof goes by induction on the length of σ.

Base case. The length of σ is 0. If we assume that for some node $\mathbf{m}$:(φ, P) with the stated property, the formula $\varphi \wedge \mathsf{pre}(\sigma, \mathsf{unsafe})$ is satisfiable, then $\mathbf{m}$ is an error node. Hence, it cannot be marked as covered or uncovered. Therefore, it is marked as unprocessed. Hence, in T_i there is an unprocessed node with depth strictly less than the depth of the node $\mathbf{n}$. This is a contradiction to the fact that $\mathbf{n}$ is a node of minimal depth marked as unprocessed.

Induction step. Let σ be of the form $c \cdot \sigma'$. By the induction hypothesis, for every node $\mathbf{r}$, such that the sum of the depth of $\mathbf{r}$ and the length of σ' is strictly less than the depth of $\mathbf{n}$, the conjunction of the reachable region of $\mathbf{r}$ and the error region for σ' is not satisfiable. Assume for contradiction that for a node $\mathbf{m}$:(φ, P), the conjunction $\varphi \wedge \mathsf{pre}(\sigma, \mathsf{unsafe})$ is satisfiable and the sum of the depth of $\mathbf{m}$ and the length of σ is strictly less than the depth of $\mathbf{n}$. Then, there is a state s that satisfies this conjunction. Hence, there is a state t, such that there is a transition labeled by c from s to t and t is an element of $\mathsf{pre}(\sigma', \mathsf{unsafe})$.

Since the depth of the node $\mathbf{m}$ is strictly less than the depth of the node $\mathbf{n}$, which the procedure processes in the current iteration, $\mathbf{m}$ should be marked either as covered or as uncovered. If we assume that $\mathbf{m}$ is marked as covered, then there exists a node $\mathbf{m}'$ with reachable region ψ' in T_i that is marked as uncovered, has depth less than the depth of the node $\mathbf{m}$ and is such that the state s satisfies ψ'. As t satisfies $\mathsf{post}(c, \psi')$, there is a node $\mathbf{m}''$ in T_i that is a child of $\mathbf{m}'$ and the edge between $\mathbf{m}'$ and $\mathbf{m}''$ is labeled by c. Let the reachable region of $\mathbf{m}''$ be ψ''. It contains the state t. If, on the other hand, we assume that $\mathbf{m}$ is marked as uncovered, there is a node $\mathbf{m}''$ in T_i that is a child of $\mathbf{m}$ and the edge between $\mathbf{m}$ and $\mathbf{m}''$ is labeled by c. If ψ'' is the reachable region of $\mathbf{m}''$, then the state t satisfies ψ''.

Hence, in both cases there is a node $\mathbf{m}''$ such that the state t belongs to its reachable region ψ'' and the sum of the depth of $\mathbf{m}''$ and the length of σ' is less than or equal to the sum of the depth of $\mathbf{m}$ and the length of σ which is strictly less than the depth of $\mathbf{n}$. Therefore, we can apply the induction hypothesis, which

Algorithm 2: LA[BF]

```
Input: a program $\mathcal{S}$ , a formula $\mathsf{unsafe}$
Output: either “CORRECT” and a formula $\theta$ or
“NOT CORRECT” and counterexample $\sigma$
$P_0 := \mathsf{atoms}(\mathsf{init}) \cup \mathsf{atoms}(\mathsf{unsafe})$;
$T$ consists of a single node $\mathbf{r}$:$(\mathsf{init}, P_0)$;
$L := \{\mathbf{r}\}$; $Reach = \mathsf{false}$;
repeat
    $\mathbf{n}$:$(\varphi, P)$ :=$\mathsf{pick\text{-}min\text{-}element}(L)$;
    remove $\mathbf{n}$:$(\varphi, P)$ from $L$;
    if $\varphi \models \neg\mathsf{unsafe}$ then
        if $\varphi \models Reach_{\mathbf{n}}$ then
            mark $\mathbf{n}$ as covered
        else
            mark $\mathbf{n}$ as uncovered;
            $Reach := Reach \vee \varphi$;
            forall $c \in C$ do
                if $\mathsf{post}(c, \varphi) \neq \emptyset$ then
                    add $\mathbf{m}$:$\mathsf{post}^{\#}(c, (\varphi, P))$ as a child of $\mathbf{n}$ in $T$;
                    label the edge from $\mathbf{n}$ to $\mathbf{m}$ with $c$;
                    mark $\mathbf{m}$ as unprocessed and add $\mathbf{m}$ to $L$;
                end
            end
        end
    else
        $\mathbf{m}$:$(\psi, Q)$ is the pivot node for $\mathbf{n}$;
        if $m = \perp$ then
            return (“NOT CORRECT”,the trace from the root to $\mathbf{n}$)
        else
            $\tau$ is the trace from $\mathbf{m}$ to $\mathbf{n}$;
            $Q' := \bigcup_{i=1}^{|\tau|+1} \mathsf{atoms}(\mathsf{pre}(\tau[i, |\tau|+1), \mathsf{unsafe}))$;
            relabel $\mathbf{m}$ with $(\psi, Q \cup Q')$;
            mark $\mathbf{m}$ as unprocessed and add it to $L$;
            delete the subtrees that start from the children of $\mathbf{m}$;
            all nodes that were marked as covered after the last time $\mathbf{m}$ was
            processed are marked as unprocessed and added to $L$;
            $Reach := \bigvee_{\mathbf{n}':(\varphi', P'):\text{uncovered}} \varphi'$
        end
    end
until $L = \emptyset$ ;
return (“CORRECT”, $Reach$)
```

yields that the intersection of ψ'' and $\mathsf{pre}(\sigma', \mathsf{unsafe})$ is empty. This contradicts to the fact that the state t is an element of both of them. This concludes the proof. $\square$

4 LA[BF] is a Decision Procedure

From now on, we assume that the program that is given as input to the procedure LA[BF] denotes a WSS $\mathcal{S}$ with a well-quasi order $\preceq$ and that the set of error states unsafe is upward-closed with respect to this order. We show that in this case the procedure LA[BF] is guaranteed to terminate.

It is easy to see that if the number of performed refinement operations is finite, then the procedure terminates. This is because with finitely many predicates we can generate only finitely many non-equivalent regions. To show that the number of iterations, at which the procedure performs a refinement, is finite, we prove that the following two properties hold. First, we prove that each particular node cannot be refined as a pivot node infinitely often. Then, we show that it is also impossible that the procedure refines infinitely many distinct pivot nodes. In both cases the proof is by contradiction. We assume that the property under consideration does not hold and show the existence of an infinite sequence of states that are pairwise incomparable with respect to the order $\preceq$. This can not be true because $\preceq$ is a well-quasi order. To show the existence of such a sequence, we make use of the fact that for each trace σ the set $\mathsf{pre}(\sigma, \mathsf{unsafe})$ is upward-closed when unsafe is upward-closed. We first show several lemmas that we use for proving the two main properties of LA[BF]. We begin with a lemma that states the progress property of the refinement. Recall that if at the i-th iteration, $\mathbf{n}$ is the pivot node and σ is the sequence of labels on the path from $\mathbf{n}$ to the corresponding error node that is processed in the current tree T_i, we say that $\mathbf{n}$ is refined w.r.t. σ in T_i.

Lemma 2. *Let the node $\boldsymbol{n}$:(φ, P) be refined in the tree T_i w.r.t. the trace σ. Let j be an index greater than i such that for every index k with $i \leq k \leq j$, the node $\boldsymbol{n}$ is not deleted from the tree T_k. Assume that the node $\boldsymbol{m}$:(ψ, Q) is in the subtree rooted at $\boldsymbol{n}$ in T_j, $\psi \models \neg\mathsf{pre}(\sigma, \mathsf{unsafe})$, the node $\boldsymbol{m}'$:(ψ', Q') is in the subtree rooted at $\boldsymbol{m}$ and the path from $\boldsymbol{m}$ to $\boldsymbol{m}'$ is labeled by $\sigma[1, l)$ for some l, i.e., with a prefix of σ. Then, $\psi' \models \neg\mathsf{pre}(\sigma[l, |\sigma|+1), \mathsf{unsafe})$.*

Proof. The proof goes by induction. For l such that $1 \leq l \leq |\sigma|+1$, let $\mathbf{m}_l$:(ψ_l, Q_l) be the node in T_j such that the path from $\mathbf{m}$ to $\mathbf{m}_l$ is labeled by $\sigma[1, l)$, if such nodes exists in T_j. For every l we show that if $1 \leq l \leq |\sigma|+1$ then $\psi_l \models \neg\mathsf{pre}(\sigma[l, |\sigma|+1), \mathsf{unsafe})$. For $l = 1$ we have $\psi_l = \psi$ and $\psi \models \neg\mathsf{pre}(\sigma[1, |\sigma|+1), \mathsf{unsafe})$. Let $l+1 \leq |\sigma|+1$. By induction hypothesis we have that $\psi_l \models \neg\mathsf{pre}(\sigma[l, |\sigma|+1), \mathsf{unsafe})$. This yields $\mathsf{post}(\sigma[l], \psi_l) \models \neg\mathsf{pre}(\sigma[l+1, |\sigma|+1), \mathsf{unsafe})$. Since by Property 1 $\mathsf{atoms}(\mathsf{pre}(\sigma[l+1, |\sigma|+1), \mathsf{unsafe})) \subseteq Q_l$, we have that $\alpha_{Q_l}(\mathsf{post}(\sigma[l], \psi_l)) \models \neg\mathsf{pre}(\sigma[l+1, |\sigma|+1), \mathsf{unsafe})$. Thus, $\psi_{l+1} \models \neg\mathsf{pre}(\sigma[l+1, |\sigma|+1), \mathsf{unsafe})$. $\square$

This lemma implies that once a node $\mathbf{n}$ is refined w.r.t. a trace σ, no node in the subtree rooted at $\mathbf{n}$ will be refined w.r.t. the same trace at the next iterations, provided that $\mathbf{n}$ is not deleted meanwhile.

Lemma 3. *Let the node $\boldsymbol{n}$ be refined in the tree T_i w.r.t. the trace σ. Assume that for some index j strictly greater than i, it holds that for every index k with $i \leq k \leq j$, the node $\boldsymbol{n}$ is not deleted from the tree T_k. Then, for every node $\boldsymbol{m}$ that is in the subtree of $\boldsymbol{n}$ in the tree T_j, it holds that $\boldsymbol{m}$ cannot be refined in the tree T_j w.r.t. the trace σ.*

Proof. Let the label of the node $\mathbf{m}$ in T_j be (ψ, Q). Assume that $\mathbf{m}$ is refined in T_j w.r.t. the trace σ. Therefore, $\psi \models \neg\mathsf{pre}(\sigma, \mathsf{unsafe})$. Also, an error node $\mathbf{m}'$:(ψ', Q') is processed in T_j and the path from $\mathbf{m}$ to $\mathbf{m}'$ is labeled with σ. According to Lemma 2, $\psi' \models \neg\mathsf{pre}(\sigma[|\sigma|+1, |\sigma|+1), \mathsf{unsafe})$, i.e. $\psi' \models \neg\mathsf{unsafe}$. This contradicts to the fact that $\mathbf{m}'$ is an error node. $\square$

Now we are ready to prove that every node can be the refined as pivot node at only finitely many iterations.

Proposition 1. *A node $\boldsymbol{n}$ cannot be refined by LA[BF] infinitely often.*

Proof. We first show that if some node is not deleted infinitely often from the tree, then it cannot be refined infinitely often. Assume that the node $\mathbf{n}$ is refined infinitely often and deleted only finitely often. Thus, there is an infinite sequence of trees $T_{k_0}, T_{k_1}, \ldots$ such that the node $\mathbf{n}$ is refined in each tree T_{k_i} w.r.t. some trace σ_i and for every index k greater or equal to k_0, the node $\mathbf{n}$ is not deleted from the tree T_k. According to Lemma 3, all traces in the sequence $\sigma_0, \sigma_1, \ldots$ must be pairwise different. Since the set of labels C is finite, w.l.o.g. we can assume that for some label c, for each index i, the first element of the trace σ_i is exactly c. Let τ_i be the trace obtained from the trace σ_i by removing its first element, and the node $\mathbf{m}$ be the child of $\mathbf{n}$ with edge from $\mathbf{n}$ to $\mathbf{m}$ labeled with c. Let the sequence of formulas $\psi_0, \psi_1, \ldots$ consist of the reachable regions of the node $\mathbf{m}$ in the trees $T_{k_0}, T_{k_1}, \ldots$ respectively. For every index i, the conjunction $\psi_i \wedge \mathsf{pre}(\tau_i, \mathsf{unsafe})$ is satisfiable because the node $\mathbf{n}$ is refined in the tree T_{k_i} w.r.t. the trace σ_i. So, for each index i we can choose a state s_i that satisfies $\psi_i \wedge \mathsf{pre}(\tau_i, \mathsf{unsafe})$. Since $\preceq$ is a well-quasi order, for the sequence of states $s_0, s_1, \ldots$, there exist indices i and j such that i is strictly smaller than j and $s_i \preceq s_j$. The set of states $\mathsf{pre}(\tau_i, \mathsf{unsafe})$ is upward-closed. Therefore, the state s_j is an element of the set $\mathsf{pre}(\tau_i, \mathsf{unsafe})$. According to Lemma 2, the intersection of the sets ψ_j and $\mathsf{pre}(\tau_i, \mathsf{unsafe})$ is empty. This contradicts to the fact that the state s_j is an element of both sets.

It now remains to show that a node cannot be deleted infinitely often. The proof is by induction on the depth of the node. The root node is never deleted. Consider a node $\mathbf{n}$ different from the root. By the induction hypothesis we have that each of the nodes on the path from the root to the node $\mathbf{n}$ is deleted only finitely many times. Hence, as we showed already, each of these nodes can be refined only a finite number of times. Since a node is deleted only when some node on the path from the root to this node is refined, $\mathbf{n}$ can be deleted only finitely many times. □

To show that it is not possible that the procedure refines infinitely many different pivot nodes we need the next lemma, which states that if this is the case then we can construct an infinite sequence of different nodes that are refined by the procedure with the property that they belong to the same branch.

Lemma 4. *If the procedure refines infinitely many pivot nodes, then there is an infinite sequence $T_{k_0}, T_{k_1}, \ldots$ of trees and an infinite sequence $\mathbf{n}_0, \mathbf{n}_1, \ldots$ of corresponding nodes in those trees, such that the following two conditions hold. In the tree $T_{k_{i+1}}$ the node $\mathbf{n}_{i+1}$ is in the subtree rooted at $\mathbf{n}_i$. Each node $\mathbf{n}_i$ is refined in the tree T_{k_i} and for each index k strictly greater than k_i, $\mathbf{n}_i$ is not deleted from or refined in T_k.*

Proof. Assume that the procedure refines infinitely many different pivot nodes. Since the number of children of each node is bounded by $|C|$ and no node is deleted infinitely often, there exists a sequence $T_{l_0}, T_{l_1}, \ldots$ of trees and a sequence $\mathbf{m}_0, \mathbf{m}_1, \ldots$ of corresponding nodes such that $\mathbf{m}_{i+1}$ is a child of the node $\mathbf{m}_i$ in the tree $T_{l_{i+1}}$. As we showed, each of these nodes is refined as a pivot node at only finitely many iterations. If we assume that only finitely many of them are refined, it follows that there exists an index j, such that $\mathbf{m}_j$ is subsumed by some $\mathbf{m}_i$ with index $i < j$, which is not possible since $\mathbf{m}_{j+1}$ is a child of $\mathbf{m}_j$. Thus, infinitely many among those nodes are refined. The fact that a node cannot be refined infinitely often implies that there is an infinite sequence $T_{k_0}, T_{k_1}, \ldots$ of trees and a corresponding subsequence $\mathbf{n}_0, \mathbf{n}_1, \ldots$ of $\mathbf{m}_0, \mathbf{m}_1, \ldots$ such that each node $\mathbf{n}_i$ is refined in the tree T_{k_i} and for each index k strictly greater than k_i, $\mathbf{n}_i$ is not deleted from or refined in T_k. This concludes the proof. □

What remains now is to show the second property of the refinement, which is stated below.

Proposition 2. *The procedure LA[BF] refines only finitely many pivot nodes.*

Proof. Assume that the procedure refines infinitely many pivot nodes and consider an infinite sequence $\mathbf{n}_0, \mathbf{n}_1, \ldots$ of different nodes and an infinite sequence $T_{k_0}, T_{k_1}, \ldots$ of trees that satisfy the conditions stated in Lemma 4. Let σ_i be the trace, with respect to which the node $\mathbf{n}_i$ is refined in the tree T_{k_i}. As every node $\mathbf{n}_i$ in the sequence is not deleted in any tree T_k with index k greater than k_i, by Lemma 3 the traces $\sigma_0, \sigma_1, \ldots$ are pairwise different. As the set of labels C is finite, we can assume w.l.o.g. that we have chosen the sequences in a way that each trace has length strictly less than the length of the next. For each trace σ_i, we denote with τ_i the trace obtained from σ_i by removing its first element. Let the sequence of nodes $\mathbf{m}_0, \mathbf{m}_1, \ldots$ be such that each node $\mathbf{m}_i$ is the child of the node $\mathbf{n}_i$ in the tree T_{k_i} with the edge between them labeled with the first element of σ_i. Let the formulas φ_i and ψ_i be the reachable regions in the tree T_{k_i} of the nodes $\mathbf{n}_i$ and $\mathbf{m}_i$ respectively. The intersection of ψ_i and $\mathsf{pre}(\tau_i, \mathsf{unsafe})$ is not empty since the node $\mathbf{n}_i$ is refined in the tree T_{k_i} w.r.t. σ_i. Therefore, there exists a sequence of states $s_0, s_1, \ldots$ such that each state s_i is an element of the corresponding intersection $\psi_i \wedge \mathsf{pre}(\tau_i, \mathsf{unsafe})$. Since $\preceq$ is a well-quasi order, there exist indices i and j such that $i < j$ and $s_i \preceq s_j$. The set of states $\mathsf{pre}(\tau_i, \mathsf{unsafe})$ is upward-closed. Therefore, as the state s_i is an element of this set and $s_i \preceq s_j$, s_j is also an element of $\mathsf{pre}(\tau_i, \mathsf{unsafe})$. The node $\mathbf{n}_j$ is refined in the tree T_{k_j} w.r.t. the trace σ_j. Hence, a node $\mathbf{n}$ with depth equal to the sum of the depth of $\mathbf{m}_j$ and the length of τ_j is processed in this tree. We can apply Lemma 1 to the node $\mathbf{n}$, the node $\mathbf{m}_j$ and the trace τ_i, because the length of the trace τ_i is strictly less than the length of the trace τ_j. Thus, for the reachable region of the node $\mathbf{m}_j$ it holds that the intersection of ψ_j and $\mathsf{pre}(\tau_i, \mathsf{unsafe})$ is empty. This contradicts to the fact that the state s_j is an element of both these sets. This completes the proof by contradiction. $\square$

Finally, the two propositions yield our main result.

Theorem 1. *The procedure LA[BF] is a decision procedure for the coverability problem for WSS.*

5 LA[$\star$] is not a Decision Procedure

We use LA[$\star$] to refer to lazy abstraction with completely non-deterministic control for building up the the abstract reachability tree. In this section we give an example of a system and a sequence of non-deterministic choices that results in non-termination.

Consider the program given below that has three variables x, y and z, each of which ranges over $\mathbb{N}$. The guarded commands are given in Table 1 (we use ”syntactic sugar” and list only the ”true” updates). The set of initial states is given by $x = 0 \wedge y = 0 \wedge z = 0$ and the set of error states is denoted by $z \geq 2$. The order $\preceq$ between the states of the corresponding transition system is the pointwise ordering between the elements of $\mathbb{N}^3$ defined by $\leq$. It is a well-quasi order according to Dickson's lemma [14]. The resulting transition system $\mathcal{S}$ equipped with the order $\preceq$ is a WSS and the set of error states is upward-closed w.r.t. this order. The system $\mathcal{S}$ is safe.

Table 1. Guarded commands

Label	**Guard**	**Update**
c_1	true	$y := 1$
c_2	$y > 1$	$x := 1 \wedge y := 0$
c_3	$x > 0$	$x := x + 1$
c_4	$x > 0$	$x := x - 1 \wedge y := y + 1$
c_5	true	$z := x$
c_6	$x > 0$	$y := y + 1$
c_7	$y > 1$	$y := y - 1 \wedge x := x + 1$
c_8	true	$z := y$

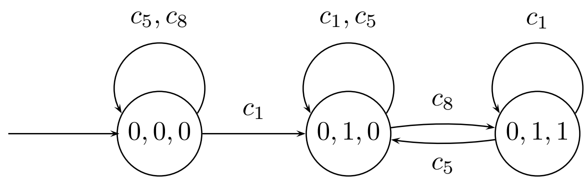
$$\begin{tikzcd}
{} \arrow[r] & 0,0,0 \arrow[loop above, "c_5, c_8"] \arrow[r, "c_1"] & 0,1,0 \arrow[loop above, "c_1, c_5"] \arrow[r, bend left, "c_8"] & 0,1,1 \arrow[loop above, "c_1"] \arrow[l, bend left, "c_5"]
\end{tikzcd}$$

Fig. 1. Transition system $\mathcal{S}$

If we execute LA[$\star$] with a sequence of non-deterministic choices that never refines the root node as pivot node, then it creates longer and longer counterexamples (and refines pivot nodes deeper and deeper in the abstract reachability tree). It adds predicates of the form $x \geq 2,\ x \geq 3, \ldots$ and $y \geq 2,\ y \geq 3, \ldots$. Thus, at each iteration it generates more and more non-subsumed regions. Since it never refines the root, the initial overapproximation that occurs by approximating $y = 1$ with $y \neq 0$ in the abstract execution of c_1 is never eliminated. Thus, the sequence of non-deterministic choices results in a non-terminating run of the iterative refinement.

It is instructive to follow the execution of LA[BF] on this example. The initial set of abstract predicates is $P_0 = \{z \geq 2, x = 0, y = 0, z = 0\}$. After a few iterations the procedure LA[BF] refines the root node as pivot node w.r.t. the trace $c_1c_2c_3c_5$ and adds the atoms $y > 1$ and $x \geq 2$ to the set of predicates for the root. After that there are no more abstract counterexamples and the procedure LA[BF] terminates.

6 Conclusion

An abstraction-based algorithm trades higher efficiency with the loss of definiteness. It may return "Don't Know" answers in some cases; it implements only a semi-test. A procedure based on counterexample-guided abstraction refinement trades a higher degree of automation with the loss of the termination guarantee. It may iterate forever without hitting a ("non-spurious") counterexample nor proving its absence; it implements only a semi-algorithm. Lazy abstraction goes one step further towards trading a potential practical gain with the risk of theoretical deficiencies (since it avoids redundant computations of abstract subtrees by localizing refinements, with the risk of having to generate the same predicate infinitely often). It is thus perhaps surprising that, as stated by our result, lazy abstraction with deterministic control is a decision procedure for the coverability problem for well-structured system.

It is not the point of this paper to advocate lazy abstraction as a promising practical alternative to existing decision algorithms for well-structured systems, including the other algorithms based on abstraction refinement. It is, however, an outcome of the work presented in this paper that a thorough experimental comparison on the wide range of well-structured systems (see, e.g., `http://www.ulb.ac.be/di/ssd/lvbegin/CST/#examples`) has come to make sense.

References

1. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL. (2002) 58–70
2. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In Emerson, E.A., Sistla, A.P., eds.: CAV. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 154–169
3. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software verification with BLAST. In Ball, T., Rajamani, S.K., eds.: SPIN. Volume 2648 of Lecture Notes in Computer Science., Springer (2003) 235–239
4. Henzinger, T.A., Jhala, R., Majumdar, R., Necula, G.C., Sutre, G., Weimer, W.: Temporal-safety proofs for systems code. In Brinksma, E., Larsen, K.G., eds.: CAV. Volume 2404 of Lecture Notes in Computer Science., Springer (2002) 526–538
5. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1-2) (2001) 63–92
6. Ganty, P., Raskin, J.F., Begin, L.V.: A complete abstract interpretation framework for coverability properties of wsts. In Emerson, E.A., Namjoshi, K.S., eds.: VMCAI. Volume 3855 of Lecture Notes in Computer Science., Springer (2006) 49–64
7. Geeraerts, G., Raskin, J.F., Begin, L.V.: Expand, Enlarge, and Check: New algorithms for the coverability problem of wsts. In Lodaya, K., Mahajan, M., eds.: FSTTCS. Volume 3328 of Lecture Notes in Computer Science., Springer (2004) 287–298
8. Geeraerts, G., Raskin, J.F., Van Begin, L.: Expand, enlarge and check... made efficient. In Rajjamani, S.K., Etessami, K., eds.: Poceedings of 17th International Conference on Computer Aided Verification – CAV 2005. Number 3576 in Lecture Notes in Computer Science, Springer Verlag (2005) 394–404 to appear.
9. Delzanno, G., Esparza, J., Podelski, A.: Constraint-based analysis of broadcast protocols. In Flum, J., Rodríguez-Artalejo, M., eds.: CSL. Volume 1683 of Lecture Notes in Computer Science., Springer (1999) 50–66
10. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: LICS. (1996) 313–321
11. Bouajjani, A., Fernandez, J.C., Halbwachs, N., Raymond, P.: Minimal state graph generation. Sci. Comput. Program. **18**(3) (1992) 247–269
12. Ball, T., Podelski, A., Rajamani, S.K.: Relative completeness of abstraction refinement for software model checking. In Katoen, J.P., Stevens, P., eds.: TACAS. Volume 2280 of Lecture Notes in Computer Science., Springer (2002) 158–172
13. McMillan, K.L.: Lazy abstraction with interpolants. In Ball, T., Jones, R.B., eds.: CAV. Volume 4144 of Lecture Notes in Computer Science., Springer (2006) 123–136
14. Dickson, L.: Finiteness of the odd perfect and primitive abundant numbers with n prime factors. Amer. J. Math. **35** (1913) 413–422