Supervisory Control of Discrete-Event Systems via IC3

Mohammad Reza Shoaei[1,⋆], Laura Kovács[2,⋆⋆], and Bengt Lennartson[1]

[1] Department of Signals and Systems
[2] Department of Computer Science and Engineering
Chalmers University of Technology, Gothenburg, Sweden
{shoaei,laura.kovacs,bengt.lennartson}@chalmers.se

Abstract. The IC3 algorithm has proven to be an effective SAT-based safety model checker. It has been generalized to other frameworks such as SMT and applied very successfully to hardware and software model checking. In this paper, we present a novel technique for the supervisory control of discrete-event systems with infinite state space via IC3. We introduce an algorithm for synthesizing maximally permissive controllers using a generalized IC3 to find (if any exists) a weakest inductive invariant predicate which holds in the initial state, is maintained as the system evolves, and implies safety and control properties. To this end, we use a variation of IC3, called Tree-IC3, as a bug finder to solve the supervisory predicate control problem by iteratively reporting all feasible counterexample traces using a tree-like search, while controlling the system to avoid them. The maximally permissiveness is achieved by finding the weakest of such controllers that is invariant under safety and control properties. Experimental results demonstrate the great potential of using IC3 technique for the purpose of the supervisory control problems.

Keywords: Discrete-event systems; Supervisory control theory; Incremental inductive verification; IC3

1 Introduction

Supervisory Control Theory (SCT), established by Ramadge and Wonham [4, 26, 25], is a formal framework for modeling and control of discrete-event systems (DES). Problems that SCT can address include dynamic allocation of resources, the prevention of system blocking, etc. and, within these constraints, maximally permissive system behavior. Traditionally, there are a certain number of modeling formalisms that can be used for investigating feedback control of DES, such as state machines (SMs), automata, and extended finite-state machines (EFSMs),

⋆ This work was supported by the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers University of Technology, and the Swedish Governmental Agency for Innovation Systems (VINNOVA).

⋆⋆ This research was supported in part by the Swedish VR grant D0497701, the Austrian FWF RiSE grant S11410-N23, and the WWTF PROSEED grant ICT C-050.

which are state machines with variables, see e.g. [28, 5, 30]. The control requirements on DES are expressed by specifications in terms of regular languages, in the case of DES modeled by SMs or automata, or in terms of constraints on the states (expressed by formula), in the case of DES modeled by EFSMs. The focus of this theory is on systematical synthesis of provably safe and nonblocking controllers for a given uncontrolled system, called *plant*.

Nevertheless, the industrial acceptance of SCT is still scarce due to the following two drawbacks: the computational complexity of synthesizing a controller and *state-space explosion*, owing to limited amount of memory when working with large state-space. Researchers in the DES community are therefore seeking effective synthesis techniques to avoid these pitfalls. For example, for systems with finite behavior, [21, 10] propose an efficient synthesis technique using BDDs, in [23] an algorithm is presented that iteratively strengthens the formula on transitions so that forbidden or blocking states become unreachable, and in [7] a SAT-based approach is presented. Furthermore, for systems with infinite behavior, [16, 24] propose approaches for synthesis of DES using predicates and predicate transformers. However, the approach in [16] works only with systems modeled by equations over an arithmetic domain and with very few uncontrollable events.

On the other hand, several researchers in the formal methods community have investigated safety of programs using verification techniques such as BMC [2], interpolation [18], k-induction [27], and recently IC3 [3]. The IC3 algorithm has proven to be an effective SAT-based safety model checker [29]. It has been generalized to other frameworks such as SMT and applied very successfully to hardware and software model checking [6, 13]. Recently, Morgenstern et al. [22] proposed a property directed synthesis method for game solving, which is also inspired by IC3. However, [22] only checks whether a system is controllable, by computing an overapproximation for the winning states, but does not construct (synthesize) a winning strategy.

In this paper, we present a novel technique for supervisory control of DES with infinite state space via IC3. To this end, we use a generalized form of IC3 as reported in [6], called *Tree-IC3*, to find (if any exists) a weakest inductive invariant predicate which holds in the initial state, is maintained as the system evolves, and implies safety and control properties. In particular, for a DES modeled by EFSMs and a safety property expressed by a set of "legal" locations, we use Tree-IC3 to find violation of safety property by searching over abstract paths of the system. Whenever a violation is found, corresponding transitions of the system are strengthened by a controller to avoid violation of the safety properties. An evaluation of the proposed IC3-based technique on standard SCT benchmarks shows a radical improvement for systems with large domains compared to BDD-based and SAT-based approaches. It should be noted that, to the best of our knowledge, this work is the first attempt to use incremental, inductive techniques, such as IC3, for solving supervisory control problems, which opens new venues of research in supervisory control theory.

The remainder of the paper is organized as follows. Section 2 briefly describes the background. In Section 3, we introduce the incremental, inductive supervisory control (IISC) algorithm and in Section 4 we discuss our experimental results. Finally, we draw some conclusions and directions for future work in Section 5.

2 Background

In this paper, we use standard notion of first-order logic (FOL) and assume quantifier-free formulas. We denote formulas with ϕ, ψ, I, T, P, variables with x, y, and sets of variables with X, Y. A literal is an atom or its negation. A *clause* is a disjunction of literals, whereas a *cube* is a conjunction of literals. A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses, and in disjunctive normal form (DNF) if it is a disjunction of cubes. With abuse of notation, we might sometimes denote formulas in CNF $\phi_1 \wedge \cdots \wedge \phi_n$ as sets of clauses $\{\phi_1, \ldots, \phi_n\}$, and vice versa. A subclause $p \subseteq c$ is a clause p whose literals are a subset of literals in c.

We write $\phi(X_1, \ldots, X_n)$ to indicate that all variables occurring in ϕ are elements of $\bigcup_i X_i$. For every variable x, we assume the existence of a unique variable x' representing the next value of x. Given a formula ϕ, we use $\phi^{\langle n \rangle}$ to denote the addition of n primes to every variable in ϕ, representing the value of variables in ϕ at n time units in the future. Similarly, for any set X of variables, we let $X' := \{x' \mid x \in X\}$ and $X^{\langle n \rangle} := \{x^{\langle n \rangle} \mid x \in X\}$.

Throughout the paper, we use the standard notion of theory, satisfiability, validity, and logical consequences. Given a theory $\mathcal{T}$, $\phi \models_{\mathcal{T}} \psi$ (or simply $\phi \models \psi$) denotes that the formula ψ is a logical consequence of ϕ in the theory $\mathcal{T}$. Furthermore, given a model $\mathcal{M} = (\mathcal{I}, \mathcal{D})$, where $\mathcal{D}$ is the domain of elements and $\mathcal{I}$ (over $\mathcal{D}$) is the interpretation function, we write $(d_1, \ldots, d_n) \models_{\mathcal{M}} \phi(X_1, \ldots, X_n)$ for some $d_i \in \mathcal{D}_{X_i}$ $(i > 0)$, or simply $d \models \phi$, to denote $(d_1, \ldots, d_n) \in (\phi)^{\mathcal{I}}$.

2.1 Modeling Discrete-Event Systems

State Machine. The behavior of a discrete-event system to be controlled, called the *plant*, is modeled as a state machine (SM) [15]. Let the tuple $G = \langle Q, \Sigma, \delta, Q^i \rangle$ denote an SM, representing a plant, where Q denotes the state set, Σ denotes the finite set of events (alphabet), $\delta \subseteq Q \times \Sigma \times Q$ denotes the transition relation, and $Q^i \subseteq Q$ denotes the subset of initial states. We say that a state $q \in Q$ is reachable in G if there is a sequence of transitions from an initial state q_0 to q, i.e., $(q_0, \sigma_0, q_1)(q_1, \sigma_1, q_2) \cdots (q_{n-1}, \sigma_{n-1}, q_n)$ in G where $q_0 \in Q^i$ and $q_n = q$. Let $C : Q \to 2^{\Sigma}$ be a function. For any $\sigma \in \Sigma$ and $q \in Q$, we say that σ is enabled by C in state q if $\sigma \in C(q)$. The *restriction* of G to C, denoted $G|_C$, is described by the state machine $G_C = \langle Q, \Sigma, \delta_C, Q^i \rangle$, where δ_C is defined according to: $(q, \sigma, r) \in \delta_C$ if and only if $(q, \sigma, r) \in \delta$ and $\sigma \in C(q)$.

Extended Finite State Machine. In practice, plants are often used for modeling programs or industrial systems with data dependency. In order to model such systems, in a compact and efficient way, we extend SMs with variables, in which FOL formulas are used to represent data flow in the system.

Let X denote the set of system variables. An *extended finite state machine* (EFSM) over X is a tuple $A = \langle L, \Sigma, \Delta, l^i, l^f, \Theta \rangle$, where L is the set of finite locations, Σ is the alphabet, Δ is the set of transitions, $l^i \in L$ is the initial location, $l^f \in L$ is a special location called *error* location (or forbidden location), and $\Theta(X)$ is a formula that describes the initial values of variables. A transition in Δ is a tuple (l, a, m), where $l, m \in L$ are respectively the entry and exit locations of the transition and $a = (\sigma, \varphi)$ is the *action of the transition*, where $\sigma \in \Sigma$ is the *event of the transition* and $\varphi(X \cup X')$ is the *formula of the transition*. When no confusion is possible, we often write $\sigma : \varphi$ instead of (σ, φ).

A path in A is a sequence of transitions of the form $(l_0, \sigma_0 : \varphi_0, l_1)(l_1, \sigma_1 : \varphi_1, l_2) \cdots (l_{n-1}, \sigma_{n-1} : \varphi_{n-1}, l_n)$. We say that the path (i) is an *error* path if $l_0 = l^i$ and $l_n = l^f$; (ii) is *feasible* iff the formula $\bigwedge_i \varphi_i^{\langle i \rangle}$ is satisfiable; and (iii) is *spurious* if it not feasible. A plant is safe when all the paths leading to l^f are not feasible.

To simplify the presentation of the algorithms, it is assumed that every location has at least one outgoing transition. This can be done, say for location $l \in L$, by adding a self-loop $(l, \varepsilon : \top, l)$ to Δ, where $\varepsilon \notin \Sigma$ is an *empty event*.

State Machine Representation: Fix a model $\mathcal{M} = (\mathcal{I}, \mathcal{D})$. Then A can be described by SM $G_A = \langle Q_A, \Sigma, \delta_A, Q_A^i \rangle$, where $Q_A = L \times \mathcal{D}$ is the (possibly infinite) set of states, each of which is a pair of location and variables valuation, $Q_A^i = \{\langle l^i, d \rangle \in Q_A \mid d \models \Theta\}$ is the set of initial states, and δ_A is defined according to: $(\langle l, d \rangle, \sigma, \langle m, d' \rangle) \in \delta_A$ iff $(l, \sigma : \varphi, m) \in \Delta$ and $(d, d') \models \varphi$.

Symbolic Representation: The EFSM A can also be described symbolically by a *symbolic transition system* (STS) $S_A = \langle \hat{X}, T(\hat{X} \cup \hat{X}'), I(\hat{X}) \rangle$, where $\hat{X}$ is the extension of variables set X by adding two special elements x_L and x_Σ, with domain L and Σ respectively, $I(\hat{X}) = (x_L = l^i) \wedge \Theta$, and $T(\hat{X} \cup \hat{X}') = \bigvee_{(l, \sigma : \varphi, m) \in \Delta} (x_L = l) \wedge (x_\Sigma = \sigma) \wedge \varphi \wedge (x'_L = m)$. In what follows, we shall drop the hat on $\hat{X}$, as context will determine the intended meaning. Given S_A, the safety of EFSM A can be shown by proving that all the reachable states of S_A are a subset of the states symbolically described by the formula $P := \neg(x_L = l^f)$, namely, S_A satisfies the invariant property P.

Note that, for any EFSM A, S_A and G_A (which both represent low level behavior of A) are related in a way that S_A also symbolically represents the state machine G_A. Thus, any property that holds on S_A holds on G_A as well. In the sequel, to simplify the mathematical representation of the problem, we shall freely switch between these two low level descriptions of A and use the one that offers the simplest representation.

Abstract Reachability Tree. Given an EFSM $A = \langle L, \Sigma, \Delta, l^i, l^f, \Theta\rangle$ over X, we further define an *abstract reachability tree* (ART) $\mathcal{A} = \langle V, E\rangle$ for A as follows: (i) V is a set of quadruples (l, ϕ, σ, k), where $l \in L$ is a location, $\phi(X)$ is a formula, $\sigma \in \Sigma$ is an event, and $k \in \mathbb{N}$ is a unique identifier; (ii) $v_\varepsilon := (l^i, \Theta, \varepsilon, 1)$ is the root of $\mathcal{A}$, where $\varepsilon \notin \Sigma$ is an empty event; (iii) for every non-leaf node $v := (l, \phi, \sigma, k) \in V$, for every transition $(l, \beta : \varphi(X \cup X'), m) \in \Delta$, v has a child node $w := (m, \psi, \beta, h)$ such that $\phi \wedge \varphi \models \psi'$ and $k < h$; and we call φ the *edge formula* of w. In the sequel, we shall use the following notations for any ART $\mathcal{A}$:

- For any node $v := (l, \phi, \sigma, k)$ in $\mathcal{A}$, ϕ and σ are called the *abstract state formula* and *event* of node v, respectively. If $l = l^f$, v is called an *error node*.
- A node $v := (l, \phi, \sigma, k)$ is said to be *covered* in $\mathcal{A}$ if either: (i) there exists an uncovered node $w := (l, \psi, \beta, h)$ in $\mathcal{A}$ such that $h < k$ and $\phi \models \psi$; or (ii) v has a proper ancestor for which (i) holds.
- $l_i \rightsquigarrow l_j$ denotes a path in $\mathcal{A}$ from a node (l_i, ψ, β, h) to a descendant node (l_j, ϕ, σ, k) with $h < k$.
- For any path $\pi := (l_0, \phi_0, \sigma_0, .) \rightsquigarrow \cdots \rightsquigarrow (l_n, \phi_n, \sigma_n, .)$ in $\mathcal{A}$, with (ab)use of notations, we let $\Sigma^A(\pi)$ and $\Delta^A(\pi)$ denote the sets of $[\sigma_0, \ldots, \sigma_n]$ events and $[\varphi_0, \ldots, \varphi_{n-1}]$ edge formulas of π, respectively. In the sequel, we sometimes view π as a subset of nodes $\pi \subseteq V$.

We say that $\mathcal{A}$ is *complete* if all its leaves are covered or their abstract state formula is equivalent to $\bot$; $\mathcal{A}$ is *safe* iff it is complete and for all error nodes (l^f, ϕ, σ, k) in $\mathcal{A}$, we have that $\phi \models \bot$. If an EFSM A has a safe ART, then A is said to be safe [12, 19][3].

2.2 Supervisory Control

This section recalls the supervisory (predicate) control framework of [25, 16]. Let $A = \langle L, \Sigma, \Delta, l^i, l^f, \Theta\rangle$ be an EFSM and let S be the STS over symbolic state space X (i.e. $\hat{X}$). The alphabet Σ is classically partitioned into the two disjoint subsets, the *controllable events* Σ_c, whose occurrence can be inhibited by the controller (also called supervisor) and the *uncontrollable events* Σ_u, which can never, or need not, be disabled. Let $P(X) := \neg(x_L = l^f)$ be the safety property that represents a set of good states. The control task is to design a static controller $C : X \to 2^\Sigma$ for S that guarantees safety property P by restricting the conduct of S, i.e., as the system (restricted to C) evolves (unrolls), it visits only the states where P holds. Note that, since a controller cannot restrict uncontrollable events, we also have that for each (symbolic) state $s \in X$, $\Sigma_u(s) \subseteq C(s)$, where $\Sigma_u(s)$ is the set of uncontrollable events defined at s.

In order to present formally the control task, we need the following notations. Given S, for any predicate $R(X)$ and for any $\sigma \in \Sigma$, let

$$\mathcal{F}_\sigma(R)(X) := \exists X_0 \, . \, R(X_0) \wedge T(X_0, X) \wedge (x_\Sigma = \sigma)$$

[3] In fact, e.g. in [19], this property is shown for systems modeled by program graphs. However, any EFSM can be transformed to the equivalent program graph by simply dropping their event set Σ and conjuncting Θ with formula of all outgoing transition of their initial location.

be the *strongest post-condition* predicate transformer for S w.r.t. σ, i.e., $\mathcal{F}_\sigma$ holds on the set of states of S that are reached by the transition with event σ from a state where R holds. We write $\mathcal{F}$, $\mathcal{F}_u$, and $\mathcal{F}^*$ for respectively $\bigvee_{\sigma\in\Sigma}\mathcal{F}_\sigma$, $\bigvee_{\sigma\in\Sigma_u}\mathcal{F}_\sigma$, and $\bigvee_{n\geq 0}\mathcal{F}^n$, where $\mathcal{F}^0$ is defined to be the identity predicate transformer. For given predicate R, $\mathcal{F}^*(R)$ holds in those states which are reachable from a state where R holds in zero or more number of transitions. Thus, $\mathcal{F}^*$ is useful in characterizing the *reachability set* of STS S. Furthermore, the restriction of $\mathcal{F}$ to R, denoted $\mathcal{F}|_R$, is a new predicate transformer defined by: $\mathcal{F}|_R(W) = \mathcal{F}(R \wedge W) \wedge R$ for any predicate $W(X)$.

Problem 1 (Supervisory Predicate Control Problem). The control task is to construct a static controller $C : X \to 2^\Sigma$ for S such that $\mathcal{F}^*_{S|_C}(I) \models P$.

That is, Problem 1 requires that the state trajectories in the controlled system $S|_C$, starting from the initial states I, remain confined to the set of states where the safety predicate P holds, and visit only the states where P holds. Thus, the controlled system guarantees safety.

Given any predicate $R(X)$, we say that R is a *controllable and safe invariant predicate (C-SIP)* for S if:

1) $R \models (\mathcal{F}|_R)^*(I)$,
2) $\mathcal{F}_u(R) \models R$, and
3) $R \models P$.

That is, 1) R is a *fixed-point* of the predicate transformer $\mathcal{F}|_R$ starting from the initial states I, 2) if S starts in a state where R holds, then under the execution of any transition with uncontrollable event it remains in a state where R holds, and 3) R implies the safety property.

The solution to the Problem 1 exists if and only if there exists a predicate R that is C-SIP for S[4]. Indeed, $\perp$ is always a possible solution. Therefore, the notion of *permissiveness* has been introduced in SCT framework to compare the quality of different (predicate) controllers for given plant. For any two predicates R_1 and R_2, we say that R_1 is more permissive (less restrictive) than R_2 if $R_1 \equiv R_1 \vee R_2$. Now, for a family of C-SIPs for S, we let $R^\uparrow := \bigvee\{R \mid R \text{ is a C-SIP for } S\}$ denotes the *maximally permissive C-SIP* for S. The real challenge is to find $R^\uparrow$ for given S.

Discussion. In SCT framework, in addition to safety and control properties, it is desired for the controlled system to be *nonblocking*. This property guarantees that at least one *marked state* (which is also referred to as accepting state or final state) is reachable from any state in the controlled system. The nonblocking property is known to be a "global" behavior, as opposed to the "local" behavior of error (forbidden) states, so the condition that a state is nonblocking cannot be expressed as a property of the state alone, without considering possible progress from the state. However, *deadlocks*, unmarked states with no outgoing

[4] We refer the interested reader to [16] for proof of the above claim.

transitions, can be expressed (locally) as a property of each state. Thus, we consider deadlocks as a special form of error states. In this paper, however, we do not discuss the full supervisory control problem but instead we will focus on an important subclass of problems, i.e., safety and controllability problems.

2.3 Incremental, Inductive Verification

The term incremental, inductive verification has been used to describe algorithms that use induction to construct lemmas in response to property-driven hypotheses, e.g., the IICTL [11] and IC3 [3] algorithms. In this section, we briefly recall the original IC3 algorithm, as formulated in [9], and its extension to SMT, as described in [6].

IC3 Algorithm. Given a STS $S = \langle X, T(X \cup X'), I(X) \rangle$, let $P(X)$ describes a set of good states. The IC3 algorithm tries to prove that S satisfies P by maintaining formulas $F := F_0(X), \ldots, F_k(X)$, where F is called a *trace* and F_i $(i \geq 0)$ are called *frames*, such that:

- $F_0 = I$;
- for all $i > 0$, F_i is a set of clauses;
- $F_{i+1} \subseteq F_i$ (thus, $F_i \models F_{i+1}$);
- $F_i(X) \wedge T(X \cup X') \models F_{i+1}(X')$;
- for all $i < k$, $F_i \models P$.

For $i > 0$, F_i represents an over-approximation of the states of S reachable in i transition steps or less. Initially, F_0 is set to the initial states I. The algorithm proceeds incrementally, by alternating two phases:
(i) *Blocking phase*: The trace is analyzed to prove that F_k and $\neg P$ do not intersect, thus $F_k \models P$. More specifically, IC3 maintains a set of pairs (s, i), where s is a cube representing a set of states that can lead to a bad state and $i > 0$ is a position in the trace. New clauses are added to (some of) the frames in the trace by (recursively) proving that a set s of a pair (s, i) is unreachable starting from F_{i-1}. This is done by checking the satisfiability of the formula

$$F_{i-1} \wedge \neg s \wedge T \wedge s' \tag{1}$$

If it is unsatisfiable, i.e. F_{i-1} blocks $\neg s$, and s does not intersect with I, then IC3 strengthens F_i by adding $\neg s$ to it. If, instead, it is satisfiable, i.e. F_{i-1} is not strong enough to block s, then IC3 computes a (generalized) cube p representing a subset of the states in $F_{i-1} \wedge \neg s$ such that all the states in p lead to a state in s' in one transition step. Afterwards, IC3 tries to block the pair $(p, i-1)$ (namely, it tries to show that p is not reachable in one step from F_{i-2}). This procedure continues recursively, possibly generating other pairs to block at earlier points in the trace, until either IC3 generates a pair $(q, 0)$, meaning that the system does not satisfy the property and a counterexample is constructed, or the trace is eventually strengthened so that the original pair (s, i) can be blocked.

(*ii*) *Propagation phase*: The trace is extended (if $F_i \models P$) with a new formula F_{i+1}, moving forward the clauses from F_i. If, during this process, two consecutive frames become identical, i.e. $F_i = F_{i+1}$, then a fixed-point is reached, and IC3 can terminate with F_i being an inductive invariant proving the property. For more elaboration on IC3 algorithm we refer to [3, 9].

IC3 Extension to SMT. For proving safety property (and later for constructing $R^{\uparrow}$) of plants modeled by EFSMs, it is more convenient to work at a higher level of abstraction, using SAT modulo theories (SMT). To this end, as described in [6], we replace the underlying SAT engine with an SMT solver[5]. With the new solver, if the formula (1) is satisfiable, then a new pair $(p, i-1)$ will be generated such that p is a cube in the *preimage of* s *w.r.t.* T. That is, to existentially quantify the variables X' in (1), eliminate the quantifiers, and then convert the result in DNF. This will generate a set of cubes $\{p_j\}_j$ which in turn generate a set of pairs $\{(p_j, i-1)\}_j$ to be blocked at $i-1$. In what follows, we shall assume that the SMT solver has a procedure PREIMAGE for computing the preimage.

3 Incremental, Inductive Supervisory Control via IC3

We now present the incremental, inductive supervisory (predicate) control (IISC) algorithm via IC3.

Outline. In high-level description, the algorithm constructs a controller (if any exists) for given plant by alternating between two phases:
(i) *Error-Finding,* which the algorithm searches for error location in the plant by *unwinding* it into an abstract reachability tree (ART). Whenever an error location is found, it tries to refute the abstract path to that location by applying a procedure that mimics the blocking phase of IC3. In case of failure to refute the error path, it returns a counterexample trace;
(ii) *Supervision,* which the algorithm tries to control the current counterexample trace from reaching the error location by strengthening the clauses attached to the *controllable nodes* (i.e. nodes with controllable event) in the path. Thus, the path is blocked from reaching the error location. In order to fulfill the maximally permissiveness criteria, the process strengthens only the nearest controllable node from the error location. This guarantees that the controller does not restrict the plant more than what is necessary to refute the error location. However, when there are no controllable nodes to be controlled (strengthen), the safety property is violated. Thus, the algorithm returns the counterexample trace. Otherwise, if the blocking process is successful, the counterexample is refuted and the algorithm continues the search.

[5] There are, however, some crucial steps which must be made before switching from SAT to SMT solver, for which we refer to [6].

```
procedure IISC(EFSM $A = \langle L, \Sigma, \Delta, l^i, l^f, \Theta\rangle$):
    global: The EFSM $A$ and an ART $\mathcal{A} = \langle V, E\rangle$
    let $v_\varepsilon := (l^i, \Theta, \varepsilon, 1)$ be the root of $\mathcal{A}$
    while there exists an uncovered leaf $v \in V$ :
        if $v := (l^f, \phi, \sigma, k)$ and $\phi \not\models \bot$ :
            $(F$, status$)$ = IC3-BLOCK-PATH($\pi := v_\varepsilon \rightsquigarrow v$)
            switch status :
                case CTX
                     /* in case $\pi$ is a feasible error path */
                    if SUPERVISE($F, \pi$) = false :
                         /* error path $\pi$ cannot be controlled */
                        return the counterexample trace $F$
                case BLOCKED
                     /* in case $\pi$ is refuted */
                    for $i = 0$ to size of $\pi$ :
                        UPDATE-INV($F[i], \pi[i]$)
        else:
            STRENGTHENING($v$) /* as in [6] */
            COVERING($v$) /* as in [6] */
            UNWINDING($v$) /* as in [6] */
    return /* the plant $A$ is successfully supervised */
```

Fig. 1. Incremental Inductive Supervisory Control Algorithm

IISC Algorithm. Fig. 1 illustrates the IISC algorithm. The algorithm is built upon the Tree-IC3 technique in [6] as it arrives to finding and refuting abstract paths to the error location. The search proceeds in an "explicit-symbolic" fashion, i.e., the given plant is unwound into an ART, following a DFS strategy.

The algorithm starts by selecting an uncovered leaf, $v := (l, \phi, \sigma, k)$. If v is an error node (namely, $l = l^f$) then the algorithm tries to refute the abstract path to this node (i.e., $\pi^f := v_\varepsilon \rightsquigarrow v$) by calling the IC3-BLOCK-PATH procedure, see Fig. 2. Note that, in order to construct a correct ART, when IC3-BLOCK-PATH checks whether a cube c is blocked by a set of clauses F_{i-1}, the inductiveness check (1) is replaced with a weaker check

$$F_{i-1} \wedge T_{i-1} \models \neg c' \tag{2}$$

However, because of this replacement, the requirement that $F_{i+1} \subseteq F_i$ is not enforced anymore. With this adaptation, the procedure tries to produce the clauses necessary to refute the abstract path and terminates successfully whenever an empty clause is generated. In case of failure to refute the path, the property is

```
procedure IC3-BLOCK-PATH($\pi := (l^i, \Theta, \varepsilon, 1) \rightsquigarrow \cdots (l_i, \phi_i, \sigma_i, .) \cdots \rightsquigarrow (l^f, \phi_n, \sigma_n, .)$):
  let $T := [\varphi_0, \dots, \varphi_{n-1}] = \Delta^A(\pi)$ /* $\varphi_i$ are the edge formulas */
  let $F := [\Theta, \dots, \phi_i, \dots, \phi_{n-1}]$ /* $\phi_i$ are the clauses of the nodes */
  while not exists $0 < j < n$ s.t. $F[j] \wedge T[j] \models \bot$ :
    let stack = $\emptyset$
    foreach bad in PREIMAGE($\phi_{n-1} \wedge T[n-1]$) :
      /* bad is a cube in the preimage of $T[n-1]$ */
      stack.push((bad, $n-1$))
    while stack is not empty :
      $c, j$ = stack.top()
      if $j = 0$ : /* $\pi$ is a feasible error trace */
        let $B = [c_0, \dots, c_{n-1}]$ be the counterexample trace
        return $(B, \text{CTX})$
      if $F[j-1] \wedge T[j-1] \models \neg c'$ :
        stack.pop() /* cube $c$ is blocked */
        /* $\neg c$ can be generalized before adding to $F[j]$ */
        $F[j] = F[j] \wedge \neg c$
      else:
        foreach $p$ in PREIMAGE($F[j-1] \wedge T[j-1] \wedge c'$) :
          stack.push(($p, j-1$))
  return ($F$, BLOCKED) /* path $\pi$ is blocked */
```

Fig. 2. IC3 blocking path procedure

violated and a counterexample trace is returned[6]. If IC3-BLOCK-PATH returns a counterexample trace, then the algorithm tries to control the path π^f from reaching the error location by calling the SUPERVISE procedure, see Fig. 3. In the supervision phase, the nearest controllable ancestor of v, say $v_i := (l_i, \phi_i, \sigma_i, h)$ s.t. $\sigma_i \in \Sigma_c$, is controlled by strengthening (conjuncting) its incoming edge formula T_{i-1}, with negation of (bad) cube c, i.e. $T_{i-1} = T_{i-1} \wedge \neg c'$. Thus, the check (2) becomes satisfiable which implies that the cube c is blocked at v_i.

In case that the leaf node v is not an error node, the following procedures are applied to v: STRENGTHENING in which v is strengthened by forward propagating the clauses of its ancestor; COVERING in which v is covered thus can be closed, whenever the set of states of v is contained in the states of some previously generated node w having the same location; and UNWINDING, which expands the

[6] Note that, our formulation of the IC3-BLOCK-PATH procedure is slightly different from the original one in [6]. In particular, we set F_0 to denote initial formula Θ instead of $\top$. We also note that $\neg c$ can be generalized before being added to F_i. Although this is quite important in practice for effectiveness of IC3, here for brevity we shall not discuss this.

```
procedure SUPERVISE($B, \pi$):
    if $\Sigma^A(\pi) \subseteq \Sigma_u$ : return false
    $T := [\varphi_0, \ldots, \varphi_{n-1}] = \Delta^A(\pi)$
    let $i$ be the size of $\pi$
    while $i > 0$ :
        let $v_i := (l_i, \phi_i, \sigma_i, .) = \pi[i]$
        UPDATE-INV($v_i, \neg B[i]$)
        if $\sigma_i \notin \Sigma_u \cup \{\varepsilon\}$ :
            /* strenghening edge formula */
            $T[i-1] = T[i-1] \wedge \neg(B[i])'$
            break
        $i = i - 1$
    return true

procedure UPDATE-INV($v_i, \psi$):
    /* $v_i := (l_i, \phi_i, \sigma_i, .)$ */
    foreach $c_j \in \psi$ s.t. $\phi_i \not\models c_j$ :
        add $c_j$ to $\phi_i$,
        uncover all nodes
        covered by $v_i$.
```

Fig. 3. Auxiliary procedures for the IISC algorithm

ART by generating the successors of v. Note that, for brevity, we have to omit several important details in each procedure, for which we refer to [6].

Finally, the IISC algorithm terminates when either the given plant couldn't be controlled or there are no uncovered leafs left, indicating that the plant A is controlled, by strengthening its transition formulas, thus guarantees the safety property. In the former case, a counterexample trace to the error location is returned. We also note that the termination of IISC algorithm is guaranteed whenever the given plant A is defined over a finite domain, see [6, 3, 9].

Theorem 1. *A maximally permissive C-SIP $R^{\uparrow}$ exists for a plant A if* IISC(A) *terminates without a counterexample trace.*

Proof. We sketch the proof as follows. Let $A^{\uparrow} :=$ IISC(A) denote the EFSM obtained from A by applying IISC algorithm in Fig. 1, and let $R_{A^{\uparrow}}$ denote the transition formula of STS $S_{A^{\uparrow}}$. Let $\mathcal{F}$ be the strongest post-condition predicate transformer defined over S_A. Since the IISC terminates without a counterexample we immediately see that $R_{A^{\uparrow}} \models (\mathcal{F}|_{R_{A^{\uparrow}}})^*(I)$ and $R_{A^{\uparrow}} \models P$, where $P := \neg(x_L = l^f)$. Also, because IISC only controls the transitions with controllable events, clearly $\mathcal{F}_u(R_{A^{\uparrow}}) \models R_{A^{\uparrow}}$. Hence, $R_{A^{\uparrow}}$ is a C-SIP for S_A (see definition of C-SIP in Section 2.2). Furthermore, the maximally permissiveness of $R_{A^{\uparrow}}$ comes from the fact that A is controlled only when a feasible error path exists, and only the nearest controllable transition to the error location is controlled (strengthened). Thus, we conclude that $R_{A^{\uparrow}}$ is a maximally permissive C-SIP for $S_{A^{\uparrow}}$.

Corollary 1. *The solution to the Problem 1 exists if IISC algorithm terminates without a counterexample trace.*

Model	IISC (s)	SC-BDD (s)	SC-SAT (s) [only verification]
CMT (1,5)	0.127	0.066	0.083
CMT (3,3)	0.430	1.639	2.128
CMT (5,5)	0.733	108	8.84
CMT (7,7)	0.975	T.O.	T.O.
EDP (5,10)	0.98	0.168	14.36
EDP (5,50)	0.124	0.374	T.O.
EDP (5,200)	0.124	1.382	T.O.
EDP (5,10E3)	0.124	16.746	T.O.
EDP (5,10E5)	0.124	T.O.	T.O.
PME	2.3	11.595	85.30

Table 1. Performance statistics on benchmark examples.

4 Experiments

The IISC algorithm has been integrated in the DES tool Supremica [1], in which we embedded Z3 [8] as our SMT solver. We use the theory of Linear Real Arithmetic for modeling formulas on transitions, which is well supported by Z3. To compute the exact preimage of a cube c and a transition formula $\varphi(X \cup X')$, we first convert the formula $\varphi \wedge c'$ to a DNF $\bigvee_i p_i$ and then use the quantifier elimination function in Z3 to project each cube p_i over current-state variables $X : \bigvee_i \exists X' . (p_i)$. The under-approximate preimage of c w.r.t. φ can then be constructed simply by picking only a subset of their exact preimage. In fact, similar to [6], we also under-approximate by simply stopping after the first cube.

Furthermore, as in work [6, 19], we applied the following improvements to the implementation: A new instance of a program variable is used only when that variable is modified. This eliminates many constraints of the form $x^{\langle i+1\rangle} = x^{\langle i\rangle}$ that occur when a variable is unmodified by a transition formula. Also, instead of always using an under-approximate preimage procedure as in [6], a threshold on the size of the clauses is introduced for deciding whether to use under-approximate or exact preimage procedure. In our industrial examples, however, this decision does not yield any substantial performance reduction. Moreover, in practice, before processing a node we first check if the node is covered. This often substantially reduces the overall run time of the algorithm.

For our evaluation, we compared the IISC algorithm with the Symbolic Supervisory Control using BDD (SC-BDD) algorithm in [21, 10] and SAT-based Supervisory Control (SC-SAT) algorithm in [7], where we used the SC-SAT algorithm for the safety and controllability verification only and not for the synthesis. To this end, we used the following set of standard benchmarks in SCT: the parallel manufacturing example (PME) [17], cat and mouse tower (CMT), and extended Dinning Philosophers (EDP) [20].

Table 1 summarized the run time performance of the algorithms[7]. In this table, CMT(n, k) denote the CMT problem with a tower composed by n identical levels, k cats, and k mice; EDP(i, j) denote the EDP problem with i philosophers and j intermediate states of each philosopher, and T.O. indicates time out (5 min). As the table shows, for those examples with smaller domain, the SC-BDD is slightly better than the IISC algorithm. The performance difference might be because of the fact that once the BDD data structure is constructed, computing a controller can be done very efficiently. Although, SC-SAT only reports if the system is safe and/or it can be controlled, it performs poorly or time out in most of the examples. One possible reason is that the SC-SAT algorithm needs to enumerate all possible solutions (within the domain) using its underlying SAT solver.

As the systems become larger (namely, domain of variables become larger), the IISC approach obviously outperforms the SC-BDD and SC-SAT. With no surprise, this owes to the fact that the BDD-based approaches suffer from exponential space blow up of BDD nodes while representing a large state space, and SAT-based approaches have the disadvantage of search for a single, often complex, queries, which can in practice overwhelm the SAT solver. In contrast, the performance of our approach (and in general, IC3-based approaches) depends on the number of variables and transition formals rather than the actual number of explicit states and transitions.

5 Conclusions and Future Work

In this paper we have presented a novel technique to synthesizing controllers for discrete-event systems via IC3. More precisely, given a plant model and a safety property, we used a variation of Abstract Reachability Trees to keep track of both the invariants of reachable states and of permissible controller actions. The reachability tree is constructed iteratively using an adaptation of the Tree-IC3 algorithm. Whenever a feasible error trace is encountered, the algorithm attempts to strengthen the controller to rule out the error trace. If an error trace cannot be removed by strengthening the controller, the system is uncontrollable; if no more error traces can be found, the plant is successfully controlled. By the properties of the construction, the controller is maximally permissive. Our experiments demonstrate the potential of IC3-based techniques in supervisory control of discrete-event systems compared to BDD-based and SAT-based approaches.

There are some promising directions of future research. IISC can be extended to cover nonblocking control problem, in which IICTL technique [11] can be exploited. Moreover, abstraction and optimization techniques can be used to improve the overall performance of the controller synthesis. We also consider the possibility of parallel implementation and using a hybrid approach that combines IISC with interpolant-based approaches, such as [19, 14], in order to get the benefits of both techniques.

[7] Benchmarks were performed on a workstation with a 2.67GHz Intel Core2 Quad processor and 2GB of available memory.

References

1. Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems. In: 8th Int. Work. Discret. Event Syst. pp. 384–385. Ann Arbor, MI, USA (2006)
2. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: Tools Algorithms Constr. Anal. Syst. pp. 193–207 (1999)
3. Bradley, A.R.: SAT-Based Model Checking without Unrolling. In: Verif. Model Checking, Abstr. Interpret. pp. 70–87 (2011)
4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer US, Boston, MA, 2nd edn. (2008)
5. Chen, Y.L., Lin, F.: Modeling of discrete event systems using finite state machines with parameters. In: IEEE Int. Conf. Control Appl. Conf. Proc. pp. 941–946 (2000)
6. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Comput. Aided Verif. pp. 277–293 (2012)
7. Claessen, K., Een, N., Sheeran, M., Sörensson, N., Voronov, A., Åkesson, K.: SAT-Solving in Practice, with a Tutorial Example from Supervisory Control. Discret. Event Dyn. Syst. 19(4), 495–524 (2009)
8. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Tools Algorithms Constr. Anal. Syst. (2008)
9. Een, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: Form. Methods Comput. Des. pp. 125–134 (2011)
10. Fei, Z., Miremadi, S., Åkesson, K., Lennartson, B.: A symbolic approach to large-scale discrete event systems modeled as finite automata with variables. In: 2012 IEEE Int. Conf. Autom. Sci. Eng. pp. 502–507. IEEE (2012)
11. Hassan, Z., Bradley, A.R., Somenzi, F.: Incremental, Inductive CTL Model Checking. In: Comput. Aided Verif. pp. 532–547 (2012)
12. Henzinger, T.a., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: ACM SIGPLAN Not. pp. 58–70 (2002)
13. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Theory Appl. Satisf. Test. pp. 157–171 (2012)
14. Hoder, K., Kovács, L., Voronkov, A.: Interpolation and symbol elimination in vampire. In: Autom. Reason., pp. 188–195. Springer (2010)
15. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Series in Computer Science, Pearson Education India, 3rd editio edn. (2007)
16. Kumar, R., Garg, V., Marcus, S.: Predicates and predicate transformers for supervisory control of discrete event dynamical systems. IEEE Trans. Automat. Contr. 38(2), 232–247 (1993)
17. Leduc, R., Lawford, M., Wonham, W.M.: Hierarchical interface-based supervisory control-part II: parallel case. IEEE Trans. Automat. Contr. 50(9), 1336–1348 (2005)
18. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Comput. Aided Verif. pp. 1–13 (2003)
19. McMillan, K.L.: Lazy abstraction with interpolants. In: Comput. Aided Verif. pp. 123–136 (2006)
20. Miremadi, S., Åkesson, K., Fabian, M., Vahidi, A.: Solving two supervisory control benchmark problems using Supremica. In: 9th Int. Work. Discret. Event Syst. pp. 131–136 (2008)

21. Miremadi, S., Lennartson, B., Åkesson, K.: A BDD-Based Approach for Modeling Plant and Supervisor by Extended Finite Automata. IEEE Trans. Control Syst. Technol. 20(6), 1421–1435 (2012)
22. Morgenstern, A., Gesell, M., Schneider, K.: Solving games using incremental induction. In: Integr. Form. Methods. pp. 177–191 (2013)
23. Ouedraogo, L., Kumar, R., Malik, R., Åkesson, K.: Nonblocking and Safe Control of Discrete-Event Systems Modeled as Extended Finite Automata. IEEE Trans. Autom. Sci. Eng. 8(3), 560–569 (2011)
24. Ramadge, P.J., Wonham, W.M.: Modular Feedback Logic for Discrete Event Systems. SIAM J. Control Optim. 25(5), 1202–1218 (1987)
25. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. 25(1), 635–650 (1987)
26. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. Proc. IEEE, Spec. Issue Discret. Event Dyn. Syst. 77(1), 81–98 (1989)
27. Sheeran, M., Singh, S., Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. In: Form. Methods Comput. Des. pp. 127–144 (2000)
28. Skoldstam, M., Åkesson, K., Fabian, M.: Modeling of discrete event systems using finite automata with variables. In: 46th IEEE Conf. Decis. Control. pp. 3387–3392 (2007)
29. Somenzi, F., Bradley, A.R.: IC3: where monolithic and incremental meet. In: Form. Methods Comput. Des. pp. 3–8 (2011)
30. Yang, Y., Gohari, P.: Embedded supervisory control of discrete-event systems. Int. Conf. Autom. Sci. Eng. pp. 410–415 (2005)