Received 27 November 2022, accepted 11 February 2023, date of publication 16 February 2023, date of current version 23 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3246123

RESEARCH ARTICLE

Model Abstraction for Discrete-Event Systems Using a SAT Solver

LIHONG CHENG[1,2] AND LEI FENG[2], (Member, IEEE)
[1]School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China
[2]Department Machine Design, ITM School, KTH Royal Institute of Technology, 10044 Stockholm, Sweden

Corresponding author: Lei Feng (lfeng@kth.se)

This work was supported in part by the China Scholarship Council under Grant 201606960044, and in part by the KTH XPRES Research Center.

ABSTRACT Model abstraction for finite state automata is beneficial to reduce the complexity of discrete-event systems (DES), enhance the readability and facilitate the control synthesis and verification of DES. Supremal quasi-congruence computation is an effective way for reducing the state space of DES. Effective algorithms on the supremal quasi-congruence relation have been developed based on the graph theory. This paper proposes a new approach to translate the supremal quasi-congruence computation into a satisfiability (SAT) problem that determines whether there exists an assignment for Boolean variables in the state-to-coset allocation matrix. If the result is satisfied, then the supremal quasi-congruence relation exists and the minimum equivalence classes is obtained. Otherwise, it indicates that there is no such supremal quasi-congruence relation, and a new set of observable events needs to be modified or reselected for the original system model. The satisfiability problem on the computation of supremal quasi-congruence relation is solved by different methods, which are respectively implemented by mixed integer linear programming (MILP) in MATLAB, binary linear programming (BLP) in CPLEX, and a SAT-based solver (Z3Py). Compared with the MILP and BLP methods, the SAT method is more efficient and stable. The computation time of model abstraction for large-scale systems by Z3Py solver is significantly reduced.

INDEX TERMS Discrete-event systems, deterministic finite automata, model abstraction, satisfiability, supremal quasi-congruence relation.

I. INTRODUCTION

In the supervisory control theory, a dynamic system driven by instantaneous events can be modeled as a discrete-event system (DES) with discrete state spaces and event-driven characteristics. Based on the state-transition structure characteristics of DESs, we usually use deterministic finite automata (DFAs) and Petri nets (PNs) to further simulate, analyze and design supervisory controllers of a DES [1], [2], [3], [4], [5], [6].

As the real system function becomes more advanced, the system complexity increases. The exponential increase of system model states and even the possibility of state explosion [7], [8], [9], [10], [11] may occur. Therefore, the research on methods supporting model abstraction for DES becomes important. Especially, it is necessary to simplify the state space of DES. Common simplification methods are developed, including partial order reduction [12], process equivalence [8], equivalence reduction [7], [13], model abstraction [9], symbolic computation and state tree structures [10], [11], which all simplify the model by analyzing the system architecture and then designing it appropriately.

In the DES theory, the equivalence relation computation is an effective method to simplify finite state automaton models. The equivalence relation [1] between states has three properties: reflexive, symmetric and transitive. The process of calculating an equivalence relation is to reduce the state space of a finite state automaton to a new automaton composed of equivalence classes (namely cosets). An equivalence class is a set of states composed of one or more states in the original state space. In general, we call the simplified automaton consisting of equivalence classes as a quotient automaton. Since the quotient automaton is obtained by dividing an equivalence relation between states, it generally has fewer states than the

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

original one and its observable language remains unchanged. The quasi-congruence equivalence relation [1], [14], [15], [16] guarantees the identity property that quotient automaton and original automaton have the same observable sequence of events. It also implies that any two states in the same equivalence class must have the same observable sequence of events in the original automaton.

The approach in [14] is to compute the coarsest equivalence relation, which is finer than that of the given causal reporter map. In this work, the algorithm on how to design an observer is developed. It is shown that this algorithm can be used to verify the observer property of natural projection and to compute an observer of an automaton in polynomial time, based on modifying the given causal reporter map that is not an observer. However, this algorithm cannot be directly applied to natural projection.

The method of minimum event extension is proposed and computed in [15], and applied to the algorithm in [14]. Although it is shown that the approach in [15] is inherently NP-hard, the polynomial-time algorithm for finding an acceptable event extension has been developed in [15]. The work in [15] makes the algorithm in [14] suitable for natural projection.

Computing the supremal quasi-congruence relation of automata is an effective approach for reducing the state space of DES. On the one hand, it can reduce the computational complexity of the system. On the other hand, it can improve the comprehensibility of control synthesis and verification of DES. Effective algorithms for the coarsest quasi-congruence have been studied in these works [1], [14], [15], [16], [17]. These contemporary approaches [14], [15], [17], [18], [19], [20] are based on graph theory and state-transition data structure.

A quasi-congruence relation is a special equivalence partition of the state set of an automaton and has the advantage of preserving the state transition relationship of the observable events of the automaton. Consequently, the state partition creates a smaller automaton whose language is equivalent to the natural projection of the language of the original automaton. Hence, quasi-congruence can be used for behavior-preserving model abstraction of a large automaton. Previous studies [1], [14], [15] have shown that the join of quasi-congruence relations is still a quasi-congruence relation. Thus, the supremal (coarsest) quasi-congruence for a given observable event set exists for an automaton. In this sense, finding the supremal quasi-congruence relation is equivalent to obtaining the least number of cosets. Therefore, the computation of supremal quasi-congruence relation plays an essential role in the equivalent simplification of complex system models.

However, the supremal quasi-congruence relation in supervisory control theory is a relatively isolated concept, which is studied and defined on the basis of state-transition data structure. There is almost no correlation between the model abstraction method in SCT and the partitioning or grouping methods intensively studied in the optimization theory. To exploit efficient optimization solvers, our previous work [16] converts the supremal quasi-congruence computation into a binary linear programming (BLP) problem to minimize the number of cosets. The novelty of the method is to formulate the state partition problem as an optimal state to coset allocation problem with linear constraints. The reformulation shows the potential of finding the supremal quasi-congruence by powerful optimization solvers.

The previous approach of using BLP suffers long computation time for large scale DES models. The reason is that most of the constraints to satisfy the properties of quasi-congruence are first order logic predicates, but they have to be converted into linear inequalities to suit the formulation of BLP. Since one logic predicate is often equivalent to many linear inequalities, the BLP problem becomes very large and this significantly increases the computation time. To reduce the complexity of the optimization problem, this paper adopts the Boolean satisfiability (SAT) solver and avoids the tedious conversion from logic predicates to linear inequalities. The optimization problem has much less constraints and hence can be solved more efficiently.

Boolean satisfiability (SAT) [21], [22] is to decide whether a propositional logic formula can be satisfied by suitable value assignments to the variables of the formula. If there exists an assignment that satisfies the propositional logic formula, then the logical proposition formula that returns true is called satisfiable and is denoted as Sat. Otherwise, the problem cannot be solved and is called Unsat. SAT solvers have the advantage of being simple and intuitive, which can directly solve problems with logical propositional constraints. The Z3Py solver used in this paper mainly refers to the Z3 API in Python, which has a wide range of applications, such as solving all kinds of equations, solving all kinds of programming problems (such as Sudoku) and solving logic problems. It has the advantage of being very efficient in dealing with some large-scale search spaces through the unique structure of the search problem itself.

As we know, SAT is essentially a decision problem, which is mainly used to solve combinatorial problems in modeling frameworks [23], [24]. Also, SAT is a NP-complete decision problem [25]. As a result, unless $P = NP$, all SAT algorithms require exponential time in the worst case. However, the performance improvements [26], [27] in SAT solvers have driven their widespread use since the mid 90s. The application of SAT methods [26], [28], [29], [30], [31], [32], [33] has been continuously expanded in recent years. It is worth mentioning that the applications of SAT have led to significant performance improvements in some real-world applications [34], [35], [36]. Therefore, the current SAT algorithm has the advantage of being very efficient in dealing with some large search spaces through the unique structure of the search problem itself. In addition, the SAT method also has important research value for some optimization problems related to decision making.

The novelty of this paper is to translate the supremal quasi-congruence computation into a Boolean satisfiability problem. Actually, the calculation of supremal quasi-congruence relation is an optimal allocation problem from states to equivalence classes. Allocating states to equivalence classes is considered as whether there is a valid set of state distribution values such that the allocation matrix satisfies all constraints. In the allocation matrix, each line represents which equivalence class a state belongs to and each column represents which states a equivalence class contains. More specifically, finding the supremal quasi-congruence relation is equivalent to finding the least number of equivalence classes. Finally, we apply a SAT-based solver Z3Py to reduce the state space of automata. It is a powerful, extensible and open source software tool that is popular today. Compared with the binary linear programming method [16] developed before, this method is more efficient, more intuitive and easier to understand.

This paper formalizes the problem of computing the supremal quasi-congruence relation as a Boolean satisfiability problem. We first assume an assignment matrix consisting of Boolean variables based on the relationships between states and equivalence classes. The constraint that all Boolean variables need to satisfy is the supremal quasi-congruence relation. The ultimate goal is to minimize the number of equivalence classes such that there is a valid set of Boolean variable values, i.e., the allocation matrix exists. The main contributions of this paper are as follows.

(1) The approach on how to translate the supremal quasi-congruence computation into a SAT problem is represented, which is based on Boolean variables in Section IV. These Boolean variables must satisfy three constraints from Sat_1 to Sat_3.
(2) An implementation of this method in the Z3Py solver is proposed in Section V. In the implementation, we convert all constraints on the supremal quasi-congruence relation into a simple logical proposition.
(3) The main algorithms of this method are proposed in Section VI. The correctness and efficiency of the proposed method are verified by comparison with other methods in Section VII.
(4) In this paper, we show how to adapt standard algorithms in Section VI for symbolic quasi-congruence relation computation to work with a SAT-based solver. Our main contributions are to obtain an efficient and reusable method for model abstraction and to provide the algorithms for removing quantifiers on Boolean variables.

The SAT method proposed in this paper is the first to use the satisfiability method to solve the supremal quasi-congruence relation problem of the supervisory control theory. A large number of experiments prove that the SAT method is efficient and stable. The experimental results show the SAT method is also reusable for different discrete-event systems. It lays the foundation for the later study of industrial scale system model abstraction, and also provides the possibility to combine with the current powerful computer software methods to study more complex problems.

In the future, we can apply the proposed method in the paper to compute model abstraction of finite automata with variables, which are easily applicable to the real industrial scale problems. Furthermore, the paper not only builds a bridge for the study of satisfiability method and supervisory control theory, but also opens a new research perspective for the study of supremal quasi-congruence relation and broadens its future application field.

Compared with the two methods of mixed integer linear programming (MILP) in MATLAB and the binary linear programming (BLP) [16] we studied before, this SAT method greatly improves the efficiency of solving problems. Especially for some large-scale problems, the SAT method we proposed greatly reduces the calculation time. Its calculation results are proved to be correct and consistent with those of TCT[1] [37] method, MILP method and BLP method [16]. The introduction of satifiability problem on the computation of supremal quasi-congruence relation brings a substantial extension of the class of systems that can be verified by algorithms. It also lays a foundation for subsequent research on automata with variables.

The structure of this paper is as follows. Section II recalls the theoretical knowledge on automata, natural projection, satisfiability and Z3Py. In Section III, the functions and concepts on the supremal quasi-congruence calculation are introduced. Main algorithms of the proposed approach are reported in Section VI. At the end of this paper, we make a summary and outlook of this research.

II. PRELIMINARIES

This section recalls some basic concepts of deterministic finite automata and satisfiablity used in the paper. Then we review the definition of quasi-congruence relation and some operations on it. More details on automata can be found in [1], [2], [14], [15], [38], [39], and [40].

A. DETERMINISTIC FINITE AUTOMATA

A *deterministic finite automaton (DFA)* [1] is represented as a five-tuple as follows.

$$G = (Q, \Sigma, \delta, q_1, Q_m) \tag{1}$$

• Q is the finite state set $\{q_1, q_2, \ldots, q_n\}$ and $n = |Q|$ is the number of states of the automaton.

• Σ is a nonempty finite event set with $\Sigma = \Sigma_o \cup \Sigma_u$, where Σ_o is an observable event set and Σ_u is the unobservable event set of G.

• Its state transition function is a partial function, denoted as $\delta : Q \times \Sigma \rightarrow Q$. The notation $\delta(q, \sigma)!$ is used to indicate that the function $\delta(q, \sigma)$ is defined on an event $\sigma \in \Sigma$.

• The initial state is $q_1 \in Q$.

[1]The software TCT is available on the website https://www.control.utoronto.ca/DES/Research

• The set of marker states is Q_m and $Q_m \subseteq Q$.

Note that Σ^* represents the set of all finite strings over Σ. ε denotes the empty string with length 0.

B. NATURAL PROJECTION

Given a DFA G, Σ is a nonempty finite event set and $\Sigma_o \subseteq \Sigma$ is an observable event set, the corresponding *natural projection* [1] function is $P : \Sigma^* \rightarrow \Sigma_o^*$, such that $P(\varepsilon) = \varepsilon$ and

$$P(\sigma) = \begin{cases} \varepsilon, \ \sigma \in \Sigma - \Sigma_o \\ \sigma, \ \sigma \in \Sigma_o \end{cases} \tag{2}$$

$$P(s\sigma) = P(s)P(\sigma), \ s \in \Sigma^*, \sigma \in \Sigma \tag{3}$$

Next, the natural projection P can be extended to the *image function* [15], $P : P_{wr}(\Sigma^*) \rightarrow P_{wr}(\Sigma_o^*)$ such that

$$\forall L \subseteq \Sigma^*, \ P(L) := \{P(s) \mid s \in L\}. \tag{4}$$

Note that $P_{wr}(A)$ is the power set of set A. Mathematically, the inverse function of natural projection P does not exist. Here, the notation of the function P^{-1} is expressed as the *inverse image function* [16], which can be defined as $P^{-1} : P_{wr}(\Sigma_o^*) \rightarrow P_{wr}(\Sigma^*)$ such that

$$\forall L_o \subseteq \Sigma_o^*, \ P^{-1}(L_o) := \{s \in \Sigma^* \mid P(s) \in L_o\}. \tag{5}$$

C. SATISFIABILITY

Given a propositional formula, the satisfiability problem is to decide whether there exists a variable assignment such that the formula evaluates to true. Satisfiability problem, is also called for Boolean Satisfiability (SAT) [21], [41], [42] and it belongs to the classic NP-complete problem [43].

For a formal definition of satisfiability problems, we need to understand the following three basic concepts [44], [45].

• Given a boolean variable x, a *literal* is seen as either x or its negation $\neg x$.

• A *clause* is the disjunctions of Boolean variables, for example $x_1 \vee x_2 \vee \neg x_3$.

• If a propositional logic formula is *conjunctive normal form* (CNF), that is, it is represented as a conjuction "and" ($\wedge$) of disjunctions "or" ($\vee$) of literals.

The satisfiability (SAT) problem is to find an assignment to the Boolean variables, such that the CNF formula evaluates to true. In short, each clause in the CNF formula has at least one literal that is true. Such a CNF formula is *satisfied* (Sat). If there is not a set of Boolean assignments such that all clauses are true, the CNF formula is said to be *unsatisfiable* (Unsat).

If a propositional formula is not in CNF formula, we can convert it to CNF in a standard way, and this process is called *clausification* [21], [43].

D. Z3Py

Z3 is a high performance theorem prover developed at Microsoft Research. Z3 is used in many applications such as software verification and testing, hardware verification and testing, constraint solving, analysis of hybrid systems, security, biology and geometrical problems. The Z3Py used in this paper mainly refers to the Z3 API in Python. The Z3 distribution also contains the C, .Net and OCaml APIs. Moreover, the source code of Z3Py is available in the Z3 distribution.

Z3 provides functions for logical expressions and all basic mathematical operations. Z3Py uses the same operator precedence of the Python language. Also, Z3 supports Boolean operators: And, Or, Not, Implies (implication), If (if-then-else). Bi-implications are represented using equality $==$. As usual, $\wedge$ is the logical and, $\vee$ is the logical or, and Z3Py solver like Python uses $=$ for assignment, operators $<$, $\leq$, $>$, $\geq$, $==$ and $! =$ for comparison. Finally, the Python Boolean constants True and False can be used to build Z3 Boolean expressions. Since Z3 can solve and crunch formulas, Z3Py displays formulas and expressions using mathematical notation.

In Z3Py, the command *solver*() creates a general purpose solver. Constraints can be added using the method *add*. We say that the constraints have been *asserted* in the solver. The method *check*() solves the asserted constraints. The result is sat (satisfiable) if a solution is found. The result is unsat (unsatisfiable) if no solution exists. We may also say that the system of asserted constraints is *infeasible*. Finally, a solver may fail to solve a system of constraints and unkown is returned.

In addition, Z3Py supports arbitrarily large numbers and only *algebraic irrational numbers*. Algebraic irrational numbers are sufficient for presenting the solutions of systems of polynomial constraints. Z3Py will always display irrational numbers in decimal notation since it is more convenient to read.

III. EQUIVALENCE RELATION AND QUASI-CONGRUENCE RELATION

This section defines three important functions and some concepts associated with the quasi-congruence computation.

A. EQUIVALENCE RELATION

Suppose that $E(Q)$ is the lattice of equivalence relations on the state set Q. Given an *equivalence relation* $\pi \in E(Q)$, it must meet the following properties [1], [46].

(1) $(\forall q \in Q)\ q\pi q$ (π is reflexive.)
(2) $(\forall q, q' \in Q)\ q\pi q' \Rightarrow q'\pi q$ (π is symmetric.)
(3) $(\forall q, q', q'' \in Q)\ q\pi q' \wedge q'\pi q'' \Rightarrow q\pi q''$ (π is transitive.)

In this paper, $q\pi q'$ is also written as $q \equiv q'(mod\ \pi)$. For any state $q \in Q$, the *coset* (or *equivalence class*) of q with respect to the equivalence relation π is denoted as $[q]_\pi$.

$$[q]_\pi := \{q' \in Q \mid q'\pi q\} \subseteq Q \tag{6}$$

By reflexivity $q \in [q]_\pi$, every coset is nonempty. In this paper, Q/π represents the set of all cosets.

B. FUNCTIONS

The following three function definitions [16] are inseparable from the computation of the supremal quasi-congruence relation. Given a DFA G, we can decide the possible reachable states from any state $q_i \in Q$ based only on observation of the events in Σ_o.

Definition 1: Let $s_o \in \Sigma_o^*$ be an observable string and $q_i \in Q$ a state. The function Δ is defined as

$$\Delta : Q \times \Sigma_o^* \rightarrow P_{wr}(Q) \tag{7}$$

and
$\Delta(q_i, s_o) := \{q' \in Q \mid (\exists s \in \Sigma^*)\, P(s) = s_o \wedge q' = \delta(q_i, s)\}$.
Note that if $\delta(q_i, s)$ is undefined for every string s with $P(s) = s_o$, then $\Delta(q_i, s_o) = \emptyset$. In addition, if $s_o = \sigma \in \Sigma_o$, we can simplify the notation of $\Delta(q_i, \sigma)$ to $\Delta_\sigma(q_i)$, which denotes *the set of reachable states* from state $q_i \in Q$ via an observable event σ. Similarly, if $s_o = \varepsilon$, it can be written as $\Delta_\varepsilon(q_i)$ which denotes the set of all reachable states from state $q_i \in Q$ via a sequence of unobservable events in $\Sigma - \Sigma_o$. Moreover, *the set of all reachable marker states* can be computed by $\Delta_m(q_i) = \Delta_\varepsilon(q_i) \cap Q_m$.

For concise presentation, we define a new event $\mu \notin \Sigma$, and define Δ_μ as Δ_m and $\Sigma_o' = \Sigma_o \cup \{\mu\}$. In addition, we denote $\mathbf{n} = \{1, 2, \cdots, n\}$.

Definition 2: Let π be an equivalence relation, $q_i \in Q$ and $\sigma \in \Sigma_o$. Denote the *canonical projection* function as $P_\pi : Q \rightarrow Q/\pi : q_i \mapsto [q_i]_\pi$. We can further define its extension projection function as

$$P_\pi : P_{wr}(Q) \rightarrow P_{wr}(Q/\pi). \tag{8}$$

Definition 3: Function $P_\pi \circ \Delta_\sigma$ is the composition function of Δ_σ and P_π,

$$P_\pi \circ \Delta_\sigma : Q \rightarrow P_{wr}(Q/\pi). \tag{9}$$

At a state $q_i \in Q$, this composed function $P_\pi \circ \Delta_\sigma$ can be further expressed as $P_\pi(\Delta_\sigma(q_i)) := \{[q]_\pi \mid q \in \Delta_\sigma(q_i)\}$.

The composite $\pi \circ \Delta_\sigma$ ($\sigma \in \Sigma_o'$) defines an *equivalence relation* on Q as follows. Suppose $i \neq j$ and $i, j \in \mathbf{n}$.

$(\forall q_i, q_j \in Q)\; q_i \equiv q_j (mod\ \pi \circ \Delta_\sigma) \Leftrightarrow$

$$\begin{cases} (\forall x \in \Delta_\sigma(q_i))\,(\exists x' \in \Delta_\sigma(q_j))\; x \equiv x'(mod\ \pi); \\ (\forall x' \in \Delta_\sigma(q_j))\,(\exists x \in \Delta_\sigma(q_i))\; x' \equiv x(mod\ \pi). \end{cases} \tag{10}$$

C. QUASI-CONGRUENCE RELATION

Assume that (Q, Δ) is a nondeterministic dynamic system where Q is the set of system states and $\Delta : Q \rightarrow P_{wr}(Q)$ is its state transition function. According to [1], [14], and [15], we can define the quasi-congruence relation as follows.

Definition 4: An equivalence relation π on Q is a *quasi-congruence* for (Q, Δ), if it satisfies the following three equivalent conditions [16].

(1) $\pi \leq \pi \circ \Delta$.
(2) $(\forall q_i, q_j \in Q)\; P_\pi(q_i) = P_\pi(q_j)$
$\Rightarrow P_\pi \circ \Delta(q_i) = P_\pi \circ \Delta(q_j)$.
(3) $(\forall q_i, q_j \in Q)\; q_i \equiv q_j(mod\ \pi)$
$\Rightarrow q_i \equiv q_j(mod\ \pi \circ \Delta)$.

D. THE SUPREMAL QUASI-CONGRUENCE

Definition 5: Given a DFA G and the observable event set $\Sigma_o \subseteq \Sigma$, the *supremal quasi-congruence* [14], [16], [47], [48] with respect to Q and Σ_o' is

$$\rho := sup\{\pi \in E(Q) \mid \pi \leq (\wedge_{\sigma \in \Sigma_o'}(\pi \circ \Delta_\sigma))\}. \tag{11}$$

E. QUOTIENT AUTOMATON

For a state set Q and the coarsest quasi-congruence relation $\rho \in E(Q)$, we have the canonical projection $P_\rho : Q \rightarrow Q/\rho$ as defined in Definition 2. Renaming the cosets in Q/ρ, we get a new state set Q' and a bijection $r : Q/\rho \cong Q'$. The composition of P_ρ and r is denoted as a function $g = r \circ P_\rho$, with equivalence kernel $kerg = \rho$.

Definition 6: Given a DFA $G = (Q, \Sigma, \delta, q_1, Q_m)$, an observable event set $\Sigma_o \subseteq \Sigma$, and an equivalence relation $\rho \in E(Q)$, the *quotient automaton* [1], [48] is

$$G' = (Q', \Sigma_o, \eta, q_1', Q_m') = G/(\Sigma_o, \rho) \tag{12}$$

where $Q' \cong Q/\rho$, $q_1' := g(q_1)$, $Q_m' := g(Q_m)$. Here, the transition function is $\eta : Q' \times (\Sigma_o \cup \{\epsilon\}) \rightarrow P_{wr}(Q')$ as follows.

$$\eta(q', \sigma) \mapsto \begin{cases} g \circ \delta(g^{-1}(q'), \sigma), & if\ \sigma \in \Sigma_o \\ g \circ \delta(g^{-1}(q'), \Sigma - \Sigma_o) - \{q'\}, & if\ \sigma = \epsilon \end{cases}$$

The quotient automaton may be nondeterministic.

IV. TRANSLATION TO SAT

This section describes the definition of Boolean matrix variables to translate the supremal quasi-congruence relation into a satisfiability problem.

Let $\pi \in E(Q)$ be a quasi-congruence relation on the state set Q. Assume that π partitions Q into $m \leq n$ equivalence classes. Then $Q/\pi = \{C_1, C_2, \cdots, C_m\}$ and C_j represents an equivalence class. Since the number of states and variables directly affects the complexity of problem in supervisory control theory, we use the permutation of equivalence class to decrease the number of unknown variables in our previous work [16].

First, the *reachable matrix* of each event σ ($\sigma \in \Sigma_o'$) is defined as a Boolean matrix R_σ, which is an $n \times n$ square matrix and $n = |Q|$. For any $i, j \in \mathbf{n}$,

$$R_\sigma(i, j) = \begin{cases} 1, & q_j \in \Delta_\sigma(q_i); \\ 0, & otherwise. \end{cases} \tag{13}$$

Second, we have shown that a quasi-congruence relation π can be expressed as a lower triangular Boolean allocation matrix $X_{n \times m}$ illustrated by Tab. 1 [16], where

$$x_{ij} = \begin{cases} 0, & q_i \notin C_j; \\ 1, & q_i \in C_j. \end{cases} \tag{14}$$

There are $\frac{m(2n-m+1)}{2}$ Boolean variables in total.

TABLE 1. Boolean variables allocation matrix $X_{n\times m}$.

Q	C_1	C_2	...	C_m
q_1	x_{11}	0	0	0
q_2	x_{21}	x_{22}	0	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	**0**
q_m	x_{m1}	x_{m2}	...	x_{mm}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_n	x_{n1}	x_{n2}	...	x_{nm}

Third, if two states q_i, q_j belong to the same coset C_k ($1 \leq k \leq m$), then they must meet the following constraints C3 [16]. For all the state pairs $q_i, q_j \in Q$ and all $\sigma \in \Sigma_o'$,

$$q_i \equiv q_j(mod\ \pi) \rightarrow P_\pi \circ \Delta_\sigma(q_i) = P_\pi \circ \Delta_\sigma(q_j). \quad (15)$$

Here, the formulas $P_\pi \circ \Delta_\sigma(q_i)$ and $P_\pi \circ \Delta_\sigma(q_j)$ can be implemented by Boolean matrix multiplication, denoted by $\bigwedge$. Its rule is the same as normal matrix multiplication, but replaces multiplication by logic $\wedge$ and addition by logic $\vee$. Given a state $q_i \in Q$, $\sigma \in \Sigma_o'$, $P_\pi \circ \Delta_\sigma(q_i)$ is represented as

$$R_\sigma(i,:) \bigwedge X_{n\times m} = X(i_1,:) \vee X(i_2,:) \vee \ldots \vee X(i_p,:).$$

Similarly, $P_\pi \circ \Delta_\sigma(q_j)$ is equivalent to the row vector

$$R_\sigma(j,:) \bigwedge X_{n\times m} = X(j_1,:) \vee X(j_2,:) \vee \ldots \vee X(j_t,:).$$

Thus, it follows that the equality $P_\pi \circ \Delta_\sigma(q_i) = P_\pi \circ \Delta_\sigma(q_j)$ is equivalent to the new vector equality as follows.

$$R_\sigma(i,:) \bigwedge X_{n\times m} = R_\sigma(j,:) \bigwedge X_{n\times m}. \quad (16)$$

Remark 1: $R_\sigma(i,:)$ is a $1 \times n$ row vector and X is an $n \times m$ matrix, then the Boolean multiplication is a $1 \times m$ row vector. The notation of $X(i_p,:)$ means that all the elements of the row correspond to state q_{i_p} in the allocation matrix $X_{n\times m}$.

Finally, we translate the supremal quasi-congruence computation into a Boolean satisfiability problem, and these Boolean variables in the allocation matrix $X_{n\times m}$ must satisfy the following constraints from Sat_1 to Sat_3.

Sat_1: There must exist a permutation of the equivalence classes such that the allocation matrix $X_{n\times m}$ is lower triangular [16], i.e.,

$$(\forall i \in \mathbf{n})\ (\forall j \in \mathbf{m})\ i < j \rightarrow x_{ij} = 0. \quad (17)$$

Sat_2: A clause ensures that the element x_{ij} for each state must belong to one and only one equivalence class, i.e., the following constraint makes sure that each state does not belong to different equivalence classes at the same time.

$$\wedge \begin{cases} (x_{11} = 1) \\ \wedge_{2\leq i<m}\{(\vee_{j=1}^{i} x_{ij} = 1) \wedge [\vee_{1\leq u<v\leq i}(x_{iu} \wedge x_{iv}) = 0]\} \\ \wedge_{m\leq i\leq n}\{(\vee_{j=1}^{m} x_{ij} = 1) \wedge [\vee_{1\leq u<v\leq m}(x_{iu} \wedge x_{iv}) = 0] \end{cases} \quad (18)$$

Sat_3: According to (15) and (16), if two states $q_i, q_j \in Q$ belong to the same equivalence class, i.e., $q_i \in C_k$, $q_j \in C_k$ ($k \in \mathbf{m}$), then the values of the elements of the two rows $X(i,:)$ and $X(j,:)$ corresponding to the two states q_i and q_j in the Boolean allocation matrix must be identical, i.e., $X(i,:) = X(j,:)$.

$$[X(i,:) = X(j,:)] \rightarrow$$

$$\{\wedge_{\sigma\in\Sigma_o'}[R_\sigma(i,:) \bigwedge X_{n\times m} = R_\sigma(j,:) \bigwedge X_{n\times m}]\}. \quad (19)$$

The correctness of the translation procedure is verified in [16].

V. IMPLEMENTATION IN THE Z3Py SOLVER

In the section, an implementation of the supremal quasi-congruence computation in the interactive theorem solver Z3Py is presented. This satisfiability method proposed in the paper is intuitive and easy to understand.

The supremal quasi-congruence relation between states and cosets is modelled by a lower triangular allocation matrix $X_{n\times m}$. Here, $n = |Q|$ is the size of states, m is the number of cosets we supposed and $m \leq n$.

Definition 7: Let x_{ij} be a Boolean variable in the allocation matrix $X_{n\times m}$ and $m \leq n$. Computing the minimum number of equivalence classes can be expressed as the following logical operation.

$$Sat(\{x_{ij}\}_{i\in\mathbf{n},j\in\mathbf{m},j\leq i}) \equiv \min \sum_{j=1}^{m}(\vee_{i=1}^{n} x_{ij}) = |QC| \quad (20)$$

Labeling the lower triangular allocation matrix $X_{n\times m}$ as shown in Tab. 1. Each row represents the distribution of which coset a state belongs to, and each column represents all states contained in the coset. Note that if there is a column of 0 elements in the matrix $X_{n\times m}$, the equivalence class does not exist and is an empty set $\emptyset$, denoted as 0 in the paper.

Definition 8: Let x_{ij} be a Boolean variable in the allocation matrix $X_{n\times m}$. Based on the three logical constraints Sat_1, Sat_2 and Sat_3 introduced in the previous section, we define the following logical formula to decide whether the supremal quasi-congruence relation is satisfied for certain automaton.

$$QC(\{x_{ij}\}_{i\in\mathbf{n},j\in\mathbf{m},j\leq i}) \equiv Sat_1 \wedge Sat_2 \wedge Sat_3 \quad (21)$$

VI. MAIN ALGORITHMS

In this section, we propose a satisfiability method to solve the problem of supremal quasi-congruence. The main algorithm functions of this method are shown below.

A. SIMPLIFIED CALCULATION

In the following Algorithm 1, *trans* is a matrix of state-transition data with three columns representing states, events, and states. Σ is the set of events and $Event_0$ represents an observable events set.

This algorithm sets all unobservable events in $\Sigma - Event_0$ as 0. A new state-transition data table is obtained for G. The purpose is to simplify the calculation and to save the subsequent search time. In order to save the storage space generated during calculation, we use the sparse matrix in this paper, according to the way of column storage.

Algorithm 1 Set All Unobservable Events as 0

```
Require: $trans$, $\Sigma$, $Event_0$.
Ensure: A new state-transition data is store in $G$.
1: for $i$ in $range(len(trans))$ do
2:   if $trans[i][1]$ not in $Event_0$ then
3:     $trans[i][1] = 0$;
4:   end if
5:   $G = trans$;
6: end for
```

B. COMPUTATION OF ALL REACHABLE STATES VIA A SEQUENCE OF UNOBSERVABLE EVENTS

According to (7), if $s_o = \varepsilon$, then $\Delta_\varepsilon(q_i)$ denotes the set of all reachable states from state $q_i \in Q$ via a sequence of unobservable events in $\Sigma - \Sigma_o$. Based on Algorithm 1, we define Algorithm 2 below to calculate all reachable states via a sequence of unobservable events.

Algorithm 2 Definition of the Function $R\_unobsv()$

```
Require: $New_1$, $G$.
Ensure: $All_1$ is equal to $\Delta_\varepsilon(Q)$.
1: def $R\_unobsv(New_1, G)$:
2: $All_1 = set()$;
3: $X = set()$;
4: while $New_1 != set()$ do
5:   $q = New_1.pop()$;
6:   $All_1.add(q)$;     /* $All_1 = All_1|q$. */
7:   for $i$ in $range(len(G))$ do
8:     if $G[i][0] == q$ then
9:       if $G[i][1] == 0$ then
10:        $s = G[i][2]$;
11:        $X.add(s)$;     /* $X = G(L, 3)'$. */
12:      end if
13:    else
14:      continue;
15:    end if
16:  end for
17:  $y = X.difference(All_1)$;
18:  $New_1 = New_1.union(y)$;
19: end while
20: return $All_1$;
```

In Algorithm 2, New_1 represents a set of new states visited by the last iteration and $set()$ represents the empty set $\emptyset$. G is a state-transition matrix with three-column: initial states, events and reachable states. Line 5 takes the first state q from New_1 every time, i.e., $q = New_1[1]$; $New_1 = New_1 - q$. Lines 7–16 search all reachable states from the state q via a sequence of unobservable events, i.e., $L = [G(:,1) == q] \wedge [G(:,2) == 0]$, and line 11 stores them in X. Line 17 represents $y = X - All_1$, this is to avoid double counting for the same state. Finally, All_1 returns all reachable states from a state set $New_1 \subseteq Q$ via a sequence of unobservable events.

C. COMPUTATION OF ALL REACHABLE STATES VIA AN OBSERVABLE EVENT

According to (7), if $s_o = \sigma \in \Sigma_o$, we can simplify the notation of $\Delta(q_i, \sigma)$ to $\Delta_\sigma(q_i)$, which denotes *the set of reachable states* from state $q_i \in Q$ via an observable event σ. Similarly, we can define Algorithm 3 using breadth-first search (BFS) as follows, to compute all reachable states via an observable event.

Algorithm 3 Definition of the Function $R\_obsv()$

```
Require: $StateSet$, $G$, $e$.
Ensure: $All_2$ is equal to $\Delta_e(Q)$. Here, $e \in Event_0$.
1: def $R\_obsv(StateSet, G, e)$:
2: $All_2 = set()$;
3: $T = set()$;
4: while $StateSet != set()$ do
5:   $q = StateSet.pop()$;
6:   for $i$ in $range(len(G))$ do
7:     if $G[i][0] == q$ then
8:       if $G[i][1] == e$ then
9:         $t = G[i][2]$;     /* $T = G(L, 3)'$. */
10:        $T.add(t)$;
11:      end if
12:    else
13:      continue;
14:    end if
15:  end for
16:  $All_2 = All_2|T$;     /* $All_2 = union(All_2, T)$. */
17: end while
18: return $All_2$;
```

In Algorithm 3, $StateSet$ represents a set of states given as the input, G is the state-transition data matrix and $e \in Event_0$ is an observable event. Line 5 takes the first state q from $StateSet$ every time, i.e., $q = StateSet[1]$; $StateSet = StateSet - q$. Lines 6–15 implement the search for states that can be reached from a state q via an observable event e and line 10 stores them in T, i.e., $L = [G(:,1) == q] \wedge [G(:,2) == e]$. Line 16 shows that All_2 returns all reachable states from a state set $StateSet \subseteq Q$ via an observable event e.

D. COMPUTATION OF ALL REACHABLE MATRICES

On the basis of the above two function definitions, we further define the function $Reachable\_Matrix()$ that can compute all reachable matrices as follows.

In Algorithm 4, G is the state-transition data matrix, n is the number of states, Q_m represents the set of marker states and $Event_0$ is an observable event set. Line 3 defines R as a list of bit vector, storing all reachable matrices. Line 6 represents that the marker reachable matrix is stored in $M_{n\times n}$. In line 7, $R\_matrix_{n\times n}$ stores the reachable matrix based on an observable event e. Line 11 computes $M\_col = intersect(All_1, Q_m)$, i.e., $\Delta_m(q_i) = \Delta_\varepsilon(q_i) \cap Q_m$. Note that the function $list()$ is required in line 12 and line 19, since it is a necessary step to convert a collection into a list. Line 18 performs $All_2 =$

Algorithm 4 Definition of Function *Reachable_Matrix*()

```
Require: $G, n, Q_m, Event_0$.
Ensure: All reachable matrices are stored in a list of bit vector $R$.
 1: def $Reachable\_Matrix(G, n, Q_m, Event_0)$:
 2: $p = len(Event_0)$;
 3: $R = [BitVec(``r\_\%s''\%t, 1)$ for $t$ in $range(p + 1)]$;
 4: for $s$ in $range(len(event_0))$ do
 5:     $e = Event_0[s]$;
 6:     $M = np.zeros((n, n), dtype = np.int)$;
 7:     $R\_matrix = np.zeros((n, n), dtype = np.int)$;
 8:     for $i$ in $range(0, n)$ do
 9:         $Q = set([i])$;
10:         $All_1 = R\_unobsv(Q, G)$;
11:         $M\_col = All_1.intersection(Q_m)$;
12:         $M\_col = list(M\_col)$;
13:         for $j$ in $range(len(M\_col))$ do
14:             $u = M\_col[j]$;
15:             $M[i][u] = 1$;
16:             $T = R\_obsv(All_1, G, e)$;
17:             $Y = R\_unobsv(T, G)$;
18:             $All_2 = T.union(Y)$;
19:             $All_2 = list(All_2)$;
20:         end for
21:         for $k$ in $range(len(All_2))$ do
22:             $R\_matrix[i][All_2[k]] = 1$;
23:         end for
24:     end for
25:     $R[s] = R\_matrix$;
26:     $R[p] = M$;
27: end for
28: return $R$;
```

$union(T, Y)$, and All_2 is the set of all reachable states via an observable event. Finally, all reachable matrices are stored in a column-compressed list $R = [R[0], R[1], \ldots, R[p-1], M]$. This is done to save storage space and simplify subsequent searches and calculations.

E. DEFINITION OF MATRIX MULTIPLICATION

According to (16), we define the Boolean matrix multiplication algorithm used in this article as follows.

In Algorithm 5, line 2 defines the number of rows and columns of matrix C is equal to the number of rows of matrix A and the number of columns of matrix B. Note that $len(B[0])$ is the number of columns of matrix B in line 4, while $len(B)$ is the number of rows of matrix B in line 5. Line 6 implements this operation: $C_{ij} = (A_{i1} \wedge B_{1j}) \vee (A_{i2} \wedge B_{2j}) \vee \ldots \vee (A_{in} \wedge B_{nj})$.

F. CALCULATION OF SUPREMAL QUASI-CONGRUENCE

Based on the definition of the above functions and algorithms, the main algorithm for how to transform the supremal quasi-congruence computation problem into a satisfiability problem is as follows.

Algorithm 5 Definition of Matrix Multiplication *Matrix_Multi*()

```
Require: Two matrices: $A$ and $B$.
Ensure: The matrix multiplication is $C = A \bigwedge B$.
 1: def $Matrix\_Multi(A, B)$:
 2: $C = [[0$ for $col$ in $range(len(B[0]))]$ for $row$ in $range(len(A))]$
 3: for $i$ in $range(len(A))$ do
 4:     for $j$ in $range(len(B[0]))$ do
 5:         for $k$ in $range(len(B))$ do
 6:             $C[i][j]| = A[i][k] \& B[k][j]$;
 7:         end for
 8:     end for
 9: end for
10: return $C$;
```

In Algorithm 6, Q_m represents the set of marker states, $Event_0$ is an observable event set. R is a list of bit vector, where all reachable matrices are stored. $X_{n\times m}$ is the allocation matrix obtained in line 3, n is the size of states, m is the number of cosets and $m \leq n$. p represents the number of observable events in line 2.

Lines 5–11 represent the first constraint Sat_1 that the allocation matrix $X_{n\times m}$ is a lower triangular matrix. Based on the constraint Sat_1, lines 12–37 ensure the constraint Sat_2 that each row of the matrix $X_{n\times m}$ has one and only one element of 1. In line 12, the first state q_1 is assumed as x_{00} in the allocation matrix $X_{n\times m}$ with a value of 1. In line 38, Z is an array of cells, and each cell element $Z[t]$ $(t = \{0, 1, 2, \ldots, p\})$ corresponds to a reachable matrix computed in line 42. Finally, lines 39–46 implement the constraint Sat_3 that is equal to the formula (19).

G. DETERMINATION OF THE MINIMUM NUMBER OF COSETS

This section presents an algorithm for choosing the appropriate value of m, to minimize the number of cosets in Algorithm 7.

In order to reduce the search and calculation time, some special search methods are used to quickly judge and choose the value of m, such as dichotomy.

VII. VERIFICATION EXAMPLES

A. AN ILLUSTRATIVE EXAMPLE

This section elaborates the complete calculation process of the SAT method to solve the supremal quasi-congruence relation with the simple automaton example shown in Fig. 1. The automaton G has the state set $Q = \{q_1, q_2, \ldots, q_7\}$, the event set $\Sigma = \{\alpha, \beta, \lambda, \gamma\}$, the initial state q_1 and the marker state set $\{q_7\}$. Given an observable event set $\Sigma_o = \{\lambda, \gamma\}$, we calculate the supremal quasi-congruence relation for the automaton G through the following steps. First, based on the formulas (7) and (13), all the reachable matrices for the automaton G are obtained through Algorithm 2, Algorithm 3

Algorithm 6 Definition of the Main Function *main*()

```
Require: $Q_m$, $Event_0$, $R$, $X$, $n$, $m$.
Ensure: $s$ is sat or unsat.
 1: def $main(Q_m, Event_0, R, X, n, m)$:
 2: $p = len(Event_0)$;
 3: $X$ = [[$BitVec$("$x[\%s][\%s]$"$\%(i, j)$, 1) for $j$ in $range(m)$]
    for $i$ in $range(n)$];
 4: $s = Solver()$;      /* Initialize the constraint. */
 5: for $i$ in $range(0, n)$ do
 6:     for $j$ in $range(0, m)$ do
 7:         if $i < j$ then
 8:             s.add(X[i][j]==0)
 9:         end if
10:     end for
11: end for
12: $s.add(X[0][0] == 1)$;
13: for $i$ in $range(1, n)$ do
14:     $d = 0$;
15:     $f = 0$;
16:     if $i < m$ then
17:         for $j$ in $range(0, i + 1)$ do
18:             $d| = X[i][j]$;
19:         end for
20:         for $u$ in $range(0, i)$ do
21:             for $v$ in $range(u + 1, i + 1)$ do
22:                 $f| = (X[i][u]\&X[i][v])$;
23:             end for
24:         end for
25:         $s.add(And(d == 1, f == 0))$;
26:     else
27:         for $j$ in $range(0, m)$ do
28:             $d| = X[i][j]$;
29:         end for
30:         for $u$ in $range(0, m - 1)$ do
31:             for $v$ in $range(u + 1, m)$ do
32:                 $f| = (X[i][u]\&X[i][v])$;
33:             end for
34:         end for
35:         $s.add(And(d == 1, f == 0))$;
36:     end if
37: end for
38: $Z$ = [$BitVec$("$z\_\%s$"$\%t$, 1) for $t$ in $range(p + 1)$];
39: for $i$ in $range(0, n - 1, 1)$ do
40:     for $j$ in $range(i + 1, n, 1)$ do
41:         for $t$ in $range(0, p + 1)$ do
42:             $Z[t] = Matrix\_Multi(R[t], X)$;
43:             $s.add(Implies(And([X[i][k] == X[j][k]$ for $k$ in
                $range(m)]), And([And([Z[t][i][k] == Z[t][j][k]$
                for $k$ in $range(m)])$ for $t$ in $range(p + 1)])))$;
44:         end for
45:     end for
46: end for
47: return $s$;
```

Algorithm 7 Definition of the Minimize Function *Minimize*()

```
Require: $m$, $n$, $X$.
Ensure: $m$ is the minimum number of cosets.
 1: def $Minimize(m, n, X)$:
 2: while $m \geq 1$ do
 3:     $s = main(Q_m, Event_0, R, X, n, m)$;
 4:     if $s.check() == sat$ then
 5:         $m = m - 1$;
 6:     else
 7:         $m = m + 1$;
 8:     end if
 9: end while
10: return $m$;
```

and Algorithm 4, including $(R_\lambda)_{7\times7}$, $(R_\gamma)_{7\times7}$ and $(R_\mu)_{7\times7}$ as in Tabs. 2– 4. Second, assume that $m = 7$, according to the formulas (14) and (17), we assume that the corresponding allocation matrix $X_{7\times7}$ of the automaton G is a lower triangular matrix with 28 Boolean variables, as shown in Tab. 5. Third, we translate the supremal quasi-congruence computation into a Boolean satisfiability problem, all Boolean variables in the supposed allocation matrix $X_{7\times7}$ must satisfy the following constraints from Sat_1 to Sat_3. Here, $n = 7$ and $m = 7$.

Sat_1: The constraint Sat_1 ensures that our assumed allocation matrix $X_{n\times m}$ is a Boolean matrix of the lower triangle, which corresponds to the algorithm implementation of lines 5–11 in Algorithm 6.

$$(\forall i \in \mathbf{n})\ (\forall j \in \mathbf{m})\ i < j \rightarrow x_{ij} = 0.$$

Sat_2: The constraint Sat_2 makes sure that each state must belong to one and only one equivalence class, which corresponds to the algorithm implementation of lines 12–37 in Algorithm 6.

$$\wedge \begin{cases} (x_{11} = 1) \\ \wedge_{2\leq i<m}\{(\vee_{j=1}^{i} x_{ij} = 1) \wedge [\vee_{1\leq u<v\leq i}(x_{iu} \wedge x_{iv}) = 0]\} \\ \wedge_{m\leq i\leq n}\{(\vee_{j=1}^{m} x_{ij} = 1) \wedge [\vee_{1\leq u<v\leq m}(x_{iu} \wedge x_{iv}) = 0]\} \end{cases}$$

For the automaton G, we can equivalently convert this constraint Sat_2 into the following set of constraints.

$$\begin{cases} x_{11} = 1 \\ \wedge_{2\leq i\leq 7}(\vee_{j=1}^{i} x_{ij} = 1) \\ \wedge_{2\leq i\leq 7}[\vee_{1\leq u<v\leq i}(x_{iu} \wedge x_{iv}) = 0] \end{cases}$$

Further equivalent conversions are shown below

$$\begin{cases} x_{11} = 1 \\ x_{k1} \vee x_{k2} \vee \cdots \vee x_{kk} = 1,\ 2 \leq k \in \mathbf{m} \\ \vee_{1\leq u<v\leq k}(x_{ku} \wedge x_{kv}) = 0,\ 2 \leq k \in \mathbf{m} \end{cases} \qquad (22)$$

Sat_3: For all pairs of $i \in \mathbf{n-1}$ and $i + 1 \leq j \leq n$, for all $\sigma \in \Sigma_o'$, the constraint Sat_3 guarantees the supremal quasi-congruence relation which must be satisfied if two different

states q_i and q_j belong to the same equivalence class. It corresponds to the algorithm implementation on lines 38–46 in Algorithm 6.

$$[X(i,:) = X(j,:)] \rightarrow \{\wedge_{\sigma\in\Sigma_o'}[R_\sigma(i,:) \bigwedge X_{7\times 7} = R_\sigma(j,:) \bigwedge X_{7\times 7}]\}.$$

According to formulas (15) and (16), the logical vector equality $R_\sigma(i,:) \bigwedge X_{n\times m} = R_\sigma(j,:) \bigwedge X_{n\times m}$ on the right side of the above constraint Sat_3 can be calculated through Boolean matrix multiplication defined in the paper. Here the matrix multiplication is implemented by Algorithm 5, corresponding to the line 42 in Algorithm 6. Note that Z is a cell list of all matrix multiplications obtained through Algorithm 5. The specific calculation process of matrix multiplication is shown as follows.

Assume that the symbol $Z_\sigma(i,t)$ represents a new vector multiplication, which can be calculated according to formula (16) and expressed as follows.

$$Z_\sigma(i,t) = R_\sigma(i,:) \bigwedge X(:,t) \tag{23}$$

$$= \vee_{s=1}^{n}[R_\sigma(i,s) \wedge X(s,t)] \tag{24}$$

Here, $\sigma \in \Sigma_o'$, $i \in \mathbf{n}$ and $t \in \mathbf{m}$. On this basis, we can compute the corresponding product matrix $(Z_\sigma)_{n\times m}$. For example,

$$\begin{aligned} Z_\lambda(2,4) &= \vee_{s=1}^{7}[R_\lambda(2,s) \wedge X(s,4)] \\ &= x_{44} \vee x_{54} \vee x_{64}. \end{aligned}$$

Thus, if $k \in \mathbf{m}$, then

$$Z_\lambda(2,:) = R_\lambda(2,:) \bigwedge X(:,k)$$

$$= \begin{bmatrix} x_{11} \vee x_{21} \vee x_{31} \vee x_{41} \vee x_{51} \vee x_{61} \\ x_{22} \vee x_{32} \vee x_{42} \vee x_{52} \vee x_{62} \\ x_{33} \vee x_{43} \vee x_{53} \vee x_{63} \\ x_{44} \vee x_{54} \vee x_{64} \\ x_{55} \vee x_{65} \\ x_{66} \\ 0 \end{bmatrix}^{T}_{7\times 1}.$$

By analyzing all the reachable matrices, we find that if two states belong to the same equivalence class, then the corresponding two rows of elements in all the reachable matrices are also one-to-one corresponding to each other. This is consistent with the property of supremal quasi-congruence relation. In general, any two states in the same equivalence class can definitely reach the same set of states after the same observable event sequence. Instead, if the condition $R_\sigma(i,:) = R_\sigma(j,:)$ is true for all reachable matrices ($\sigma \in \Sigma_o'$), then the two states q_i and q_j must belong to an equivalence class. From Tabs. 2– 4, we can see that $[R_\lambda(4,:) = R_\lambda(6,:)] \wedge [R_\gamma(4,:) = R_\gamma(6,:)] \wedge [R_\mu(4,:) = R_\mu(6,:)]$, so we can determine that q_4 and q_6 must belong to an equivalence class. Similarly, from all reachable matrices, we can also see that states q_1, q_2, q_3 and q_5 belong to an equivalence class in the automaton G, based on the observable event set $\Sigma_o = \{\lambda, \gamma\}$.

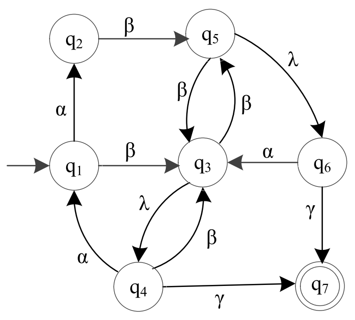


FIGURE 1. Automaton G, and an observable event set $\Sigma_o = \{\lambda, \gamma\}$.

Therefore, this example is formalized as the following satisfiability problem. The objective function for the supremal quasi-congruence computation problem is:

$$|QC| = \min \sum_{j=1}^{m}(\vee_{i=1}^{n} x_{ij}), \ n = m = 7 \tag{25}$$

subject to the following constraints.

$$(\forall i \in \mathbf{n})\,(\forall j \in \mathbf{m})\ i < j \rightarrow x_{ij} = 0 \tag{26}$$

$$x_{11} = 1 \tag{27}$$

$$x_{k1} \vee x_{k2} \vee \cdots \vee x_{kk} = 1, \ 2 \le k \in \mathbf{m} \tag{28}$$

$$\vee_{1\le u<v\le k}(x_{ku} \wedge x_{kv}) = 0, \ 2 \le k \in \mathbf{m} \tag{29}$$

$$[X(i,:) = X(j,:)] \rightarrow \{\wedge_{\sigma\in\Sigma_o'}[Z_\sigma(i,:) = Z_\sigma(j,:)]\} \tag{30}$$

$$x_{ij} \in \{0,1\}, \ i \ge j \in \mathbf{n} \tag{31}$$

In total, this simple example has 28 variables, 1 linear equality and 34 logical propositional formulas. The problem is solved by Z3Py with a PC with Intel(R) i7-4600U CPU @2.10GHz 2.70 GHz and 16.0GB installed memory (RAM). The computation time is 0.0083 second. The final result of the supremal quasi-congruence relation for G is shown in Tab. 6, and its state partition is $\pi = \{\{q_1, q_2, q_3, q_5\}, \{q_4, q_6\}, \{q_7\}\}$, as shown in Fig. 2. All the states in a dotted box represent an equivalence class. Fig. 3 further illustrates the simplified quotient automaton of G. The abstracted model is a nondeterministic finite automaton, because there is an unobservable transition from state 1 to state 0. This suggests that the selected observable event set is not proper for simplifying the original DFA as a smaller DFA. In this case, we may need to consider re-selecting an observable event set [15]. Moreover, the result has been confirmed by the standard method in TCT [1], [37].

B. COMPARATIVE ANALYSIS OF A SIMPLE EXAMPLE

In this section, a simple example is studied to illustrate the efficiency of the proposed method in the paper. In Fig. 4, the automaton M has the state set $Q = \{q_0, q_1, \ldots, q_{11}\}$, the event set $\Sigma = \{10, 11, 12, 13, 20, 21, 22, 23, 32\}$, the state space size $n = 12$, the initial state q_0 and the marker state set $\{q_0\}$.

TABLE 2. Reachable matrix $(R_\lambda)_{7\times7}$ of G.

$(R_\lambda)_{7\times7}$	q_1	q_2	q_3	q_4	q_5	q_6	q_7
q_1	1	1	1	1	1	1	0
q_2	1	1	1	1	1	1	0
q_3	1	1	1	1	1	1	0
q_4	1	1	1	1	1	1	0
q_5	1	1	1	1	1	1	0
q_6	1	1	1	1	1	1	0
q_7	0	0	0	0	0	0	0

TABLE 3. Reachable matrix $(R_\gamma)_{7\times7}$ of G.

$(R_\gamma)_{7\times7}$	q_1	q_2	q_3	q_4	q_5	q_6	q_7
q_1	0	0	0	0	0	0	0
q_2	0	0	0	0	0	0	0
q_3	0	0	0	0	0	0	0
q_4	0	0	0	0	0	0	1
q_5	0	0	0	0	0	0	0
q_6	0	0	0	0	0	0	1
q_7	0	0	0	0	0	0	0

TABLE 4. Reachable matrix $(R_\mu)_{7\times7}$ of G.

$(R_\mu)_{7\times7}$	q_1	q_2	q_3	q_4	q_5	q_6	q_7
q_1	0	0	0	0	0	0	0
q_2	0	0	0	0	0	0	0
q_3	0	0	0	0	0	0	0
q_4	0	0	0	0	0	0	0
q_5	0	0	0	0	0	0	0
q_6	0	0	0	0	0	0	0
q_7	0	0	0	0	0	0	1

TABLE 5. Supposed matrix $X_{7\times7}$ of G.

Q	C_1	C_2	C_3	C_4	C_5	C_6	C_7
q_1	x_{11}	0	0	0	0	0	0
q_2	x_{21}	x_{22}	0	0	0	0	0
q_3	x_{31}	x_{32}	x_{33}	0	0	0	0
q_4	x_{41}	x_{42}	x_{43}	x_{44}	0	0	0
q_5	x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	0	0
q_6	x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	0
q_7	x_{71}	x_{72}	x_{73}	x_{74}	x_{75}	x_{76}	x_{77}

TABLE 6. Allocation matrix $X_{7\times7}$ of G.

$X_{7\times7}$	C_1	C_2	C_3	C_4	C_5	C_6	C_7
q_1	1	0	0	0	0	0	0
q_2	1	0	0	0	0	0	0
q_3	1	0	0	0	0	0	0
q_4	0	1	0	0	0	0	0
q_5	1	0	0	0	0	0	0
q_6	0	1	0	0	0	0	0
q_7	0	0	1	0	0	0	0

Given several different sets of observable events and assuming that the initial number of equivalence classes is $m = n$ for facilitating the comparison of calculation results, the supremal quasi-congruence is computed by three different solvers in Tab. 7. Among them, the first two methods are mixed integer linear programming (MILP) [49] in MATLAB and binary linear programming (BLP) in CPLEX which are proposed in our previous work [16]. The other one is SAT method in Z3Py proposed in this paper.

TABLE 7. Comparison of three computational methods based on different observable event sets Σ_o of automaton M.

			Time/s		
Σ_o	TCT	Size of C3	MILP	BLP	SAT
$\{10, 12, 32\}$	3	65×90	0.0253	0.0138	0.0090
$\{12, 22, 32\}$	3	157×90	0.0503	0.0223	0.0090
$\{13, 23, 32\}$	6	241×90	0.1007	0.0421	0.0170
$\{10, 20, 22\}$	6	787×90	0.1269	0.0433	0.0150
$\{11, 13, 21, 23\}$	10	791×90	0.0636	0.0951	0.0190
$\{13, 20, 23, 32\}$	10	910×90	0.2214	0.0810	0.0170
$\{10, 13, 20, 23\}$	9	972×90	0.1070	0.0393	0.0200
$\{12, 21\}$	3	1848×90	0.0708	0.0404	0.0080
$\{11, 13, 21\}$	7	1999×90	0.0620	0.0318	0.0140

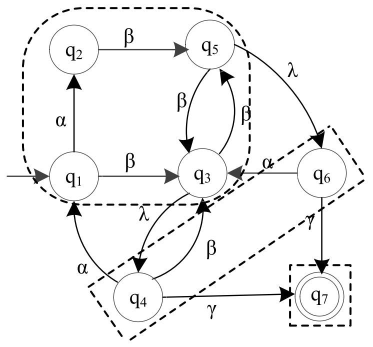


FIGURE 2. G model abstraction based on $\Sigma_o = \{\lambda, \gamma\}$.

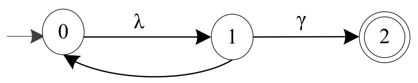


FIGURE 3. The quotient automaton of G with respect to $\Sigma_o = \{\lambda, \gamma\}$.

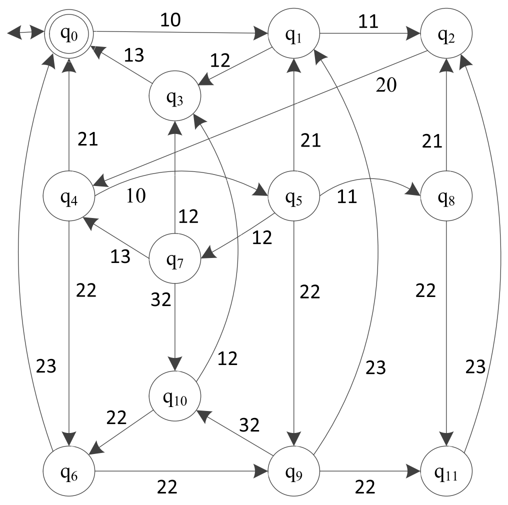


FIGURE 4. Automaton M.

Tab. 7 is a comparison of three computational methods based on different observable event sets of automaton M.

TABLE 8. Comparison of calculation results.

Example	$\lvert QC\rvert$	m	Size of C3	Time/s		
				MILP	BLP	SAT
SMALLFACT [15]		12	157×90	0.0503	0.0200	0.0100
$n = 12, \lvert\Sigma\rvert = 9$	3	6	129×63	0.0302	0.0200	0.0090
$\Sigma_o = \{12, 22, 32\}$		3	78×36	0.0204	0.0000	0.0090
M3		21	1261×252	73.0757	0.0600	0.0480
$n = 21, \lvert\Sigma\rvert = 22$	8	10	1092×175	8.5508	0.0500	0.0460
$\Sigma_o = \{13, 31, 53, 71, 81\}$		8	975×148	1.6661	0.0300	0.0440
M3		21	3312×252	55.7098	1.6600	0.0490
$n = 21, \lvert\Sigma\rvert = 22$	8	10	1620×175	10.6396	0.4200	0.0470
$\Sigma_o = \{15, 33, 55, 75\}$		8	1310×148	2.4277	0.2500	0.0460
M3		21	995×252	0.1757	0.0200	0.0150
$n = 21, \lvert\Sigma\rvert = 22$	3	10	905×175	0.1579	0.0100	0.0140
$\Sigma_o = \{51\}$		3	387×63	0.0249	0.0100	0.0140
M4		54	8121×1539	1.7758	0.1900	0.0460
$n = 54, \lvert\Sigma\rvert = 53$	3	27	6750×1134	1.4337	0.1700	0.0430
$\Sigma_o = \{15, 25, 35, 45\}$		3	1002×162	0.0248	0.0200	0.0400
M4		54	10446×1539	35.7073	0.3800	0.0830
$n = 54, \lvert\Sigma\rvert = 53$	4	10	3885×505	0.9188	0.1100	0.0810
$\Sigma_o = \{28, 38, 48\}$		4	1662×214	0.1867	0.0500	0.0800
M4		54	19519×1539	7209.08	1.0500	0.8190
$n = 54, \lvert\Sigma\rvert = 53$	10	27	16969×1134	7200.51	0.9000	0.8100
$\Sigma_o = \{10, 16, 41, 50, 52\}$		10	9350×505	7199.90	0.6600	0.8037
FMS [47]						-
$n = 180, \lvert\Sigma\rvert = 13$						-
$\Sigma_o = \{211, 231, 241\}$	3	3	276732×540	328.5743	102.0500	1.9241
$\Sigma_o = \{230, 251, 371\}$	6	6	877048×1071	10138.733	901.2900	3.4174
PRODCELL [10]						-
$n = 2478, \lvert\Sigma\rvert = 23$	2	2	1503438×4957	18.3476	0.9400	0.0372
$\Sigma_o = \{610\}$						-

Column 1 lists different observable event sets Σ_o based on the automaton M. Column 2 lists the result of computing the supremal quasi-congruence relation using the classical graph-based partition method TCT [1], [37] for each observable event set. Column 3 represents the size of the complex linear inequality constraints C3 (Number of linear inequalities $\times$ Number of variables) in the methods MILP [16] and BLP [16], since the complexity of the constraint C3 is the main factor affecting the computational efficiency of methods MILP and BLP in our previous work [16]. Columns 4–6 show the computation time using three different methods, the MILP method in MATLAB and the BLP method implemented by calling CPLEX Studio 12.8 in MATLAB. Note that the problem is solved by MATLAB with a PC with Intel(R) i7-4600U CPU @2.10GHz 2.70 GHz and 16.0GB installed memory (RAM). From the above Tab. 7, we can draw the following conclusions.

- The calculation results of these three methods are consistent with those of the classical TCT method [1], [37] in supervisory control theory, which indicates their correctness.
- For the same observable event set, it is obvious that the three methods are MILP, BLP, SAT in order from slow to fast.
- On different observable event sets, when the final number of optimal equivalence classes is the same, more constraints imply longer time consumed by the first two methods MILP and BLP, while the time difference consumed by the SAT method proposed in this paper is very small.
- When the number of minimum equivalence classes is larger, the computation time of the above three methods becomes relatively longer. However, the SAT method is the most computationally efficient overall.

C. COMPARATIVE ANALYSIS OF DIFFERENT EXAMPLES

This section mainly compares the computational efficiency of three different methods in calculating the supremal quasi-congruence relation, including MILP, BLP and SAT. Note that the $\lvert QC\rvert$ column values in Tab. 8 are consistent with those obtained by the classical TCT [1], [37] method in the supervisory control theory. This verifies the correctness of the proposed method.

Tab. 8 mainly compares the calculation of some large cases, using the three methods mentioned above. Column 1 mainly lists the state space of different automata, where n represents the size of state space, $\lvert\Sigma\rvert$ is the size of event set and Σ_o is the set of observable events. Column 2 is the real minimum number of equivalence classes $\lvert QC\rvert$ obtained by these methods. Columns 3–4 respectively list the number of equivalence classes m ($m \leq n$) we assumed and the size of linear inequalities for constraint C3 (Number of linear inequalities $\times$ Number of variables) in the methods MILP and BLP, because different values of m and the size of constraint C3 affect the later computational efficiency in our previous work [16]. Columns 5–7 compare the running times of three different calculation methods: the MILP method [16], the BLP method [16], and the SAT method proposed in the paper. Each row represents the time consumed by the above three different methods to compute the supremal quasi-congruence relation for the same automaton and the same set of observable events when different values of m are assumed. In addition, the automata M3 and M4 used in Tab. 8 are designed by us for comparison and test. M3 has 21 states, 22 events, and 26 transitions. M4 has 54 states, 53 events, and 100 transitions. Note that when the running time of BLP method is less than 0.01s, the system automatically reduces it to 0.00s. Through comparative analysis of the data in Tab. 8, the following conclusions are obtained.

- For different automata, when the size of state space n is small and $m > |QC|$, the SAT method takes significantly less time than the other two methods MILP and BLP. However, for small values of n and $m = |QC|$, the BLP method runs a little faster than the SAT method.
- For the same automaton and the same observable event set, the calculation results of the three methods are consistent, which proves their correctness.
- For the same automaton, no matter what value m takes, the running time difference of the SAT method is very small as long as $m \geq |QC|$, almost no more than 0.01s. It indicates that the computation time of the SAT method is relatively stable compared with the BLP method and the MILP method.
- When n is relatively large, whether it is $m > |QC|$ or $m = |QC|$, the SAT method proposed in this paper takes the shortest time. In general, the SAT method is more efficient for complex problems.

VIII. CONCLUSION

In this paper, the supremal quasi-congruence relation is assumed as a state-to-coset Boolean allocation matrix $X_{n\times m}$, where n represents the size of state space, m denotes the number of assumed equivalence classes, and $m \leq n$. The computation of the supremal quasi-congruence relation is translated as a satisfiability problem and solved by an efficient SAT solver Z3Py. The task of computing the supremal quasi-congruence relation is equivalent to determining whether there exists an assignment for Boolean variables such that its three constraint CNF formulas ($Sat_1 - Sat_3$) evaluate to true. If the result is satisfied, then the supremal quasi-congruence relation exists and the minimum equivalence classes is obtained. Otherwise, it indicates that there is no such supremal quasi-congruence relation, and a new set of observable events needs to be modified or reselected for the original system model. The efficiency and correctness of the SAT method proposed in this paper are verified by comparing with the two methods of MILP and BLP proposed in our previous work and the classical TCT method in supervisory control theory.

In conclusion, the SAT method proposed in this paper has obvious advantages for relatively large size of state space. It not only has significant improvement in computational efficiency, but also has relatively stable computational time when the value of m ($m \leq n$) is different. Therefore, the research of the SAT method not only lays a foundation for model abstraction of complex systems, but also establishes a bridge for the subsequent study of supervisory control theory and satisfiability theory.

REFERENCES

[1] W. M. Wonham and K. Cai, *Supervisory Control of Discrete Event Systems*. Berlin, Germany: Springer, 2018.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Berlin, Germany: Springer, 2008.

[3] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[4] W. M. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history," *Annu. Rev. Control*, vol. 45, pp. 250–256, Jan. 2018.

[5] H. Gharsellaoui and M. Khalgui, "Dynamic reconfiguration of intelligence for high behaviour adaptability of autonomous distributed discrete-event systems," *IEEE Access*, vol. 7, pp. 35487–35498, 2019.

[6] M. Khalgui, O. Mosbahi, and Z. W. Li, "On reconfiguration theory of discrete-event systems: From initial specification until final deployment," *IEEE Access*, vol. 7, pp. 18219–18233, 2019.

[7] J. Billington, G. E. Gallasch, L. M. Kristensen, and T. Mailund, "Exploiting equivalence reduction and the sweep-line method for detecting terminal states," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 34, no. 1, pp. 23–37, Jan. 2004.

[8] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dyn. Syst.*, vol. 17, no. 4, pp. 475–504, Nov. 2007.

[9] R. Su, J. H. Schuppen, and J. E. Rooda, "Model abstraction of nondeterministic finite-state automata in supervisor synthesis," *IEEE Trans. Autom. Control*, vol. 55, no. 1, pp. 2527–2541, Sep. 2010.

[10] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Trans. Autom. Control*, vol. 51, no. 5, pp. 782–793, May 2006.

[11] D. Wang, X. Wang, and Z. Li, "Nonblocking supervisory control of state-tree structures with conditional-preemption matrices," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 3744–3756, Jun. 2020.

[12] D. Peled, "Partial order reduction: Model-checking using representatives," in *Proc. IEEE Symp. Math. Found. Comput. Sci.*, Sep. 1996, pp. 93–112.

[13] K. Peeva, "Equivalence, reduction and minimization of finite automata over semirings," *Theor. Comput. Sci.*, vol. 88, no. 2, pp. 269–285, Oct. 1991.

[14] K. C. Wong and W. M. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 14, no. 1, pp. 55–107, Jan. 2004.

[15] L. Feng and W. M. Wonham, "On the computation of natural observers in discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 20, no. 1, pp. 63–102, Mar. 2010.

[16] L. H. Cheng, L. Feng, and Z. W. Li, "Model abstraction for discrete-event systems by binary linear programming with applications to manufacturing systems," *Sci. Prog.*, vol. 104, no. 3, pp. 1–32, Jul. 2021.

[17] R. Y. Zhang, Y. M. Gan, W. J. Chao, and Z. A. Wang, "Improved algorithm of quasi-congruence in discrete-event system," *IET Control Theory A.*, vol. 29, pp. 151–156, Feb. 2012.

[18] K. C. Wong and W. M. Wonham, "Hierarchical control of discrete event systems," *Discrete Event Dyn. Syst.*, vol. 6, no. 3, pp. 241–273, Jul. 1996.

[19] K. C. Wong, "On the complexity of projections of discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 1, pp. 201–208, Sep. 1998.

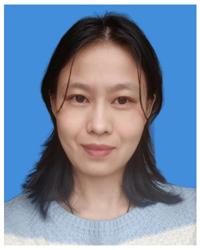
[20] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Sci. Comput. Program.*, vol. 13, nos. 2–3, pp. 219–236, May 1990.

[21] J. Marques-Silva, "Practical applications of Boolean satisfiability," in *Proc. IEEE Conf. 9th (WODES)*, Goteborg, Sweden, Aug. 2008, pp. 429–436.

[22] R. Kindermann, T. Junttila, and I. Niemelä, "Bounded model checking of an MITL fragment for timed automata," 2013, *arXiv:1304.7209*.

[23] L. Cordeiro and B. Fischer, "Bounded model checking of multi-threaded software using SMT solvers," *Comput. Sci. Rev.*, vol. 12, no. 2, pp. 37–49, 2010.

[24] K. Hameed, S. Garg, M. B. Amin, and B. Kang, "Towards a formal modelling, analysis and verification of a clone node attack detection scheme in the Internet of Things," *Comput. Netw.*, vol. 204, Feb. 2022, Art. no. 108702.
[25] V. Dahllöf, P. Jonsson, and R. Beigel, "Algorithms for four variants of the exact satisfiability problem," *Theor. Comput. Sci.*, vol. 320, nos. 2–3, pp. 373–394, Jun. 2004.
[26] L. C. Wu and C. Y. Tang, "Solving the satisfiability problem by using randomized approach," *Inf. Process. Lett.*, vol. 41, no. 4, pp. 187–190, Mar. 1992.
[27] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 10, pp. 1606–1621, Oct. 2005.
[28] O. Bailleux, Y. Boufkhad, and O. Roussel, "A translation of pseudo Boolean constraints to SAT," *JSAT*, vol. 2, pp. 183–192, Feb. 2006.
[29] A. Biere, M. Heule, H. V. Maaren, and T. Walsh, *Handbook of Satisfiability*. Amsterdam, The Netherlands: IOS Press, 2009.
[30] H. Katebi, K. A. Sakallah, and I. L. Markov, *Symmetry and Satisfiability: An Update*. Berlin, Germany: Springer-Verlag, 2010.
[31] W. Kong, N. Katahira, W. Qian, M. Watanabe, T. Katayama, and A. Fukuda, "An SMT-based approach to bounded model checking of designs in communicating state transition matrix," in *Proc. Int. Conf. Comput. Sci. Appl.*, Jun. 2011, pp. 946–957.
[32] A. Carioni, S. Ghilardi, and S. Ranise, "Automated termination in model-checking modulo theories," *Int. J. Found. Comput. Sci.*, vol. 24, pp. 211–232, Feb. 2013.
[33] J. Tan and Y. Li, "A topology graph algorithm based on lattice-valued logic to solve satisfiability problems," *Math. Problems Eng.*, vol. 2022, pp. 1–9, Jan. 2022.
[34] P. A. Abdulla, P. Bjesse, and N. Een, "Symbolic reachability analysis based on SAT-solvers," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Germany: Springer, Aug. 2000, pp. 411–425.
[35] A. Tlili, K. Belahcène, O. Khaled, V. Mousseau, and W. Ouerdane, "Learning non-compensatory sorting models using efficient SAT/MaxSAT formulations," *Eur. J. Oper. Res.*, vol. 298, no. 3, pp. 979–1006, May 2022.
[36] H. Fu, J. Liu, G. Wu, Y. Xu, and G. Sutcliffe, "Improving probability selection based weights for satisfiability problems," *Knowl-Based Syst.*, vol. 245, pp. 108–119, Jun. 2022.
[37] L. Feng and W. M. Wonham, "TCT: A computation tool for supervisory control synthesis," in *Proc. 8th Int. Workshop Discrete Event Syst.*, 2006, pp. 388–389.
[38] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.
[39] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res. Develop.*, vol. 3, no. 2, pp. 114–125, 1959.
[40] J. E. Hopcroft and R. Motwani, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Boston, MA, USA: Addison Wesley, 2012.
[41] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods Syst. Des.*, vol. 19, no. 1, pp. 7–34, 2001.
[42] K. Claessen, N. Een, M. Sheeran, and N. Sorensson, "SAT-solving in practice," in *Proc. 9th Int. Workshop Discrete Event Syst.*, 2008, pp. 442–449.
[43] S. V. Popov, "On the complexity of derivations in classical propositional calculus," *Sov. Math., Dokl.*, vol. 17, pp. 876–880, Jan. 1976.
[44] H. R. Andersen and H. Hulgaard, "Boolean expression diagrams," in *Proc. 12th Annu. IEEE Symp. Logic Comput. Sci.*, Warsaw, Poland, 1997, pp. 88–98, doi: 10.1109/LICS.1997.614938.
[45] G. Audemard, A. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani, "A sat based approach for solving formulas over Boolean and linear mathematical propositions," in *Automated Deduction*. Berlin, Germany: Springer, 2002, pp. 195–210.
[46] S. Maclane and G. Birkhoff, *Algebra*. Providence, RI, USA: AMS Chelsea Publishing, 1988.
[47] L. Feng, "Computationally efficient supervisor design in discrete-event systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, Jul. 2008.
[48] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Trans. Autom. Control*, vol. 53, no. 6, pp. 1449–1461, Jul. 2008.
[49] P. Kuendee and U. Janjarassuk, "A comparative study of mixed-integer linear programming and genetic algorithms for solving binary problems," in *Proc. 5th Int. Conf. Ind. Eng. Appl. (ICIEA)*, Apr. 2018, pp. 284–288.



LIHONG CHENG was born in Yuncheng, Shanxi, Xinjiang, China, in 1984. She received the B.S. degree from Xinzhou Teachers University, Shanxi, in 2007, and the M.S. degree from Shaanxi Normal University, Xi'an, China, in 2010. She is currently pursuing the Ph.D. degree in control theory and control engineering with Xidian University, Xi'an.

From 2015 to 2019, she was a joint training Ph.D. candidate with the School of Electro-Mechanical Engineering, Xidian University, and the Department of Machine Design, ITM School, KTH Royal Institute of Technology, Stockholm, Sweden. Her main research interests include model abstraction, formal verification, and control synthesis of discrete-event systems.

Ms. Cheng received the Doctoral joint training program of China Scholarship Council (CSC), in 2016.

LEI FENG (Member, IEEE) was born in Xi'an, Shaanxi, China, in 1976. He received the B.S. and M.S. degrees from the Department of Mechanical and Electronic Engineering, Xi'an Jiaotong University, Xi'an, China, in 1998 and 2001, respectively, and the Ph.D. degree from the Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, in 2007.

In 2012, he joined the Mechatronics and Embedded Control System Division, KTH Royal Institute of Technology, Stockholm, Sweden, where he is currently an Associate Professor. He is the author of one U.S. patent, more than 30 refereed journal articles in prestigious scientific journals, including IEEE Transactions and Elsevier. He is currently a visiting professor position of the National 111 Project base on electromechanical coupling theory and key technology for electronic equipment with Xidian University, Xi'an. His main research interests include energy management control of mechatronic systems, autonomous driving, verification and control synthesis of cyber-physical systems, and supervisory control of discrete-event systems.