IEEE Control Systems Letters paper presented at
2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Transformational Supervisor Localization

Sander Thuijsman[a], Kai Cai[b], and Michel Reniers[a]

***Abstract*— Supervisor localization can be applied to distribute a monolithic supervisor into local supervisors. Performing supervisor localization can be computationally costly. In this work, we consider systems that evolve over time. We study how to reuse the results from a previous supervisor localization, to more efficiently compute local supervisors when the system is adapted. We call this approach transformational supervisor localization, and present algorithms for the procedure. The efficiency of the procedure is experimentally evaluated.**

I. Introduction

Supervisory control theory, as introduced by Ramadge and Wonham [1], is a model-based approach to control discrete event (dynamic) systems. Typically, cyber-physical systems are modeled. By applying *supervisor synthesis* on a model of an uncontrolled system (plant) and system requirements, a correct-by-construction supervisor is obtained. This supervisor enables/disables events such that the requirements are always adhered to, and some more behavioral properties apply to the controlled system such as: nonblockingness, controllability, and maximal permissiveness [2]. The most straightforward approach is *monolithic* supervisor synthesis, which computes a single global supervisor that controls all components and enforces all requirements.

Large, global controllers may be undesirable in practice. As such, many modern control systems are distributed over a number of agents [3]. These agents may act locally based on their own observations and control strategies. Through *supervisor localization* (SL) [4], local supervisors for the individual agents are computed from the monolithic supervisor, that together achieve the same controlled behavior as the monolithic supervisor. SL is an extension to *supervisor reduction*, which converts a supervisor automaton to a smaller automaton (with less states) that is control equivalent to the original automaton [5]. We present preliminaries on supervisor localization/reduction in Section II.

In this work, we slightly modify the SL algorithm from [5], [4] to be able to initialize it in a way such that it has to do less calculations/loops, which benefits the method that we are going to introduce. Furthermore, because it is desirable to obtain small (in terms of number of states) local supervisors, we show that the local supervisors obtained by SL are maximally reduced. These novel extensions to SL are presented in Section III.

In recent work, transformational approaches for supervisory control algorithms, such as transformational supervisor synthesis [6] and transformational nonblocking verification [7], are investigated. Such transformational approaches deal with cyber-physical systems that evolve over time. Results of previous computations, such as synthesis or verification, may not be valid anymore once the system is adapted. In this case, tranformational methods can be applied that reuse the output of previous calculations to more efficiently compute the result of some algorithm, rather than computing it from scratch. The general idea is that the previous result is *transformed* into the new result, using knowledge on how the system is adapted.

In this paper, we investigate *transformational supervisor localization* (TSL). We assume a *base model*, on which (T)SL has already been performed. The base model is adapted such that a *variant model* is obtained. The goal is to use the localization output of the base model, to more efficiently compute local supervisors for the variant model. The formal problem definition is given in Section IV. We present algorithms for TSL and prove their correctness in Section V. The computational benefit of TSL is evaluated by a use case in Section VI.

II. Preliminaries

In the following we discuss the preliminaries on SL. We first provide automata definitions for plant and (monolithic) supervisor. The plant is assumed to be a composition of agents. The goal is to generate local supervisors that each supervise an agent. This is done by grouping states of the monolithic supervisor into cells, that are consistent in the enablement and disablement of events controlled by the respective agent. These cells are the states of the local supervisor. Together, the behavior of the system under control by the local supervisors is the same as that of the monolithic supervisor. For details we refer to [4].

The plant is defined by a finite state automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite state set, Σ is the finite event set, q_0 is the initial state, and Q_m is the subset of marked states. $\delta : Q \times \Sigma \to Q$ is the (partial) transition function. We denote $\delta(q, \sigma)!$ if $\delta(q, \sigma)$ is defined. We extend this notation to $\delta : Q \times \Sigma^* \to Q$, and write $\delta(q, s)$ for $s \in \Sigma^*$ to indicate sequences of transitions. We consider the case that G is formed by a composition of n agents, that each have local events: $\dot{\bigcup}_{k\in\{1,\dots,n\}} \Sigma_k = \Sigma$, from which a subset are locally controllable $\Sigma_{c,k} \subseteq \Sigma_k$. We assume a monolithic supervisor is provided for plant G, defined by finite state automaton $S = (X, \Sigma, \xi, x_0, X_m)$. For the purpose of the algorithms in this work, we assume the states are numbered/indexed, i.e., $X = \{x_0, x_1, ...\}$.

We use the following functions [4]:

[a]Eindhoven University of Technology, Eindhoven, Netherlands; {s.b.thuijsman, m.a.reniers}@tue.nl
[b]Osaka Metropolitan University, Osaka, Japan; cai@omu.ac.jp
This work was supported in part by JSPS KAKENHI Grant nos. 21H04875, 22KK0155

- $E : X \to 2^{\Sigma}$, where $E(x) = \{\sigma \in \Sigma | \xi(x, \sigma)!\}$
- $D_k : X \to 2^{\Sigma_{c,k}}$, and $D_k(x) = \{\sigma \in \Sigma_{c,k} | \neg\xi(x, \sigma)! \wedge (\exists s \in \Sigma^*) : (\xi(x_0, s) = x \wedge \delta(q_0, s\sigma)!)\}$
- $M : X \to \{0, 1\}$, where $M(x) = 1$ iff $x \in X_m$
- $T : X \to \{0, 1\}$, where $T(x) = 1$ iff $(\exists s \in \Sigma^*) : (\xi(x_0, s) = x \wedge \delta(q_0, s) \in Q_m)$

E indicates events enabled by the supervisor in state x. D_k indicates the events from $\Sigma_{c,k}$ disabled by the supervisor in state x. M determines if a state is marked in S, and T determines if some corresponding state is marked in G.

We define control consistency relation $\mathcal{R}_k \subseteq X \times X$ (for agent k): for every $x, x' \in X$, $(x, x') \in \mathcal{R}_k$ iff:

$$E(x) \cap D_k(x') = \emptyset = E(x') \cap D_k(x) \quad (1)$$

$$T(x) = T(x') \implies M(x) = M(x') \quad (2)$$

Cover $\mathcal{C}_k = \{X_i \subseteq X | i \in I_k\}$ with suitable index set I_k is called a *control cover* with respect to some Σ_k iff:

(i) $(\forall i \in I_k, \forall x, x' \in X_i)(x, x') \in \mathcal{R}_k$

(ii) $(\forall i \in I_k, \forall \sigma \in \Sigma)\big[\big((\exists x \in X_i)\xi(x, \sigma)!\big) \implies \big((\exists j \in I_k)(\forall x' \in X_i)\xi(x', \sigma)! \implies \xi(x', \sigma) \in X_j\big)\big]$

If a control cover $\mathcal{C}$ is a partition on X, it is called a *control congruence*.

In this work we frequently address a *singleton cover* $\mathcal{C} = \{\{x\} | x \in X\}$, which trivially always is a control congruence.

We call a set of states in a cover a *cell*. In our notation we use $[x]_{\mathcal{C}}$ to refer to the set of states contained in the same cell as x in cover $\mathcal{C}$, or simply $[x]$ if there is no ambiguity.

Given a control congruence $\mathcal{C}_k$, a local supervisor LOC_k is computed as follows (simplified from [5]): $LOC_k = (\mathcal{C}_k, \Sigma, \eta_k, y_{0,k}, Y_{m,k})$, where: $\eta_k : \mathcal{C}_k \times \Sigma \to \mathcal{C}_k$, with $\eta_k(\pi_1, \sigma) = \pi_2$ iff $(\exists x \in \pi_1) : \xi(x, \sigma) \in \pi_2$; $y_{0,k} = [x_0]$; and $Y_{m,k} = \{[x] | x \in X_m\}$. A local supervisor is deterministic as a result of condition (ii) for the control cover.

The set of local supervisors $\{LOC_k | 1 \leq k \leq n\}$ constructed in this way is *control equivalent* to S with respect to G [4]:

$$L(G) \cap \bigcap\nolimits_{1 \leq k \leq n} L(LOC_k) = L(S) \cap L(G) \quad (3)$$

$$L_m(G) \cap \bigcap\nolimits_{1 \leq k \leq n} L_m(LOC_k) = L_m(S) \cap L_m(G) \quad (4)$$

$L(A)$ and $L_m(A)$ respectively denote the language and the marked language of automaton A [2].

III. Supervisor localization

In the process of SL, for each agent, a control congruence is computed and subsequently the local supervisor is generated. We can use the definitions and functions from Section II to perform the localization algorithm, shown in Algorithm 1, which makes calls to Algorithm 2 [4][1]. Note that, e.g., the X on line 1 implicitly originates from automaton S. A 'continue' ends current execution and the function goes to the next iteration of the nearest enclosing for-loop. A 'return' ends current call to the algorithm and the specified values are returned to the parent routine.

[1]Relative to [5], [4] some minor changes have been made to lines 1,2, and 7 of Algorithm 2 for correctness.

Algorithm 1 `localize`

Input: G, S, $\Sigma_{c,k}$, initial $\mathcal{C}_k$
Output: Control congruence $\mathcal{C}_k$

1: **for** $i = 0$ to $|X| - 2$ **do**
2: **if** $i > min(\{m | x_m \in [x_i]\})$ **then continue; end**
3: **for** $j = i + 1$ to $|X| - 1$ **do**
4: **if** $j > min(\{m | x_m \in [x_j]\})$ **then continue; end**
5: $W = \emptyset$
6: $(flag, W) =$ `check_merge`$(x_i, x_j, W, i, \xi, \mathcal{C}_k)$
7: **if** *flag* **then**
8: $\mathcal{C}_k = \Big\{[x] \cup \bigcup\{[x'] | \{(x, x'), (x', x)\} \cap W \neq \emptyset\} \Big| [x], [x'] \in \mathcal{C}_k\Big\}$
9: **end**
10: **end**
11: **end**
12: **return** $\mathcal{C}_k$

Algorithm 2 `check_merge`

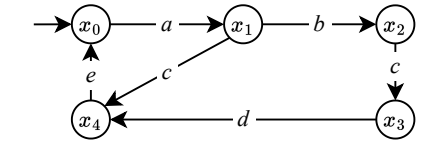
Input: x_i, x_j, waiting list W, i, ξ, $\mathcal{C}_k$
Output: mergeability Boolean *flag*, W

1: **for all** $x_p \in [x_i] \cup \bigcup\{[x] | \{(x, x'_i), (x'_i, x)\} \cap W \neq \emptyset, x'_i \in [x_i]\}$ **do**
2: **for all** $x_q \in [x_j] \cup \bigcup\{[x] | \{(x, x'_j), (x'_j, x)\} \cap W \neq \emptyset, x'_j \in [x_j]\}$ **do**
3: **if** $\{(x_p, x_q), (x_q, x_p)\} \cap W \neq \emptyset$ **then continue; end**
4: **if** $(x_p, x_q) \notin \mathcal{R}_k$ **then return** $(false, W)$**; end**
5: $W = W \cup \{(x_p, x_q)\}$
6: **for all** $\sigma \in E(x_p) \cap E(x_q)$
7: **if** $[\xi(x_p, \sigma)] = [\xi(x_q, \sigma)]$ **or** $\{(\xi(x_p, \sigma), \xi(x_q, \sigma)), (\xi(x_q, \sigma), \xi(x_p, \sigma))\} \cap W \neq \emptyset$ **then continue; end**
8: **if** $min(\{m | x_m \in [\xi(x_p, \sigma)]\}) < i$ **or** $min(\{m | x_m \in [\xi(x_q, \sigma)]\}) < i$ **then return** $(false, W)$**; end**
9: $(flag, W) =$ `check_merge`$(\xi(x_p, \sigma), \xi(x_q, \sigma), W, i, \xi, \mathcal{C}_k)$
10: **if not** *flag* **then return** $(false, W)$**; end**
11: **end**
12: **end**
13: **end**
14: **return** $(true, W)$

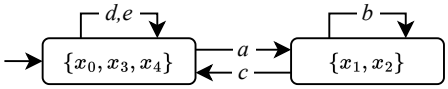
Example 1: We consider the supervisor automaton shown in Fig. 1(a). The states are represented by circles. The dangling incoming arrow indicates x_0 is the initial state. Transitions are shown by arrows between states with the respective event label. To simplify the examples, no states are marked and all events are controllable.

We consider the case that there is an agent (numbered 1) whose set of local controllable events includes all events, i.e., $\Sigma_{c,1}{=}\Sigma_1{=}\Sigma{=}\{a, b, c, d, e\}$. Let us consider the case that the supervisor disables event c in state x_0, and disables event a in state x_2. There are no disablements in the other states, i.e., the supervisor permits the same events as the plant in those states. So, $D_1(x_0){=}\{c\}, D_1(x_2){=}\{a\}$, $D_1(x_1){=}D_1(x_3){=}D_1(x_4){=}\emptyset$.

To compute the local supervisor, we perform the localization algorithm initialized with a singleton cover $\{\{x_0\}, ..., \{x_4\}\}$. First, mergeability of x_0 and x_1 is checked. These states are not mergeable, since event c is disabled in



(a) Supervisor



(b) Local supervisor

Fig. 1: Automata of Example 1

x_0 but enabled in x_1. Also x_0 and x_2 are not mergeable. x_0 is mergeable with x_3 and they are subsequently merged. Next, $\{x_0, x_3\}$ is merged with x_4 to form cell $\{x_0, x_3, x_4\}$. Finally x_1 and x_2 are merged, and no more merges are possible so the algorithm terminates. Using the resulting control congruence, a local supervisor is constructed, which is displayed in Fig. 1(b). ♦

In [4] the localization algorithm is initiated with a singleton cover. However, in this work we will also initialize the algorithm with non-singleton covers, to benefit the efficiency of the transformational method that we are going to introduce. We present Lemma 1 on this initialization.

Lemma 1: If Algorithm 1 is initiated with a control congruence $\mathcal{C}_{k,init}$, the output cover $\mathcal{C}_k$ is a control congruence.

Proof: Correctness of Algorithm 1 initiated by a singleton cover is proven in [5]. The singleton cover is a special instance of a control congruence. The same proof of [5] applies here, when we generalize the algorithm to be initialized with any control congruence. ■

It is desirable to have small (in terms of number of states) local supervisors. Therefore, we want to compute control congruences which cannot be reduced further, i.e., any further merging of cells would result in an invalid control cover. We call such a cover *maximally reduced*, see Definition 1. Reducedness of the control congruences obtained by Algorithm 1 is addressed in Lemma 2.

Definition 1: Cover $\mathcal{C}_k$ is *maximally reduced* w.r.t. $G, S, \Sigma_{c,k}$ iff $\forall \pi_1, \pi_2 \in \mathcal{C}_k$, if $\pi_1 \neq \pi_2$, then $(\mathcal{C}_k \setminus \{\pi_1, \pi_2\}) \cup \{\pi_1 \cup \pi_2\}$ is not a control congruence w.r.t. $G, S, \Sigma_{c,k}$. ♦

Lemma 2: $\mathcal{C}_k$ obtained by Algorithm 1, is maximally reduced w.r.t. $G, S, \Sigma_{c,k}$.

Proof: Algorithm 1 iterates over all pairs of states, and only skips pairs of states when mergeability between some pair of states contained in the respective cells has already been checked. Thus, if Algorithm 1 outputs a control congruence containing individual cells π_1 and π_2, then mergeability has been checked between some pair of states $x_1 \in \pi_1, x_2 \in \pi_2$. Let us say x_1, x_2 respectively were in cells ρ_1, ρ_2 at the point their mergeability was checked. Since x_1 and x_2 were not merged, `check_merge` has returned false for this evaluation, which means that some pair of states x_3, x_4 respectively in ρ_1, ρ_2 were not mergeable. Since Algorithm 1 only merges cells (i.e., never splits a cell), we know that for the resulting control congruence $x_3 \in \rho_1 \subseteq \pi_1$ and $x_4 \in \rho_2 \subseteq \pi_2$. Since x_3 and x_4 are not mergeable, π_1 and π_2 cannot be merged to form a control congruence. ■

Note that Lemma 2 does not mean that the smallest control congruence is found by Algorithm 1. A control congruence (and resulting local supervisor) is generally non-unique, and which is found by Algorithm 1 depends on the order in which mergeability of the states is checked, which depends on their indexing. Unfortunately, finding a control congruence with the smallest number of cells is an NP-hard problem [5].

Lemmas 1 and 2 are applicable for supervisor localization [4] and supervisor reduction [5] (which also uses Algorithms 1 and 2, i.e., not only applicable to the transformational approach we present next.

IV. Problem definition

We assume a base system G consisting of n agents, a supervisor S, and a partition $\dot{\bigcup}_{k\in\{1,\ldots,n\}} \Sigma_{c,k} = \Sigma_c \subseteq \Sigma$ of controllable events. This base system has been localized, i.e., a control congruence $\mathcal{C}_k$ was obtained for each agent k.

Now the system changes to variant system G' consisting of n' agents, a supervisor S', and a partition of controllable events $\dot{\bigcup}_{k\in\{1,\ldots,n'\}} \Sigma'_{c,k} = \Sigma'_c \subseteq \Sigma'$. We compute $\mathcal{C}'_k$ and LOC'_k for all k from 1 to n' based on the control congruences of the base system, rather than starting localization from scratch. We call this procedure *transformational supervisor localization* (TSL). TSL is to correctly localize the variant system, as defined in Problem 1. Note that in this problem definition, any adaptation can be made to the base system (that generates a well-defined variant system).

Problem 1: Use $\mathcal{C}_k$ for k from 1 to n of the base system G, S to transformationally compute new local supervisors LOC'_k for all k from 1 to n' that are control equivalent (Equations 3 and 4) to S' with respect to G'. ♦

Since a set of local supervisors can be constructed from a set of control covers, in our work we mainly focus on finding control covers (in this case, control congruences) for the variant system in a transformational approach.

Furthermore, it is desirable to have small local supervisors. Therefore, TSL will compute maximally reduced control covers to use in the construction of the local supervisors.

V. Transformational supervisor localization

In this section, we first discuss an algorithm that is used to transform a cover $\mathcal{C}_k$ to a control congruence in case the system has been adapted. Next, we use this algorithm in the general procedure used for TSL.

A. Isolating conflicts

We consider the case that a control congruence $\mathcal{C}_k$ has been computed for some base system $S = (X, \Sigma, \xi, x_0, X_m)$, $G = (Q, \Sigma, \delta, q_0, Q_m)$. Now the system is adapted to form variant system $S' = (X', \Sigma', \xi', x'_0, X'_m)$, $G' = (Q', \Sigma', \delta', q'_0, Q'_m)$. In our notation, we use E', D'_k, ... to indicate that the function E, D_k, ... are applied to the variant automaton. I.e., E' is a function $E' : X' \to 2^{\Sigma'}$.

Algorithm 3 constructs a control congruence $\mathcal{C}'_k$ based on $\mathcal{C}_k$. First, states that are removed from X to create X'

Algorithm 3 `isolate`

Input: $\mathcal{C}_k$, S, G', S', $\Sigma'_{c,k}$
Output: $\mathcal{C}'_k$

1: $\mathcal{C}'_k = \{\pi \setminus (X \setminus X') | \pi \in \mathcal{C}_k\} \cup \bigcup\{\{x\} | x \in X' \setminus X\}$
2: $flag = true$
3: **while** $flag$ **do**
4: $\quad flag = false$
5: $\quad$ **for all** $x \in X' \cap X$ **do**
6: $\quad\quad$ **if** $\exists x' \in [x]_{\mathcal{C}'_k} : \big((x,x') \notin \mathcal{R}'_k \vee (\exists \sigma \in E'(x) \cap E'(x')) : ([\xi'(x,\sigma)]_{\mathcal{C}'_k} \neq [\xi'(x',\sigma)]_{\mathcal{C}'_k})\big)$ **then**
7: $\quad\quad\quad flag = true$
8: $\quad\quad\quad \mathcal{C}'_k = (\mathcal{C}'_k \setminus \{[x]_{\mathcal{C}'_k}\}) \cup \{[x]_{\mathcal{C}'_k} \setminus \{x\}\} \cup \{\{x\}\}$
9: $\quad\quad$ **end**
10: $\quad$ **end**
11: **end**
12: **return** $\mathcal{C}'_k$

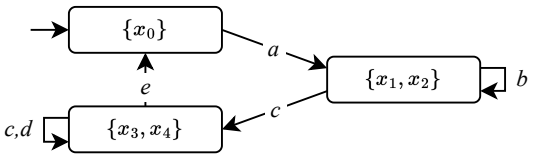


Fig. 2: Isolated state x_0

are removed from the cells they were in in $\mathcal{C}_k$. New states are added as singleton cells. Next, the algorithm looks for states x that do not satisfy condition (*i*) or (*ii*) of a control cover from Section II anymore with a state x' in the same cell. If such a state x is found, it is *isolated*: it is removed from its initial cell and placed in a singleton cell. Note that conditions (*i*) and (*ii*) are always satisfied for states in a singleton cell. Finally, all states that induce such a control consistency conflict are isolated, and the resulting cover is a control congruence.

We first present Example 2 to demonstrate the functioning of Algorithm 3. Next, we prove correctness of Algorithm 3 in Theorem 1.

Example 2: Let us consider the case the system of Example 1 is adapted. In addition to the disablements $D_1(x_0)=\{c\}, D_1(x_2)=\{a\}$ in the base system, the variant system has an additional disablement: $D_1(x_3)=\{a\}$. As a result, for the variant system $(x_0, x_3) \notin \mathcal{R}_1$. Therefore, the cover found in Example 1 is not valid anymore. This conflict is found in line 6 of Algorithm 3, and subsequently x_0 (or x_3 depending on order of iteration) is removed from its previous cell and placed in a singleton cell. No more conflicts exist in the resulting cover. Constructing a local supervisor for this cover yields the automaton shown in Fig. 2. ♦

Theorem 1: Given $N = |X \cap X'|$, Algorithm 3 terminates, has a worst-case time complexity of $\mathcal{O}(|\Sigma| \cdot N^3)$, and the generated cover $\mathcal{C}'_k$ is a control congruence w.r.t. $G', S', \Sigma'_{c,k}$.

Proof: A state in a singleton cell is trivially control consistent. If in the for-loop (lines 5-10) a state is found that is not control consistent with another state in the same cell, it is placed in a singleton cell and removed from its original cell, and the algorithm iterates over all states in $X \cap X'$ again. Eventually, since N is finite, there are no more non-control consistent states, the for-loop terminates with $flag = false$, the while-loop breaks, and the algorithm terminates.

Checking the if-condition on line 6 has a worst-case cost of $|\Sigma| \cdot N$. The for-loop (lines 5-10) is performed N times in worst-case. The while-loop (lines 3-11) is performed N times in worst-case. Therefore, the time complexity is $\mathcal{O}(|\Sigma| \cdot N^3)$.[2]

The while-loop only breaks when conditions (*i*) and (*ii*) are both met for all states in $X \cap X'$. Also all states in $X' \setminus X$ are control consistent as they are placed in singleton cells. There is no overlap between cells in $\mathcal{C}'_k$ as all cells in $X' \setminus X$ are placed in singleton cells and no merges are performed for states in $X \cap X'$, which are initially partitioned by $\mathcal{C}_k$. Thus, $\mathcal{C}'_k$ is a control congruence w.r.t. $G', S', \Sigma'_{c,k}$. ■

Algorithm 4 `TSL`

Input: $\{\mathcal{C}_k | 1 \leq k \leq n\}$, S, G', S', $\{\Sigma'_{c,k} | 1 \leq k \leq n'\}$, $\mathcal{M}$
Output: $\{LOC'_k | 1 \leq k \leq n'\}$, $\{C'_k | 1 \leq k \leq n'\}$

1: **for** $k = 1$ to n' **do**
2: $\quad$ **if** $\mathcal{M}(k) \neq 0$ **then**
3: $\quad\quad \mathcal{C}'_{k,init} = \texttt{isolate}(\mathcal{C}_{\mathcal{M}(k)}, S, G', S', \Sigma'_{c,k})$
4: $\quad$ **else**
5: $\quad\quad \mathcal{C}'_{k,init} = \{\{x\} | x \in X'\}$
6: $\quad$ **end**
7: $\quad \mathcal{C}'_k = \texttt{localize}(G', S', \Sigma'_{c,k}, \mathcal{C}'_{k,init})$
8: $\quad$ Compute LOC'_k based on $\mathcal{C}'_k$
9: **end**
10: **return** $\{LOC'_k | 1 \leq k \leq n'\}$, $\{C'_k | 1 \leq k \leq n'\}$

B. General procedure

In this section we present the TSL procedure, show in Theorem 2 that TSL solves Problem 1, and in Theorem 3 that the resulting control congruences are maximally reduced.

The TSL procedure is sketched in pseudo-code in Algorithm 4. We assume a mapping $\mathcal{M} : \{1, ..., n'\} \to \{0, ..., n\}$, that maps every agent in the variant system to either an agent of the base system, or to '0'. If $\mathcal{M}(k)$=0, it means no base control cover is selected and the inital control congruence is set to a singleton cover. In case $\mathcal{M}(k)$ is nonzero, control congruence $\mathcal{C}_{\mathcal{M}(k)}$ is selected from the base system to perform `isolate` to find an initial control congruence. After performing `isolate`, the resulting cover might not be maximally reduced. This is why, after performing `isolate`, the cover is used to initialize `localize` in order to merge cells whenever possible. The reasoning for the TSL procedure is that `isolate` produces a control congruence in which generally states will already be merged into cells, limiting the work that needs to be done during `localize`. This is demonstrated in Example 3. `TSL` also returns covers $\{C'_k | 1 \leq k \leq n'\}$ so that they can be used in a next TSL if the system is further adapted.

Example 3: This is a continuation of Example 2, in which a variant system was presented to the base system of Example 1, and `isolate` was performed to compute a

[2]To achieve this cost in implementation, instead of storing cells as state sets, a cell index number is stored for each state. A state can be isolated by simply assigning it with a new cell index. Since all cells are non-overlapping, comparing whether two cells are the same can be done by comparing the cell index of one state from each cell.

control congruence for the variant system, yielding the local supervisor of Fig. 2. However, the cover can be further reduced, resulting in a local supervisor with less states. We perform `localize` initialized with the cover found in Example 2. $\{x_0\}$ cannot merge with $\{x_1, x_2\}$ for multiple reasons: x_1 and x_2 both enable event c, which is disabled in x_0, and x_0 enables event a, which is disabled in x_2. $\{x_0\}$ cannot merge with $\{x_3, x_4\}$ as x_0 enables a, which is disabled in x_3 in the variant system. $\{x_1, x_2\}$ can be merged with $\{x_3, x_4\}$: there are no conflicts. After merging these cells, no further merges are possible, leading to control congruence $\{\{x_0\}, \{x_1, x_2, x_3, x_4\}\}$. Constructing a local supervisor for this cover yields the automaton in Fig. 3. ♦

Theorem 2: Algorithm 4 terminates, has worst-case complexity $\mathcal{O}(n' \cdot |\Sigma'| \cdot |X'|^4)$, and solves Problem 1.

Proof: Algorithm 4 terminates because `isolate` (Theorem 1) and `localize` ([5], [4]) terminate.

In worst-case, `isolate` is called n' times, and its complexity is $\mathcal{O}(|\Sigma'| \cdot |X \cap X'|^3)$ (Theorem 1). `localize` is called n' times, and its complexity is $\mathcal{O}(|\Sigma'| \cdot |X'|^4)$ [5], [4]. Therefore, the complexity of `TSL` is $\mathcal{O}(n' \cdot |\Sigma'| \cdot |X'|^4)$.

For each agent in the variant system, `localize` is initiated with a control congruence, since line 3 constructs a control congruence (Theorem 1) and the singleton cover constructed in line 5 is a control congruence. Thus, the covers computed by `localize` are control congruences following from Lemma 1. It is shown in [4] that local supervisors constructed from control congruences satisfy Problem 1. ■

Clearly, SL and TSL have the same complexity. The idea is that TSL is quicker in practice, when the variant system is sufficiently similar to the base system. Unfortunately, at the moment we can not predict whether TSL will be quicker than SL. We present some experiments in Section VI to study the computational benefit in practice.

In addition to correctness of the result, `TSL` also produces maximally reduced control congruences.

Theorem 3: All $\mathcal{C}'_k \in \{\mathcal{C}'_k | 1 \leq k \leq n'\}$ obtained by Algorithm 4 are maximally reduced w.r.t. $G', S', \Sigma'_{c,k}$.

Proof: Every control congruence $\mathcal{C}'_k$ that is returned by Algorithm 4 is constructed by performing Algorithm 1. Control congruences constructed by Algorithm 1 are maximally reduced (Lemma 2). Thus, the theorem holds. ■

VI. Case study: Cat and Mouse Tower

As a case study to evaluate the efficiency of TSL relative to SL, we take the Cat and Mouse Tower (CMT) from [8]. There are n floors, and on each floor of the tower there are five rooms as shown in Fig. 4. Cats and mice can move between the rooms as indicated by the arrows. Between each level there is a connection for both cats and mice. This connection is between room j of level $5 \cdot i + j$ to room j of level $5 \cdot i + j + 1$, for $i \in \mathbb{N}_0$, $j \in \{1, 2, 3, 4, 5\}$, and $5 \cdot i + j < n$. So room 1 level 1 is connected to room 1 level 2; room 2 level 2 is connected to room 2 level 3; and so forth, essentially forming a spiraling staircase. All doors can be controlled, except for the bidirectional cat door between rooms 2 and 4. There are k cats and k mice, and consequently

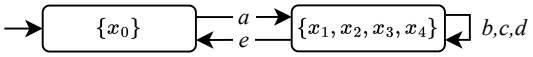


Fig. 3: Local supervisor of variant system

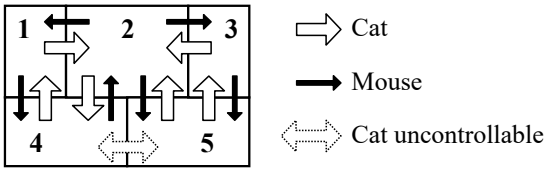


Fig. 4: CMT room layout of a level [9]

each room can also hold between 0 and k cats and/or mice. The cats start in room 1 of level 1, and the mice start in room 5 of level n. The requirement of this system is that there can never be a cat and a mouse in the same room at the same time.

As base system, we take a tower with four levels, one cat, and one mouse. The monolithic supervisor of this system has 362 states and 1159 transitions. For localization, we consider each level as a separate agent. An agent controls all events of the cat and mouse that originate in that level, e.g., the level 1 agent controls all doors on that level, and the movements from level 1 room 1 to level 2 room 1 (but not the other way around; these are controlled by the level 2 agent).

We construct five variant systems (each modifies the base system directly, i.e., the adaptations are not cumulative):

1) Removed cat door from room 3 to room 4 on level 2.
2) Made all doors controllable.
3) Added requirement that cats should never reach level 4.
4) Removed room 5 of level 1.
5) Added a room 6 to level 1 with bidirectional controllable doors for cat and mouse to room 5 of level 1.

The models and a proof-of-concept implementation of the algorithms have been made in Matlab[3]. We performed SL for the base system, and SL and TSL for each variant system. For TSL, each agent (floor) of the variant system is mapped to the same floor in the base system. A standard personal computer with i7 processor was used. Matlab used less than 2 GB of memory. Since we draw conclusions on relative and not absolute runtimes, the conclusions are not influenced by the hardware. Because the results are influenced by state indexing order, the experiments are performed for ten random index orders and mean values over those runs are presented.

In the left side of Table I we compare the computation time in seconds of performing SL and TSL for the agents in the variant system. To provide further detail, we show how much time of performing TSL is spent on the `isolate` and `localize` portion of the procedure. The percentage change comparing TSL to SL is displayed, where a negative or positive value respectively indicates how much quicker or slower TSL is compared to SL.

In the right side of Table I we compare the number of cells between the result of SL and TSL for the agents in the

[3]All used models and algorithms can be found here: `https://github.com/sbthuijsman/TSL`.

TABLE I: CMT experimental results

variant system	agent	mean runtime					mean # cells			
		SL [s]	`isolate` [s]	initialized `localize` [s]	TSL [s] (sum)	% change	SL	initial guess	isolated	TSL
1 (362 states, 1142 trans.)	1	1.22	0.05	0.15	0.20	-83%	11.6	11.6	11.6	11.6
	2	1.11	0.04	0.21	0.25	-78%	14.3	14.3	14.3	14.3
	3	1.16	0.04	0.15	0.18	-84%	13.7	13.7	13.7	13.7
	4	2.47	0.06	0.08	0.14	-94%	10.9	10.9	10.9	10.9
2 (375 states, 1214 trans.)	1	1.84	0.03	1.73	1.76	-4%	11.5	23.9	295.6	18.8
	2	1.10	0.05	1.26	1.30	+18%	15.0	27.8	231.6	22.1
	3	1.32	0.04	1.21	1.26	-5%	14.7	27.5	232.7	21.7
	4	3.78	0.03	1.92	1.95	-48%	11.1	24.3	292.7	17.2
3 (270 states, 853 trans.)	1	0.78	0.04	0.10	0.14	-82%	9.4	8.8	8.8	8.8
	2	0.71	0.02	0.09	0.11	-85%	12.8	14.1	14.1	13.9
	3	0.88	0.03	0.10	0.12	-86%	10.8	14.3	14.7	13.9
	4	6.90	0.04	3.68	3.72	-46%	2.5	10.0	10.0	9.0
4 (309 states, 986 trans.)	1	2.19	0.05	0.65	0.69	-68%	8.9	11.2	11.2	10.6
	2	0.82	0.03	0.10	0.12	-85%	12.8	14.3	14.3	14.3
	3	0.79	0.03	0.15	0.17	-78%	13.9	14.1	14.1	14.1
	4	1.78	0.04	0.08	0.13	-93%	10.6	11.4	11.4	11.4
5 (403 states, 1304 trans.)	1	2.23	0.05	1.49	1.54	-31%	12.1	51.5	327.4	14.7
	2	1.41	0.05	0.66	0.71	-50%	14.6	55.2	269.4	15.9
	3	1.79	0.05	0.61	0.66	-63%	14.3	55.1	262.0	14.4
	4	4.59	0.03	1.49	1.52	-67%	11.2	52.4	323.2	11.7

variant system. The numbers under 'initial guess' indicate the number of cells of $\mathcal{C}'_k$ after line 1 of `isolate`, before any states are isolated. The numbers under 'isolated' indicate the number of cells after completing `isolate`, but before `localize` is performed.

For the first variant system, we observe that no states need to be isolated during `isolate` and no further merges of cells can be performed when performing `localize` initialized by the control cover of the base system. Compared to performing SL initialized by a singleton cover, TSL is much quicker. For the second variant system, there is much less computational benefit. Here, a local system is found were TSL is slower than SL, i.e., in this case localization is quicker when initialized by a singleton cover. At the moment, we have no way to predict when this will be the case. We observe that for this system a lot of states need to be isolated for all subsystems. Even so, isolation is performed relatively quickly. Because the isolated cover is relatively close to the singleton cover (which has 375 cells), TSL runtimes are relatively close to the SL runtimes. Another observation is that TSL computes covers with more cells than SL, because it starts with a coarser cover which limits the cell merges that can be made during `localize`. For variant systems 3, 4, and 5 TSL is consistently quicker than SL, even though for variant system 5 a lot of states require to be isolated.

The same experiments have been performed for larger instances of CMT, with 6 levels (842 states) and 8 levels (1525 states). Because of space constraints we cannot fully present those results in this paper, they are available in the repository linked above. The same conclusions can be made for these larger instances. Respectively, the average percentage change over all local systems for CMT with 4, 6, and 8 levels, were -61%, -54%, and -57%.

From a monolithic point of view the adaptations made to the CMT system are considerable (reflected in the change in number of states and transitions). Regardless, these experiments suggest that TSL is more efficient than performing SL from scratch.

VII. Conclusions

We presented a TSL procedure, that reuses control congruences from a previous SL to more efficiently compute these control congruences for a system once it is adapted. Correctness of the algorithms is shown, and examples are provided. The method is evaluated by means of some experiments on the CMT system. For these experiments, the runtime of TSL is shown to be lower than SL.

References

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

[2] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 3rd ed. Springer, 2021.

[3] L. Moormann, R. Schouten, J. van de Mortel-Fronczak, W. Fokkink, and J. Rooda, "Synthesis and implementation of distributed supervisory controllers with communication delays," in *Conf. Autom. Sci. Eng.* IEEE, 2021.

[4] K. Cai and W. Wonham, "Supervisor localization: A top-down approach to distributed control of discrete-event systems," *Trans. Autom. Control*, vol. 55, no. 3, pp. 605–618, 2010.

[5] R. Su and W. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Syst.*, vol. 14, no. 1, pp. 31–53, 2004.

[6] S. Thuijsman and M. Reniers, "Transformational supervisor synthesis for evolving systems," *Discrete Event Dynamic Syst.*, vol. 32, no. 2, pp. 317–358, 2022.

[7] S. Thuijsman, M. Reniers, and K. Cai, "Transformational nonblocking verification," *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 256–263, 2022.

[8] C. Ma and W. Wonham, "STSLib and its application to two benchmarks," in *IEEE Workshop Discrete Event Syst.*, 2008, pp. 119–124.

[9] S. Thuijsman, M. Reniers, and D. Hendriks, "Efficiently enforcing mutual state exclusion requirements in symbolic supervisor synthesis," in *Conf. Autom. Sci. Eng.* IEEE, 2021.