Supervisory Control of Concurrent Discrete Event Systems with Local Linear Temporal Logic Specifications

Ami Sakakibara[1] and Toshimitsu Ushio[2]
[1,2]Graduate School of Engineering Science, Osaka University
Toyonaka, Osaka 560–8531, Japan
E-mail : [1]sakakibara@hopf.sys.es.osaka-u.ac.jp, [2]ushio@sys.es.osaka-u.ac.jp

Abstract: We consider a concurrent discrete event system, where each subsystem has its local specification described by a linear temporal logic formula. Then, we propose an algorithm to synthesize a supervisor for the concurrent system such that each subsystem satisfies a given linear temporal logic formula, and any subsystem never reaches a deadlock state.

1. Introduction

Linear temporal logic (LTL) is often used to describe control specifications. For example, in robot motion planing, temporal constraints such as mission completeness, deadlock avoidance, safety properties are given by LTL formulas [2]. There have been several studies tackling control problems with LTL requirements. Jiang and Kumar [1] design controllers of discret event systems by an automata-based approach. We proposed a design method of a controller for a quantitative discrete event system that satisfies a given LTL constraint [3].

In practice, however, it is common to consider a number of systems, rather than a single system. For example, some lines in a factory consist of a series of more than two robots running in cooperation. Then, it is required to control such systems appropriately, i.e., we have to control the entire system so that each subsystem will surely satisfy its given control specification. For a control problem of these systems, it is common way to compose the systems themselves in the beginning, and then control them under an LTL formula describing a global specification [4]. Unfortunately, this approach has difficulty in adapting some changes among the systems or the local specifications.

In this paper, we study control of a concurrent discrete event system consisting of N subsystems, where the local control requirement is written by an LTL formula for each subsystem. Our objective is to design a supervisor for the concurrent system ensuring that all subsystems satisfy their own specifications and that any subsystem never enters a deadlock state. The proposed design procedure is as follows. First, for each LTL formula, we construct a generalized Rabin automaton that accepts all words satisfying the formula. Next, we compute the composition of each subsystem and its corresponding Rabin automaton. Then, we construct a tree which unfolds the behaviors of the composite system. Finally, we obtain a supervisor by eliminating illegal behaviors from the tree.

2. Concurrent Discrete Event Systems

We consider a concurrent discrete event system (DES) consisting of N subsystems modeled by automata G_i $(i = 1, 2, \ldots, N)$:

$$G_i = (X_i, \Sigma_i, \delta_i, x_{i,0}, L_i, AP_i),$$

where X_i is the set of states, $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}$ is the set of events and partitioned into sets of controllable and uncontrollable events, a partial function $\delta_i : X_i \times \Sigma_i \to X_i$ is a transition function, $x_{i,0} \in X_i$ is the initial state, $L_i : X_i \to 2^{AP_i}$ is a labeling function, and AP_i is the set of atomic propositions. We assume that $AP_i \cap AP_j = \emptyset$ if $i \neq j$. The set of all active events at state $x \in X_i$ is denoted by $\Sigma_i(x)$.

Behaviors of a system G_i are represented by sequences consisting of states and events. An infinite sequence $x_0\sigma_1 x_1 \ldots \in X_i(\Sigma_i X_i)^\omega$ is called a *run* if , for any $j \in \mathbb{N}$, $x_{j+1} = \delta_i(x_j, \sigma_{j+1})$. A finite sequence $x_0\sigma_1 x_1 \ldots \sigma_n x_n \in X_i(\Sigma_i X_i)^*$ is called a *history* if , for any $j \in \{0, 1, \ldots, n-1\}, x_{j+1} = \delta_i(x_j, \sigma_{j+1})$. A run or a history is *initialized* if it starts from the initial state $x_{i,0}$. $\mathsf{Runs}(G_i)$ (resp., $\mathsf{His}(G_i)$) is the set of all initialized runs (resp., histories) in the plant. The last state of each history $h = x_0\sigma_1 x_1 \ldots x_n$ is denoted by $\mathsf{last}(h)$, that is, $\mathsf{last}(h) = x_n$. A state $x \in X_i$ is *deadlock* if $\Sigma(x) = \emptyset$, i.e., x has no outgoing transition. A history $x_0\sigma_1 ... x_n$ is called a cycle if $x_0 = x_n$. If there exist indices $i, j \in \mathbb{N}$ for a history h such that $c = x_i\sigma_{i+1} \ldots x_j$ is a cycle, we say that the history h contains the cycle c.

We assume that parts of the subsystems have some events in common and such events are called *shared events*. For each event σ, let $In(\sigma) = \{i : \sigma \in \Sigma_i\}$. Shared events σ can occur only when σ is enabled in all related subsystems, that is,

$$\forall i \in In(\sigma), \delta_i(x_i, \sigma)!, \tag{1}$$

where $\delta_i(x_i, \sigma)!$ means that there exists a transition from x_i with σ. The concurrent system [5] of all subsystems is given by

$$G = (X, \Sigma, \delta, x_0, L, AP),$$

where $X = X_1 \times \ldots \times X_N$, $\Sigma = \cup_{i=1}^N \Sigma_i$, $x_0 = (x_{1,0}, \ldots, x_{N,0})$, $AP = \cup_{i=1}^N AP_i$. The transition function δ is defined as follows: for $(x_1, \ldots, x_N) \in X$ and $\sigma \in \Sigma$,

$$\delta((x_1, \ldots, x_N), \sigma) = \begin{cases} (x'_1, \ldots, x'_N) & \text{if Eq. (1) holds,} \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{2}$$

where $x'_i = \delta(x_i, \sigma)$ if $i \in In(\sigma)$; $x'_i = x_i$ otherwise. $L : X \to 2^{AP}$ is the labeling function of the entire system such that $L((x_1, \ldots, x_N)) = \cup_{i=1}^N L_i(x_i)$.

Remark: We allow the existence of uncontrollable shared events, unlike the assumption in [5] that $\Sigma_{i,u} \cap \Sigma_j = \emptyset$ for any $i \neq j$.

3. Formulation

3.1 Linear temporal logic

In this paper, each subsystem has its local specification described by a linear temporal logic (LTL) formula.

Syntax of LTL: Let AP be a set of atomic propositions. An LTL formula over AP is defined as

$$\varphi ::= \mathbf{tt} \,|\, \mathbf{ff} \,|\, a \,|\, \neg\varphi \,|\, \varphi_1 \wedge \varphi_2 \,|\, \varphi_1 \vee \varphi_2 \,|\, \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

where $a \in AP$.

For an infinite word $w \in (2^{AP})^\omega$, we denote $w \models \varphi$ if the word w satisfies the formula φ. The satisfaction relation $\models$ is formally defined in [6]. Intuitively, each operator can be interpreted as follows: $\mathbf{X}\varphi$ means that φ holds along a word starting at the next letter; $\mathbf{F}\varphi$ says that φ will be true in the future; $\mathbf{G}\varphi$ represents that φ always holds along a word; $\varphi_1 \mathbf{U} \varphi_2$ implies that a word keeps to satisfy φ_1 until φ_2 turns to be true.

We say a system G satisfies an LTL formula φ, denoted $G \models \varphi$, if

$$\forall \rho \in \mathsf{Runs}(G), L(\rho) \models \varphi$$

where $\rho = x_0 \sigma_1 x_1 \ldots$ and $L(\rho) = L(x_0)L(x_1)\ldots$.

In this paper, we construct a generalized deterministic Rabin automaton (GDRA) for an LTL formula φ according to the method proposed in [6]. Any LTL formula φ can be translated into a GDRA of the form

$$R = (Q, \delta_R, q_0, Acc),$$

where Q is the set of states, q_0 is the initial state, $\delta_R : Q \times 2^{AP} \to Q$ is a transition function, and $Acc = \{Acc_j = (\mathcal{E}^j, \mathcal{F}^j) : j \in \mathcal{I}\}$ is the set of acceptance conditions with an index set $\mathcal{I}$, and the generalized Rabin pair $(\mathcal{E}_i, \mathcal{F}_i) \subseteq \delta_R \times 2^{\delta_R}$. $\mathcal{F}^j$ is of the form $\mathcal{F}^j = \{F^{j1}, \ldots, F^{jK_j}\}$. An infinite word w is accepted if, for some $j \in \mathcal{I}$,

$$\mathsf{Inf}_\mathsf{R}(w) \cap \mathcal{E}^j = \emptyset \wedge 1 \leq \forall k \leq K_j, \mathsf{Inf}_\mathsf{R}(w) \cap \mathcal{F}^{jk} \neq \emptyset$$

where $\mathsf{Inf}_\mathsf{R}(w)$ is the set of transitions that occur infinitely often along w. Note that Rabin conditions are not given by pairs of state but those of transitions.

3.2 Supervisory control

The best-known theory for the control of DESs is supervisory control, initiated by Ramadge and Wonham [7]. A supervisor is defined as a mapping $S : \mathsf{His}\,(G) \to \Gamma$ where $\Gamma = \{\gamma \in 2^\Sigma : \Sigma_u \subseteq \gamma\}$, i.e., a supervisor observes the history from the initial state to the current state and determines a control pattern.

Let φ_i be an LTL formula that represents a specification for a subsystem G_i. Our objective is to synthesize a supervisor for the concurrent system G such that 1) each subsystem satisfies a given LTL formula, and 2) any subsystem never reaches a deadlock state.

4. Algorithm

4.1 Product automata

First, specification formulas $\varphi_1, \ldots, \varphi_N$ are translated into equivalent GDRAs $R_1, \ldots, R_N$. To investigate the relation between behaviors of a considered system and an LTL requirement, we often take the product of the system and the GDRA corresponding to the specification [3]. Formally, for each i, the composition of the subsystem and the GDRA is

$$G_i \otimes R_i = (X_i \times Q_i, \Sigma_i, \delta_i^\otimes, x'_{i,0}, Acc'_i),$$

where $X_i \times Q_i$ is the set of states, $\delta_i^\otimes : (X_i \times Q_i) \times \Sigma_i \to (X_i \times Q_i)$ is the transition defined as

$$\delta_i^\otimes((x,q),\sigma) = (\delta_i(x,\sigma), \delta_{R,i}(q, L_i(x)))$$

for each $(x,q) \in X_i \times Q_i$ and $\sigma \in \Sigma$, $x'_{i,0} := (x_{i,0}, \delta_{R,i}(q_{i,0}, L_i(x_{i,0})))$ is the initial state, Acc'_i is the set of acceptance conditions.

The desired runs are accepted by the automaton $G_1 \otimes R_1 \| \ldots \| G_N \otimes R_N$. It is necessary, therefore, to find a control action through which all acceptance conditions $Acc'_1, \ldots, Acc'_N$ are satisfied in the product automata.

4.2 Searching tree

Algorithm 1 briefly describes how to construct a searching tree. From the subsystems $G_1, \ldots, G_N$ and the GDRAs $R_1, \ldots, R_N$, we build the concurrent system but dynamically writes down its behaviors in a tree structure. The output of the algorithm is $T = (V, E)$, where V is the set of nodes and E is the set of edges.

To represent which conditions have been satisfied, an $N \times M$ matrix is introduced for each node, denoted by $acc(v) = \left(acc_{ij}^v\right)$, where $M = \max_{1 \leq i \leq N} |\mathcal{I}_i|$. All elements are initialized with 0 and acc_{ij}^v turns to 1 if the Acc_i^j, the j-th acceptance condition of R_i, is satisfied.

For each node v, let $\mathsf{cycle}(v)$ be the maximal cyclic subsequence of v^{his}. We say v is accepted by R_j if it holds that

$$\mathsf{cycle}(v) \cap \mathcal{E}^j = \emptyset \ \wedge \ \forall \mathcal{F}' \in \mathcal{F}^j, \mathsf{cycle}(v) \cap \mathcal{F}' \neq \emptyset \quad (3)$$

since $\mathsf{cycle}(v)$ is the subsequence of v^{his} that occurs infinitely often. If Eq. (3) holds for all $j \in \mathcal{I}$, then it is guaranteed that the entire system meets all of the acceptance conditions through $\mathsf{cycle}(v)$. When checking the acceptance conditions, the matrix $acc(v)$ plays an important role. Indeed, $acc(v)$ reflects the evaluation of Eq. (3).

The main part of Algorithm 1 works as follows. From the initial state of the concurrent system, namely $x_0 = (x_{1,0}, \ldots, x_{N,0})$, the algorithm writes down the possible histories of G by function $Expand$. The label of node v, denoted v^{his}, keeps the history from the initial state to the current state of the entire system. Whenever a cycle is detected, i.e., the current sequence ends with a cycle, the algorithm examines whether the current node is accepted or not. If v is accepted, then we add it to V_L, which is the set of accepted leaf nodes; otherwise, we consider the next conditional execution. If the currently detected cycle contains bad transitions,

Algorithm 1 Construction of a searching tree

Require: $G_1, \ldots, G_N, R_1, \ldots, R_N$
  Let $v_0 = (x_{1,0}, \ldots, x_{N,0})$, $v^{his} = v_0$, and $acc(v_0) = (\mathbf{0}, \ldots, \mathbf{0})$. Put v into V and U. Let V_c, V_u, V_L, and $V_{\neg\varphi}$ be empty sets.
  while $U \neq \emptyset$ **do**
    for all $v \in U$ **do**
      Let $U' = \emptyset$.
      if the current node v ends with a cycle **then**
        Check the acceptance conditions by $IsAcc(v)$
        if $\forall i, \exists j, acc_{ij}^v = 1$ (v is accepted) **then**
          Put v into V_L (label v as a leaf node).
        else
          if $\exists i, \forall j, acc_{ij}^v = -\infty$ or Eq. (4) holds **then**
            Put v into $V_{\neg\varphi}$ (label v as a bad node).
          end if
        end if
      else
        for all $p \in \{c, u\}$ **do**
          if $\Sigma_p(\mathsf{last}(v)) \neq \emptyset$ and Eq. (4) does not hold **then**
            $U' \leftarrow Expand(v, p)$
          end if
        end for
      end if
    end for
    Update U by U'.
  end while
  return $T = (V, E)$

function $IsAcc(v)$
  Let c be the cycle that is currently detected.
  if c contains a transition that belongs to $\mathcal{F}_i^j$ **then**
    Let $acc_{ij}^v = 1$.
  end if
  if c contains a transition that belongs to $\mathcal{E}_i^j$ **then**
    Let $acc_{ij}^v = -\infty$.
  end if

function $Expand(v, p)$
  Create a new node $v^p \in V_p$.
  Let $\tilde{U}$ be an empty set and $(v, \pi_p, v^p) \in E$.
  for all $\sigma \in \Sigma_p(\mathsf{last}(v))$ **do**
    if $vis(v, \sigma) = 0$ **then**
      Create a new node u according to the transition rule (Eq. (2)). $u^{his} = v^{his}\sigma\delta(\mathsf{last}(v^{his}), \sigma)$. Let $acc(u) = acc(v)$ and $(v^p, \sigma, u) \in E$. Put u into V and $\tilde{U}$.
    end if
  end for
  return $\tilde{U}$

function $vis(v, \sigma)$
  if $\exists (v', \sigma, v'') \in E$ s.t. $\mathsf{last}(v) = \mathsf{last}(v')$ and v' is an ancestor of v **then**
    return 1
  else
    return 0
  end if

further search will result in nothing. Stated differently, this suffix itself violates at least one of the acceptance conditions. Then, v is added to $V_{\neg\varphi}$, which means that v is labeled as a bad node. Nodes in V_L and $V_{\neg\varphi}$ will not be expanded any more. All of the leaf nodes correspond to runs of G_i that end with cycles. It is noted that every transition in such cyclic behaviors will occur infinitely often. Hence, we need to analyze the cycles and thus we seek cyclic runs.

If the suffix of the current node does not correspond to a cyclic path, then this node undergoes the expansion process. The expansion process will continue until a cycle is detected. When the first cycle is found at the current node v, examined here is whether the system satisfies the specification or not. If the system has not fulfilled the acceptance conditions yet, the algorithm continues to search. Otherwise, we pick up one of the successors of $\mathsf{last}(v^{his})$ to be investigated in the next iteration. However, at the next time we pick up one of the successors of v, we need to avoid tracing the same path again. To ensure this, function $vis(v, \sigma)$ indicates if the pair of state v and the event σ was already searched. Possible options to be tested in the next step will be restricted to those which have not been visited yet, i.e., the node corresponding to $\delta(\mathsf{last}(\mathsf{v}^{\mathsf{his}}), \sigma)$ with $vis(v, \sigma) = 0$. If we have

$$\wedge_{\sigma \in \Sigma(v)} vis(v, \sigma) = 1, \tag{4}$$

then there is no chance to obtain a preferable path from v. In this case, therefore, the expansion also ends.

Moreover, it is obvious by function $IsAcc$, that

$$\forall i, \exists j, acc_{ij}^v = 1$$

if v is accepted. To the contrary, the system violates the formula if at least one acceptance condition is refuted, i.e.,

$$\exists i, \forall j, acc_{ij}^v = -\infty.$$

In this case, further transitions from the current state will never influence the acceptance in light of the acceptance conditions.

4.3 Synthesis of a supervisor

The leaf nodes in $V_{\neg\varphi}$ correspond to the behaviors violating the acceptance conditions. By eliminating edges that lead to such bad nodes from the output tree of Algorithm 1, we obtain a supervisor.

Algorithm 2 roughly explains the procedure of designing a supervisor. Let V' be the output set of V after the operations $elim$ in the for loop. The outcome has the following properties.

Property 1 For any node v in $V' \cap V_L$, Eq. (3) holds.

Proof: From the operation rule, all nodes violating Eq. (3) are eliminated since they belong to $V_{\neg\varphi}$. Consequently, the remaining nodes are those satisfying Eq. (3).

Property 2 For any node v in $V' \cap V_L$,

$$1 \leq \forall i \leq N, \mathsf{cycle}(v) \cap \delta_i \neq \emptyset.$$

Proof: From Algorithm 1, node v does not belong to V_L whenever $\exists i, \forall j, acc_{ij}^v \neq 1$. If there exists an index i such that

Algorithm 2 Synthesis of a supervisor

Require: The output of Algorithm 1 $T = (V, E)$
  for all $v \in V_{\neg\varphi}$ **do**
    $elim(v)$
  end for
  if $V = \emptyset$ **then**
    return There exists no supervisor.
  else
    return Construct a supervisor.
  end if

function $elim(v)$
  if $pre(v) \in V_c \wedge pre(v)$ has another successor **then**
    Remove $(pre(v), \sigma, v)$ from E, and v from V
  else
    $elim(pre(v))$
  end if

subsystem G_i reaches a deadlock state, acc_{ij}^v remains 0 for any j. Moreover, they are eliminated by the operation $elim$. Then, there does not exist a node in V' with a cyclic path through which at least one subsystem remains suspended.

If $V' = \emptyset$, then there does not exist a supervisor that meets our objective. Otherwise, we construct a supervisor as follows. For nodes in $V_L \setminus V_{\neg\varphi}$, we reconnect appropriate nodes so that the corresponding cycles are achieved. A supervisor for G is implemented by a pair of an automaton and a mapping $\mathcal{S} = ((X_S, \Sigma, \delta_S, x_{S,0}), S)$, where X_S is the set of the states representing the supervised histories, Σ is the set of events, δ_S is the transition function, $x_{S,0}$ is the initial state with $x_{S,0} = x_0$, and $S : \mathsf{His}(G) \to \Gamma$ is given by

$$S(h) = \{\sigma \in \Sigma_c : \exists v, v' \in V' \text{ s.t. } v^{his} = h, h\sigma\mathsf{last}(v'^{his}) = v'^{his}, \text{ and } (v, \sigma, v') \in E'\}.$$

From Properties 1 and 2, the resulting supervisor meets our objective: $\mathcal{S}$ controls the entire system G so that 1) each subsystem satisfies a given LTL formula, and 2) any subsystem never reaches a deadlock state.

5. Example

We consider a control problem of a system consisting of two robot arms, where Arm 1 (resp., Arm 2) can execute Tasks 1 and 3 (resp., Tasks 2 and 3) [8]. Here, the control objective is to realize that 1) Arm 1 carries on Tasks 1 and 3 (infinitely often), and 2) Arm 2 executes Task 3, in cooperation with Arm 1, while avoiding Task 2. These local specifications are written by the following LTL formulas:

$$\varphi_1 = \mathbf{GF}t_1 \wedge \mathbf{GF}t_3, \quad \varphi_2 = \mathbf{G}\neg t_2 \wedge \mathbf{GF}t_3,$$

where t_i represents Task i for $i = 1, 2, 3$.

From the resulting supervisor, shown in Fig. 2 it is ensured that, whenever a_2 is active, the supervisor disables a_2 since it leads Arm 2 to Task 2.

6. Conclusion

We considered a concurrent discrete event system consisting of N subsystems, where each subsystem has its local specification described by a linear temporal logic formula. We proposed an algorithm to synthesize a supervisor for the concurrent system such that each subsystem satisfies a given linear temporal logic formula, and any subsystem never reaches a deadlock state.

Future work includes the extension to the work where a global specification is also given in addition to local ones.

Acknowledgement This research was supported in part by JSPS KAKENHI Grant Number 24360164.

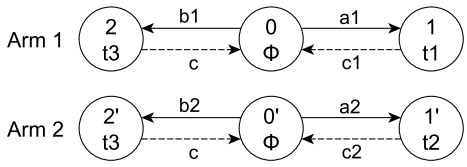

Figure 1. Automata representing the considered subsystems. $\Sigma_{c,1} = \{a_1, b_1\}, \Sigma_{u,1} = \{c_1, c\}, \Sigma_{2,c} = \{a_2, b_2\}$, and $\Sigma_{2,u} = \{c_2, c\}$. c is the only shared event. The lower label of each state stands for the atomic proposition that holds there.

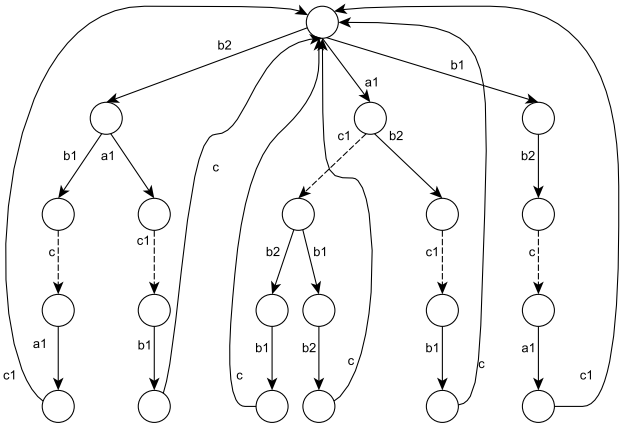

Figure 2. The resulting supervisor.

References

[1] S. Jiang and R. Kumar, *SIAM J. Control Optim.*, vol. 44, no. 6, pp. 2079–2103, 2006.
[2] X. Ding *et al.*, *IEEE Trans. Automat. Contr.*, vol. 59, no. 5, pp. 12441257, May 2014.
[3] A. Sakakibara *et al.*, in *Proc. 20th IEEE ETFA*, 2015.
[4] T. Wongpiromsarn *et al.*, *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 229–236, Mar. 2012.
[5] Y. Willner and M. Heymann, *Int. J. Control*, vol. 54, no. 5, pp. 1143–1169, 1991.
[6] J. Esparza *et al.*, submitted for publication, 2015.
[7] P. J. Ramadge and W. M. Wonham, *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.
[8] S. Takai and T. Ushio, in *Proc. 7th Int. Work. Discret. Event Syst.*, vol. E87-A, no. 4, pp. 181–186, 2004.