On Specification Informatics in Discrete-event Systems
State-transparency for Clarity of Finite Automata as Control Specifications

Amrith Dhananjayan and Kiam Tian Seow
School of Computer Engineering, Nanyang Technological University, 639798, Singapore, Singapore

Keywords: Discrete-event Systems, Supervisory Control, Finite Automata, Specification Informatics, State Transparency, Human Perception.

Abstract: In control of discrete-event systems (DES's), the formalization of control requirements from natural language statements is essentially a human endeavor. Without automated support tools, human designers often face the uncertainty of not knowing if the control requirements formalized in the rudimentary DES formalism of automata are as intended, motivating the automata-theoretic study of specification informatics in the field of DES's. A specification automaton that renders its linguistic description more transparent should help designers ascertain the prescribed requirement. Such transparency may be formalized in either the state space or the event space of the DES. In this paper, treating the former as fundamental, a state-transparent specification automaton is conceptualized with respect to a full specification automaton ('full' in the sense of having all the a priori transitional constraints of the DES embedded in it). It contains only specification relevant states called specification epochs. Each epoch denotes a "well-defined" disjoint subset of states of the full specification automaton in the same phase of execution, meaningfully aggregated such that the resultant specification automaton retains the original restrictiveness on the DES. The problem of maximizing the state-transparency of specification automata for DES's is then theoretically formulated. Subject to human perceptive or cognition limits, we believe that such a maximally state-transparent specification automaton could be more comprehensible, showing clearly the compliant execution of the system through a minimum number of specification epochs, and should aid designers in clarifying if the requirement prescribed is the one intended.

1 INTRODUCTION

Many man-made systems can be modeled as discrete-event systems (DES's). Supervisory control theory for DES's (Ramadge and Wonham, 1987) provides a theoretical framework for synthesizing controllers (called supervisors) for DES's. A DES is modeled as a finite automaton, with its possible executions generating a formal language over a finite event set. A supervisor, also modeled as a finite automaton, restricts the DES to certain sequences of events or states such that a given set of control specifications is conjunctively satisfied.

In practice, control requirements are first given in natural language statements. In our opinion, the potential benefits of supervisory control theory could be better harnessed provided the specifications in automata correctly formalize these verbal or textual statements (Ou and Hu, 2000). The translation from a natural language statement of a control requirement to a specification in automata is often a non-trivial task requiring the working knowledge of the entire DES (Cassandras and Lafortune, 2008). Often, designers understand the control requirements in natural language and manually prescribe the corresponding specification automata. The lack of formal tools to help designers in turn interpret and comprehend the specifications prescribed has been identified as a factor limiting the effective use of supervisory control theory (Grigorov et al., 2011). Without automated support tools, designers are generally faced with the uncertainty of whether a prescribed specification is as intended. This has been encountered in many applications of the automata-based DES framework (Grigorov et al., 2010) – in automated manufacturing (Magnusson et al., 2011; Cheriaux et al., 2010), robotics (Košecká and Bajcsy, 1994; Ricker et al., 1996), communicating transaction processes (Feng et al., 2007), information systems (Hinze et al., 2006) and intelligent service transportation (Seow and Pasquier, 2004), to name a few. In fact, with advances in computing technologies mitigating complexity to a practically feasible extent, human comprehensibility is turning out to be more challenging than computa-

Dhananjayan A. and Seow K..
On Specification Informatics in Discrete-event Systems - State-transparency for Clarity of Finite Automata as Control Specifications.
DOI: 10.5220/0003977403570367
In *Proceedings of the 9th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2012), pages 357-367
ISBN: 978-989-8565-21-1

tional issues in solving industrial problems using supervisory control theory (Wonham, 2003). This motivates the automata-theoretic study of specification informatics in the field of DES's.

Psychological studies have shown that human beings tend to first abstract the representation of a problem when addressing complex real world problems (Rasmussen, 1986). In studying a problem, an on-the-fly model abstraction of the problem could often help a human's limited mental capacity better understand the essence of the problem (Bredereke and Lankenau, 2005). From this empirical study, we infer that it might be useful to have an algorithm that can abstract an input specification automaton, and present human designers with an output automaton that is structurally clearer and thus easier to comprehend, but retains the essential expressions of the requirement prescribed by the original automaton. Such clarity or transparency comes from accentuating specification epochal information, and correctly suppressing information irrelevant to the specification without oversimplifying that could lead to ambiguity. The development of a systematic approach to obtain such transparent specification automata is the aim of this paper. Not unlike any study involving human cognition, the proposed approach is assistive and provides a necessity basis to individual designers for understanding specification automata.

Research on specification transparency for DES's originates with the work of (Pham et al., 2010). Therein, the specification transparency maximization problem introduced attempts to highlight the precedence ordering among a minimal set of events deemed relevant to the specification, by 'hiding' in self-loops all the events that are considered irrelevant to the specification but can occur in the DES. For many applications, it might be more desirable to specify for the DES in terms of state sequences or trajectories (Du and Wang, 1988; Ou and Hu, 2000; Faraut et al., 2011). In fact, it has been noted that while designers of communication systems would prefer to specify restrictions on event sequences generated by a DES model, designers of manufacturing systems would prefer to specify restrictions as constraints on state trajectories traced by the DES (Cao and Ho, 1990). A specification automaton in conjunction with a DES can equivalently be viewed as prescribing a set of 'legal' state trajectories of the DES. A specification automaton that traces state trajectories suitably projected from this legal set might thus furnish a clearer state-linguistic description that aids in easier interpretation of the prescribed control requirement. This motivates an alternative but complementary formulation of the specification transparency maximization problem studied in (Pham et al., 2010), namely, one that is based on states rather than events.

In our theoretical development, we first introduce the concept of state-transparency of control specifications prescribed in finite automata. A state-transparent specification automaton is an abstraction of a full specification automaton ('full' in the sense of having all the a priori transitional constraints of the DES embedded in it), with each of its states representing a unique epoch (or execution phase) of the DES that is relevant to the specification. As will be formalized and explained, each relevant epoch distills a disjoint subset of states of the full specification automaton that is intraconnected. By such intraconnectivity, any pair of states in each disjoint subset is "graphically connected" by the automaton via only the states in the subset, such that the member states reached by an automaton transition from the same nonmember state are all temporally strung by a DES string of events transiting through only the states in the subset. It is postulated that specification epochs defined and constructed this way would have well-defined meanings for a state-transparent specification automaton. For an intuitive understanding, the reader might want to skip ahead to Section 5 for an example of abstracting the states of a full specification automaton [Fig. 4(a)] into specification epochs, and yielding a state-transparent specification automaton [Fig. 4(b)].

Different state-transparent specification automata resulting from a full specification automaton can represent the control requirement using different sets of specification epochs; and any specification automaton is said to be the most (or maximally) state-transparent if it can retain the original restrictiveness of the specification on the DES using the least number of specification epochs. Subject to human perceptive or cognition limits, we believe that such a maximally state-transparent specification automaton could be more human comprehensible. Constructing such an automaton is a state-transparency maximization problem. In this paper, we formalize this problem, which is shown to be NP-hard elsewhere. As a polynomial time algorithm cannot be expected for this problem, we propose an algorithm that can run in polynomial time and achieve maximal state-transparency in individual cases but not in general.

The rest of this paper is organized as follows. Section 2 reviews relevant concepts in languages and automata theory, and in some graph basics. The new concept of state-transparency is defined, and the problem and NP-hardness of finding a maximally state-transparent specification automaton are formally stated in Section 3. A polynomial time algorithm that can compute state-transparent specification automata

is proposed in Section 4. An illustrative example is presented in Section 5 to demonstrate the concept of a state-transparent specification automaton. Section 6 presents some related work. Section 7 concludes the paper.

2 PRELIMINARIES

2.1 Formal Languages and Automata

We follow the framework of Ramadge and Wonham (Ramadge and Wonham, 1987). Let Σ be a finite set of events. A string denotes a finite sequence of events from Σ. Let Σ^* be the set of all finite strings from Σ including the empty string ε. A string t' is a prefix of t, if there exists a string s such that $t's = t$.

A language L over Σ is a subset of Σ^*. We say L_1 is a sublanguage of L_2 if $L_1 \subseteq L_2$. For a language L, its prefix closure $\overline{L}$ is the language consisting of all prefixes of its strings. As any string s in Σ^* is a prefix of itself, we have $L \subseteq \overline{L}$. A language L is considered prefixed-closed if $L = \overline{L}$.

Any regular language can be generated by an automaton. An automaton G is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite set of discrete states, Σ is the finite set of discrete events, $\delta : \Sigma \times Q \to Q$ is the (partial) transition function, q_0 is the initial state and $Q_m \subseteq Q$ is the set of marker states.

Write $\delta(\sigma, q)!$ to mean that $\delta(\sigma, q)$ is defined and $\neg\delta(\sigma, q)!$ to mean otherwise. The definition of δ can be extended to $\Sigma^* \times Q$ in the usual way as follows.

$$\delta(\varepsilon, q) = q,$$

$$(\forall \sigma \in \Sigma)(\forall s \in (\Sigma)^*)\delta(s\sigma, q) = \delta(\sigma, \delta(s, q)).$$

The prefix-closed language $L(G)$ and the marked language $L_m(G)$ describe the behavior of automaton G. Formally,

$$\begin{aligned} L(G) &= \{s \in \Sigma^* \mid \delta(s, q_0)!\}, \\ L_m(G) &= \{s \in L(G) \mid \delta(s, q_0) \in Q_m\}. \end{aligned}$$

A state $q \in Q$ is reachable if $(\exists s \in \Sigma^*)\ \delta(s, q_0) = q$, and coreachable if $(\exists s \in \Sigma^*)\ \delta(s, q) \in Q_m$. Automaton G is reachable if all its states are reachable, and is coreachable if all its states are coreachable, i.e., $\overline{L_m(G)} = L(G)$. If G is both reachable and coreachable then it is said to be trim.

2.2 Graph Basics

Following (Diestel, 2006), an undirected graph J is a 2-tuple (V_J, E_J), where V_J is the set of nodes and $E_J \subseteq V_J \times V_J$ is the set of edges. A path from node $u \in V_J$ to node $v \in V_J$ in J is a sequence of edges in E_J, starting at node u and ending at node v:

$(u, u_1), (u_1, u_2) \ldots (u_{k-1}, u_k)(u_k, v)$.

Two nodes u and v are said to be connected if such a path exists. A graph is connected if every pair of nodes in V_J is connected.

Finally, let $|P|$ denote the cardinality of the set P.

3 PROBLEM CONCEPTS AND DESCRIPTION

3.1 Specification Automaton and State-transparency

Definition 1. *(Pham et al., 2010) Given a DES $G = (Q, \Sigma, \delta, q_0, Q_m)$, and a regular language L such that $L = L_m(A)$, where automaton $A = (X, F, \xi, x_0, X_m)$. If A is said to be a specification automaton (of L for DES G), then 1) $F = \Sigma$, 2) $\overline{L_m(A) \cap L_m(G)} = L(A) \cap L(G)$, and 3) A is trim.*

Intuitively, a specification automaton for DES G models a (marked) sublanguage of G over event set Σ. The sublanguage $L_m(A) \cap L_m(G)$ is well modeled such that every common prefix string in $L(A) \cap L(G)$ can be extended to a marked string in $L_m(A) \cap L_m(G)$, thereby specifying an uninhibited sequence of executions to complete some task.

Given a specification automaton A for G, a trim automaton H such that $L_m(H) = L_m(A) \cap L_m(G)$ is said to be a full automaton, in that it embodies the a priori transitional constraints of G, and thus represents the full nonblocking behavioral specification for G under A. By Definition 1, H can be called a specification automaton.

Given a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for DES $G = (Q, \Sigma, \delta, q_0, Q_m)$, we define the following (cf. (Su and Wonham, 2004)).

Let $E : Y \to 2^\Sigma$ be the set of events defined at $y \in Y$ such that $E(y) = \{\sigma \in \Sigma \mid \zeta(\sigma, y)!\}$.

Let $D : Y \to 2^\Sigma$ be the set of events not permitted by the specification at $y \in Y$ such that $D(y) = \{\sigma \in \Sigma \mid \neg\zeta(\sigma, y)!$ *and* $(\exists s \in \Sigma^*)[\zeta(s, y_0) = y$ *and* $\delta(s\sigma, q_0)!]\}$.

Let $T : Y \to \{true, false\}$ with $T(y) =$ *true* if $(\exists s \in \Sigma^*)[\zeta(s, y_0) = y$ *and* $\delta(s, q_0) \in Q_m]$. $T(y)$ is *true* if y is reachable by a string in the marked language of G.

Let $M : Y \to \{true, false\}$ with $M(y) = true$ if $y \in Y_m$. $M(y)$ is *true* if y is a marker state.

We say two states in a full specification automaton H are specification compatible with each other if their associated event permission and nonpermission

in a DES G as well as their markings are consistent, formally defined as follows.

Definition 2. *Given a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for a DES G, let $C \subseteq Y \times Y$ be the set of pairs of specification compatible states of H on G. Then:*

For $y, y' \in Y$, $(y, y') \in C$ iff

1. $E(y) \cap D(y') = E(y') \cap D(y) = \emptyset$;
2. $T(y) = T(y') \Rightarrow M(y) = M(y')$.

Condition 1 states that for $(y, y') \in C$, an event permitted at y should not be denied permission at y' and vice versa. According to Condition 2, states $y, y' \in Y$ of H such that $(y, y') \in C$ are consistently marked *true* or *false* if both states are reachable by some strings in the marked language of G (i.e., when $T(y) = T(y') = true$), or if neither is reachable by a string in the marked language of G (i.e., when $T(y) = T(y') = false$).

In what follows, a procedure called 13 is developed to check if a state pair is specification compatible. Lemma 1 is immediate.

Procedure : *ChkIfInC*(y, y').

```
   Input: Two states y,y' ∈ Y of H = (Y,Σ,ζ,y0,Ym);
   Output: true if (y,y') ∈ C; false, otherwise;
 1 begin
 2     if (E(y) ∩ D(y')) ∪ (E(y') ∩ D(y)) = ∅ then
 3         if T(y) = T(y') then
 4             if M(y) = M(y') then
 5                 return true;
 6             end
 7         end
 8         else
 9             return true;
10         end
11     end
12     return false;
13 end
```

Lemma 1. *Let C be the set of pairs of specification compatible states of a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for a DES G. Then for a pair of input states $y, y' \in Y$, Procedure 13(y, y') returns true iff $(y, y') \in C$.*

A cover of a set Y is a family of subsets of Y whose union is Y. Each element of a cover is called a cell. A cover is a partition if its cells are pairwise disjoint.

We now define what is called a specification-equivalent partition P over the state set of a full specification automaton H.

Definition 3. *Let C be the set of pairs of specification compatible states of a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for a DES G. For an index set X, a partition $P = \{Y_x \subseteq Y \mid x \in X\}$ is said to be a specification-equivalent partition of H if*

1. $(\forall x \in X)(\forall y, y' \in Y_x)(y, y') \in C$;
2. $(\forall x \in X)(\forall \sigma \in \Sigma)(\exists x' \in X)[(\forall y \in Y_x)\zeta(\sigma, y)! \Rightarrow \zeta(\sigma, y) \in Y_{x'}]$.

For an index set X and a state $y \in Y$, we denote the cell of $P = \{Y_x \subseteq Y \mid x \in X\}$ containing y as $[y]$, i.e., for $y \in Y_x$, we have $[y] = Y_x$.

Condition 1 of Definition 3 states that all pairs of states within the same cell must be specification compatible. Condition 2 asserts that, for two arbitrary cells Y_x and $Y_{x'}$ of P, the states reachable via the same event from states of the same cell Y_x must all belong to the same cell $Y_{x'}$.

According to Definition 3, two states $y, y' \in Y$ are to be placed together in a cell of a specification-equivalent partition if the state pair is specifiation compatible and all the state pairs reachable by identical strings from y and y' are also specification compatible. Any two states y and y' that can be placed within a cell as such are said to be pairable. It follows that all state pairs reachable by identical strings from pairable y and y' are also pairable. State pairability is formally defined as follows.

Definition 4. *Let C be the set of pairs of specification compatible states of a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$ for a DES G. Two states $y, y' \in Y$ are said to be pairable if*

1. $(y, y') \in C$
2. $(\forall s \in \Sigma^*)[\zeta(s, y)! \text{ and } \zeta(s, y')! \Rightarrow (\zeta(s, y), \zeta(s, y')) \in C]$

In what follows, we present a procedure (cf. (Su and Wonham, 2004)) called : *PTran* that computes and returns an induced automaton $A = (X, \Sigma, \xi, x_0, X_m)$ from a given specification-equivalent partition $P = \{Y_x \subseteq Y \mid x \in X\}$ defined on a full specification automaton $H = (Y, \Sigma, \zeta, y_0, Y_m)$, such that each state $x \in X$ of A uniquely corresponds to cell Y_x of P. The initial state of A corresponds to the cell containing the initial state of H. A state of A is marked if the corresponding cell contains any marked state of H. In A, a transition of event $\sigma \in \Sigma$ from a state x to a state x' is defined if there is a transition of the same event in H, from a state in Y_x to a state in $Y_{x'}$. The procedure has a worst case complexity of $O(|Y||\Sigma|)$.

Theorem 1. *Given a full specification automaton H for a DES G and a specification-equivalent partition P of H. Then under G, $A = PTran(H, P)$ is a specification automaton modeling $L_m(H)$ such that $L_m(H) = L_m(A) \cap L_m(G)$.*

Remark 1. *A specification-equivalent partition (of Definition 3), though mathematically similar, is conceptually different from a control congruence (Su and Wonham, 2004). For a different purpose as will be*

discussed in Section 6, the latter notion requires H to be controllable; and we note that for a control congruence P_c defined on a controllable H, $PTran(H,P_c)$ becomes a procedure that returns a nonblocking and possibly state-reduced supervisor (Su and Wonham, 2004) realizing $L_m(H)$ for DES G. Exposing this technical correspondence, it is not unexpected that certain definitions in this paper have some similarity with those in (Su and Wonham, 2004).

Procedure : $PTran(H,P)$.

Input: A full specification automaton $H=(Y,\Sigma,\zeta,y_0,Y_m)$ and a specification-equivalent partition $P=\{Y_x\subseteq Y \mid x\in X\}$ of H;
Output: An induced specification automaton A such that each state of A uniquely represents a cell of P;

```
1  begin
2      x_0 = x ∈ X such that y_0 ∈ Y_x;
3      X_m = {x ∈ X | Y_x ∩ Y_m ≠ ∅};
4      ξ : Σ × X → X (pfn) with ξ(σ,x) = x' for (x,x' ∈ X) and
       (σ ∈ Σ), such that (∃y ∈ Y_x)ζ(σ,y) ∈ Y_x';
5      return A = (X,Σ,ξ,x_0,X_m);
6  end
```

Given an automaton $H=(Y,\Sigma,\zeta,y_0,Y_m)$ and a subset $Y'\subseteq Y$ of states, an undirected graph denoted by $I(H,Y')$ can be induced such that $I(H,Y')=(Y',E')$ with $E'=\{(y,y') \mid (y,y'\in Y')$ *and* $(\exists\sigma\in\Sigma)[\zeta(\sigma,y)=y']\}$.

For a partition P of the state set Y of H, we say a cell $p\in P$ is intraconnected if the states it contains are all "graphically connected" in the automaton via only the states within the cell, i.e., the induced graph $I(H,p)$ is connected.

For any given input cell $p\in P$, Procedure *CheckIfConnected* conducts a breadth-first search on the graph $I(H,p)$ and returns *true* if $I(H,p)$ is connected. The procedure starts by adding a random node of $I(H,p)$ to a list *con_nodelist*. The procedure then adds to *con_nodelist* all nodes in p that share an edge with the selected node. This step is repeated for each node that is added to *con_nodelist*. Once after all nodes in *con_nodelist* are considered and there are no further nodes to be added to *con_nodelist*, if *con_nodelist* equals p, then $I(H,p)$ is connected. The procedure has a complexity of $O(|p||\Sigma|)$.

In what follows is the important definition of a state-transparent partition over a full specification automaton.

Definition 5. *Let X be an index set. Given a full specification automaton $H=(Y,\Sigma,\zeta,y_0,Y_m)$ for a DES G, a state-transparent partition of H is a specification-equivalent partition $P=\{Y_x\subseteq Y|x\in X\}$ of H with the following cell intraconnectivity property: $(\forall x\in X)[I(H,Y_x)$ is connected$]$.*

Procedure : $CheckIfConnected(p)$.

Input: A cell p of a partition on the state set of an automaton H;
Output: *true*, if p is intraconnected; *false*, otherwise;

```
1  begin
2      (p,E') = I(H,p);
3      Select any y'' ∈ p;
4      Add y'' to a queue u;
5      con_nodelist := {y''};
6      while u is not empty do
7          Dequeue y from u;
8          foreach (y1,y2) ∈ E' such that y ∈ {y1,y2} do
9              Let y' = {y1,y2} − {y};
10             if y' ∉ con_nodelist then
11                 con_nodelist := con_nodelist ∪ {y'};
12                 Add y' to u;
13             end
14         end
15     end
16     if p = con_nodelist then
17         return true;
18     end
19     else
20         return false;
21     end
22 end
```

According to Definition 5, a state-transparent partition is a specification-equivalent partition containing only intraconnected cells over a full specification automaton. We postulate that these cells can be abstracted to reveal well-defined specification-relevant epochs, in what is defined as a state-transparent specification automaton.

Definition 6. *Given a full specification automaton H for a DES G. Let $\mathcal{P}$ be the set of all state-transparent partitions on H. Then for $P\in\mathcal{P}$, a specification automaton $A=PTran(H,P)$ (for DES G) is said to be a P-transparent specification automaton of H.*

A P-transparent specification automaton formalizes a state-transparent specification. Intuitively, under DES G, a P-transparent specification automaton A is an abstract representation of a full specification H, and each state of A is a specification epoch abstracting a cell of the state-transparent partition P of H. By Theorem 1, A possesses the same restrictiveness on G as H.

We postulate that the most (or maximally) state-transparent specification automaton A should express the control requirement using the least number of specification epochs, i.e., the state-transparent partition P should be of minimal cardinality.

3.2 Problem Statement

The problem of finding a maximally state-transparent

specification automaton A for a full specification automaton H on DES G can now be formally stated.

Problem 1. *Given a full specification automaton H for a DES G. Let $\mathcal{P}$ be the set of all state-transparent partitions on H. Construct a P-transparent specification automaton $A = PTran(H,P)$ of H such that $(\forall P' \in \mathcal{P})|P'| \geq |P|$.*

Theorem 2. *Problem 1 is NP-hard.*

4 PROCEDURES AND SOLUTION ALGORITHM

Since the state-transparency maximization problem is NP-hard, a polynomial-time algorithm that can always return a specification automaton of maximal state-transparency is not expected. In this section, a polynomial-time algorithm that cannot guarantee maximal state-transparency, but can often produce maximal solution in individual cases is proposed.

By Theorem 1 and Definition 6, a state-transparent partition induces a state-transparent specification automaton. A procedure to compute a state-transparent partition is, therefore, essential for computing a state-transparent specification automaton. Such a procedure, : *GetSTPartition*, is presented in Section 4.3. The procedure utilizes another procedure (Section 4.1) to check for state pairability and yet another procedure (Section 4.2) for checking the cell intraconnectivity property. In Section 4.4, a solution algorithm for Problem 1 is then presented.

4.1 Checking of State Pairability

For two states $y,y' \in Y$ of a full specification automaton $H = (Y,\Sigma,\zeta,y_0,Y_m)$, Procedure : *ChkIfPairable* checks if the input states y and y' are pairable. For this, initially, the procedure checks if the input state pair is specification compatible (Condition 1 of Definition 4) by invoking Procedure 13 and thereafter recursively checks if all state pairs reachable by identical strings from the input state pair are also specification compatible (Condition 2 of Definition 4). The procedure returns *true* if all the state pairs considered are specification compatible; and *false*, otherwise. A list *waitlist* (that is initialized to the empty set $\emptyset$ whenever Procedure : *ChkIfPairable* is invoked by Procedure : *GetSTPartition*) is updated with each state pair that is considered. The procedure has a complexity of $O(|Y|^2)$

Procedure : *ChkIfPairable*$(y,y',waitlist)$.

Input: Two states y,y' of input specification automaton H, a list *waitlist* of states pairs considered so far;
Output: *true*, if y and y' are pairable; *false*, otherwise; with *waitlist* updated;

```
1  begin
2    Let W(y) = {y'' | {(y,y''),(y'',y)} ∩ waitlist ≠ ∅};
3    foreach y1 ∈ [y] ∪ ⋃_{y'1 ∈ W(y)} [y'1] do
4      foreach y2 ∈ [y'] ∪ ⋃_{y'2 ∈ W(y')} [y'2] do
5        if {(y1,y2),(y2,y1)} ∩ waitlist =
         ∅ and [y1] ≠ [y2] then
6          if ChkIfInC(y1,y2) = false then
7            return false;
8          end
9          waitlist := waitlist ∪ {(y1,y2)};
10         foreach
           σ ∈ Σ such that ζ(σ,y1)! and ζ(σ,y2)!
           do
11           flag = ChkIfPairable(ζ(σ,y1),
             ζ(σ,y2),waitlist);
12           if flag = false then
13             return false;
14           end
15         end
16       end
17     end
18   end
19   return true;
20 end
```

Procedure : *GetCnctPartition*$(waitlist,P)$.

Input: A list *waitlist* of pairable state pairs and a state-transparent partition P;
Output: A partition P' that augments P such that the states paired in *waitlist* are placed within same cells, whenever P' has only intraconnected cells; P, otherwise;

```
1  begin
2    Let W(y) = {y'' | {(y,y''),(y'',y)} ∩ waitlist ≠ ∅};
3    P' = {[y] ∪ ⋃_{y' ∈ W(y)} [y'] | [y],[y'] ∈ P};
4    foreach p ∈ P' do
5      flag = CheckIfConnected(p);
6      if flag = false then
7        return P;
8      end
9    end
10   return P';
11 end
```

4.2 Checking of Cell Intraconnectivity

For a list *waitlist* of pairable state pairs and a state-transparent partition P, Procedure : *GetCnctPartition* augments P to a partition of states of H such that the state pairs in *waitlist* are placed within same cells of the partition. Each cell of the newly computed partition is checked for intraconnectivity by invoking

Procedure : *CheckIfConnected*. If each of its cells is intraconnected, the newly computed partition is a state-transparent partition and is returned by Procedure : *GetCnctPartition*. Otherwise, the input state-transparent partition is returned without any modification. The procedure has a complexity of $O(|Y|^2)$.

Procedure : *GetSTPartition*(H,G).

Input: A full specification automaton $H = (Y,\Sigma,\zeta,y_0,Y_m)$ on DES G;
Output: A state-transparent partition P of H;

```
1  begin
2      P = {[y] | [y] = {y} for y ∈ Y};
3      foreach y ∈ Y do
4          foreach σ ∈ Σ do
5              if ζ(σ,y)! then
6                  Let y' = ζ(σ,y);
7                  waitlist := ∅;
8                  flag = ChkIfPairable(y,y',waitlist);
9                  if flag = true then
10                     P = GetCnctPartition(waitlist,P);
11                 end
12             end
13         end
14     end
15     return P;
16 end
```

4.3 Computation of a State-transparent Partition

For an input full specification automaton H on DES G, Procedure : *GetSTPartition* computes a state-transparent partition P of H. Initially, the procedure defines P such that each state belongs to a distinct cell. Then, for each pair of states that are connected in H by a transition, the procedure invokes Procedure *ChkIfPairable* to check if the state pair is pairable. Procedure : *ChkIfPairable* returns *true* if its input state pair is pairable, with *waitlist* having a list of state pairs that are reachable by identical strings from the input state pair. Whenever, Procedure : *ChkIfPairable* returns *true*, Procedure : *GetCnctPartition* is invoked. As explained in the preceding subsection, Procedure : *GetCnctPartition* updates P by placing within same cells those states that are paired in *waitlist*, provided the cell intraconnectivity property is satisfied. Otherwise, the state-transparent partition is not updated. The procedure terminates after it has considered all pairs of states in H connected by a transition. The complexity of the procedure is $O(|Y|^3|\Sigma|)$

4.4 Solution Algorithm

For an input full specification automaton H on DES G, Algorithm 1 computes a state-transparent specification automaton A of H. Procedure : *GetSTPartition* computes and returns a state-transparent partition P of H. The algorithm returns automaton $A = PTran(H,P)$, which by Definition 6, is a P-transparent specification automaton of H. The complexity of Algorithm 1 is $O(|Y|^3|\Sigma|)$. The complexity of Algorithm 1 is the sum of complexities of procedures : *GetSTPartition* and : *PTran*, i.e., $O(|Y|^3|\Sigma| + |Y||\Sigma|) \approx O(|Y|^3|\Sigma|)$.

Algorithm 1: Computation of a state-transparent specification automaton.

Input: A full specification automaton H on DES G;
Output: A state-transparent specification automaton A of H;

```
1 begin
2     P = GetSTPartition(H,G);
3     A = PTran(H,P);
4     return A;
5 end
```

Theorem 3. *For a full specification automaton H on DES G, Algorithm 1 returns a state-transparent specification automaton A of H such that $L_m(H) = L_m(A) \cap L_m(G)$.*

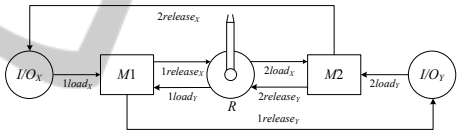


Figure 1: FMS layout.

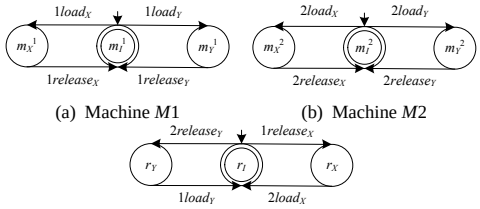
5 ILLUSTRATIVE EXAMPLE

In this section, we illustrate the concept of state-transparency using an example of a flexible manufacturing system (FMS) adapted from (Uzam, 2004). As shown in Fig. 1, the FMS consists of two machines $M1$ and $M2$, and a robot R, all capable of handling parts of two different types X and Y. Parts of type X enter and leave the FMS through an infinite-size buffer I/O_X and those of type Y through an infinite-size buffer I/O_Y. Each machine can process only one part at a time, and the robot can hold only one part at a time. The production cycles for part types X and Y are as follows.

$$X : M1 \rightarrow R \rightarrow M2$$

$$Y : M2 \rightarrow R \rightarrow M1$$

In the former cycle, a part of type X is first loaded into machine $M1$ for initial-stage processing, before robot R takes and loads it into machine $M2$ for final-stage processing. In the latter cycle, a part of type Y is first



(a) Machine $M1$ (b) Machine $M2$

(c) Robot R

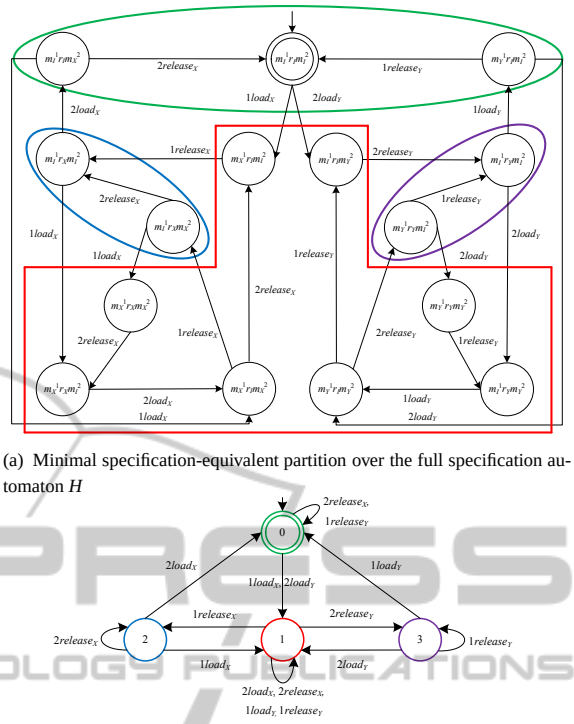
Figure 2: FMS models.

loaded into machine $M2$ for initial-stage processing, before robot R takes and loads into machine $M1$ for final-stage processing.

As depicted in Figs. 2(a) and 2(b), machines M1 and M2 transit among states idling (m_I^1, m_I^2), processing X (m_X^1, m_X^2) and processing Y (m_Y^1, m_Y^2), respectively. The robot R transits among states idling (r_I), holding X (r_X) and holding Y (r_Y), as depicted in Fig. 2(c). Note that in the directed graph representation of an automaton, a state is represented by a node, and a state-to-state transition by a directed edge labeled with an event. The initial state is represented by a node with an entering arrow, and a marker state by a node drawn as a double-concentric circle. For $i \in \{1,2\}$, event i$load_X$ represents the loading of X into Mi for processing and event i$release_X$ represents the release of X by Mi after processing. Similarly, event i$load_Y$ represents the loading of Y into Mi and event i$release_Y$ represents the release of Y by Mi.

The DES G for the FMS is formed by the synchronous product (Cassandras and Lafortune, 2008) of $M1$, $M2$ and R. A control requirement for G is that, once the processing of a part of type X starts, the processing of a part of type Y cannot start until the processing of the part of type X is completed, and vice versa. The full specification automaton H of this requirement for G is shown in both Figs. 3(a) and 4(a). It is derived from some specification automaton B prescribed by a system designer such that $L_m(B) \cap L_m(G) = L_m(H) \subseteq L_m(G)$.

Fig. 3 shows a specification-equivalent partition of minimal cardinality [Fig. 3(a)] over H that induces a minimal-state specification automaton MIN [Fig. 3(b)] of H. It has been pointed out in (Wonham, 2003) that the control requirement prescribed in MIN is not easy to comprehend. The specification-equivalent partition in Fig. 3(a) used to construct MIN is not a state-transparent partition. Notice that the cell that induces state 1 of MIN does not satisfy the cell intraconnectivity property; intuitively, it means the cell contains states in clearly different phases of execution, such as $m_X^1 r_I m_I^2$ that occurs when a part of type X is processed, and $m_I^1 r_I m_Y^2$ that occurs when a part of type Y is processed. Due to this, both the events $1load_X$



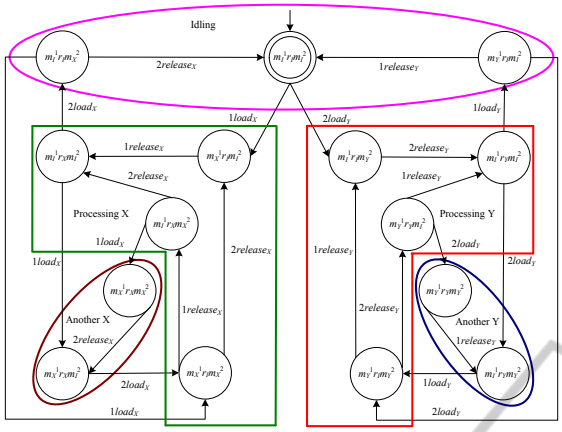
(a) Minimal specification-equivalent partition over the full specification automaton H

(b) The resultant minimal state specification automaton MIN

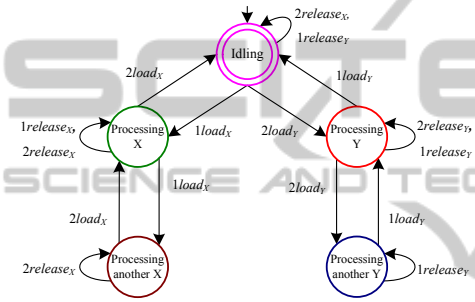
Figure 3: Specification state minimization.

and $2load_Y$, that initiate respectively the processing of parts of type X and Y, lead to state 1 in MIN. As a result, this state does not form a meaningful specification epoch in MIN as it represents either "type-X part processing" or "type-Y part processing". This is ambiguous, in that type-X part processing could be confused with executing event $1load_X$ followed by event $2release_Y$, when the latter event should be executed only following a type-Y part processing.

Algorithm 1 returns a state-transparent specification automaton A (which incidentally is maximal) over H, as shown in Fig. 4(b). The state-transparent partition computed on H is shown in Fig. 4(a). Each state of A abstracts a cell of the computed state-transparent partition of H. As will be explained, each such cell has states in the same phase of execution. Note in Fig. 4(a) that when event $1load_X$ brings the system from the initial state $m_I^1 r_I m_I^2$ of H to state $m_X^1 r_I m_I^2$, it means that the system has started processing a part of type X. To help better understand specification H, in conformance with Definition 5, our algorithm aggregates the state $m_X^1 r_I m_I^2$ with other states in H, forming a cell of pairable and intraconnected states, to distill a well-defined specification epoch meaningfully interpreted as "Processing X". Other specification epochs that result can also be interpreted meaningfully. Effectively, the output state-transparent



(a) Minimal state-transparent partition over the full specification automaton H



(b) The resultant maximally state-transparent specification automaton A

Figure 4: Specification state-transparency maximization.

specification automaton A 'summarizes' state information in H as specification epochs, highlighting only the essential restrictions on the DES. The specification automaton A is easier to comprehend, since the temporal essence of the specification - that once the processing of a part of type X starts, the processing of a part of type Y cannot start until the processing of the part of type X is completed, and vice versa - is clearly depicted in A.

6 RELATED WORK

Within the DES control community, design comprehensibility is receiving increasing attention and recognition. Current efforts focus primarily on clarifying supervisor design, whereas our work focuses on clarifying given specifications:

- In the former line of research, state reduction of synthesized supervisors has been proposed as a technique that might render the control actions due to a specification more readily comprehensible to designers (e.g., (Su and Wonham, 2004)). Over a controllable specification automaton generating a sublanguage of the DES, the supervisor reduction algorithm in (Su and Wonham, 2004) constructs a control congruence, which essentially is a partition of states of the input automaton, and uses it to compute a state-reduced supervisor automaton. By "virtually setting" all events as controllable, one could adapt the algorithm to achieve state size reduction for a full specification automaton. In some cases a state-reduced automaton constructed may be easier to understand, but the output of such a state reduction procedure (Su and Wonham, 2004) is usually not appealing to human engineers (Wonham, 2003). The reason is that the output often also abstracts away those a priori DES constraints essential for interpreting the specification. The fact of the matter is that the main aim in (Su and Wonham, 2004) is to construct a supervisor automaton to achieve economy of implementation, using the least possible number of control states by removing constraints already enforced by the DES. Also related is the research (Miremadi et al., 2008) that generates and attaches propositional formulae called guards to a given supervisor to aid in design clarity.
- In this paper, along our latter and complementary line of research, state-transparency is introduced and developed. Our aim is to re-express a specification automaton using the minimum number of specification relevant states manifesting as specification epochs, so as to aid in clarifying if it specifies the intended requirement.

Finally, the concept of state-transparency provides a different language perspective to clarifying specifications from that of event-transparency introduced in (Pham et al., 2010). This is manifested by two main differences as follows. First, in the case of maximal state-transparency, a specification is recast using the least possible number of specification relevant states, with each state representing some unique specification epoch. In the case of maximal event-transparency, the precedence ordering among a minimal set of events deemed relevant to the specification is highlighted. Second, states in a state-transparent specification automaton correspond to a state partition of a full specification, but are not so for an event-transparent specification automaton. The event-transparency maximization algorithm developed in (Pham et al., 2010) allows unrestricted state splitting, causing in some cases, an event that is not permitted and therefore undefined in a state of a given input full specification automaton to be undefined at several states of the output event-transparent automaton. This could result in a cluttered state structure

of the automaton. In other words, event-transparency tends to facilitate clarity in event ordering information, whereas state-transparency facilitates clarity in temporal state information.

7 CONCLUSIONS

The study of informatics for the clarification of discrete-event control specifications in automata is motivated, and the informatics notion of state-transparency is developed. A state-transparent specification automaton is formally defined and a solution algorithm to construct a specification automaton of significant state-transparency is proposed. An illustrative example is given to highlight how such an automaton can enhance the intuitive understanding of a specification in automata.

To enhance specification comprehensibility, our future work will incorporate temporal logic to extend our state-transparency specification framework to a dual-language framework. Temporal logic is a natural language readable and expressive formalism for writing specifications (Manna and Pnueli, 1992). An algorithm is proposed in (Seow, 2007) to translate a state-based (finitary) temporal logic specification for a DES G to a full specification automaton H. An algorithm that directly translates such a temporal logic specification to a state-transparent specification automaton A for which $L_m(H) = L_m(A) \cap L_m(G)$ should be of great practical utility to designers prescribing specification for DES's. Such a dual-language framework can give designers added confidence in ascertaining whether a prescribed specification is as intended by rendering the control requirement of the translated specification automaton easier to understand from the integrated perspective of automaton state-transparency clarified with readable temporal state information.

ACKNOWLEDGEMENTS

This research is funded by the Singapore Ministry of Education, under NTU-AcRF Tier 1 Grant No: RG65/07.

REFERENCES

Bredereke, J. and Lankenau, A. (2005). Safety-relevant mode confusions–modelling and reducing them. *Reliability Engineering & System Safety*, 88(3):229 – 245.

Cao, X.-R. and Ho, Y.-C. (1990). Models of discrete event dynamic systems. *IEEE Control Systems Magazine*, 10(4):69 – 76.

Cassandras, C. G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer.

Cheriaux, F., Picci, L., Provost, J., and Faure, J.-M. (2010). Conformance test of logic controllers of critical systems from industrial specifications. In *Proceedings of the European Conference on Safety and Reliability*.

Diestel, R. (2006). *Graph theory*. Springer.

Du, Y. and Wang, S. H. (1988). Translation of output constraint into event constraint in the control of discrete event systems. In *Proceedings of the IEEE Conference on Decision and Control*, volume 2, pages 1119 –1124.

Faraut, G., Piétrac, L., and Niel, E. (2011). Process tracking by equivalent states in modal supervisory control. In *Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation*, pages 1 – 8.

Feng, L., M.Wonham, W., and Thiagarajan (2007). Designing communicating transaction processes by supervisory control theory. *Formal Methods in System Design*, 30:117–141.

Grigorov, L., Butler, B., Cury, J., and Rudie, K. (2010). Conceptual design of discrete-event systems using templates. *Discrete Event Dynamic Systems*, pages 1–47.

Grigorov, L., Butler, B., Cury, J., and Rudie, K. (2011). Conceptual design of discrete-event systems using templates. *Discrete Event Dynamic Systems*, 21:257–303.

Hinze, A., Malik, P., and Malik, R. (2006). Interaction design for a mobile context-aware system using discrete event modelling. In *Proceedings of the Australasian Computer Science Conference*, pages 257–266.

Košecká, J. and Bajcsy, R. (1994). Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12(3-4):187–198.

Magnusson, P., Sundström, N., Bengtsson, K., Lennartson, B., Falkman, P., and Fabian, M. (2011). Planning transport sequences for flexible manufacturing systems. In *Proceedings of the IFAC World Congress*.

Manna, Z. and Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems*. Springer-Verlag.

Miremadi, S., Akesson, K., and Lennartson, B. (2008). Extraction and representation of a supervisor using guards in extended finite automata. In *International Workshop on Discrete Event Systems*, pages 193 –199.

Ou, Y.-C. and Hu, J. (2000). A modified method for supervisor specification and synthesis of a class of discrete event systems. *Asian Journal of Control*, 2(4):263–273.

Pham, M. T., Dhananjayan, A., and Seow, K. T. (2010). On the transparency of automata as discrete-event control specifications. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1474–1479.

Ramadge, P. J. and Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230.

Rasmussen, J. (1986). *Information processing and human-machine interaction: an approach to cognitive engineering*. North-Holland.

Ricker, S. L., Sarkar, N., and Rudie, K. (1996). A discrete event systems approach to modeling dextrous manipulation. *Robotica*, 14(5):515–525.

Seow, K. T. (2007). Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata. *IEEE Transactions on Automation Science and Engineering*, 4(3):451–464.

Seow, K. T. and Pasquier, M. (2004). Supervising passenger land-transport systems. *IEEE Transactions on Intelligent Transportation Systems*, 5(3):165–176.

Su, R. and Wonham, W. M. (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 14(1):31–53.

Uzam, M. (2004). An optimal deadlock prevention policy for flexible manufacturing systems using petri net models with resources and the theory of regions. *The International Journal of Advanced Manufacturing Technology*, 19(3):192–208.

Wonham, W. M. (2003). Supervisory control theory: Models and methods. In *Proc. ATPN - Workshop on Discrete Event Systems Control, International Conference on Application Theory of Petri Nets*, pages 1–14.