Integrating Temporal Logic as a State-Based Specification Language for Discrete-Event Control Design in Finite Automata

Kiam Tian Seow

Abstract—This paper presents and analyzes a correct and complete translation algorithm that converts a class of propositional linear time temporal logic (PTL) formulae to deterministic finite (-trace) automata. The translation algorithm is proposed as a specification interface for finitary control design of discrete-event systems (DES’s). While there has been a lot of computer science research that connects PTL formulae to \( \omega \)-automata, there is relatively little prior work that translates state-based PTL formulae, in the context of a finite-state DES model, to event-based finite automata - the formalism on which well-established control synthesis methods exist. The proposed translation allows control requirements to be more easily described and understood in temporal logic, widely recognized as a useful specification language for its intuitively appealing operators that provide the natural language expressiveness and readability needed to express and explain these requirements. Adding such a translation interface could therefore effectively combine specifiability and readability in temporal logic with prescriptiveness and computability in finite automata. The former temporal logic features support specification while the latter automata features support the prescription of DES dynamics and algorithmic computations. A practical implementation of the interface has been developed, providing an enabling technology for writing readable control specifications in PTL that it translates for discrete-event control synthesis in deterministic finite automata. Two application examples illustrate the use of the proposed temporal logic interface. Practical implications of the complexity of the translation algorithm are discussed.

Note to Practitioners—Current software technology for finitary control design of discrete-event systems (DES’s) requires control requirements to be specified in event-based finite (-trace) automata [2], [3]. This paper is motivated by the fact that specification in automata can be a non-trivial problem, since a system designer may not always know for sure if a specification automaton captures the intended control requirement correctly and completely, nor can another designer readily interpret the intended control meaning of the automaton. As propositional linear time temporal logic (PTL) is syntactically closer to natural language, it suggests specifying a class of control requirements as state-based PTL formulae to render them more readable and easily understood, and proposes a correct and complete algorithm to convert these formulae into event-based finite automata. NantA, a practical implementation of the algorithm, has been developed as an interface to enable writing control requirements in PTL that it translates into finite automata for direct use with TCT [3], a freely available, finite automata-based control design software for DES’s. The interface would help reduce design errors and costs associated with incorrect, unnecessarily restrictive or misinterpreted specifications. In future research, to support larger control systems design found in many real-world manufacturing and automation systems (e.g., [4]), we will attempt to make NantA computationally more efficient, and generate more compact specification automata.

Index Terms—Discrete-Event Systems, Propositional Linear Time Temporal Logic, Finite Automata, Automation, Supervisory Control

I. INTRODUCTION

Supervisory control theory, initiated by Ramadge and Wonham [5], addresses the problem of synthesizing controllers for discrete-event systems (DES’s) by focussing on the high-level characterizations of different existence conditions of controllers, as well as the associated algorithms for computing them from formalized control requirements. The algorithmic considerations have included the different control architectures and/or the DES inherent mathematical structures, mitigating the complexity of such control computations.

Although steady progress continues to be made in the study of supervisory control (e.g., see recent proceedings [6], [7], [8]), specifying the control requirements for a DES remains a non-trivial problem that must be resolved if controller synthesis is to become a formal method that is widely used. The natural prerequisites of a language for specifying control requirements are natural language expressiveness and readability. The former implies that it can specify complex requirements more readily, while the latter implies that statements written in it can be easily understood.

Most synthesis methods for supervisory control require the control requirements and DES dynamics to be represented by finite (-trace) automata [9]. In DES formulation [5], a finite automaton generates finite traces or strings of events. The finite automata DES framework [2] has been a pervasive formalism in supervisory control research due to its rudimentary, prescriptive and computation-oriented features. The DES dynamics can be prescribed explicitly in automata to show the events and the states in which they can occur. However, a control requirement (on a DES) is often more easily written and readily understood - correctly - from a descriptive or declarative rather than a prescriptive viewpoint. In general, writing a specification in the prescriptive language of automata might not always be straightforward. This is evident in many earlier applications of the finite automata DES framework (e.g., in automated manufacturing [4], [10], task-level robotics [11], [12], intelligent service transportation [13], and computer
and communication networks [14]), where a system designer is often confronted by the following specification problem: how do we know that a specification in automata does indeed capture the intended control requirement?

Specifications of control requirements, in our opinion, can be correctly written down easily in temporal logic, since it offers simple syntax and semantics for descriptively writing formulae that are paraphrastic in natural language. In fact, temporal logic has long been recognized as an expressive and readable language for specifying and verifying the properties of reactive systems [15], including DES’s (e.g., [16], [17], [18], [19], [20]).

Nonetheless, the dominance of automata in supervisory control theory has led to extensive development of increasingly powerful automata-theoretic synthesis tools (e.g., TCT [3]). These tools require both the control requirements and DES to be modelled by finite automata. So, to incorporate the specification merits of temporal logic in order to better exploit these existing tools, the idea advocated in this paper is to add an interface that accepts temporal logic specifications of a class of control requirements for a given DES, and automatically translates them into finite automata. The specifications considered are expressed in terms of state information of the DES, hence termed state-based. The proposed translation proceeds in two steps: first, convert a state-based temporal logic formula into an $S$-automaton, one whose transitions are associated with propositional state formulae, and second, convert the $S$-automaton into an $E$-automaton, a ‘conventional’ finite automaton [5] whose transitions are associated with symbols of events of a given DES modelled by a transition system. A transition system is an $E$-automaton incorporated with state information. The DES model, $E$-automaton and $S$-automaton will be formally defined later in Section III. The proposed translation is contextual in that the finite automaton obtained selects a sublanguage of (the marked language generated by) the DES model, the context of interest. By this selection, the sublanguage is also said to be marked by the $S$-automaton.

In contrast, most current algorithms [21] in computer science are concerned with translating a temporal logic formula to an $\omega$-automaton accepting an infinite language, not a finite one. Of these, including the few prior work [22] translating a temporal logic formula to a finite automaton, none is like the proposed translation framework, cast in a DES formalism that 1) distinguishes event information as model transitions from state information, and 2) contextually connects state-based temporal logic to (event-based) $E$-automata for finitary control synthesis of a class of temporal-finitary control requirements. Thus, finite or infinite, no related algorithm in computer science is directly suited as an interface to existing finitary control design tools. Importantly, in our opinion, the proposed interface would help reduce design errors and costs associated with incorrect, unnecessarily restrictive or misinterpreted specifications for finitary control systems design, covering many manufacturing and automation systems in general.

Perhaps, the DES research efforts most closely related to the proposed translation are those of Du and Wang [23], Lin [19] and Sanchez [24]:

Du and Wang propose that a control requirement be specified directly by an automaton, with transitions defined on state symbols instead of event symbols of a given DES; their translation algorithm converts this automaton into an $E$-automaton that marks a sublanguage of the DES model. Their proposed specification automaton differs from an $S$-automaton in two aspects. Firstly, its transitions are defined on enumerated DES states instead of propositional state formulae that the latter’s are associated with. Secondly, all its states are apparently assumed to be marked unlike the latter’s, thereby limiting the specifications to an equivalent of prefixed-closed languages.

Lin proposes to translate or synthesize an $E$-automaton from a given temporal logic formula. However, the adapted temporal logic framework only allows a formula to be expressed over the events, and not the state information, of a DES. This apparently limits the use of temporal logic features, namely, its natural language expressiveness and readability, as well as a richer description of the DES.

Sanchez develops a design method that translates a small class of temporal logic formulae into what he calls an $\alpha$-machine, namely, an automaton incorporated or ‘labelled’ with state information. It is similar to the transition model used herein, differing only in the way the state variables are defined. However, this class is rather restrictive as its formulae are expressed in certain predefined formats according to which a different translation subprocedure is designed. As a result, Sanchez’s design procedures offer a more rigid specification interface than the generic translation algorithm proposed in this paper.

Also related is the infinitary control synthesis research of Barbeau et al [20], and Ziller and Schneider [25]:

With control requirements specified in metric temporal logic, Barbeau et al propose a method which composes a useful event-based automaton that realizes a controller satisfying these requirements, but which is not necessarily maximally permissive. Their method can be viewed as building the translation of temporal logic to automata directly into the control synthesis process. This differs from our approach which treats the translation and control synthesis problems as distinctly separate, and is concerned with finitary synthesis. Similar to ours, the synthesis method is contextual, i.e., it is done with respect to (the transitions of) a DES. However, it synthesizes controllers in the context of uncontrollable events characterized by nondeterminism. It is therefore not appropriate for the supervisory control framework [5] considered in this paper, since control problems such as controllability do not naturally arise in their setting.

Ziller and Schneider propose that a control requirement be specified jointly in automata and temporal logic. The part in automata specifies the usual desired marked language, while that in temporal logic asserts a specific liveness condition, generalizing the nonblocking condition as formulated in finitary supervisory control [5]. Their method can be viewed as translating the temporal logic liveness expression, along with the system marked states, into a system of $\mu$-calculus equations from which a set of states is computed in the initial part of the control synthesis process. The process is completed by way of restricting to these states, the cartesian product of
an $\mathcal{E}$-automaton (specifying a desired marked language) and a DES automaton. Our translation interface can be used to directly produce this cartesian product, where the input to the translation interface is a control requirement in temporal logic corresponding to the desired marked language.

In another related work, Jiang and Kumar [26] claim that it is possible to first convert a propositional linear time temporal logic formula to a nondeterministic Büchi automaton [27], which in turn is translated into a deterministic Rabin automaton [28], and finally fully synchronized with a DES automaton to yield a deterministic Rabin automaton. However, a procedure like that in [27] can only convert a temporal logic formula that is expressed over the events, and not the state information of a DES, to an event-based Büchi automaton. In any case, this approach only supports infinitary control synthesis. Many practical DES's (e.g., [4], [10], [11], [12], [13], [14]) can be conveniently modelled by finite automata (generating marked finite languages or finite event-based behaviours), not $\omega$-automata. So toward the design automation and engineering of discrete-event control systems, we believe it would be practically very useful to equip the well-established (finite) $\mathcal{E}$-automata control framework [2] with an interface that can automatically translate a class of readable (state-based) temporal logic formulae to $\mathcal{E}$-automata. A theoretical analysis and a practical development for this DES-contextual interface are the main contributions of this paper.

The rest of the paper is organized as follows. Section II presents the preliminaries on languages and automata. Section III covers the background on $\mathcal{E}$-automata for DES modelling, the syntax and semantics of propositional temporal logic proposed as a control specification language, as well as on $\mathcal{S}$-automata that can furnish an intermediate representation for algorithmically converting a class of temporal logic formulae to $\mathcal{E}$-automata. Section IV presents the translation algorithm, and establishes the finitary correctness and completeness of the algorithm in generating deterministic automata. Section V examines the implications of the algorithm as a finitary specification interface, along with discussions to distinguish it from similar work. Section VI presents a practical implementation of the algorithm as a specification interface for finitary control synthesis, and illustrates the usefulness of the developed temporal logic interface with two application examples. Section VII concludes the paper.

II. LANGUAGES & DETERMINISTIC AUTOMATA

Let $T$ be a non-empty finite set of symbols. The Kleene closure $T^*$ denotes the set of all finite strings composed of elements in $T$, including the empty trace $\varepsilon$. A subset $L \subseteq T^*$ is called a language over $T$; $2^{T^*}$ is the power set of $T^*$, and denotes the set of all languages over $T^*$. If $s, s', s'' \in T^*$ with $s's'' = s$, then we call $s' T \subseteq T^*$ a prefix of $s \in T^*$. The closure $\overline{L}$ of $L$ is the language consisting of all the prefixes of strings in $L$; if $L = \emptyset$, then $\overline{L} = \emptyset$, and if $L \neq \emptyset$, then $\varepsilon \in \overline{L}$. $L$ is closed if $L = \overline{L}$.

Let finite (-trace) automaton $A$ [9] be a generator (due primarily to [5]) given by a 5-tuple:

$$A \triangleq (Y, T, \delta, y_0, y_m)$$

where $Y$ denotes the finite state set, $T$ denotes the finite set of symbols, $\delta : T \times Y \to Y$ denotes the partial state transition function, $y_0 \in Y$ is the initial state and $Y_m \subseteq Y$ is the set of marked states. $\delta$ is ‘partial’ in the sense that for each fixed $y \in Y$, $\delta(t, y)$ is defined (denoted $\delta(t, y)$) only for some subset of $T$, denoted $T(y) \subseteq T$, i.e., each element $t \in T(y)$ is defined at $y$.

$\delta$ can be recursively extended to $\delta : T^* \times Y \to Y$ as follows. For $t \in T$, $s \in T^*$,

$$\delta(\varepsilon, y) = y \quad \delta(st, y) = \delta(t, \delta(s, y))$$

provided $\delta(t, y)$ and $\delta(t, \delta(s, y))$. For any $s' \in T^*$, $y' \in Y$, $\delta(s', y')$ denotes that $\delta(s', y')$ is defined. Whenever we write $\delta(s, y_0) \in Y$ or $\delta(s, y_0) \in Y_m$, we imply $\delta(s, y_0)$.

Explicitly, $A$ is a directed graph with node set $Y$ and an edge $y \xrightarrow{t} y'$ labelled $t$ for each triple $(t, y, y')$ such that $\delta(t, y) = y'$. Such an edge is called a $t$-transition.

**Definition 1:** A finite automaton $A$ is said to be deterministic (as induced by $\delta$) in the sense that for any two identical symbols $t_1, t_2 \in T$ (i.e., $t_1 = t_2$) defined at a given state $y \in Y$, $\delta(t_1, y) = \delta(t_2, y)$.

In the directed graph of $A$, the initial state $y_0 \in Y$ is labelled with an entering arrow, while a marked state $y_m \in Y_m$ is labelled as a darkened node or a double-concentric circle.

The strings generated by automaton $A$ are characterized by two languages: $L(A)$, the closed language generated by $A$, and $L_m(A)$, the language marked by $A$. More formally, we have:

$$L(A) = \{s \in T^* : \delta(s, y_0) \in Y\}$$

$$L_m(A) = \{s \in L(A) : \delta(s, y_0) \in Y_m\}$$

$L_m(A) \subseteq L(A)$ is the subset of strings in $L(A)$ which end in any of the states in $Y_m$ and is a distinguished subset that is meant to represent the satisfaction of some property that model $A$ is intended to represent.

A string $s \in T^*$ is said to be accepted or marked by a finite automaton $A$ if $\delta(s, y_0) \in Y_m$. A language $L \subseteq T^*$ is termed regular if and only if every string $s \in L$ can be marked or accepted by $A$ [9]. A state $y \in Y$ is reachable in $A$ if there exists a string $s \in T^*$ such that $\delta(s, y_0) = y$; and coreachable in $A$ if there exists a string $s \in T^*$ such that $\delta(s, y) \in Y_m$. Generator $A$ is said to be trim if every string in $Y$ is both reachable and coreachable. Trim ($A$) defines a trim version of $A$, i.e., it is automaton $A$ but with all states $\delta(t, y) \in Y$ that are not reachable or co-reachable, and the respective transitions $t \in T$ removed. Unless otherwise stated, an arbitrary finite automaton $A$ referred to henceforth is assumed to be trim.

III. BACKGROUND

A. DISCRETE-EVENT SYSTEM & FINITE AUTOMATON MODEL

Formally, let transition system

$$G \triangleq (II, G)$$

(4)
be a DES model such that $G \overset{\text{def}}{=} (Q, \Sigma, \delta_e, q_0, Q_m)$ is a finite automaton (1), where

1) $\Pi$ denotes the finite set of propositional state symbols of $G$, i.e., for each (non-temporal) variable $u \in \Pi$, the domain $\text{Range}(u)$ over which it ranges is \{true, false\}.

2) $Q$ denotes the finite state set, defined to be the cartesian product of the ranges of the variables in $\Pi$, i.e., $Q \overset{\text{def}}{=} \prod_{u \in \Pi} \text{Range}(u)$, such that for any state $q \in Q$, we denote the value of $u \in \Pi$ assigned by $q$ to be $q[u]$ over its domain. Syntactically, the formula $\prod_{u \in \Pi} (u = q[u])$ uniquely characterizes the state information in $q \in Q$.

3) $\Sigma$ denotes the finite event set so that model $G$ can be interpreted as an event-driven system that starts from $q_0 \in Q$ and executes events $\sigma \in \Sigma$ defined at each state $q \in Q$, thus generating a string of events.

4) $L_m(G) \subseteq L(G)$ is meant to represent completed ‘tasks’ (or strings of tasks) carried out by the physical process that $G$ is intended to model.

Remark 1: $G$ is a DES model with both state and event information incorporated. For notational convenience, unless necessary, we simply use symbol $G$ to refer to any deterministic DES model, with $\Pi$ implicitly assumed. An event-generating automaton, $G$ is referred to as an $E$-automaton - one that is said to be event-based.

B. DES as a Synchronous Product of Automata

A complex DES $G$ is usually modelled as a system of several interacting component (discrete-event) processes, each modelled by an $E$-automaton $G_i$. Composing a complex DES $G$ from its component processes $G_i$’s requires the synchronous operator $\parallel$ [2]. The synchronous operator for regular languages with state information is presented in the appendix.

C. Control Specification & Temporal Logic

The temporal logic adopted for specifying control requirements is propositional linear time temporal logic PTL [15].

1) Syntax: PTL formulae are constructed from a set of propositional symbols $\mathcal{P}$; the Boolean connectives $\land$ (and) and $\neg$ (not); and the temporal operators $\bigcirc$ (next), $\square$ (always) and $U$ (until).

The formula formation rules are:

1) every propositional symbol $p \in \mathcal{P}$ is a formula;

2) if $\omega, \omega_1$ and $\omega_2$ are formulae, so are $\neg \omega, \omega_1 \land \omega_2, \bigcirc \omega, \square \omega$ and $\omega_1 U \omega_2$.

In addition, the following equivalences ($\equiv$) are used: about which related connectives $\lor$ (or), $\rightarrow$ (implies), and operator $\diamond$ (eventually) are defined.

\[
\begin{align*}
\omega_1 \lor \omega_2 & \equiv \neg \neg \omega_1 \land \neg \omega_2, \\
\omega_1 \rightarrow \omega_2 & \equiv \neg \omega_1 \lor \omega_2, \\
\diamond \omega & \equiv \text{true} U \omega.
\end{align*}
\]

The language also includes propositional constants true (validity) and false (inconsistency) defined respectively by the equivalences

\[
\begin{align*}
\text{true} & \equiv \neg \omega \lor \omega, \\
\text{false} & \equiv \neg \omega \land \omega.
\end{align*}
\]

A PTL formula $\omega$ is satisfiable provided it is not false.

2) Semantics: A string over an event set $\Sigma$ can be viewed as a mapping

\[ e : \{0, \ldots, i, \ldots, \} \rightarrow \Sigma \] (5)

such that

\[ e \overset{\text{def}}{=} e(0)e(1)\ldots e(i) \ldots, \ e(i) \in \Sigma. \]

Then, in the context of DES $G = [\Pi, \Sigma]$ , $e$ is an event string generated by $E$-automaton $G$ provided there exists a ‘labelling’ of the string by states

\[ \rho : \{0, \ldots, i, \ldots, \} \rightarrow Q \] (6)

such that

\[ \rho \overset{\text{def}}{=} \rho(0)\rho(1)\ldots \rho(i) \ldots, \ \rho(i) \in Q, \]

for which

1) $\rho(0) = q_0$ (the initial label is the initial state);

2) $\rho(i + 1) = \delta_e(e(i), \rho(i)).$

Such a labelling $\rho$ is an arbitrary state trajectory of $G$. The $k$-suffix of $\rho$ is

\[ \rho(k)\rho(k + 1)\ldots \rho(i) \ldots, \ k \geq 0, \]

and denoted by $\rho(k)$. Note that $\rho(0) = \rho$. The $k$-prefix of $\rho$ is

\[ \rho(0)\rho(1)\ldots \rho(k), \ k \geq 0, \]

a finite state-trajectory and denoted by $\rho_k$. It is said to be marked if $\rho(k) \in Q_m$.

PTL formulae expressed over a given DES $G$ are interpreted over models of the form $(\rho, \pi)$, where

\[ \pi : \{0, \ldots, k, \ldots, \} \times \mathcal{P} \rightarrow \{\text{true}, \text{false}\} \] (7)

is a binary function that evaluates a propositional symbol $p$ in $k$-th state $\rho(k)$, i.e.,

\[ \pi(k, p) = \begin{cases} 
\text{true} & \text{if } p \text{ holds in } \rho(k), \\
\text{false} & \text{otherwise.}
\end{cases} \]

The model $(\rho, \pi)$ is understood, and so we simply write $\models_{\rho(k)} p$ if a propositional symbol $p$ holds (i.e., is true) in state $\rho(k)$.

We write $\models_{\rho(k)} \omega$ if the $k$-suffix of an arbitrary trajectory $\rho$ makes true or satisfies formula $\omega$. It should be clear that the evaluations of a propositional symbol $p$ over a $k$-suffix $\rho(k)$ and in $k$-th state $\rho(k)$ are logically equivalent, i.e., $\models_{\rho(k)} p \iff \models_{\rho(k)} p$.

In addition to the standard rules for Boolean connectives, PTL uses the following rules for temporal operators that establish the satisfaction of a suffix trajectory over a PTL formula: For a state index $k$, $k \geq 0$, a propositional symbol $p$, formulae $\omega, \omega_1$ and $\omega_2$,

- $\models_{\rho(k)} p$ iff $\pi(k, p) = \text{true}$;
- $\models_{\rho(k)} \bigcirc \omega$ iff $\models_{\rho(k+1)} \omega$;
The rules above for \( k = 0 \) define the semantics of PTL over infinite state-trajectories. As mentioned in [29], the semantics of PTL can be restricted to finite state-trajectories as follows:

\[ \vDash^0 \varphi \iff \text{for all } j \geq k, \:vDash^j \varphi; \]
\[ \vDash^0 \varphi \land \omega \iff \text{there exists a } j \geq k \text{ such that } \vDash^j \varphi \land \omega; \]
\[ \vDash^0 \varphi_1 \land \omega_2 \iff \text{there is a } j, j \geq k, \text{ such that } \vDash^j \varphi_1 \land \omega_2 \text{ and for all } i, k \leq i < j, \vDash^i \varphi_1 \land \omega_1. \]

The rules above for \( k = 0 \) define the semantics of PTL over infinite state-trajectories. As mentioned in [29], the semantics of PTL can be restricted to finite state-trajectories as follows: \( \vDash^0 \varphi \iff \varphi^\infty \), where \( \varphi^\infty \) is an infinite extension of \( \rho_n \) by an empty trace \( \varepsilon \) at state \( \rho(n) \in Q \), such that \( \delta_\varepsilon(\varepsilon, \rho(n)) = \rho(n). \)

3) Expansion Rules: PTL uses the following rules to expand temporal formulae accordingly (see right-hand side of these rules).

\[ \bigcirc \omega \equiv \omega \land \bigcirc \omega; \]
\[ \omega \equiv \omega \lor (\omega \land \bigcirc \omega); \]
\[ \omega_1 \land \omega_2 \equiv \omega_2 \lor (\neg \omega_2 \land \omega_1 \land \bigcirc (\omega_1 \land \omega_2)); \]
\[ \neg \omega \equiv \neg \omega \lor \omega \land \bigcirc \neg \omega; \]
\[ \neg \omega \equiv \omega \lor \neg \omega \land \bigcirc \omega; \]
\[ \neg (\omega_1 \land \omega_2) \equiv (\neg \omega_1 \land \neg \omega_2) \lor (\omega_1 \land \neg \omega_2 \land \bigcirc \neg (\omega_1 \land \omega_2)). \]

It is based on these expansion rules that one could construct an \( S \)-automaton from a PTL formula, as will be presented in Section IV-A.

4) Finitary Control Requirements & Normal Form: Marked sublanguages are clearly the specification of interest in finitary nonblocking control [2]. For such control, our intention of writing a control requirement in PTL is to ‘select’ a desired marked sublanguage of a DES model \( G \) (4). Essentially, this means we need to specify a PTL formula \( \omega \) that ‘selects’ marked finite state-trajectories \( \rho_m \) for which \( \vDash^m \omega \) (read: \( \rho_m \) satisfies \( \omega \)). However, \( \vDash^m \omega \) must be marked semantics consistent with the infinitary semantics of PTL. To elaborate, let \( Q^*Q_m \) denote the set of finite state-strings ending in a marked state (i.e., marked finite state-trajectories of DES \( G \)), and \( (Q^*Q_m)^\infty \), the set of infinite state-trajectories said to be in the limit (of \( Q^*Q_m \)). Then to be consistent, all finite trajectories \( \rho_m \in Q^*Q_m \), with \( \vDash^m \omega \), need to be the infinitely many prefixes of some state trajectories \( \rho \) in the limit for which \( \vDash^\rho \omega \). This is found to be so provided \( \omega \) is a response formula [15]. It means a finitary control requirement that we specify in PTL is necessarily a response formula.

A PTL formula is said to be in disjunctive form if it is expressed by a disjunction of a finite number of subformulae. Then the PTL formula on the left-hand side of each expansion rule above is said to be in (finitary) normal form, if every constituent formula (i.e., \( \omega_1 \) or \( \omega_2 \) within the scope of an outermost temporal operator), when expressed in disjunctive form, does not contain a persistence modality. This modality is \( \bigcirc \) for the first three expansion rules, and \( \bigcirc \bigcirc \) for the last three. Additionally, if the outermost operator is \( \bigcirc \) or \( \neg \bigcirc \), the PTL formula is said to be in invariance normal form.

The normal form is a conveniently recognizable structure of a response PTL formula. At this juncture, the reader may skip ahead to Section VI-A.1 for a glance at a useful class (14) in this form.

With no essential loss of practical expressiveness\(^1\), we consider a class of PTL formulae representing (finitary) control requirements in conjunctive normal form (CNF). A CNF is defined as a conjunction of \( k \) subformulae, \( k \geq 1 \), where each subformula is in normal form. The class of response formulae is closed under conjunction [15]; thus a CNF is (still) a response PTL formula. A CNF can be said to be in invariance normal form if all its subformulae are in invariance normal form, since for two arbitrary PTL formulae \( \omega_1 \) and \( \omega_2 \), \( \Box (\omega_1 \land \omega_2) \equiv (\Box \omega_1) \land (\Box \omega_2) \) [15].

D. Control Specification & \( S \)-Automaton

Control requirements can also be specified by automata with their transitions defined on state formulae (i.e., formulae without containing any temporal operators). Such state-based specification models can represent subsets of the state trajectories of DES \( G \).

Formally, let

\[ C \equiv [C, F] \]

be a state-based specification model such that \( C \equiv (X, V, \delta_s, x_0, X_m) \) is a finite automaton (1), where \( V \) denotes a finite set of transition symbols labelled by a finite set \( S \) of satisfiable state formulae under a map \( F \), i.e., \( F : V \rightarrow S \); and has the following property:

Property 1: For an arbitrary \( x \in X \) and two non-identical \( v_1, v_2 \in V \), \( \delta_s(v_1, x) \) and \( \delta_s(v_2, x) \) implies \( F(v_1) \land F(v_2) \equiv \text{false}. \]

Remark 2: For notational convenience, henceforth, unless necessary, we will simply use automaton \( C \) to refer to a state-based specification model with the map \( F \) implicitly assumed. A state-generating automaton, \( C \) is referred to as an \( S \)-automaton.

IV. \( G \)-Based Translation & Analysis

For a PTL formula \( \omega \) on a DES \( G \), the translation operator

\[ \otimes \overset{\text{def}}{=} \times_G \circ \otimes_\Pi \]

converts the PTL formula \( \omega \) to an \( \varepsilon \)-automaton, with operation \( \otimes_\Pi \) translating this formula \( \omega \) (expressed in terms of variables in \( \Pi \)) to an \( S \)-automaton, and operation \( \times_G \) translating the \( S \)-automaton to an \( \varepsilon \)-automaton (in the event language space of \( G \)). This is depicted as a commutative diagram in Fig. 1.

![Fig. 1. Commutative diagram illustrating \( G \)-based translation via operations \( \otimes_\Pi \) and \( \times_G \).](image)

The following two sections present algorithmic procedures that implement \( \otimes_\Pi \) and \( \times_G \).

\(^1\)This is in the sense that following the usual approach [15, p. 295], requirement specification is written as a conjunction of several PTL subformulae; and in practice, each response subformula that one could think of in natural language for a finitary requirement is often expressible in normal form.
A. Translating PTL Formula to \( S \)-Automaton

1) The Procedure: The procedure S-XLAT implements the operation \( \otimes_\Pi \). Intuitively, using the expansion rules (and logic arithmetic), the procedure expands an initial formula \( \omega \) into subformulae that are either atomic (i.e., without temporal operators) or have next \( \Box \) as their outermost operator, where the atomic subformulae are transitional conditions while the rest within each \( \Box \) are next state conditions. In turn, it expands the latter next state subformulae similarly if they are new (i.e., not identical to those already formed). Thus, by repeatedly applying expansion rules, it performs construction state by state, until all next state subformulae are not new, yielding an \( S \)-automaton. The procedure’s termination condition - when all next state subformulae are not new - will eventually hold since the length (in terms of the number of propositional symbols or their negation) of the initial formula \( \omega \) is finite [30].

Formally, given a PTL formula \( \omega \), the procedure constructs an \( S \)-automaton \( C \) from it as in the following. The states of the \( S \)-automaton created are labelled, under a map \( T \), by a finite set of satisfiable PTL formulae \( S \) generated under formula expansion. The map \( T : X \rightarrow S \) has the property that \( T(x_1) = T(x_2) \) if \( x_1 = x_2 \). For a formula \( \varpi \) that labels a state \( x \), we write \( \varpi /\{x\} \).

Proc S-XLAT: Input is \( \omega \), Output is \( C \) (8)

1) Create and label initial state \( x_0 \) as \( \omega \), i.e., \( T(x_0) = \omega \).
2) Let \( X = \{x_0\} \), \( E_{\text{new}} = \{\omega\} \) and \( E_{\text{old}} = \emptyset \).
3) If \( T(x_0) \) is in invariance normal form, then add \( x_0 \) to \( X_m \).
4) While \( E_{\text{new}} \neq \emptyset \), do the following.
   a) For each formula \( \varpi /\{x\} \) in \( E_{\text{new}} \),
      i) expand it using the expansion rules (see Section III-C.3);
      ii) do logic arithmetic on expanded formula to reduce it to a sum of a finite number of \( \Pi \)-product terms.
      Each term is of the form \( f^j \land \Box \varpi^j \), where \( f^j \) is a satisfiable state formula with the conjunction of any two such state formulae being false, and \( \varpi^j \) is a different satisfiable PTL formula.
   b) Empty all \( \varpi \)'s from \( E_{\text{new}} \) into \( E_{\text{old}} \).
   c) For each product term \( f^j \land \Box \varpi^j \) (generated in Step (4a)), do the following.
      i) Create a new symbol \( v^j \) for \( f^j \), for which \( F(v^j) = f^j \), and add it to \( V \).
      ii) If \( \varpi^j \not\in E_{\text{old}} \), then
         A) create a new state \( x^j \) for which \( T(x^j) = \varpi^j \), and add it to \( X \);
         B) add \( x^j \) to \( E_{\text{new}} \);
         C) define \( \delta_s(x^j, x) = x^j \);
         D) if \( T(x^j) \) is in invariance normal form (including \( \Box(\text{true}) \)), then add \( x^j \) to \( X_m \).
      iii) Else
         A) get the old state \( x^j \) in \( X \) for which \( T(x^j) = \varpi^j \);
         B) define \( \delta_s(v^j, x) = x^j \).

Remark 3: That procedure S-XLAT can construct a deterministic state-based automaton (i.e., \( S \)-automaton \( C \)) is facilitated by logic arithmetic (including using appropriate validity assertions) and expansion rules rewritten under equivalence for temporal operators \( \Diamond \) and \( U \) (see Section III-C.3). The resulting automaton constructed would have transitions at the same state labelled by different satisfiable state formulae, and hence can be associated with different transition symbols for which \( F \) is deterministic and Property 1 holds (see Section III-D).

Remark 4: If an initial PTL formula \( \omega \) has a satisfiable subformula containing a persistence modality, \( \omega \) is not a response PTL formula. S-XLAT, upon expanding such a formula, removes nondeterminism by essentially recombining (using disjunction) the next state subformulae \( \Box \varpi' \)'s of all \( f' \land \Box \varpi' \)'s generated, one \( \varpi' \) of which is the subformula containing the persistence modality, into one next state which it does not mark. In the ensuing logic arithmetic process, an empty marked state set \( X_m \) will result when the procedure terminates. Thus, in constructing ‘determinism’, and marking only those states labelled by PTL formulae in invariance normal form, the (trim) output \( C \) can exist (i.e., its \( X_m \neq \emptyset \)) under S-XLAT only for an initial response PTL formula \( \omega \). In other words, although procedure S-XLAT accepts any state-based PTL formula \( \omega \), it can return an \( S \)-automaton \( C \) only if \( \omega \) is a response PTL formula. This S-XLAT translation is therefore sufficient for our purpose, since a finitary control requirement in PTL, the focus of this paper, is necessarily a response formula which we assume is a CNF (see Section III-C.4).

2) Translation Examples: To illustrate the translation using Proc S-XLAT, consider the following two examples.

Example 1: PTL formula \( \Box[f_1 \rightarrow f_2 U f_3] \), where \( f_1, f_2 \) and \( f_3 \) are state formulae. Let \( \omega /\{x_0\} \equiv \Box[f_1 \rightarrow f_2 U f_3] \). Clearly, \( x_0 \in X_m \).

Let \( \phi \equiv (f_2 U f_3) \). Then, applying expansion rules and logic arithmetic, we have

\[
\omega \equiv \Box[f_1 \rightarrow f_2 U f_3] \\
\equiv \neg f_1 \lor f_1 \land (f_2 U f_3) \land \Box(\omega) \\
\equiv \neg f_1 \land (\Box(\omega) \lor f_1 \land (f_2 U f_3) \land \Box(\omega)) \\
\equiv \neg f_1 \land (\Box(\omega) \lor f_1 \lor (f_2 \lor \neg f_3) \land \Box(\phi)) \land \Box(\omega) \\
\equiv \neg f_1 \land \Box(\omega) \lor (f_2 \lor \neg f_3) \land \Box(\omega \land \phi) \lor \Box(\omega /\{x^2\})].
\]

Create \( v^0, v^1 \) such that

\[
F(v^0) = (\neg f_1 \lor f_3), \\
F(v^1) = (f_1 \land f_2 \land \neg f_3).
\]

Define

\[
\delta_s(v^0, x_0) = x_0, \\
\delta_s(v^1, x_0) = x_1 \not\in X_m.
\]

Next, expanding similarly, we have

\[
\omega \land \phi \equiv \Box[f_1 \rightarrow f_2 U f_3] \land [f_2 U f_3] \\
\equiv [\neg f_1 \lor f_3] \land (\Box(\omega) \lor (f_1 \land f_2 \land \neg f_3) \\
\land \Box(\omega \land \phi)) \lor (f_2 \lor \neg f_3) \land \Box(\phi)) \\
\equiv (f_2 \land \neg f_3) \land \Box(\omega \land \phi).
\]

Create \( v^2, v^3 \) such that

\[
F(v^2) = (f_2 \land \neg f_3), \\
F(v^3) = f_3.
\]
Define
\[ \delta_s(v^2, x^1) = x^1, \]
\[ \delta_s(v^3, x^1) = x_0. \]

Hence the S-automaton graphically depicted in Fig. 2.

**Example 2:** PTL formula \( \omega_1 \land \omega_2 \), with \( \omega_1 \equiv \Box[f_1 \rightarrow f_2 U f_3] \) and \( \omega_2 \equiv \Box[g_1 \rightarrow g_2 U g_3] \), where \( f_1, f_2, f_3 \) and \( g_1, g_2, g_3 \) are all state formulae.

Let \( \omega/\{x_0\} \equiv \omega_1 \land \omega_2 \equiv \Box[(f_1 \rightarrow f_2 U f_3) \land (g_1 \rightarrow g_2 U g_3)]. \)

Clearly, \( x_0 \in X_m \).

Let \( \phi_1 \equiv (f_2 U f_3), \phi_2 \equiv (g_2 U g_3) \) and \( \phi \equiv \phi_1 \land \phi_2. \) Then, following from Example 1,
\[ \omega \equiv \Box[f_1 \rightarrow f_2 U f_3] \]
\[ \equiv [(\neg f_1 \lor f_3) \land \diamond \omega_1] \lor (f_1 \land f_2 \land \neg f_3) \land \diamond (\omega_1 \land \phi_1)], \]
and
\[ \omega_2 \equiv \Box[g_1 \rightarrow g_2 U g_3] \]
\[ \equiv [(\neg g_1 \lor g_3) \land \diamond \omega_2] \lor (g_1 \land g_2 \land \neg g_3) \land \diamond (\omega_2 \land \phi_2)]. \]

Thus expanding, it can be shown that
\[ \omega \equiv \omega_1 \land \omega_2 \]
\[ \equiv (\neg f_1 \lor f_3) \land (\neg g_1 \lor g_3) \land \diamond (\omega_1 \land \phi_1) \]
\[ \lor (f_1 \land f_2 \land \neg f_3) \land (\neg g_1 \lor g_3) \land \diamond (\omega_1 \land \phi_1)/\{x^1\} \]
\[ \lor (\neg f_1 \lor f_3) \land (g_1 \land g_2 \land \neg g_3) \land \diamond (\omega_2 \land \phi_2)/\{x^2\} \]
\[ \lor (f_1 \land f_2 \land \neg f_3) \land (g_1 \land g_2 \land \neg g_3) \land \diamond (\omega_1 \land \phi_1)/\{x^3\} \].

Create \( v^{00}, v^{01}, v^{02}, v^{03} \) such that
\[ F(v^{00}) = (f_1 \lor f_3) \land (\neg f_1 \lor g_3), \]
\[ F(v^{01}) = (f_1 \land f_2 \land \neg f_3) \land (\neg g_1 \lor g_3), \]
\[ F(v^{02}) = (f_1 \lor f_3) \land (g_1 \land g_2 \land \neg g_3), \]
\[ F(v^{03}) = (f_1 \land f_2 \land \neg f_3) \land (g_1 \land g_2 \land \neg g_3). \]

Define
\[ \delta_s(v^{00}, x_0) = x_0, \]
\[ \delta_s(v^{01}, x_0) = x^1 \not\in X_m, \]
\[ \delta_s(v^{02}, x_0) = x^2 \not\in X_m, \]
\[ \delta_s(v^{03}, x_0) = x^3 \not\in X_m. \]

Next, expanding, it can be shown that
\[ \omega \land \phi_1 \equiv (\omega_1 \land \phi_1) \land \omega_2 \]
\[ \equiv (\neg g_1 \lor g_3) \land \diamond (\omega_1 \land \phi_1) \]
\[ \lor (f_2 \land \neg f_3) \land (\neg g_1 \lor g_3) \land \diamond (\omega_1 \land \phi_1) \]
\[ \lor (f_2 \land \neg f_3) \land (g_1 \land g_2 \land \neg g_3) \land \diamond (\omega_1 \land \phi_1) \]
\[ \lor (g_2 \land \neg g_3) \land (f_1 \land f_2 \land \neg f_3) \land \diamond (\omega_1 \land \phi_1). \]

Hence the S-automaton graphically depicted in Fig. 3.

3) Dealing with Conjunction of Formulae: As Example 2 might have revealed, using Proc S-XLAT to directly translate a conjunctive formula \( \omega \) could be tedious. However, by observation of the process in Proc S-XLAT, one can define a binary operator \( \uparrow \) to combine the individual S-automata \( C_i \) for \( \omega_i \) to yield an S-automaton \( C \) for \( \omega \), \( \omega = \bigwedge \omega_i \). For \( \omega = \omega_1 \land \omega_2 \), where \( C \equiv \bigcup_{i \in \omega} C_i \equiv [\{X, V, \delta_v, x_0, X_m, F\}, C_1 \equiv \bigcup_{i \in \omega_1} C_i \equiv [\{X, V, \delta_v^1, x_0^1, X_m1, F_1\}, F_1 \] and
of propositional symbols contained in the formulae; formally, it is possible to build a Büchi automaton (the parallel of which is the \(S\)-automaton) of a very reasonable size for any PTL formula that the system designer could think of for a program property, or in the DES context, for a control requirement. In other words, the inherently exponential nature of PTL translation to \(S\)-automata is of little practical significance [21]. As Section VI will show, the significance diminishes further in our implementation approach, where we use \(S\)-automaton templates representing PTL formulae of practically useful forms. These \(S\)-automaton templates are instantiated accordingly and further translated into \(E\)-automata using the procedure presented in Section IV-B.

B. Translating \(S\)-Automaton to \(E\)-Automaton

In further translating to an \(E\)-automaton, effectively, we are selecting a set of marked strings generated by a (trim) DES \(G\), whose corresponding state trajectories satisfy a PTL formula \(\omega\) represented by an \(S\)-automaton \(C\).

Given a DES \(G = \langle Q, \Sigma, \delta, \omega, q_0 \rangle\), and an \(S\)-automaton \(C = \langle X, V, \delta_s, \omega, x_0, X_m \rangle\), converted from a PTL formula \(\omega\) expressed in terms of the propositional variables in \(\Pi\), the procedure E-xLAT that implements \(\times_G\) to yield the translation, a cross product of \(G\) and \(C\), is as follows.

**Proc E-XLAT:** Input is \((G, C)\), Output is \(\text{Trim}(G \times C)\)

1. Determine the initial state: 
   \((q_0, \delta_s(v_0, x_0)) \in Q \times X\) is the initial state iff \(\exists v_0 \in V, \delta_s(v_0, x_0) \models \omega\).
2. Define the state transition function \(\delta : \Sigma \times Q \times X \to Q \times X:\)
   \[
   \delta(\sigma, q, x) = \begin{cases} 
   (\delta_s(\sigma, q), \delta_s(v, x)) & \text{if } \delta_s(\sigma, q) \land \delta_s(v, x) \\
   \text{undefined} & \text{otherwise}
   \end{cases}
   \]
3. Set \(Q_m \times X_m\) as the set of marked states.
4. Trim \(G \times C\).

Following state initialization (Step 1), the pairing of states in \(G \times C\) product synthesis (Step 2) is graphically depicted in Fig. 4.

**Fig. 4.** \(G \times C\) synthesis: for \(i > 0\), \((q_i, x_{i+1}) \in Q \times X\) if \(\models q_i \models F(v_i)\)

A resulting string \(u = v_0v_1v_2 \cdots v_m \in V^*\) is accepted by the \(S\)-automaton \(C\) if \(\delta_s(u, x_0) \in X_m\), i.e., in a standard fashion [9] for finite strings. Due essentially to the expansion rules (see Section III-C.3) used in Proc S-xLAT to produce \(C\), it follows that if \(C\) exists (i.e., its \(X_m \neq \emptyset\)) for a PTL
formula $\omega$, the (finitary) satisfaction or correctness relation of trajectory $\rho_m$ over the formula $\omega$ can be written as

\[ \models_{Gm} \omega \iff \text{there exists } v_0 v_1 v_2 \cdots v_m \in L_m(C) \]

As discussed in Section III-C.4, marked semantics consistency of $\models_{Gm} \omega$ holds (and thus $C$ can exist) only for a response PTL formula $\omega$.

The construction of product $G \times C$ is a translation algorithm of worst-case complexity $O(|Q| |X|||V|||)$.

C. Determinism of Translation

Theorem 1: $\text{Trim}(G \times C)$ is a deterministic $\mathcal{E}$-automaton.

Proof: By Property 1 of $\mathcal{S}$-automaton $C$, for an arbitrary $x \in X$, two non-identical $v', v'' \in V(x)$ and $\delta(x, q) \in Q$, $\models\delta(x,q) F(v')$ implies $\models\delta(x,q) F(v'')$. Hence, there exists at most one $v \in V(x)$ for which $\models\delta(x,q) F(v)$. Thus from function $\delta$ (12), $\delta(x, (q, x))$ holds (and thus (13), $\delta(x, (q, x))$ holds (and thus only if $\omega$ is a response PTL formula; it follows that $C$ is an equivalent representation (that exists) for a response PTL formula $\omega$. A CNF, by definition, is a response PTL formula. Hence the result.

Following Theorem 2, the whole translation algorithm (consisting of $\mathcal{S}$-XLAT followed by $\mathcal{E}$-XLAT) is said to be correct and complete.

D. Correctness and Completeness of Translation

Naturally, two important properties of a translation are its correctness and completeness, defined as follows.

Definition 2: Let $\rho_m = \rho(0)\rho(1)\cdots \rho(i)\cdots \rho(m)$ be a marked finite state-trajectory, where $\rho(i) \in Q \times X$ for all $i$, $0 \leq i \leq m$, for which there exists a corresponding string $e_{m-1} = e(0)e(1)\cdots e(m-1) \in L_m(G \times C)$ such that $\rho(i+1) = \delta(e(i), \rho(i))$ and $q_{i+1} = \delta(x, (q_i, x_i))$. Then $\text{Trim}(G \times C)$ is said to be correct with respect to $C$ if

1) $\models^{\rho_m} F(v_i)$
2) for every $e_{m-1} \in L_m(G \times C)$, there exists a $v_0v_1v_2\cdots v_m \in L_m(C)$ such that $\models_{G} F(v_i)$ for all $i$, $0 \leq i \leq m$.

The second condition of Definition 2 asserts that every (finite) state trajectory on a (marked) string of $L_m(G \times C)$ must satisfy some PTL formula translated as $\mathcal{C}$.

Definition 3: Let $\rho_m = \rho(0)\rho(1)\cdots \rho(i)\cdots \rho(m)$ be a marked finite state-trajectory, where $\rho(i) = q_i \in Q$ for all $i$, $0 \leq i \leq m$, for which there exists a corresponding string $e_{m-1} = e(0)e(1)\cdots e(m-1) \in L_m(G)$ such that $\rho(i+1) = \delta(x, (q_i, x_i))$. Then $\text{Trim}(G \times C)$ is said to be complete with respect to $G$ if

1) $\models^{\rho_m} F(v_i)$ and $(q_0, \delta(x, (q_0, x_0))) \in Q \times X$ is the initial state of $G \times C$;
2) for every $\rho_m$ of $G$ with $\rho(m) \in Q_m$, if there exists a $v_0v_1v_2\cdots v_m \in L_m(C)$ such that for all $i$, $0 \leq i \leq m$, $\models_{G} F(v_i)$, then $e_{m-1} \in L_m(G \times C)$.

Note that $\models_{G} F(v_i)$ and antecedent of the second condition of Definition 3 (or the consequent of the second condition of Definition 2) define the (finitary) satisfaction or correctness relation (13).

Theorem 2: $\text{Trim}(G \times C)$ is correct with respect to CNF $\omega$, and complete with respect to DES $G$.

Proof: By $\mathcal{E}$-XLAT translation to $G \times C$, it follows straightforwardly by Definitions 2 and 3 that $\text{Trim}(G \times C)$ is correct with respect to $C$, and complete with respect to $G$.

V. STATE-BASED TEMPORAL LOGIC & EVENT-BASED FINITE AUTOMATA: A DISCUSSION

Our technical goal has been to build a standard $\mathcal{E}$-automaton that accepts completely a marked sublanguage whose corresponding finite state trajectories for a DES satisfy a finitary control requirement written in state-based PTL. The proposed $\mathcal{G}$-based translation algorithm is developed for a class of response PTL formulae (in CNF) that characterizes finitary control requirements, and abides by the finitary notions of correctness and completeness as defined in Section IV-D. Importantly, marked semantics consistency, and hence validity of the translation under infinitary semantics, holds for response PTL (see Section III-C.4).

Automaton $C$, the output of procedure $\mathcal{S}$-XLAT, is necessarily different from a B"uchi automaton. For, in the latter’s acceptance condition, infinite trajectories must visit some state in an acceptance subset in the $\omega$-automaton infinitely often. Thus although the idea underlying procedure $\mathcal{S}$-XLAT is due essentially to Wolper [30]’s that outputs a B"uchi automaton, we avoid calling $\mathcal{S}$-XLAT’s output $C$ a B"uchi automaton since, while structurally similar, its marked state set is not interpreted as an acceptance set formulated in the original definition for B"uchi automata. Besides, while a B"uchi automaton generated by Wolper’s original procedure is not deterministic, the $\mathcal{S}$-automaton generated by $\mathcal{S}$-XLAT is (see Remark 3). Importantly, that $\mathcal{S}$-automaton $C$ is deterministic leads us to a fruitful result, namely, Theorem 1, as presented in Section IV-C.

Apparently, the final deterministic $\mathcal{E}$-automaton translated seems less complete (in some representative sense) than its $\omega$-version, as it admits only marked finite strings in contrast to the latter accepting infinite ones. However, for finitary construction, the former’s proposed translation algorithm can do away with the latter’s having to translate the liveness part of a PTL formula. In other words, it does not need to deal directly with the temporal issue of liveness associated with infinite strings [21]. Despite the (simpler) finitary translation, it turns out that this need not lead to loss of completeness. For, given a CNF $\omega$ on a DES $G$ whose state space is bounded, an appropriate assignment or reassignment of marked states
in DES $G$ is all that is required to establish marked semantics completeness, i.e., $(|=^\rho \omega) \iff (|=^{\rho_m} \omega)$, where $\rho_m$, with state $\rho(m) \in Q_m$, is any marked finite trajectory of an infinite DES state-trajectory $\rho$. Importantly, this means that as long as there is an appropriate marking of DES states (such as that suggested later in Section VI-A.1 for a useful class of response PTL formulae), the proposed translation algorithm can be used as a specification interface, without essential loss of temporal completeness for discrete-event control synthesis founded on the finitary case [2].

Thus in principle, using the proposed translation algorithm, any PTL formula $\omega$ that a system designer could think of can be specified, provided it is a CNF, and expressed using only propositional variables in $\Pi$ characterizing the states of a DES model $[\Pi, G]$ (4). By an appropriate marking of DES states, the resultant $\mathcal{E}$-automaton (or equivalently its generated marked sublanguage) can, in the limit, also represent all infinite event strings on state trajectories satisfying $\omega$. In practice, such marking is often subject to a designer’s notion of completed tasks for a specific DES, which determines the DES marked language $L_m(G)$. We believe that the design synergy between DES marking and specifying nonblocking control requirements in PTL could often be found, facilitating well-understood finitary design.

VI. A SPECIFICATION INTERFACE & APPLICATION EXAMPLES

A. An Interface for Control Synthesis

A software package called TCT [3] has been developed as a tool for designing supervisors based on the conventional (or finitary) automata-theoretic framework [2]. The proposed translation algorithm has been implemented as a specification interface called NanTA for use with TCT, integrated as depicted in Fig. 5.

![Diagram](image)

Fig. 5. NanTA Interface for TCT

1) Specifications Considered: S-XLAT can actually be implemented as similarly done in Fujita et al [31]. But as a practical approach, for NanTA, only E-XLAT has been implemented. Instead of using S-XLAT to translate an arbitrary PTL formula, we consider a useful class of response PTL formulae of the (normal) form:

$$\Box[f_1 \rightarrow f_2 U f_3]$$ (14)

where $f_1$, $f_2$ and $f_3$ are propositional state formulae. Typically, a control requirement is expressed as some condition - an invariant $f_2$ - that must hold whenever its precondition $f_1$ occurs, and the invariant cannot be relaxed until its postcondition $f_3$ occurs, as is embodied by the form (14) that is thus not unfamiliar to system designers.

The $S$-automaton for formulae of the form considered, synthesized using S-XLAT, is shown in Fig. 2; the detailed S-XLAT translation steps to construct it were given earlier in Example 1 of Section IV-A.2. In NanTA, it has been implemented as a computation `template' from which the $S$-automaton for each specific formula in this form can be `instantiated'. To support the conjunctive combination of PTL formulae, the operator $\Box$ has also been implemented to combine the respective $S$-automata.

Finally, to deal with $f_3$-liveness conditional upon $f_1$ as asserted in the PTL form (14), it is easily seen to be necessary but quite natural to interpret the truth of $f_3$ as progress towards some task completion represented by DES marked states. Hence we can always assign or reassign some states in DES $G$ as marked, so that there is at least one marked state $q \in Q_m$ in which $f_3$ is true.

It should be pointed out that two other useful forms are subsumed by (14):

$$\left\{ \begin{array}{l}
\Box[\neg f_1] \quad \text{if } f_3 = \text{false}, \\
\Box[f_1 \rightarrow \Diamond f_3] \quad \text{if } f_2 = \text{true}.
\end{array} \right.$$

The former asserts a forbidden-state formula, while the latter, a conditional response formula.

2) Class of DES’s Considered: The finite automata framework [2], [3] for system design admits DES modelling as a collection of individual finite automata prescribing the various component processes. Such a DES model is not uncommon (e.g., in manufacturing systems [4]). It is usually because of interacting component processes that a designer needs to specify control requirements to constrain and ensure proper operation of the DES as a whole.

In the current version of NanTA, every state in a component process $G_i$ of a DES model $G$ is uniquely characterized by a propositional symbol (or state variable), i.e., the symbol defined is true only at that state, and false elsewhere. This, however, is not an assumption of the general translation algorithm proposed in Section IV, and can be relaxed in a future version of NanTA. Nonetheless, it is believed that many practical DES’s can be modelled with this assumption which helps simplify the implementation of propositional satisfiability checks.

B. Application Examples

To demonstrate the usefulness of the NanTA interface, we consider two application examples, chosen to show how useful (temporal) requirements can be specified in PTL, leaving the tedium of translation to the interface. The first example is also
simple enough so that we can visually inspect the outputs of the NanTA translator.

In both the examples, an automaton modelling a discrete-event process is graphically depicted with a propositional symbol (or state variable) placed next to each node representing a state; each symbol is true only at that state, and false elsewhere.

1) Example 1 (Resource Allocation): Two users USER1 and USER2 operate asynchronously to access a resource [5]. The automaton for each user is shown in Fig. 6(a). The symbols used are defined in Table I.

![Fig. 6. NanTA translation: Illustrative example 1 (resource allocation)](image)

(a) Two users (b) \( E \)-automaton for First-Come-First-Serve requirement

TABLE I

<table>
<thead>
<tr>
<th>SYMBOL DEFINITIONS FOR USER MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
</tr>
<tr>
<td>iː idling;</td>
</tr>
<tr>
<td>rː resource requested;</td>
</tr>
<tr>
<td>uː resource in-use;</td>
</tr>
</tbody>
</table>

The integer subscripted to each symbol identifies the user (see Fig. 6(a)). The system \( G \) as a whole is USER1 \( \parallel \) USER2.

Consider a First-Come-First-Serve control requirement. In temporal logic, it may be simply written as a conjunction of the following two formulae:

1) \( \Box [r_1 \land i_2 \rightarrow \neg u_2 U i_1] \), 2) \( \Box [r_2 \land i_1 \rightarrow \neg u_1 U i_2] \).

The first formula may be paraphrased as ‘whenever USER1 requests to use the resource when USER2 is idling (i.e., \( r_1 \land i_2 \) is true), USER2 will not be allowed to use it (i.e., \( u_2 \) will remain false) until USER1 has finished with it, which occurs once USER1 returns to its idling state (i.e., \( i_1 \) becomes true). The second formula may be paraphrased similarly.

The conjunctive translation produced by NanTA is as shown in Fig. 6(b).

2) Example 2 (Manufacturing System): A manufacturing system adapted from [2] consists of two machines \( M1 \) and \( M2 \), and a one-slot buffer \( B \), connected as shown in Fig. 7, along with the automaton for each process. The symbols used are defined in Table II.

![Fig. 7. NanTA translation: Illustrative example 2 (a simple manufacturing system)](image)

The event set for \( M_j, j \in \{1, 2\} \), is \( \Sigma_j = \{a_j, b_j, g_j, h_j\} \) and that for \( B \) is \( \Sigma \), where \( \Sigma = \Sigma_1 \cup \Sigma_2 \). It is assumed that \( M1 \) takes a workpiece from an infinite buffer; also, when \( B \) is filled, any attempt to add another workpiece to it will fail as the workpiece will immediately drop off \( B \). The system \( G \) as a whole is \( M1 \parallel B \parallel M2 \).

Consider the conjunction of the following control requirements expressed in PTL:

1) \( \Box [e \land \neg w_2 \rightarrow \neg w_2 U f] \) (Underflow avoidance);
2) \( \Box [f \land \neg i_1 \rightarrow \neg i_1 U (e \lor d_1)] \) (Overflow avoidance);
3) \( \Box [d_1 \land d_2 \rightarrow \neg i_1 U i_2] \) (Repair priority of \( M2 \) over \( M1 \)).

The first formula may be paraphrased as ‘whenever \( B \) is empty and \( M2 \) is not processing any workpiece, \( M2 \) must not take any workpiece from \( B \) to process until \( B \) is filled’.

The second formula may be paraphrased as ‘whenever \( B \) is filled and \( M1 \) is not idling, \( M1 \) must not put a workpiece it is holding into \( B \) until either \( B \) is emptied, or \( M1 \) breaks down without placing the workpiece into \( B \)’. The last formula may be paraphrased as ‘whenever \( M1 \) and \( M2 \) both break down, \( M1 \) will not be repaired until \( M2 \) is’.

A sample translation produced by NanTA is as shown in Fig. 8.

3) Some Remarks: The two examples have demonstrated the purpose of NanTA for use with TCT, namely, NanTA helps, if not reduce the tedium of crafting error-free specification automata, provide an alternative means to describing and understanding control requirements, through enabling the use of (natural language) expressive and readable PTL.

As a final remark, we note that a system designer conversant with finite automata might be able to write simple \( E \)-automata corresponding to the required specifications for the two examples. It however does not eradicate the fact that in general, the designer may not always know for sure if a specification \( E \)-automaton captures the intended control requirement correctly.
TABLE II
SYMBOL DEFINITIONS FOR MACHINE AND BUFFER MODELS

<table>
<thead>
<tr>
<th>Process</th>
<th>State</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_j$</td>
<td>$i_j$: idling;</td>
<td>$a_j$: workpiece taken;</td>
</tr>
<tr>
<td>$j \in {1,2}$</td>
<td>$w_j$: workpiece processing;</td>
<td>$b_j$: workpiece put into buffer;</td>
</tr>
<tr>
<td></td>
<td>$d_j$: non-operating;</td>
<td>$g_j$: machine broken down;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$h_j$: machine repaired.</td>
</tr>
<tr>
<td>$B$</td>
<td>$e$: buffer emptied;</td>
<td>$f$: buffer filled;</td>
</tr>
</tbody>
</table>

Fig. 8. Example 2: Priority of repair: $[[d_1 \land d_2 \rightarrow \neg i_1 U i_2]]$

and completely, nor can another designer readily interpret the intended control meaning of the $\mathcal{E}$-automaton. This potential problem might be mitigated if the specifications are written in readable (and thus more easily understood) PTL formulae, with their translation to correct and complete $\mathcal{E}$-automata handled by an automation tool such as NanTA.

VII. CONCLUSIONS

This paper has proposed a translation algorithm as an interface for specifying and converting a class of control requirements in PTL for automata-theoretic finitary synthesis of discrete-event supervisors. The resulting interface is a new DES-contextual translator which can automatically convert a class of state-based response PTL formulae into event-based finite automata that, importantly, are deterministic, correct and complete. In our opinion, its immediate impact is significant: it mitigates, if not solve, the long-standing problem of specification for finitary control synthesis of DES’s.

NanTA, a practical implementation of the interface, has been developed to provide an enabling technology for writing control requirements in a more readily understood PTL form that it translates into $\mathcal{E}$-automata for use with the finitary control synthesis package, TCT [3]. Two simple but practical application examples illustrate the usefulness of the NanTA interface. The interface represents a step towards a more ‘specification-friendly’ design framework for finitary control of DES’s. It would help reduce design errors and costs associated with incorrect, unnecessarily restrictive or misinterpreted specifications. It should, in principle, also lend mutual support to the usage (by a larger community of application researchers and engineers) and further enhancement (by researchers and collaborators) of TCT’s underlying capabilities for automated systems design and control synthesis.

The proposed translation algorithm returns a trim $\mathcal{E}$-automaton $A_f$ of $G \times C$ representing the full nonblocking behaviour captured by the corresponding PTL formula $\omega$ for a DES $G$. However, the automaton $A_f$ can be larger in state size than is necessary because it incorporates all the a priori transitional constraints embodied in DES model $G$ itself. This automaton $A_f$ represents the full nonblocking behaviour of a supervisor if it is controllable. Su and Wonham [33] have developed Supreduce, a heuristic reduction procedure of polynomial complexity that can often find a greatly reduced-state supervisor based on a given DES $G$ and an $\mathcal{E}$-automaton representing the full nonblocking behaviour of a supervisor. A procedure, Autreduce, can therefore be developed that essentially is Supreduce but with all events ‘virtually set’ as controllable, to convert $A_f$ to a state reduced (trim) automaton $A_p$ for which

$$L_m(A_f) = L_m(A_p) \cap L_m(G)$$

(15)

But more useful would be a translation to be developed that is partially contextual (with respect to DES $G$), in that it produces $A_p$ in accordance to (15), and that works ‘on the fly’, in that $A_p$ is produced without the need to explicitly construct $A_f$ altogether.

Active research in computer science has led to efficient
but non-contextual translation algorithms converging a PTL formula to an $\omega$-automaton (see e.g., [34]), and they have apparently already been implemented in tools like SPIN [35]. Future work on PTL translation to $E$-automata for finitary control might benefit from adapting such developments, since the worst-case complexity of constructing automata from PTL formulae is known to be PSPACE-complete [30].

APPENDIX

The $\parallel$-operation is implemented via the synchronous product of two arbitrary $E$-automata. Consider $G_1 = (\Pi_1, G_1)$ and $G_2 = (\Pi_2, G_2)$, with $\Pi_1 \cap \Pi_2 = \emptyset$ (i.e., no shared variable), and $E$-automata $G_1 = (Q_1, \Sigma_1, q_0, Q_{m1})$ and $G_2 = (Q_2, \Sigma_2, q_0', Q_{m2})$. Then, to yield $G = [G_1 \parallel G_2]$, with $E$-automaton $G = (Q, \Sigma, q_0, Q_{m})$ as the synchronous product of $G_1$ and $G_2$, conveniently denoted by $G = G_1 \parallel G_2$, we have $\Pi = \Pi_1 \cup \Pi_2$, and $Q = Q_1 \times Q_2$, $Q_{m} = Q_{m1} \times Q_{m2}$ such that

$$\prod_{u_1 \in \Pi_1} (u_1^1 = q_1^1 \parallel u_1^2 = q_2) \equiv \prod_{u_2 \in \Pi_2} (u_2^1 = q_1^2 \parallel u_2^2 = q_2^2)$$

uniquely characterizes the state information in $q = (q_1^1, q_2^2) \in Q$. The rest follows the conventional definition of $\parallel$ [2].

ACKNOWLEDGEMENTS

The author would like to thank his undergraduate research students, Ming Gai and Tong Lee Lim, for co-prototyping the NanTA software. He would also like to thank the Associate Editor and all the anonymous referees for their critical but constructive comments on the review versions of this manuscript. The responsibility for the integrity of this work remains solely with the author.

REFERENCES


