Control of State Tree Structures

Chuan Ma and W. M. Wonham

Abstract— **It is well known that the nonblocking supervisory control problem is NP-hard, subject in particular to state space explosion that is exponential in the number of system components. In this paper we propose to manage complexity by organizing the system as a State Tree Structure (STS). STS are an adaptation of state charts to supervisory control theory. Based on STS we present an efficient recursive algorithm that can perform nonblocking supervisory control design (in reasonable time and memory) for systems of state size 10^{24} and higher.**

Keywords— **State Tree Structure (STS), Discrete Event System, Nonblocking Supervisory Control, Symbolic Computation, Recursive Algorithm.**

I. Introduction

THE central problem of supervisory control theory (SCT) is that of optimal nonblocking supervisory control [1]. This problem is well known to be NP-hard [2]: specifically it is unlikely that any algorithm can be found to solve it that circumvents state space explosion that is exponential in the number of system components. This means that realistic industrial problems (say with state set sizes of 10^{20} and higher), if formulated naively, may well exceed the computational capacity available.

It is therefore attractive to explore structured system architectures with the property that, if the given system can be modelled in the selected framework, the required computations can be carried out with greater efficiency than by a naive 'monolithic' approach. Apart from the effort needed to adapt a specific structural form to the system to be modelled, the price to be paid may include relaxing the (monolithic) requirement of optimality, viz. maximal permissiveness of controlled behavior. However, as long as controlled behavior is 'legal' and nonblocking, some reduction in permissiveness may well be acceptable for the sake of increased tractability. In addition, structured modelling may confer advantages of model transparency and modifiability. One instance is Leduc's [3] *Hierarchical Interface-based Supervisory Control* theory (HISC).

Statecharts [4] offer a compact representation of hierarchy and concurrency in finite state machines (FSM). Here the system state set is structured top-down into successive layers of cartesian products (AND superstates) alternating with disjoint unions (OR superstates). On this basis, Wang [5] introduced State Tree Structures consisting of a hierarchical state space or State Tree, equipped with dynamic modules called *holons* (inspired by [6] and [7]). Gohari [8] formalized Wang's model in linguistic terms. In [5], however, AND states had to be converted by synchronous product of factors into OR states at a higher level before computations could effectively be carried out; and [8] was similarly restricted to a purely OR state expansion. By contrast, in this paper we treat both AND and OR states on an equal footing, and in AND states allow shared events among the factors. Statecharts also underlie the Asynchronous Hierarchical State Machine (AHSM) model of [9], but shared events among AND components were ruled out. Marchand *et al.* [10] introduce another simplified version of statecharts, the Hierarchical Finite State Machines (HFSM). But HFSM rule out shared events, are apparently restricted to be OR structures at the topmost level, and allow only specifications of forbidden state type.

Control design with STS depends on efficient computational representation. Borrowing from symbolic model checking [11] we employ binary decision diagrams [12] (BDDs). In [13] BDDs were used for a DES with 10^6 states; by means of STS hierarchies encoded with BDDs we report on a version of the AIP example [16], [3] having on order 10^{24} states.

The authors are with the Systems Control Group, Dept. of Electrical and Computer Engineering, University of Toronto, ON, Canada M5S 3G4. E-mail: (cma, wonham)@control.utoronto.ca.

II. State Tree Structure

Let X be a finite set of objects, called *states*. Let $x \in X$ and $Y = \{x_1, x_2, \ldots, x_n\} \subset X$, $x \notin Y$. If x can be represented by the union (Cartesian product) of states in Y, we call x an *OR (AND) superstate* and each x_i an *OR (AND) component* of x. Also call x a *parent* of any x_i and x_i a *child* of x. All other states in X are called *simple states*. Say X is a *structured state set*. Formally, we define $\mathcal{M} : X \to \{and, or, simple\}$ as the *mode* function and $\mathcal{E} : X \to 2^X$ as the *expansion* function such that

$$\mathcal{E}(x) := \begin{cases} Y, & \text{if } \mathcal{M}(x) \in \{and,\, or\} \\ \emptyset, & \text{if } \mathcal{M}(x) = simple \end{cases},$$

where $\emptyset$ denotes the empty set. Let $R \subset X$. Write $\mathcal{M}_R : R \to \{and, or, simple\}$ as the restriction of $\mathcal{M}$ to R, and $\mathcal{E}_R : R \to 2^R$ as the restriction of $\mathcal{E}$. The *reflexive and transitive closure* of $\mathcal{E}$ is written $\mathcal{E}^*$. That is, $\mathcal{E}^*(x) - \{x\}$ is the set of *all descendants* of x, while x is an *ancestor* of states in $\mathcal{E}^*(x) - \{x\}$.

Now we can define the *state tree* by recursion.

Definition 1 (State Tree) A *state tree* is a 4-tuple $(X, x_o, \mathcal{M}, \mathcal{E})$, where

- X is a finite structured state set with $X = \mathcal{E}^*(x_o)$,
- $x_o \in X$ is the *root state*.

$\mathbf{ST} = (X, x_o, \mathcal{M}, \mathcal{E})$ is a *state tree* if

1. (terminal case) $X = \{x_o\}$, or
2. (recursive case) $(\forall x_i \in \mathcal{E}(x_o))\mathbf{ST}^{x_i} = (\mathcal{E}^*(x_i), x_i, \mathcal{M}_{\mathcal{E}^*(x_i)}, \mathcal{E}_{\mathcal{E}^*(x_i)})$ is also a state tree, where $(\forall x_i, x_j \in \mathcal{E}(x_o), x_i \neq x_j)\mathcal{E}^*(x_i) \cap \mathcal{E}^*(x_j) = \emptyset$ and $\bigcup_{x_i \in \mathcal{E}(x_o)} \mathcal{E}^*(x_i) = X - \{x_o\}$.

Say $\mathbf{ST}^{x_i}$ is a *child state tree of* x_o in $\mathbf{ST}$, rooted by x_i.

A *well-defined* state tree must also satisfy

$$(\forall x \in X)\mathcal{M}(x) = and \;\&\; x_i \in \mathcal{E}(x) \Rightarrow \mathcal{M}(x_i) = or.$$

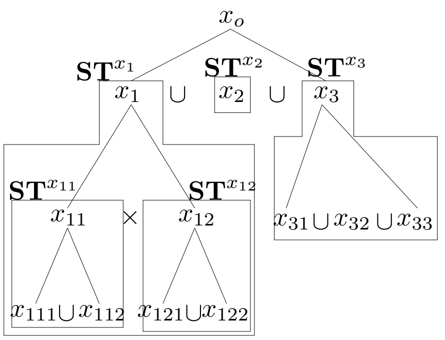

Fig. 1. Recursively define the State Tree **ST**

That is, all AND components must be OR superstates. We assume all state trees are well-defined.

For convenience, say $\mathbf{ST}$ is the *empty state tree* if $X = \emptyset$, and write $\mathbf{ST} = \emptyset$.

$\diamond$

An example is shown in Fig. 1.

Definition 2 (Sub State Tree) Let $\mathbf{ST} = (X, x_o, \mathcal{M}, \mathcal{E})$ be a state tree, and let $Y \subseteq X$. $\mathbf{subST} = (Y, x_o, \mathcal{M}_Y, \mathcal{E}')$ is a *sub state tree* of $\mathbf{ST}$ if $\mathbf{subST}$ is a well-defined state tree with $\mathcal{E}' : Y \to 2^Y$ defined by

$$\begin{cases} \mathcal{E}'(y) := \mathcal{E}(y), & \text{if } \mathcal{M}_Y(y) \in \{and, simple\} \\ \emptyset \subset \mathcal{E}'(y) \subseteq \mathcal{E}(y), & \text{if } \mathcal{M}_Y(y) = or \end{cases},$$

for all $y \in Y$. Trivially, the empty state tree $\emptyset$ and $\mathbf{ST}$ itself are sub state trees of $\mathbf{ST}$. A *proper* sub state tree of $\mathbf{ST}$ is one with $Y \subset X$. Denote by $\mathcal{ST}(\mathbf{ST})$ the set of all sub state trees of $\mathbf{ST}$.

$\diamond$

Two examples of sub state trees are $\mathbf{ST}_\beta$ and $\mathbf{ST}_\sigma$ in Fig. 5.

Let $\mathbf{ST}_1, \mathbf{ST}_2 \in \mathcal{ST}(\mathbf{ST})$. Define

$$\mathbf{ST}_1 \leq \mathbf{ST}_2 \text{ iff } \mathbf{ST}_1 \in \mathcal{ST}(\mathbf{ST}_2).$$

That is, $\mathbf{ST}_1 \leq \mathbf{ST}_2$ if and only if $\mathbf{ST}_1$ is a sub state tree of $\mathbf{ST}_2$. One can prove that $(\mathcal{ST}(\mathbf{ST}), \leq)$ is a *lattice* in which the meet and join of any two elements always exist. In particular the bottom element $\perp = \emptyset$ and the top element $\top = \mathbf{ST}$. Fig. 2 illustrates the meet operator. Notice that x_2 is not on $\mathbf{ST}_3$ because $\mathbf{ST}_1$ and $\mathbf{ST}_2$ do not agree on any of x_2's descendants.

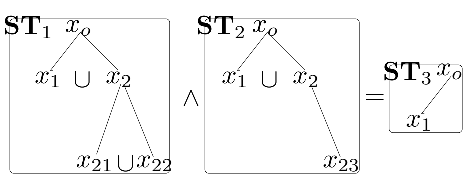

Fig. 2. $\mathbf{ST}_1 \wedge \mathbf{ST}_2 = \mathbf{ST}_3$

Instead of the abstract concept of *configuration* in statecharts, we use the simpler $\mathcal{ST}(\mathbf{ST})$ as the universe for the behavior of STS. In our setting, an eligible event is capable of transforming a sub state tree of $\mathbf{ST}$ into another one.

To measure the size of $\mathcal{ST}(\mathbf{ST})$, we introduce the function *count*, defined recursively along the sub state tree $\mathbf{ST}_1 \in \mathcal{ST}(\mathbf{ST})$ such that $\text{count}(\mathbf{ST}_1) :=$

$$\begin{cases} \prod_{\forall x_i \in \mathcal{E}(x_o)} \text{count}(\mathbf{ST}_1^{x_i}), & \text{if } \mathcal{M}(x_o) = and \\ \sum_{\forall x_i \in \mathcal{E}(x_o)} \text{count}(\mathbf{ST}_1^{x_i}), & \text{if } \mathcal{M}(x_o) = or \\ 1, & \text{if } \mathcal{M}(x_o) = simple \end{cases}$$

Trivially, $\text{count}(\emptyset) = 0$. Say $\mathbf{subST}$ is a *basic* sub state tree of $\mathbf{ST}$ if $\text{count}(\mathbf{subST}) = 1$. Write $\mathcal{B}(\mathbf{ST})(\subseteq \mathcal{ST}(\mathbf{ST}))$ for the set of all basic sub state trees of $\mathbf{ST}$. A basic sub state tree is a minimal nonempty element, or *atom*, of $\mathcal{ST}(\mathbf{ST})$. It is equivalent to a state of FSM in describing system behavior. A simple way of defining STS behavior is to assign transitions to each element in $\mathcal{B}(\mathbf{ST})$. However, the size of $\mathcal{B}(\mathbf{ST})$ can be so large for complex systems that the assignment may be infeasible to carry out. Instead, we introduce the concept of *holon*, the local behavior, and then build the global behavior structurally.

A holon is a generalized FSM with possibly more than one initial state.

Definition 3 (Holon) A *Holon* H is a 5-tuple

$$H := (X, \Sigma, \delta, X_o, X_m),$$

where

- X, the finite state set, is the disjoint union of *external state set* X_E and *internal state set* X_I.
- Σ, the event set, is the disjoint union of *boundary event set* Σ_B and *internal event set* Σ_I. Of course, an event in Σ can be controllable or uncontrollable.
- The transition structure $\delta : X \times \Sigma \to X$ is a partial function; it is the disjoint union [1] of the *internal transition structure* $\delta_I : X_I \times \Sigma_I \to X_I$ and the *boundary transition structure* δ_B; δ_B is again the disjoint union of two transition structures
  - $\delta_{BI} : X_E \times \Sigma_B \to X_I$ (incoming boundary transitions)
  - $\delta_{BO} : X_I \times \Sigma_B \to X_E$ (outgoing boundary transitions)

Write $\delta(x, \sigma)!$ if $\delta(x, \sigma)$ is defined. We require the transition structure to be deterministic.

- $X_o \subseteq X_I$ is the *initial state set*, where X_o has only the target states of incoming boundary transitions if δ_{BI} is defined. Otherwise X_o is a selected nonempty subset of X_I. Formally,

$$X_o := \begin{cases} \{\delta_{BI}(x, \sigma) | \delta_{BI}(x, \sigma)!\}, & \text{if } X_E \neq \emptyset \\ Z, \text{ where } \emptyset \subset Z \subseteq X_I, & \text{if } X_E = \emptyset \end{cases}$$

From now on , write $\delta_{BI} : X_E \times \Sigma_B \to X_o$ (pfn).

- $X_m \subseteq X_I$ is the *marker state set*, where X_m has only the source states of the outgoing boundary transitions if δ_{BO} is defined. Otherwise X_m is a selected nonempty subset of X_I. Formally,

$$X_m := \begin{cases} \{x | \delta_{BO}(x, \sigma)!\}, & \text{if } X_E \neq \emptyset \\ Z, \text{ where } \emptyset \subset Z \subseteq X_I, & \text{if } X_E = \emptyset \end{cases}$$

Write $\delta_{BO} : X_m \times \Sigma_B \to X_E$ (pfn).

$\diamond$

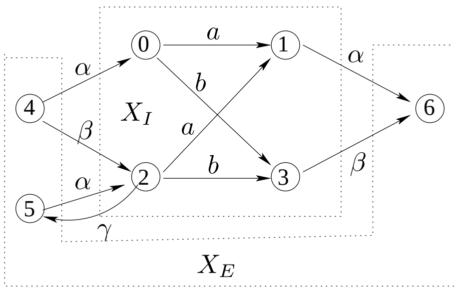

Fig. 3. An example holon

An example holon is given in Fig. 3. Here $X_E = \{4, 5, 6\}$ and $X_I = \{0, 1, 2, 3\}$; $\Sigma_B = \{\alpha, \beta, \gamma\}$ and $\Sigma_I = \{a, b\}$; $X_o = \{0, 2\}$; $X_m = \{1, 2, 3\}$.

For each OR superstate y in the state tree $\mathbf{ST} = (X, x_o, \mathcal{M}, \mathcal{E})$, a holon $H^y = (X^y_E \cup X^y_I, \Sigma^y, \delta^y, X^y_o, X^y_m)$ is said to be *matched* to y if

- internal structure matches, i.e., $X^y_I = \mathcal{E}(y)$.
- external structure matches. Let x be the nearest OR ancestor of y on $\mathbf{ST}$, i.e., $x < y$ and $\mathcal{M}(x) = or$. Then $\begin{cases} X^y_E = \emptyset, & \text{if } x \text{ does not exist} \\ X^y_E \subset \mathcal{E}(x), & \text{if } x \text{ exists} \end{cases}$

Suppose a holon H^x is also matched to the OR superstate x. We say H^x is the *parent holon* of H^y and H^y the *child holon* of H^x. Notice that if x exists, $X^y_E \subset \mathcal{E}(x)$ implies that the superstate y cannot be a boundary state in $(X^x_o \cup X^x_m)$, i.e., all boundary states of matched holons must be simple states. This helps us to limit vertical communication to be only between parent/child holons.

Definition 4 (Boundary Consistency) Let $H^x = (X^x, \Sigma^x, \delta^x, X^x_o, X^x_m)$ and $H^y = (X^y, \Sigma^y, \delta^y, X^y_o, X^y_m)$ be the holons matched to x and y, respectively. H^x is the parent holon of H^y. As illustrated in Fig. 4, there are only

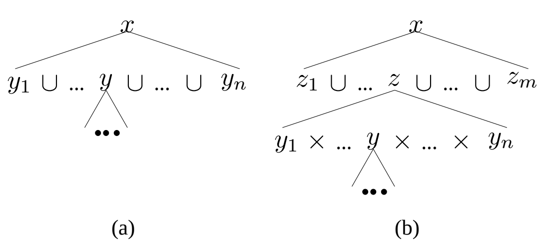

Fig. 4. Relation between y and x

two possible cases

1. $y \in \mathcal{E}(x) = X^x_I$ as in case (a) of Fig. 4, or
2. $(\exists z, \mathcal{M}(z) = and) y \in \mathcal{E}(z)$ & $z \in \mathcal{E}(x) = X^x_I$ as in (b).

In both cases, there is exactly one *representative superstate* of y in X^x_I. Denote the representative superstate $\hat{y}$ by

$$\hat{y} = \begin{cases} y, & \text{if } y \in X^x_I \\ z, & \text{if } y \in \mathcal{E}(z) \text{ and } z \in X^x_I \end{cases}$$

Then the pair (H^x, H^y) is *boundary consistent* if

- (State consistency) The external states of H^y are those connected with the superstate $\hat{y}$ at H^x, i.e.,

$$X^y_E = \{a \in X^x_I | (\exists \sigma \in \Sigma^x_I) \delta^x_I(a, \sigma) = \hat{y} \text{ or } \delta^x_I(\hat{y}, \sigma) = a\}.$$

- (Event consistency) The boundary events of H^y are internal events of H^x, i.e., $\Sigma^y_B \subseteq \Sigma^x_I$. More precisely, the boundary events are those events which point to or leave the superstate $\hat{y}$ at H^x, i.e.,

$$\Sigma^y_B = \{\sigma \in \Sigma^x_I | (\exists a \in X^x_I) \delta^x_I(a, \sigma) = \hat{y} \text{ or } \delta^x_I(\hat{y}, \sigma) = a\}.$$

- (Boundary transition consistency) The incoming/outgoing boundary transitions of H^y are consistent with those of the superstate $\hat{y}$ at H^x, i.e.,

$$(\forall a \in X^y_E, \sigma \in \Sigma^y_B)(\exists b \in X^y_o)(\delta^y_{BI}(a, \sigma) = b \text{ iff } \delta^x_I(a, \sigma) = \hat{y}).$$

$$(\forall a \in X^y_E, \sigma \in \Sigma^y_B)(\exists b \in X^y_m)(\delta^y_{BO}(b, \sigma) = a \text{ iff } \delta^x_I(\hat{y}, \sigma) = a).$$

◇

It is very easy to verify boundary consistency. Intuitively, it means you can "plug" the low level holon into the high level holon without changing the boundary transitions of its representative superstate in the high level.

Definition 5 (State Tree Structure (STS)) A *state tree structure* (STS) is a 6-tuple $(\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_o, \mathcal{ST}_m)$, where

- $\mathbf{ST} := (X, x_o, \mathcal{M}, \mathcal{E})$ is a state tree;
- $\mathcal{H} := \{H^a | a \in X \;\&\; \mathcal{M}(a) = or\}$ is the set of holons assigned to the OR superstates in $\mathbf{ST}$;
- Σ is the set of events occurring in $\mathcal{H}$;
- $\Delta : \mathcal{ST}(\mathbf{ST}) \times \Sigma \to \mathcal{ST}(\mathbf{ST})$ is the transition function;
- $\mathbf{ST}_o \in \mathcal{ST}(\mathbf{ST})$ is the *initial state tree*;
- $\mathcal{ST}_m \subseteq \mathcal{ST}(\mathbf{ST})$ is the *marker state tree set*.

$\mathbf{G} = (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_o, \mathcal{ST}_m)$ is a *state tree structure* if

1. (Boundary consistency) all parent-child pairs in $\mathcal{H}$ are boundary consistent, and
2. (Loose coupling) events of inner transition structure can only be shared among those holons matched to the children of an AND superstate. Formally, for all superstates $a \neq b$ with matching holons $H^a, H^b \in \mathcal{H}$, we require

$$\Sigma^a_I \cap \Sigma^b_I \neq \emptyset \Rightarrow (\exists z \in X, \mathcal{M}(z) = and) a \in \mathcal{E}(z) \;\&\; b \in \mathcal{E}(z).$$

◇

Event sharing in a STS is bounded vertically by boundary consistency and horizontally by loose coupling. This modularity makes recursive synthesis of STS possible.

The dynamics of STS, given by the Δ function, is defined as follows. [2] Let $\mathbf{ST}_1 \in \mathcal{ST}(\mathbf{ST})$ and $\sigma \in \Sigma$. Define the total function [3]

$$\Delta(\mathbf{ST}_1, \sigma) := \text{replace\_source}_{\mathbf{G},\sigma}(\mathbf{ST}_1 \wedge \text{Elig}_{\mathbf{G}}(\sigma)).$$

$\text{Elig}_{\mathbf{G}}(\sigma) \in \mathcal{ST}(\mathbf{ST})$ is the largest sub state tree of $\mathbf{ST}$ that allows σ to happen. $\mathbf{ST}_1 \wedge \text{Elig}_{\mathbf{G}}(\sigma)$ is also a sub state tree

[1] $\delta_i : X \times \Sigma \to X, i = 1, 2$ are disjoint if the sets $\{(x, \sigma, \delta_1(x, \sigma)) | \delta_1(x, \sigma)!\}$ and $\{(x, \sigma, \delta_2(x, \sigma)) | \delta_2(x, \sigma)!\}$ are disjoint, i.e., δ_1, δ_2 have no transitions in common.

[2] Space does not permit explanation in detail.

[3] $\Delta(\mathbf{ST}_1, \sigma)$ is the empty state tree if σ cannot occur on $\mathbf{ST}_1$.

because $(\mathcal{ST}(\mathbf{ST}), \leq)$ is a lattice. If p is any source state on the tree $\mathbf{ST}_1 \wedge \mathrm{Elig}_{\mathbf{G}}(\sigma)$ such that $\delta^x(p,\sigma)!$ for some holon H^x, the function replace_source$_{\mathbf{G},\sigma}(\cdot)$ just replaces p by its target state $\delta^x(p,\sigma)$ to get another sub state tree. In other words, $\Delta(\mathbf{ST}_1,\sigma)$ is the largest sub state tree (wrt. $\leq$) in which the system could reside if event σ occurred at $\mathbf{ST}_1$. Because in general count$(\mathbf{ST}_1) \geq 1$, one can look on $\mathbf{ST}_1$ as the "symbol" of $\mathcal{B}(\mathbf{ST}_1)$, its set of basic sub state trees. So $\Delta(\mathbf{ST}_1,\sigma)$ takes care of all transitions labelled by σ in the set $\mathcal{B}(\mathbf{ST}_1)$, i.e., our Δ function is computationally more efficient than the δ function of FSM, which computes only one transition at a time.

A simple example is shown in Fig. 5. We use the graphical notation of statecharts (see [4]) to draw our STS model in (a). Its state tree is given in (b), while (c) and (d) display what the target state trees will be if events β and σ occur at $\mathbf{ST}_\beta$ and $\mathbf{ST}_\sigma$, respectively.

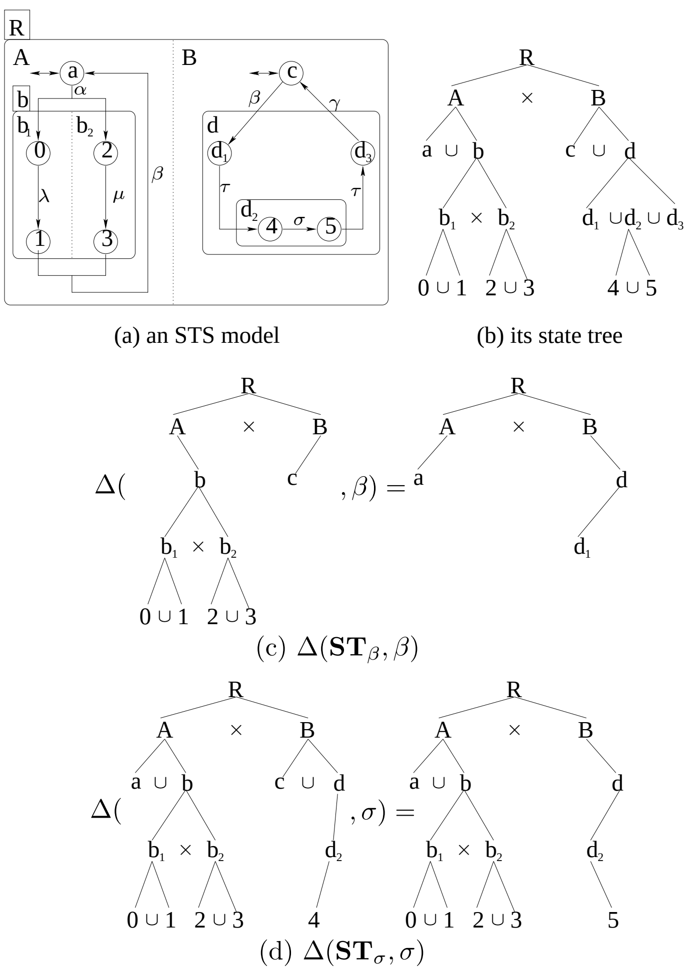

Fig. 5. Example for the Δ function

Following a dual route, we can define the function Γ, which is more important for the synthesis of STS.

Definition 6 (Γ) Let $\mathbf{ST}_1 \in \mathcal{ST}(\mathbf{ST})$ and $\sigma \in \Sigma$. Define

$$\Gamma(\mathbf{ST}_1,\sigma) := \mathrm{replace\_target}_{\mathbf{G},\sigma}(\mathbf{ST}_1 \wedge \mathrm{Next}_{\mathbf{G}}(\sigma)).$$

$\diamond$

$\mathrm{Next}_{\mathbf{G}}(\sigma) := \Delta(\mathbf{ST},\sigma)$ is the largest sub state tree of $\mathbf{ST}$ that the event σ is targeting. The function replace_target$_{\mathbf{G},\sigma}(\cdot)$ just replaces target states with source states to get a new sub state tree. Thus, $\Gamma(\mathbf{ST}_1,\sigma)$ is the largest sub state tree of $\mathbf{ST}$ that could reach a sub state tree of $\mathbf{ST}_1$ if event σ occurs. Normally $\Gamma(\Delta(\mathbf{ST}_1,\sigma),\sigma) \neq \mathbf{ST}_1$. We call Γ a *backward transition function.*

III. Symbolic Representation of STS

A. Encode State Space

Let $\mathbf{G} = (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_o, \mathcal{ST}_m)$ be a state tree structure with $\mathbf{ST} = (X, x_o, \mathcal{M}, \mathcal{E})$. A predicate P defined on $\mathcal{B}(\mathbf{ST})$ (or simply on $\mathbf{ST}$) is a function $P : \mathcal{B}(\mathbf{ST}) \to \{0,1\}$. Also a predicate can be *identified* by a set of basic state trees, say B_P, such that $B_P := \{b \in \mathcal{B}(\mathbf{ST}) | P(b) = 1\}$.

Let $b \in \mathcal{B}(\mathbf{ST})$. We say predicate P *holds*, or is *satisfied* for b, i.e., $b \models P$ if and only if $b \in B_P$. Similarly, predicate P *holds*, or is *satisfied* for $\mathbf{ST}_1$, i.e., $\mathbf{ST}_1 \models P$ if and only if $\mathcal{B}(\mathbf{ST}_1) \subseteq B_P$. Write $Pred(\mathbf{ST})$ for the set of all predicates defined on $\mathbf{ST}$. Standard logic operators are defined in the usual way:

$$\begin{array}{lll} (\neg P)(b) = 1 & \text{iff} & P(b) = 0 \\ (P_1 \wedge P_2)(b) = 1 & \text{iff} & P_1(b) = 1 \text{ and } P_2(b) = 1 \\ (P_1 \vee P_2)(b) = 1 & \text{iff} & P_1(b) = 1 \text{ or } P_2(b) = 1 \end{array}$$

We define a partial order on $Pred(\mathbf{ST})$ from subset containment: $P_1 \preceq P_2$ iff $P_1 \wedge P_2 = P_1$; say P_1 is a *subpredicate* of P_2. Under this definition, $(Pred(\mathbf{ST}), \preceq)$ is a lattice. The top element $\top$ is identified with $B_\top = \mathcal{B}(\mathbf{ST})$ and the bottom element $\perp$ with $B_\perp = \emptyset$.

The following function Θ encodes a sub state tree of $\mathbf{ST}$ to the predicate it satisfies.

Definition 7 (Θ) Denote by v_x the state variable for the OR superstate x . Let $\mathbf{ST}_1 = (X_1, x_o, \mathcal{M}_1, \mathcal{E}_1)$ be a sub state tree of $\mathbf{ST}$. Define $\Theta : \mathcal{ST}(\mathbf{ST}) \longrightarrow Pred(\mathbf{ST})$ recursively by $\Theta(\mathbf{ST}_1) :=$

$$\begin{cases} \bigwedge_{y\in\mathcal{E}_1(x_o)} \Theta(\mathbf{ST}_1^y), & \text{if } M(x_o) = and \\ \bigvee_{y\in\mathcal{E}_1(x_o)} ((v_{x_o} = y) \ \wedge \Theta(\mathbf{ST}_1^y)), & \text{if } M(x_o) = or \\ 1, & \text{if } M(x_o) = simple \end{cases},$$

where $\mathbf{ST}_1^y$ denotes the child state tree of $\mathbf{ST}_1$ that is rooted by y. Notice that if x_o is an OR superstate, we can exploit the tautology [4]

$$(\bigvee_{y\in\mathcal{E}(x_o)} (v_{x_o} = y)) = 1$$

to simplify $\Theta(\mathbf{ST}_1)$.

$\diamond$

For example, in (d) of Fig. 5,

$$\Theta(\mathbf{ST}_\sigma) := (v_B = c) \vee (v_B = d \wedge v_d = d_2 \wedge v_{d_2} = 4),$$

where v_B, v_d, v_{d_2} are state variables for states B, d, d_2, respectively. The variables for the OR superstates on the

[4]This says that the predicate $\Theta(\mathbf{ST}_1)$ is independent of the state variable v_{x_o} if all descendants of x_o are on the state tree.

child state tree $\mathbf{ST}^A_\sigma$ do not appear in $\Theta(\mathbf{ST}_\sigma)$ because the above tautology has been applied.

Now for any set B_P of basic state trees, its predicate representation is given by $P := \bigvee_{b \in B_P} \Theta(b)$. If $B_P = \mathcal{B}(\mathbf{ST}_1)$, then $P := \Theta(\mathbf{ST}_1)$.

In the STS $\mathbf{G}$, the *initial predicate* $P_o := \Theta(\mathbf{ST}_o)$, and the *marker predicate* $P_m := \bigvee_{\forall \mathbf{ST}_i \in \mathcal{ST}_m} \Theta(\mathbf{ST}_i)$. Now we can rewrite the plant STS by $\mathbf{G} = (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, P_o, P_m)$, where $\mathbf{ST}_o$ and $\mathcal{ST}_m$ are replaced by their predicate counterparts.

B. *Encode* Γ

Let x be an OR superstate. Call v_x the *normal* state variable of x. Also denote by $v_x{}'$ the *prime* state variable of x. In a transition relation, v_x will be used to record target state information, while $v_x{}'$ is for source states.

Following the method of [14], we encode the entire set of transitions labelled by a given event σ as a triple $(N_\sigma, \mathbf{v}_{\sigma,\mathrm{s}}, \mathbf{v}_{\sigma,\mathrm{t}})$. $N_\sigma(\mathbf{v}_{\sigma,\mathrm{s}}{}', \mathbf{v})$ is the transition relation with $\mathbf{v}_{\sigma,\mathrm{s}}{}'$ the set of prime variables for the source states where $\delta(\cdot,\sigma)!$ and $\mathbf{v}$ the set of all variables in $\mathbf{G}$. $\mathbf{v}_{\sigma,\mathrm{t}}$ is the set of normal variables for those target states which $\delta(\cdot,\sigma)$ hits (takes a value).

Definition 8 ($\hat{\Gamma}$) Let $\sigma \in \Sigma$. Let $(N_\sigma, \mathbf{v}_{\sigma,\mathrm{s}}, \mathbf{v}_{\sigma,\mathrm{t}})$ represent the transitions labelled with σ. $\hat{\Gamma} : \mathit{Pred}(\mathbf{ST}) \times \Sigma \to \mathit{Pred}(\mathbf{ST})$ is defined by [5]

$$\hat{\Gamma}(P,\sigma) := (\exists \mathbf{v}_{\sigma,\mathrm{t}}(P \wedge N_\sigma))[\mathbf{v}_{\sigma,\mathrm{s}}{}' \to \mathbf{v}_{\sigma,\mathrm{s}}].$$

We can omit ˆ to write $\Gamma(P,\sigma)$ if no ambiguity.

◇

Notice that $P[\mathbf{v}_{\sigma,\mathrm{s}}{}' \to \mathbf{v}_{\sigma,\mathrm{s}}]$ means replacing all prime variables in $\mathbf{v}_{\sigma,\mathrm{s}}{}'$ by their respective normal variables in $\mathbf{v}_{\sigma,\mathrm{s}}$.

Our computation of Γ is slightly different from the one given in [14] but very computationally efficient because (1) the expensive existential operator is applied to a smaller variable set $\mathbf{v}_{\sigma,\mathrm{t}}$ instead of the entire variable set $\mathbf{v}$; and (2) we have a much smaller transition relation N_σ because it does not have redundant terms such as $(v_x = v_x{}')$ if v_x does not change value under any transition labelled by σ.

IV. Symbolic Synthesis of STS

A specification $\mathcal{S}$ will be given as a set of *illegal* sub state trees of $\mathbf{ST}$. Of course, $\mathcal{S}$ can be encoded by a suitable predicate P. Then our synthesis objective is to find the largest subpredicate of $\neg P$ that is nonblocking and controllable.

In this section, we demonstrate our hierarchical control of STS by the structural computation of the *coreachability predicate* $\mathit{CR}(\mathbf{G},P)$.

Definition 9 ($\mathit{CR}(\mathbf{G},P)$) Let $P \in \mathit{Pred}(\mathbf{ST})$. The *coreachability predicate* $\mathit{CR}(\mathbf{G},P)$ is defined to hold all basic state trees that can reach $b_m \models P_m$ via trees satisfying P, according to the inductive definition:

1. $(b_m \models P_m \wedge P) \Rightarrow b_m \models \mathit{CR}(\mathbf{G},P)$
2. $b \models \mathit{CR}(\mathbf{G},P)$ & $b \neq \emptyset$ & $\sigma \in \Sigma$ & $\Delta(b',\sigma) = b$ & $b' \models P \Rightarrow b' \models \mathit{CR}(\mathbf{G},P)$
3. No other basic state trees satisfy $\mathit{CR}(\mathbf{G},P)$.

◇

From the definition, we can immediately provide a 'naive' algorithm.

Algorithm 1: (Basic algorithm for $\mathit{CR}(\mathbf{G},P)$)
1. $K_o := P \wedge P_m$.
2. $K_{i+1} := K_i \vee (P \wedge \bigvee_{\sigma\in\Sigma} \Gamma(K_i,\sigma))$.
3. If $K_{n+1} = K_n$, then $\mathit{CR}(\mathbf{G},P) := K_n$. Otherwise go to step 2.

◇

Our experience shows that the basic algorithm can succeed for moderately complex systems with state space size as large as 10^8. However, during the synthesis, the number of BDD nodes in the intermediate predicates is much larger than that in $\mathit{CR}(\mathbf{G},P)$ (cf. also [15]). If we can "control" the BDD size for intermediate predicates, we can handle much bigger systems; to some extent this is achieved by our second algorithm.

Algorithm 2 (Recursive algorithm for $\mathit{CR}(\mathbf{G},P)$)
Let $P, R \in \mathit{Pred}(\mathbf{ST})$. Denote $\mathit{CR}^x(\mathbf{G},P,R)$ as the fixpoint for the superstate x such that for each $b_o \models \mathit{CR}^x(\mathbf{G},P,R)$, there is a sequence $\{b_i | i = 0,1,\ldots,n\}$ for b_o to reach $b_n \models R$ by local transitions in x and every b_i satisfies P. Precisely, we have

1: **function** $\mathit{CR}^x(\mathbf{G},P,R) =$
2: $\quad K \leftarrow R$
3: $\quad$ **if** $\mathcal{M}(x) = \mathit{or}$ **then**
4: $\quad\quad$ **repeat**
5: $\quad\quad\quad K' \leftarrow K$
6: $\quad\quad\quad K \leftarrow K \vee (P \wedge \bigvee_{\sigma\in\Sigma^x_I} \Gamma(K,\sigma))$
7: $\quad\quad\quad$ **foreach** $y \in \mathcal{E}(x)$ & $\mathcal{M}(y) \in \{\mathit{or}, \mathit{and}\}$
8: $\quad\quad\quad\quad K \leftarrow \mathit{CR}^y(\mathbf{G},P,K)$
9: $\quad\quad$ **until** $K' = K$
10: $\quad$ **else if** $\mathcal{M}(x) = \mathit{and}$ **then**
11: $\quad\quad$ **repeat**
12: $\quad\quad\quad K' \leftarrow K$
13: $\quad\quad\quad$ **foreach** $y \in \mathcal{E}(x)$
14: $\quad\quad\quad\quad K \leftarrow \mathit{CR}^y(\mathbf{G},P,K)$
15: $\quad\quad$ **until** $K' = K$
16: **end** $\mathit{CR}^x(\mathbf{G},P,K)$
17: **return** K

◇

Remark
1. $\mathit{CR}(\mathbf{G},P) := \mathit{CR}^{x_o}(\mathbf{G},P,P \wedge P_m)$.
2. The computation takes care of two cases. If x is OR state, line 4-9 computes the fixpoint. If x is AND state, line 11-15 does the job. The computation will terminate as the change of K is monotone and the system is finite.
3. It is a recursive algorithm. Each fixpoint $\mathit{CR}^x(\mathbf{G},P,K)$ depends on $(\forall y)\mathit{CR}^y(\mathbf{G},P,K)$, where y is a superstate-child of x. All of x's descendants must have been computed too before getting $\mathit{CR}^x(\mathbf{G},P,K)$. The reason for choosing this direction is twofold. First, we want to apply the tautology $(\bigvee_{y\in\mathcal{E}(x)} (v_x = y)) = 1$ as early as possible, to control the size of intermediate predicates. If y is a superstate under x, we want to compute $\mathit{CR}^y(\mathbf{G},P,K)$ first to try to quantify out the state variables inside y by applying the

[5] For the detailed explanation of this formula, refer to [14].

above tautology, and therefore make it possible to apply the above tautology to x too. Second, by having a largest fixpoint $CR^x(\mathbf{G}, P, K)$, we add into K as many basic state trees as possible from the computation of transitions under x. Then in the subsequent computation of another superstate z, each call of Γ can add more basic state trees to speed up the computation, because Γ is monotone in the sense that $K_1 \preceq K_2 \Rightarrow (\forall\sigma)\Gamma(K_1, \sigma) \preceq \Gamma(K_2, \sigma)$.
4. This recursive algorithm requires iterations back and forth between x and its descendants. The fewer the iterations, the faster the algorithm terminates. Some efficient techniques have been applied in our computer program to limit the number of iterations.

V. AIP Example

The AIP example is first presented by Brandin *et al.* in [16]. The whole system is modelled as the synchronous product of 100 automata. Modular control has been applied to the synthesis. However, modular control is not yet a completely formal method, because it involves human judgement during the synthesis, and so is error-prone. In this example, we compute the controller without human intervention.

Leduc *et al.* [3] successfully applied *Hierarchical Interface-based Supervisory Control* theory to the AIP example. That program could automatically synthesize a controller for a simplified version of the AIP example, where the maximum allowed number of pallets on external loop 1 or 2 was limited to be one (the original requirement was ten). However, without using symbolic computation, the program exceeded memory when buffer size was increased from 1 to 2.

We model the AIP example with Brandin's original requirement as a STS with state space of order 10^{24}. Our BDD-based program can automatically compute the controller in around 45 minutes, using a fixed allocation of memory (required by the BDD package we use) on an ordinary personal computer with Athlon CPU and 256MB RAM. The BDD size of the intermediate predicates during the computation of $CR(\mathbf{G}, P)$ is sampled and displayed in Fig. 6. Notice that the BDD size is fairly well controlled, the maximum size being just below 3 times that of the resulting BDD.

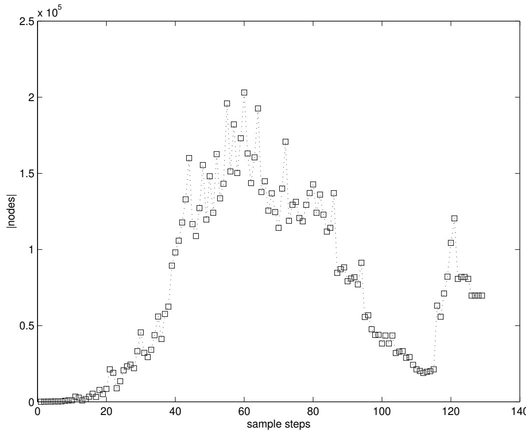

Fig. 6. The BDD size of intermediate predicates

VI. Conclusions

In this paper, we introduced a state-based version of Wang's State Tree Structure (STS). In order to perform control design efficiently, we employed a symbolic representation of STS and developed a recursive algorithm that succeeds with the AIP example having state space of order 10^{24}. The state space explosion problem is effectively controlled.

The symbolic approach should clear the way for the industrial application of our STS framework. Work continues on new recursive algorithms to make the synthesis faster and less demanding of memory.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[2] Peyman Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on DES*, vol. 30(5), pp. 643–652, 2000.
[3] R.J. Leduc, M. Lawford, and W.M. Wonham, "Hierarchical interface-based supervisory control: AIP example," in *Proc. of the 39th Allerton Conf. on Comm., Contr., and Comp*, October 3-5, 2001, pp. 396 – 405.
[4] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, June 1987.
[5] B. Wang, "Top-down design for RW supervisory control theory," M.S. thesis, Department of Electrical and Computer Engineering, Univ. of Toronto, 1995.
[6] Arthur Koestler, *The Ghost in the Machine*, Penguin Group Ltd., 1989.
[7] Ling Gou, Tetsuo Hasegawa, and Peter Luh, "Holonic planning and scheduling for a robotic assembly testbed," in *Proc. of the fourth international conference on computer integrated manufacturing and automation technology*, Oct. 10-12, 1994, pp. 142–149.
[8] Peyman Gohari and W. M. Wonham, "A linguistic framework for controlled hierarchical DES," in *4th International Workshop on Discrete Event Systems (WODES '98), IEE,*, 1998, pp. 207–212.
[9] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," *IEEE Transactions on Automatic Control*, vol. 38, no. 12, pp. 1803–1819, Dec. 1993.
[10] H. Marchand and B. Gaudin, "Supervisory Control Problems of Hierarchical Finite State Machines," *Proc. of the 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada USA, 2002, pp. 1199–1204.
[11] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98, pp. 142–170, June 1992.
[12] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, August 1986.
[13] G. Hoffmann and H. Wong-Toi, "Symbolic synthesis of supervisory controllers," in *Proc. of 1992 American Control Conference*, Chicago, IL, USA, 1992, pp. 2789–2793.
[14] Sergey Berezin, Sérgio Campos and Edmund M. Clarke, "Compositional Reasoning in Model Checking," in *Lecture Notes in Computer Science*, vol. 1536, pp. 81–102, 1998.
[15] Z.H. Zhang and W.M. Wonham, "STCT: An efficient algorithm for supervisory control design," in *Symposium on Supervisory Control of Discrete Event Systems*, Paris, July, 2001.
[16] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 1994 4th Intl. Conf. on Computer Integrated Manufacturing and Automation Technology*, Troy, 1994, pp. 319–324.