State Tree Structures

F.J.W. Zeelen (Frank) 514785

August 2007

...

Contents

Chapter 1
State Tree Structures (STS)

In SCT a supervisor is computed that restricts the behaviour of an uncontrolled system, the *plant*, to meet some behavioral requirements, the *specifications*, and to ensure that the controlled system is deadlock free. This task is called supervisor synthesis and is essentially a series of reachability searches. In large industrial cases, this method is often impossible or infeasible to execute due to the state-space explosion problem. By using symbolic computation the size of the state-spaces that can be explored in reasonable time and memory usage can be much larger and interesting for industrial cases (Vahidi, Fabian and Lennartson, 2006). Another way to improve the synthesis is by structuring the system models by using hierarchy and concurrency. State tree structures (STS) are a structured and compact representation of discrete-event systems that include both hierarchy and concurrency. A state tree (ST) is a formal description of a state space based on *Statecharts* (Harel (1987)). The local behaviour of a ST can be described by an automata, which is called a *holon*. Wang (1995) introduced STS and combined the structure of state trees with holons that describes their local behaviour. Ma and Wonham (2005) further refined the STS for supervisor synthesis.

The STS as described in this chapter are based on the work of Ma and Wonham (2005). We start this chapter by introducing basic State Trees in Section 1.1. Second, Section 1.2 introduces holons that describe the local behaviour of a cluster of states within a ST. The global behaviour of a ST equipped with holons is described in Section 1.3. Using this global behaviour, the non-blocking supervisory control of STS is discussed in Section 1.4. To see how STS work in practice, we use the non-blocking control of STS for the supervisory control of the reject wafer logistics case in Section **??**. Finally, we conclude this chapter in Section 1.5.

1.1 State Trees

An intuitive and natural way of describing the dynamic behavior and structure of such systems are *state transitions diagrams* (Cassandras and Lafortune, 2004). Representing large systems using this type of diagram generally suffers from an exponential growth in the number of states. This phenomenon is also known as the *state-space explosion* problem. This problem is caused by the fact that the states in a state transition diagram are represented using only exclusive *OR* nodes. That is, the system is either in state x *or* in state y but not in both. Because of this exclusive representation, the state space of a DES can be too large to explicitly store each state. State trees differ from state transition diagrams in the sense that they represent the state space using both *OR* and *AND state clusters* that are called *superstates*.

Definition (OR superstate)
Let X be a finite set of states, state $Y = \{y_1, y_2, \ldots, y_{n-1}, y_n\} \in \mathcal{P}(X)\backslash\{\emptyset\}$, $x \in X$ and and $x \notin Y$ then state x is defined as an OR superstate iff:

$$x = \bigcup_{\forall y \in Y} y \tag{1.1}$$

In words, x is the disjoined union of all states in Y. Here, each $y \in Y$ is called an *OR component* of x. For the system to be in OR superstate x it must be in *exactly* one state of Y. Hence, each OR superstate represents a cluster of exclusive states, i.e. hierarchy.

▪

Definition (AND superstate)
Let X be a finite set of states, state, $Y = \{y_1, y_2, \ldots, y_{n-1}, y_n\} \in \mathcal{P}(X)\backslash\{\emptyset\}$, $x \in X$ and $x \notin Y$ then state x is defined as an AND superstate iff:

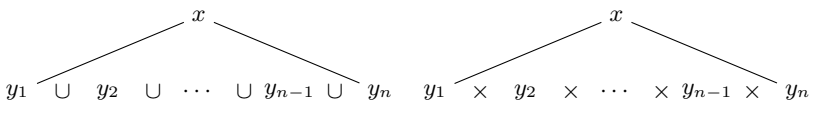
$$x = \prod_{\forall y \in Y} y \tag{1.2}$$

In words, x is the Cartesian product of all states in Y. Here, each y is called an *AND component* of x. For the system to be in AND superstate x it must be in *all* states of Y simultaneously. Hence, each AND superstate represents a cluster of parallel states, i.e. concurrency.

▪

Examples of graphically representing AND and OR superstates are given in Figure 1.1. In both so-called *expansions*, superstate x is called the parent of y and y is called the child of x. All states other then superstates, i.e. states that have no children, are called *simple states*. Using these different types of states, X is defined as a *structured state set* in which hierarchy and concurrency are included.

Graphical representation of AND and OR superstates



(a) OR superstate x (b) AND superstate x

Figure 1.1

To formally define this structured representation of superstates, two functions are introduced: (i) the *type function* that returns the type of a state and (ii) the *expansion function* that returns the set of all child states of a state.

Definition (*type function:* $\mathcal{T}(x)$) $\mathcal{T} : X \to \{and, or, simple\}$ such that for all $x \in X$:

$$\mathcal{T}(x) = \begin{cases} and, & \text{if } x \text{ is an AND superstate} \\ or, & \text{if } x \text{ is an OR superstate} \\ simple, & \text{otherwise} \end{cases} \tag{1.3}$$

▪

Definition (*expansion function:* $\varepsilon(x)$) $\varepsilon : X \to \mathcal{P}(X)$ such that for all $x \in X$:

$$\varepsilon(x) = \begin{cases} Y = \{y | \text{ iff } y \text{ is a child of } x\}, & \text{if } \mathcal{T}(x) \in \{and, or\} \\ \emptyset, & \text{if } \mathcal{T}(x) = simple \end{cases} \tag{1.4}$$

In words, the expansion function $\varepsilon(x)$ returns the *complete set of child states* of state x.

▪

Definition ($\varepsilon^*(x)$) The transitive and reflexive closure $\varepsilon^*(x)$ of $\varepsilon(x)$ is computed recursively, with $n, m \in \mathbb{N}^+$, as:

$$\begin{array}{rll} \text{(basis)} & \varepsilon^0(x) = \{x\}, & \text{where } x \in X \\ \text{(recursive step)} & \varepsilon^n(x) = \varepsilon^0(x) \cup \bigcup_{\forall y \in \varepsilon^{n-1}(x)} \varepsilon(x) & \\ \text{(closure)} & \varepsilon^m(x)\backslash\varepsilon^{m+1}(x) = \emptyset, & \text{where } m \leq |X| - 1 \end{array} \tag{1.5}$$

In words, $\varepsilon^*(x)$ is the set of *all descendants* of state x (transitive) including x itself (reflexive).

▪

The state $x \in X$ for which $\mathcal{E}^*(x) = \{X\}$ holds is called the root state and is often denoted as x_0. With this root state, all formal elements of a state tree are defined.

Definition (state tree) A state tree is a 4-tuple $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$, where:

- $x_0 \in X$ is the root state.
- X is a finite structured state set, $\mathcal{E}^*(x_0) = X$.
- $\mathcal{T} : X \to \{and, or, simple\}$ is the type function.
- $\mathcal{E} : X \to \mathcal{P}(X)$ is the expansion function.

An empty state tree, i.e. $X = \emptyset$, is denoted as $\mathrm{ST} = \emptyset$.

▪

Graphical representation of a state tree

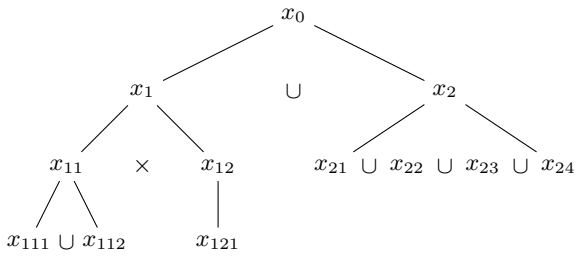


Figure 1.2

An example of a state tree is shown in Figure 1.2. Using this definition of a state tree, two individual states in a state tree can be compared to each other using the following relations.

Definition (partial order $\langle X; \leq \rangle$) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ and $x, y \in X$ then:

$$x \leq y \text{ iff } y \in \mathcal{E}^*(x) \tag{1.6}$$

is a partial order on the state set X. In words, $x \leq y$ means that x is equal to y or x is an ancestor of y. For those states $x, y \in X$ that cannot be compared, i.e. $x \not\leq y \vee y \not\leq x$, we write $x <> y$. When re-examining Figure 1.2 it follows that, for example, $x_1 \leq x_1$, $x_1 \leq x_{11}$ and $x_{12} <> x_{21}$.

▪

Definition (nearest common ancestor (NCA)) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ and $a, b, c \in X$ then c is the nearest common ancestor of a and b iff c satisfies the following two conditions:

$$\begin{array}{ll} \text{(i)} & (c < a) \wedge (c < b) \\ \text{(ii)} & \forall x \in \mathcal{E}^*(c), \quad (x \not< a) \vee (x \not< b) \end{array} \tag{1.7}$$

In words, c is the NCA of a and b iff: (i) c is an ancestor of both a and b (ii) no descendant of c is an ancestor of both a and b. From Figure 1.2 it follows that, for example, x_1 is the NCA of states x_{111} and x_{12}.

▪

Definition (parallel: $x \parallel y$) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ and $x, y \in X$ s.t. $x <> y$, then x and y are parallel iff the NCA of x and y is an AND superstate. From Figure 1.2 it follows that, for example, states x_{121} and x_{111} are parallel.

▪

Definition (exclusive: $x \mid y$) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \varepsilon)$, $x, y \in X$ and $x <> y$ then x and y are exclusive iff the NCA of x and y is an OR superstate. From Figure 1.2 it follows that, for example, states x_{21} and x_{22} are exclusive.

▪

The complete state space of a DES can be represented as a state tree. It is also possible to represent sub spaces of a state tree as state trees. We distinguish two types of such sub spaces: (i) child state trees and (ii) sub state trees.

Definition (child state tree) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \varepsilon)$ be a state tree then, $\mathrm{ST}^y = (\varepsilon^*(y), y, \mathcal{T}, \varepsilon)$ is called a child state tree of ST rooted at y iff $y \in X$. The reflexive and transitive closure of child state trees of ST is defined as $\mathrm{ST}^* = \{\mathrm{ST}^y | \forall y \in X\}$.

In words, a child state tree is defined by its root state, say y, and represents the state tree of *all* its descendants and itself, i.e. the state set $\varepsilon^*(y)$.

▪

Definition (sub state tree) A sub state tree, say ST_1, is *always* rooted at the root state of the complete state tree, say ST, and represents a subsection of a state tree by selecting a subset of children from each OR superstate of ST. Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \varepsilon)$ be a state tree and $X_1 \subseteq X$, then $\mathrm{ST}_1 = (X_1, x_0, \mathcal{T}, \varepsilon')$ is a proper sub state tree of ST iff $\forall q \in X_1$:

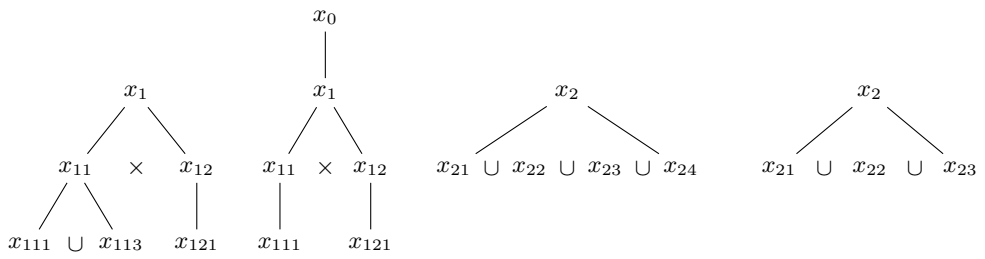
$$\varepsilon'(q) = \begin{cases} \varepsilon(q), & \text{if } \mathcal{T}(q) = and \\ Z, \text{ s.t. } Z \subseteq \varepsilon(q), & \text{if } \mathcal{T}(q) = or \\ \emptyset, & \text{if } \mathcal{T}(q) = simple \end{cases} \tag{1.8}$$

The reflexive and transitive closure of all sub state trees of ST is:
$\mathrm{ST}_* = \{\mathrm{ST}_1 | X_1 \subset X, X_1 = \varepsilon'^*(x_0) : \mathrm{ST}_1 = (\varepsilon'^*(x_0), x_0, \mathcal{T}, \varepsilon')\}$.

In words, a sub state tree ST_1 is proper sub state tree of ST iff for each AND superstate q, with $q \in X_1$, ST and ST_1 have the same set of child states, i.e. $\varepsilon'(q) \equiv \varepsilon(q)$ *and* iff for each OR superstate r, with $r \in X_1$, zero or more child states of ST are present in ST_1, i.e. $\varepsilon'(r) \subseteq \varepsilon(r)$.

▪

Examples of child state trees are depicted in Figure 1.3. Here, ST^{x_1} and ST^{x_2} are child state trees of the state trees shown in Figure 1.2. In this figure, there are also two sub state trees shown: (i) ST_1 denotes a sub state tree where the system is in parallel states x_{111} and x_{121} and (ii) $\mathrm{ST}_2^{x_2}$ denotes the sub state tree of child state tree ST^{x_2} in which the system is in exclusive state x_{21}, x_{22} or x_{23}.



Child and sub state trees of Figure 1.2

ST^{x_1} ST_1 ST^{x_2} $\mathrm{ST}_2^{x_2}$

Figure 1.3

The number of possible exclusive states represented by a (sub) state tree can be calculated using the $count(\mathrm{ST})$ function.

Definition (count(ST)) Let ST be a state tree and $\mathrm{ST}_1 \in \mathrm{ST}_*$. Then the count function $count(\mathrm{ST}_1)$ recursively counts the state size of ST_1 :

$$count(\mathrm{ST}_1) = \begin{cases} 0, & \text{if } \mathrm{ST}_1 = \emptyset \\ 1, & \text{if } \mathcal{T}(x_0) = simple \\ \prod_{\forall y \in \mathcal{E}'(x_0)} count(\mathrm{ST}_1^y) & \text{if } \mathcal{T}(x_0) = and \\ \sum_{\forall y \in \mathcal{E}'(x_0)} count(\mathrm{ST}_1^y) & \text{if } \mathcal{T}(x_0) = or \end{cases} \tag{1.9}$$

▪

Definition (basic state tree) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ be a state tree and $\mathrm{ST}_1 = (X_1, x_0, \mathcal{T}, \mathcal{E}')$ be a sub state tree of ST. Then ST_1 is a *basic state tree* of ST iff $count(\mathrm{ST}_1) = 1$. For convenience, we define the set of all basic state trees of ST as $\mathcal{B}(\mathrm{ST})$, s.t. $|\mathcal{B}(\mathrm{ST})| = count(\mathrm{ST})$.

In words, a basic sub state tree represents an *exclusive state* of the state space described in ST.

▪

Using the hierarchical structure of the state tree and its sub state trees, a partial order ($\leq$) on sub state trees can also be defined.

Definition (Partial order $\langle \mathrm{ST}_*; \leq \rangle$) Let $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ and $\mathrm{ST}_1, \mathrm{ST}_2 \in \mathrm{ST}_*$ then:

$$\mathrm{ST}_1 \leq \mathrm{ST}_2 \text{ iff } \mathrm{ST}_1 \text{ is a sub state tree of } \mathrm{ST}_2 \tag{1.10}$$

It is also possible to define this relation as: $\mathrm{ST}_1 \leq \mathrm{ST}_2$ iff $X_1 \subseteq X_2$. Since $\subseteq$ is a partial order on the state power set $\mathcal{P}(X)$ its easy to verify that $\leq$ is a partial order on ST_*.

▪

we also define two binary operations on sub state trees: (i) the join operation and (ii) the meet operation.

Definition (join operation ($\vee$)) Let ST be a state tree and let ST_1 and ST_2 be sub state trees, s.t. $\mathrm{ST}_1, \mathrm{ST}_2 \in \mathrm{ST}_*$. Then the join of these sub state trees, $\mathrm{ST}_1 \vee \mathrm{ST}_2$ always exists and is defined as:

(case 1) If $\mathrm{ST}_1 = \emptyset$ then:
$\mathrm{ST}_1 \vee \mathrm{ST}_2 = \mathrm{ST}_2$.

(case 2) If $\mathrm{ST}_2 = \emptyset$ then:
$\mathrm{ST}_1 \vee \mathrm{ST}_2 = \mathrm{ST}_1$.

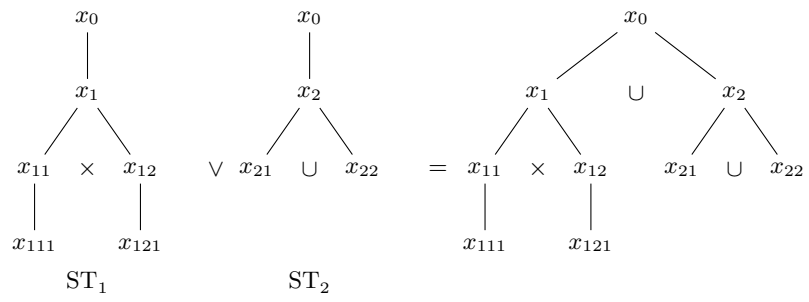
(case 3) If $\mathrm{ST}_1 \neq \emptyset$ and $\mathrm{ST}_2 \neq \emptyset$, then:
Let $\mathrm{ST}_1 = (X_1, x_0, \mathcal{T}, \mathcal{E}_1)$, $\mathrm{ST}_2 = (X_2, x_0, \mathcal{T}, \mathcal{E}_2)$ and define $\mathrm{ST}_3 = \mathrm{ST}_1 \vee \mathrm{ST}_2 = (X_3, x_0, \mathcal{T}_3, \mathcal{E}_3)$ such that $\forall x \in X_1 \cup X_2$:

$$\mathcal{E}_3 = \begin{cases} \mathcal{E}_1 & \text{if } x \in X_1 \backslash X_2 \\ \mathcal{E}_2 & \text{if } x \in X_2 \backslash X_1 \\ \mathcal{E}_1 \cup \mathcal{E}_2 & \text{if } x \in X_1 \cap X_2 \end{cases}$$

Define, $X_3 = \mathcal{E}_3^*(x_0)$ and $\mathcal{T}_3$ as the restriction of $\mathcal{T}$ to X_3 (1.11)

▪

An example of the join operation shown in Figure 1.4.



ST$_1 \vee$ ST$_2$

Figure 1.4

Definition (meet operation ($\wedge$)) Let ST be a state tree and ST$_1$ and ST$_2$ sub state trees, s.t. ST$_1$, ST$_2 \in$ ST$_*$. Then the meet of these sub state trees, ST$_3$ = ST$_1 \wedge$ ST$_2$ always exists and is defined as:

(case 1) If ST$_1 = \emptyset \vee$ ST$_2 = \emptyset$ then:
ST$_1 \wedge$ ST$_2 = \emptyset$.

(case 2) If ST$_1 \neq \emptyset$ and ST$_2 \neq \emptyset$, then:
Let ST$_1 = (X_1, x_0, \mathcal{T}, \varepsilon_1)$, ST$_2 = (X_2, x_0, \mathcal{T}, \varepsilon_2)$ and define ST$_3$ = ST$_1 \wedge$ ST$_2 = (X_3, x_0, \mathcal{T}, \varepsilon_3)$. Then $\forall y \in \varepsilon_1(x_0) \cap \varepsilon_2(x_0)$:

If $\mathcal{T}(y) = simple$ then state y is an element of $\varepsilon_3^*(x_0)$ (1.12)

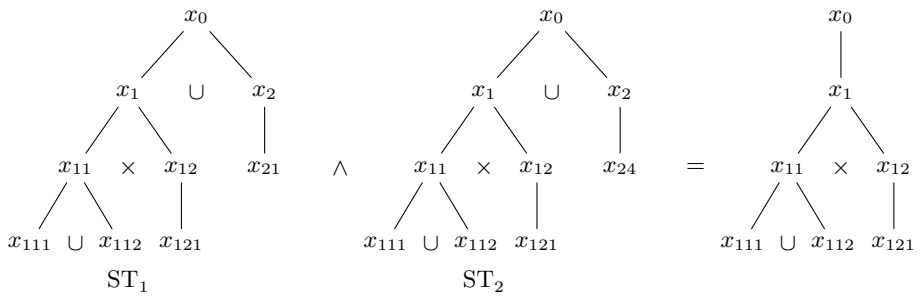
If $\mathcal{T}(y) = or$ then state y is an element of $\varepsilon_3^*(x_0)$ iff $\exists q \in \varepsilon_1^*(y) \cap \varepsilon_2^*(y) : \mathcal{T}(q) = simple$

If $\mathcal{T}(y) = and$ then state y is an element of $\varepsilon_3^*(x_0)$ iff $\varepsilon_1(y) \equiv \varepsilon_2(y) \equiv \varepsilon(y)$

Define, $X_3 = \varepsilon_3^*(x_0)$ and $\mathcal{T}_3$ as the restriction of $\mathcal{T}$ to X_3

▪

An example of the meet operation is shown in Figure 1.5.

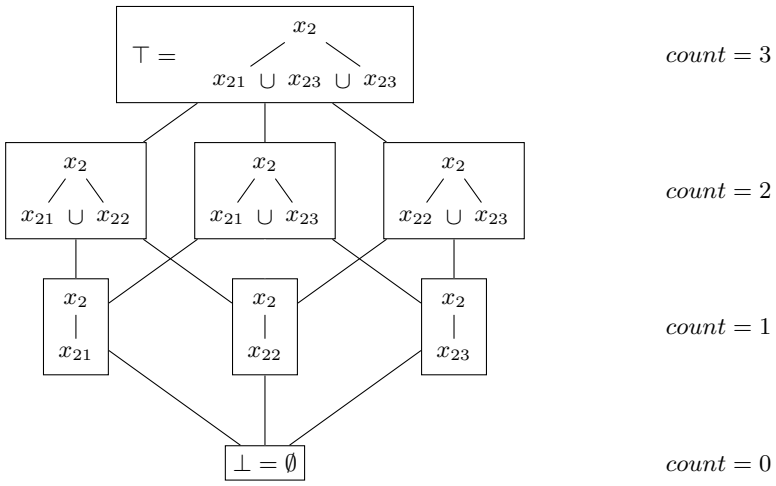


ST$_1 \wedge$ ST$_2$

Figure 1.5

In the partial ordered set of sub state trees $\langle$ST$_*; \leq\rangle$ there always exists a least element or bottom, i.e. $\perp = \emptyset$, and a greatest element or top, i.e. $\top$ = ST. For every pair of sub state trees ST$_1$, ST$_2 \in$ ST$_*$ we can also define an unique *least upper bound* and an unique *greatest lower bound* by using the join and meet operations respectively. Because these bounds always exist and the fact that they are unique for

each pair of sub state trees in ST_*, the ordered set of sub state trees is a lattice. From such a lattice, we can construct a Hasse diagram (Davey and Priestly, 2002). In this type of diagram, the nodes represent the elements and the edges represent the ordering of elements. For instance, consider sub state tree $\mathrm{ST}_2^{x_2}$ from Figure 1.3. Then, the Hasse diagram of the lattice of partially ordered sub state tree set $\langle \mathrm{ST}_2^{x_2}; \leq \rangle$ with binary operations meet ($\wedge$) and join ($\vee$) is shown in Figure 1.6.



Lattice of sub state tree $\mathrm{ST}_2^{x_2}$

Figure 1.6

This section defined state trees and relations over state trees. With these state trees, the state space of a DES can be efficiently structured and stored. The next step is to include a specification of the behaviour of DES. This is done by first defining holons that describe the local behaviour of an OR superstate and its children. Next, the global behaviour can be defined by 'connecting' the different holons according to the structure of the state tree.

1.2 Local behaviour

To describe the behaviour of a DES represented as a state tree, the local behaviour of each OR superstate and its children is described by a holon.

Definition (Holon) For an OR superstate $x \in X$, the holon H^x represents the transitions between the children of superstate x, i.e. internal transitions, and transitions *from* or *to* other holons, i.e. *external* or *boundary* transitions.

Formally a holon is defined as a 5-tuple: $H = (X, \Sigma, \delta, X_0, X_m)$ where:

- X is the state set of H. This state set is disjoint by the (possibly empty) external state set X_E and the non-empty internal state set X_I.
- Σ is the event set. This set is disjoint by the boundary events Σ_B and the internal event set Σ_I. The event set is also disjoint by the controllable and uncontrollable event sets Σ_c and Σ_u respectively.
- δ is the transition structure $\delta : X \times \Sigma \to X$ which is a partial function and for which we write $\delta(x, \sigma)!$ iff $\delta(x, \sigma)$ is defined. The transition function is disjoint by the internal (δ_I), incoming boundary (δ_{IB}) and outgoing boundary (δ_{OB}) transitions s.t:

$$\delta(x,\sigma) = \begin{cases} \delta_I(x,\sigma), & \text{if } x \in X_I \text{ and } \sigma \in \Sigma_I \\ \delta_{IB}(x,\sigma), & \text{if } x \in X_E \text{ and } \sigma \in \Sigma_B \\ \delta_{OB}(x,\sigma), & \text{if } x \in X_I \text{ and } \sigma \in \Sigma_B \end{cases} \tag{1.13}$$

- X_0 is the set of initial states. If there exist incoming boundary transitions then X_0 is exactly the set of states that can be reached by these transitions. Otherwise, X_0 is a nonempty subset of X.

- X_m is the set of marked (or final) states. If there exist outgoing boundary transitions then X_m is exactly the set of source states of these boundary transitions. Otherwise, X_m is a nonempty subset of X.

In words, a holon defines the internal and external behaviour of an OR-superstate.

▪

To check whether the internal and external transitions of a holon agree with the state structure as defined in the state tree a holon needs to be match to an OR superstate.

Definition (matching holon) Each holon needs to be *matched* with an OR superstate of the state tree. For a state tree $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ and state $y \in X$. Then a holon $H^y = (X^y, \Sigma^y, \delta^y, X_0^y, X_m^y)$ is matched with superstate y iff the following two conditions hold:

(i) The internal structure of H matches:

$$\begin{array}{rcl} X_I^y & = & \mathcal{E}(y) \\ X_0^y & \subseteq & \mathcal{E}(y) \\ X_m^y & \subseteq & \mathcal{E}(y) \\ \delta_I^y & : & X_I^y \times \Sigma_I^y \to X_I^y \end{array} \tag{1.14}$$

(ii) The external structure matches:

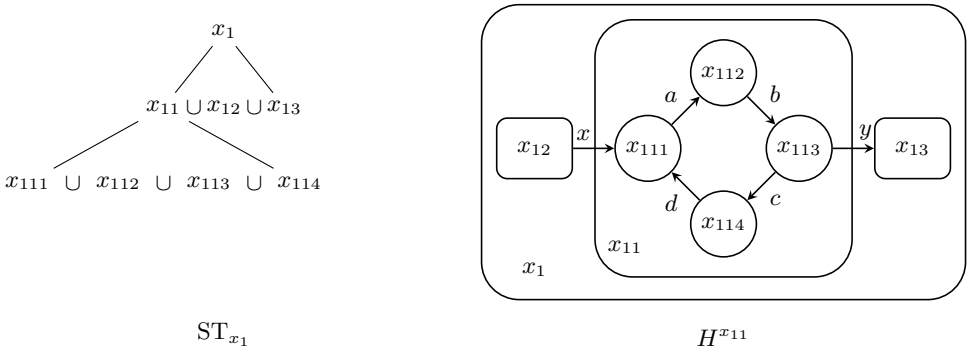
$$X_E^y = \begin{cases} \emptyset, & \text{if } y \text{ does not have an OR ancestor} \\ Z \text{ s.t. } Z \subseteq \mathcal{E}(x) \backslash \{y\}, & \text{if } y \text{ has an OR ancestor } x. \end{cases} \tag{1.15}$$

$$\begin{array}{rcl} \delta_{BI}^y & : & X_E^y \times \Sigma_B^y \to X_0^y \\ \delta_{BO}^y & : & X_m^y \times \Sigma_B^y \to X_E^y \end{array} \tag{1.16}$$

In words, for OR superstate x, with $x \in X$, the internal state set of H^x must be the same as the state set defined by the expansion function $\mathcal{E}(x)$ and the boundary transitions within a holon must be vertically bounded by its OR ancestor.

▪

In Figure 1.7, for example, the holon of a state x_{11} is given. In this figure, states x_{12} and x_{13} can be simple AND or OR superstates and state x and state y are boundary transitions. Notice that x_1 is the parent state of x_{11}, x_{12}, x_{13} and x_{14}.



State tree $\mathrm{ST}x_1$ **with matched holon** $H^{x_{11}}$

Figure 1.7

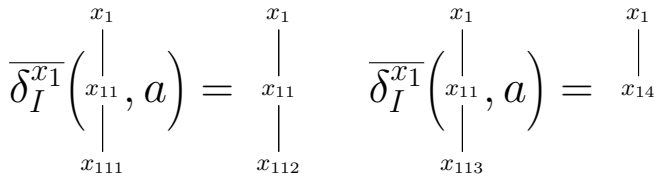
Definition ($\overline{\delta_I^x}$ local transition function) Let H^x be the holon matched to OR superstate $x \in \mathrm{ST}$ then:

$$\overline{\delta_I^x} : \mathcal{B}(\mathrm{ST}^x) \times \Sigma_I^x \to \mathcal{B}(\mathrm{ST}^x) \tag{1.17}$$

is a partial map that transforms a basic state tree into another basic state tree based on the transition structure of holon H^x. The difference between the normal transition function δ and transition function $\overline{\delta}$ is that the structure of the state tree is included in the transitions.

▪

Some examples of this function for the holon of Figure 1.7 are given in Figure 1.8.



Local transition function examples

Figure 1.8

With the state tree and its local behaviour (matched holons) we can now formally define a state tree structure and its global behaviour.

1.3 Global behaviour

An STS is a combination of a *state tree* (structure) and a set of *holons* (local behaviour) that describe the global behaviour of the system.

Definition (state tree structure) An STS is a 6-tuple $\mathbf{G} = (\mathrm{ST}, \mathcal{H}, \Sigma, \Delta, \mathrm{ST}_0, \mathcal{ST}_m)$ where:

- $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ is a state tree representing the state space.
- $\mathcal{H} = \{H^x | x \in X, \mathcal{T}(x) = \mathit{or} \text{ with } H^x = (X^x, \Sigma^x, \delta^x, X_0^x, X_m^x)\}$ is the set of matching holons for each OR superstate in ST.
- $\Sigma = \bigcup_{\forall H^x \in \mathcal{H}} \Sigma^x$ is the complete event set of the STS
- $\Delta : \mathrm{ST}_* \times \Sigma \to \mathrm{ST}_*$ the global transition function of ST.
- $\mathrm{ST}_0 \in \mathrm{ST}_*$ is the initial basic sub state trees.
- $\mathrm{ST}_m \subset \mathrm{ST}_*$ is the set of marked basic state trees.

▪

For an STS to be well-formed, it needs to satisfy some consistency rules. These rules define whether or not the states, events and boundary transitions are consistent with the structure of the state tree and whether the boundary events are loosely coupled. Loose coupling is a constraint on events that are shared such that events can only be shared among holons that have a common AND ancestor.

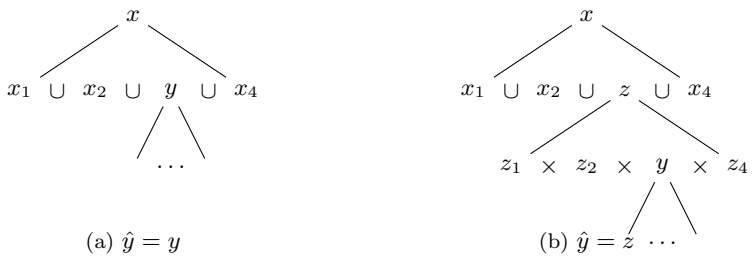
Definition (Consistency) A STS G is well-formed if all holon pairs H^x and H^y, with $x, y \in X$, in which state x is the nearest OR ancestor of y satisfy the (i) *boundary consistency rules* and (ii) if all shared events are *loosely coupled*. There are two possible cases for H^x to be the nearest OR ancestor of H^y:

$$\begin{array}{ll} \text{(case 1)} & y \in X_I^x \\ \text{(case 2)} & \exists z \big(\mathcal{T}(z) = \mathit{and} \wedge y \in \mathcal{E}(z) \wedge z \in X_I^x \big) \end{array} \tag{1.18}$$

In Figure 1.9 both case 1 and case 2 are shown in (a) and (b) respectively. For simplicity we define $\hat{y}$ as:

$$\hat{y} = \begin{cases} y, & \text{if } y \in X_I^x \\ z, & \text{if } y \in \mathcal{E}(z) \wedge z \in X_I^x \end{cases} \tag{1.19}$$

Parent child relation between H^x and H^y



(a) $\hat{y} = y$ (b) $\hat{y} = z$

Figure 1.9

(i) Boundary consistency

This constraint can be split into four sub-constraints: (a) state consistency, (b) event consistency, (c) incoming boundary transition consistency and (d) outgoing boundary transition consistency.

$\forall x, y \in X$ such that H^x is the nearest OR ancestor of H^y:

(a) state consistency:
state x and state y are state consistent if all external states in H^y are also in H^x, that is:

$$\forall a \in X_E^x, \exists \sigma \in \Sigma_I^x \big(\delta_I^x(a,\sigma) = \hat{y} \vee \delta_I^x(\hat{y},\sigma) = a\big) \tag{1.20}$$

(b) event consistency:
state x and state y are event consistent if all boundary events of y are internal events of x:

$$\forall \sigma \in \Sigma_E^x, \exists a \in X_I^x \big(\delta_I^x(a,\sigma) = \hat{y} \vee \delta_I^x(\hat{y},\sigma) = a\big) \tag{1.21}$$

(c) incoming boundary transition consistency:
All incoming boundary transitions of y must lead to an initial state:

$$\forall a \in X_E^y, \sigma \in \Sigma_B^y, \exists b \in X_0^y \big(\delta_{BI}^y(a,\sigma) = b \wedge \delta_I^x(a,\sigma) = \hat{y}\big) \tag{1.22}$$

(d) outgoing boundary transition consistency:
All outgoing boundary states must have a marker state as source state:

$$\forall a \in X_E^y, \sigma \in \Sigma_B^y, \exists b \in X_m^y \big(\delta_{BO}^y(b,\sigma) = a \wedge \delta_I^x(\hat{y},\sigma) = a\big) \tag{1.23}$$

(ii) Loose coupling

Events can only be shared among those holons that have the same *AND parent*. Thus, for $x, y \in X$, $x \neq y$ and $H^x, H^y \in \mathcal{H}$:

$$\Sigma_I^x \cap \Sigma_I^y = \emptyset \rightarrow \exists z \in X \big(\mathcal{T}(z) = and \wedge x \in \mathcal{E}(z) \wedge y \in \mathcal{E}(z)\big) \tag{1.24}$$

▪

Using these consistency rules, event sharing is bounded 'vertically' by the boundary consistency rules and 'horizontally' by the loose coupling rule.

The next obvious step should be to formally define the global transition function. But, since the supervisory controller synthesis in STS is symbolic and uses different type of functions, we prefer, for now, to give a informal view on how this function works.

Definition (global transition function $\Delta(\mathrm{ST}_1, \sigma)$) The global transition function $\Delta(\mathrm{ST}_1, \sigma)$ returns the target (basic) state tree given a source (basic) state tree and an event σ:

$$\Delta : \mathrm{ST}_* \times \Sigma \rightarrow \mathrm{ST}_* \tag{1.25}$$

Simply speaking, the target state tree is computed by first defining in which states event σ is eligible, say ST_σ. Then, the intersection of this so called eligible state tree and the source state tree is computed, $\mathrm{ST}_2 = \mathrm{ST}_1 \wedge \mathrm{ST}_\sigma$. Finally, the source state $x \in \mathrm{ST}_2$ of event σ is replaced with the target state(s) according to the local transition function $\overline{\delta}_I^x(\mathrm{ST}_2^x, \sigma)$.

▪

Definition (global reverse-transition function $\Gamma(\mathrm{ST}_1, \sigma)$) The global reverse-transition function is similar to $\Delta(\mathrm{ST}_1, \sigma)$ only this function replaces the target state tree ST_1 with its source state tree:

$$\Gamma : \mathrm{ST}_* \times \Sigma \rightarrow \mathrm{ST}_* \tag{1.26}$$

▪

1.4 Non-blocking supervisory control

In this section, the theory and methods behind non-blocking supervisory control of STS are explained. We begin this section with a short introduction to different reachability properties. After this introduction, the supervisor synthesis is explained in more detail. Finally, the algorithms behind the synthesis are explained.

For now, we will use state sets to represent predicates in this Section. If, for example we have state set P={1,2}, then we write predicate $P(x)$ such that $P = \mathit{true}$ iff $x \in P$ and *false* otherwise.

First, we will explain the basic idea behind the non-blocking, i.e. deadlock free, control of STS. In essence, supervisor synthesis is nothing more than a series of reachability searches. First, we define a predicate that describes the set of all states that satisfy some requirements, say set P, and that can be reached from the initial state, the so-called reachability predicate.

Definition (reachability predicate, $Re(\mathbf{G}, P)$) Let $\mathbf{G} = (\mathrm{ST}, \mathcal{H}, \Sigma, \Delta, P_0, P_m)$ be an STS then the reachability predicate $Re(\mathbf{G}, P)$ contains all basic state trees that can be reached from the root state of $\mathbf{G}$ via state trees satisfying predicate P. A basic state tree b is in $Re(\mathbf{G}, P)$ iff:

i $P \wedge P_0 = false \rightarrow Re(\mathbf{G}, P) = false$ and

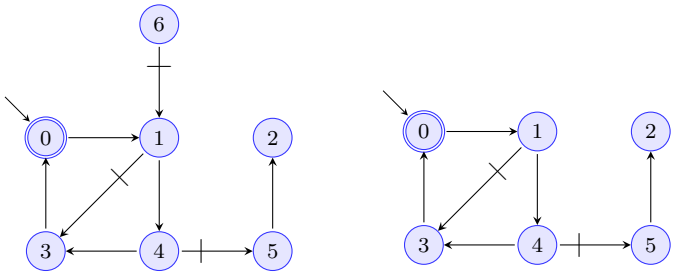
ii $b_0 \models P \wedge P_0 \rightarrow b_0 \models Re(\mathbf{G}, P)$ and

iii $b \models Re(\mathbf{G}, P), \sigma \in \Sigma, \Delta(b, \sigma) \neq \emptyset, \Delta(b, \sigma) \models P \rightarrow \Delta(b, \sigma) \models Re(\mathbf{G}, P)$.

▪

Let us consider the system $\mathbf{G}$ as depicted in Figure 1.10(a). Then, we can define the reachability predicate of the complete system, i.e. $P = \{0, 1, 2, 3, 4, 5, 6\} = \mathit{true}$ as the state set $Re(\mathbf{G}, \mathit{true}) = \{0, 1, 2, 3, 4, 5\}$ because state 6 is the only state that cannot be reached from the initial state 0. The resulting (reachable) behaviour of $Re(\mathbf{G}, \mathit{true})$ is shown in Figure 1.10(b).

By reversing this procedure, we can also define a predicate that describes the states that satisfy some requirements, say P, and from which a marker or final state can be reached, the so-called co-reachability predicate.



(a) system $\mathbf{G}$ (b) $R(\mathbf{G}, true)$

Supervisor synthesis example

Figure 1.10

Definition (co-reachability predicate, $Co(\mathbf{G}, P)$) Let $\mathbf{G} = (\mathrm{ST}, \mathcal{H}, \Sigma, \Delta, P_0, P_m)$ be an STS then the co-reachability predicate $Co(\mathbf{G}, P)$ contains all basic state trees from which a marker state of $\mathbf{G}$ can be reached via state trees satisfying predicate P. A basic state tree b is in $Co(\mathbf{G}, P)$ iff:

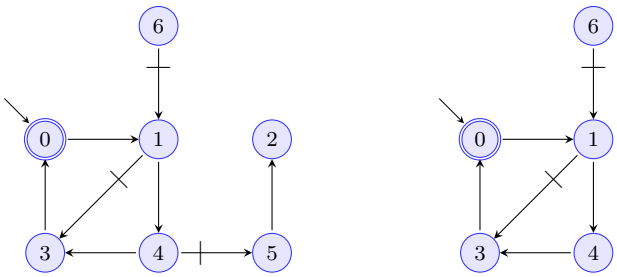
i $P \wedge P_m = false \rightarrow Co(\mathbf{G}, P) = false$ and

ii $b_m \models P \wedge P_m \rightarrow b_m \models Co(\mathbf{G}, P)$ and

iii $b \models Co(\mathbf{G}, P), \sigma \in \Sigma, \Delta(b', \sigma) = b, b' \models P \rightarrow b' \models Co(\mathbf{G}, P)$.

▪

Again, lets consider the system $\mathbf{G}$ (Fig. 1.11(a)). Now, we can define the co-reachability predicate of the complete system, i.e. $P = \{0, 1, 2, 3, 4, 5, 6\} = true$ as the state set $Co(\mathbf{G}, true) = \{0, 1, 3, 4, 6\}$ because states 2 and 5 are the only states from which the marker state 0 cannot be reached. The resulting (co-reachable) behaviour of $Co(\mathbf{G}, true)$ is shown in Figure 1.11(b).



(a) system $\mathbf{G}$ (b) $CR(\mathbf{G}, true)$

Supervisor synthesis example

Figure 1.11

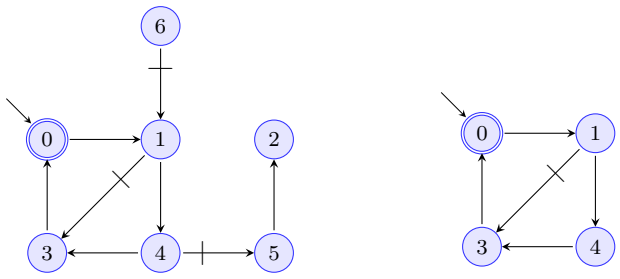
Definition (nonblocking) Let $\mathbf{G} = (\mathrm{ST}, \mathcal{H}, \Sigma, \Delta, P_0, P_m)$ be an STS and $Re(\mathbf{G}, P)$ and $Co(\mathbf{G}, P)$ be the reachable and co-reachable predicates of P, respectively. Then predicate P is non-blocking iff: $Re(\mathbf{G}, P) \preceq Co(\mathbf{G}, P)$. Thus, P is nonblocking iff all states in P that are reachable are also co-reachable. In other words, each state in P can be reached from the initial state *and* for every state in P a marker state can be reached, i.e. no deadlock.

▪

Let us consider the system $\mathbf{G}$ again (Fig. 1.12(a)). Using the reachability and co-reachability predicates as defined in the previous examples we can define a non-blocking predicate as:

$$P_{nb} = Re(\mathbf{G}, true) \wedge Co(\mathbf{G}, true) = \{0,1,3,4,6\} \wedge \{0,1,2,3,4,5\} = \{0,1,3,4\}$$

The resulting (non-blocking) behaviour is shown in Figure 1.12(b).



(a) system $\mathbf{G}$ (b) P_{nb}

Supervisor synthesis example

Figure 1.12

The next step is to define the controllability of a set of states within an STS. Ramadge and Wonham (1987) identified two types of events that are important for the control of a discrete event system: (i) controllable events and (ii) uncontrollable events. Controllable events are those events that can be blocked or postponed by the user or control system, e.g. executing a transport task. Events that cannot blocked or postponed are called un-controllable events, e.g. the break-down of a machine. In other words, uncontrollable events are those events that, iff eligible, *must* be enabled. Here, uncontrollable events are represented in a transition graph as edges *with a tick*. All other transitions are controllable.

A system is said to be controllable iff we can define a non-blocking subset of states within the system in which all uncontrollable edges lead to states that are also within this subset. In other words, a subset that is invariant under the occurrence of uncontrollable events. To explain this principle, we first define the weakest liberal precondition.

Definition (weakest liberal precondition, $M_\sigma(P)$) The *weakest liberal precondition* is a function $M_\sigma(P) : Pred(\mathrm{ST}) \rightarrow Pred(\mathrm{ST})$ that transforms the set of predicates of ST and is defined as $b \models M_\sigma(P)$ iff $\Delta(b,\sigma) \models P$.

In other words, a basic state tree b is in $M_\sigma(P)$ iff the the basic state tree reached from b by event σ is in predicate P.

▪

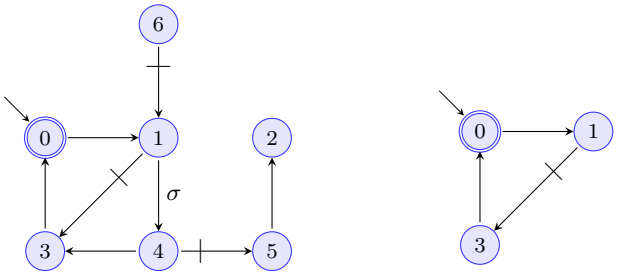
Definition (controllable) Let $\mathbf{G}$ be an STS and $P \in Pred(\mathrm{ST})$. Then, predicate P is *controllable* with respect to $\mathbf{G}$ iff:

$$P \preceq Re(\mathbf{G}, P) \wedge \forall_{\sigma \in \Sigma_u}, P \preceq M_\sigma(P) \tag{1.27}$$

In other words, a predicate P is controllable iff it is reachable from the initial state and *invariant* under the occurrence of uncontrollable events.

▪

Now, lets consider system $\mathbf{G}$ again (Fig. 1.13(a)). If we consider the non-blocking predicate $P_{nb} = \{0,1,3,4\}$ we can conclude that this predicate is not controllable since there exists an uncontrollable transition from state 4 to state 5 which is not in P_{nb}. If we remove state 4 from the non-blocking predicate, then the state set $P_{nbc} = \{0,1,3\}$ is non-blocking, i.e. $Re(\mathbf{G}, \{0,1,3\}) \preceq Co(\mathbf{G}, \{0,1,3\})$, *and* controllable, i.e. $\{0,1,3\} \preceq Re(\mathbf{G}, \{0,1,3\}) \wedge \forall_{\sigma \in \Sigma_u}, \{0,1,3\} \preceq M_\sigma(\{0,1,3\})$.



(a) system $\mathbf{G}$ (b) P_{nbc}

Supervisor synthesis example

Figure 1.13

So, using the reachability, co-reachability and controllability definitions we can check if we can define a predicate P over $\mathcal{G}$ such that P is non-blocking and controllable.

When considering the controllability of a system, control functions can be defined that prevent the system to 'leave' a controllable subset of states. These control function define when (at which state) to block which (controllable) events to stay within the controllable state set.

Definition (state feedback control) Let $\mathbf{G}$ be a STS and $\sigma \in Sigma$ then $f_\sigma : \mathcal{B}(\mathrm{ST}) \rightarrow \mathbb{B}$ denotes the *state feedback control* (SFBC) for event σ in system $\mathbf{G}$. This SFBC function enables event σ at state $b \in \mathcal{B}$ iff $f_\sigma(b) = true$. We define $\{$ as the set of all control functions.

An STS under control of $\{$ is denoted as $\mathbf{G}^f = \{\mathrm{ST}, \mathcal{H}, \Sigma, \Delta^f, P_0^f, P_m\}$. Where, Δ^f contains all transition enabled by $\{$ and $P_0^f = P_0 \wedge P$ contains all eligible initial state.

A set of control functions $\{$ for $\mathbf{G}$ is nonblocking iff all states in $\mathbf{G}^f$ are nonblocking: $Re(\mathbf{G}^f, true) \preceq Co(\mathbf{G}^f, true)$.

▪

When considering the controllable and non-blocking system as shown in Figure 1.13(b) then a state feedback control function can be defined that blocks event σ (see Fig. 1.13(a)) at state 1 in order to prevent the system to leave the controllable and non-blocking sub-space.

So far, all definitions stated in this section define the basic idea behind non-blocking controllable supervisors: (i) define all states that can be reached from the initial state and from which a final state can be reached, (ii) check if this set is invariant under the occurrence of uncontrollable events and (iii) define control functions that ensures that system stays within the set of non-blocking states.

For large industrial systems the reachability predicate can be extremely large and thus expensive to compute. In order to increase the efficiency of this process, Ma and Wonham (2005) introduced a so-called weakly controllable supervisor that avoids the expensive computation of the reachability predicate. A proof is also given that this weakly controllable supervisor has the same non-blocking controlled behavior as a controllable non-blocking supervisor that is computed with the reachability predicate.

To explain this new method in more detail, we first define weakly controllable.

Definition (Weakly controllable) Let $\mathbf{G}$ be a STS and $P \in Pred(\mathrm{ST})$. Then, predicate P is *weakly controllable* with respect to $\mathbf{G}$ iff:

$$\forall_{\sigma \in \Sigma_u}, P \preceq M_\sigma(P) \tag{1.28}$$

In words, P is invariant under the occurrence of uncontrollable events. Iff P is weakly controllable then SFBC $f : Q \rightarrow \Gamma$ is defined as:

$$\forall_{\sigma \in \Sigma_c} f_\sigma = M_\sigma(P) \tag{1.29}$$

▪

Using this predicate, we can now define a weakly controllable supervisor.

Definition (weakly controllable supervisor) The set of weakly controllable sub predicates that are stronger then P are:

$$\mathcal{CP}(P) = \{K \in Pred(\mathrm{ST}), K \preceq P, K \text{ is weakly controllable}\} \tag{1.30}$$

The supremal element of these predicates is defined as the join of all individual predicates:

$sup\mathcal{CP}(P) = \bigvee\{K | K \in \mathcal{CP}(P)\}$.

To compute $sup\mathcal{CP}(P)$ we define a function $[P]$ that transforms predicate P into a predicate that includes both P itself and $\forall_{\sigma \in \Sigma_u} M_\sigma(P)$.

i $b \models P \rightarrow b \models [P]$ and

ii $b \models P, b \neq \emptyset, \sigma \in \Sigma_u, \Delta(b', \sigma) = b \rightarrow b' \models [P]$ and

In other words, $[P]$ contains all states in P and all states from which an uncontrollable event leads to P. Using this function we can define the supremal element of weakly controllable elements as:

$$sup\mathcal{CP}(P) = \neg[\neg P] \tag{1.31}$$

In words, P is the set of possible (according to the restrictions) states and $\neg P$ represents the set of all illegal state within the system. Consequently, $\neg[\neg P]$ contains all legal states from which no uncontrollable events lead to an illegal state.

▪

Intuitively, for non-blocking control of state trees, the supervisor also needs to be non-blocking.

Definition (weakly controllable and non-blocking supervisor) We define the weakly controllable and non-blocking predicate $\mathcal{NBCP}(P)$ as:

$$\mathcal{NBCP}(P) = \{K \in Pred(\mathrm{ST}), K \preceq P, K \text{ is weakly controllable and non-blocking}\} \tag{1.32}$$

We can compute the supremal of these predicates by:

$$sup\mathcal{NBCP}(P) = P \wedge Re(\mathbf{G}, Co(\mathbf{G}, \neg[\neg P]))$$

In words, this supremal element contains all weakly controllable states that are both reachable and co-reachable.

▪

As said before, computing the reachability predicate is expensive. Ma and Wonham (2005) verified that the controlled behavior of weakly controllable and non-blocking supervisor is exactly the same if we leave out the reachability computation. The only difference between this, so-called, co-reachable supervisor and the non-blocking supervisor is that the co-reachable supervisor contains co-reachable states that cannot be reached from the initial state. Because these states cannot be reached, the reachability computation can be left out without changing the controlled behavior of the system.

Definition (weakly controllable and co-reachable supervisor)

$$\mathcal{C}^2\mathcal{P}(P) = \{K \in Pred(\mathrm{ST}), K \preceq P, K \text{ is weakly controllable and co-reachable}\} \tag{1.33}$$

Supremal $sup\mathcal{C}^2\mathcal{P}(P)$ is defined by the following algorithm:

$K_0 = P$

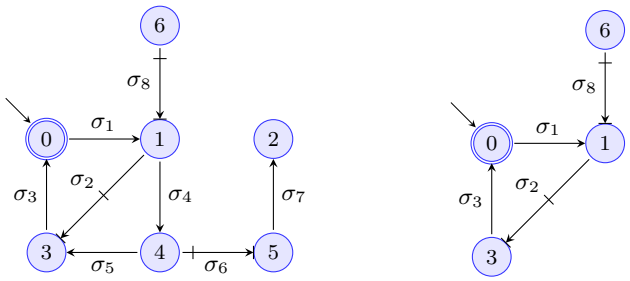
while $K_{i+1} \neq K_i$
  $K_i = K_{i+1}$
  $K_{i+1} = \Omega_P(K_i)$

$sup\mathcal{C}^2\mathcal{P}(P) = K_i$

where,
$\Omega_P(K) = P \wedge Co(\mathbf{G}, sup\mathcal{CP}(P)) = P \wedge Co(\mathbf{G}, \neg[\neg K])$

▪

The algorithm for the computation of $sup\mathcal{C}^2\mathcal{P}(P)$ as shown above is used by (Ma and Wonham, 2005) as basic supervisor synthesis algorithm. To illustrate how this algorithm precisely works we consider the non-blocking control of the system $\mathbf{G}$ as shown in Figure 1.14(a).



(a) system $\mathbf{G}$ (b) $sup\mathcal{C}^2\mathcal{P}(\mathbf{G}, P)$

Supervisor synthesis example

Figure 1.14

Since there are no special restrictions to our example problem the set of legal states is defined as: $P = \{0, 1, 2, 3, 4, 5, 6\}$. The following step are taken during automatic synthesis:

step 0
$K_0 = P = \{0, 1, 2, 3, 4, 5, 6\}$
$\neg P = \emptyset \Rightarrow [\emptyset] = \emptyset \Rightarrow \neg[\neg P] = \{0, 1, 2, 3, 4, 5, 6\}$
$K_1 = P \wedge Co(\mathbf{G}, \{0, 1, 2, 3, 4, 5, 6\}) = \{0, 1, 3, 4, 6\}$

step 1
$K_1 = \{0, 1, 3, 4, 6\}$
$\neg K_1 = \{2, 5\} \Rightarrow [\{2, 5\}] = \{2, 4, 5\} \Rightarrow \neg[\{2, 4, 5\}] = \{0, 1, 3, 6\}$
$K_2 = P \wedge Co(\mathbf{G}, \{0, 1, 3, 6\}) = \{0, 1, 3, 6\}$

step 2
$K_2 = \{0, 1, 3, 6\}$
$\neg K_1 = \{2, 4, 5\} \Rightarrow [\{2, 4, 5\}] = \{2, 4, 5\} \Rightarrow \neg[\{2, 4, 5\}] = \{0, 1, 3, 6\}$
$K_3 = P \wedge Co(\mathbf{G}, \{0, 1, 3, 6\}) = \{0, 1, 3, 6\}$

step 3
$K_3 = K_2 \Rightarrow sup\mathcal{C}^2\mathcal{P}(P) = \{0, 1, 3, 6\}$

The resulting supervisor state space is shown in Figure 1.14(b). In this result, state 6 is not reachable but it is still in the supervisor because we dropped the reachability predicate. The behavior when we start in the initial state 0 is the same whether we include state 6 or not. So, in essence the resulting

supervisor is not the same as a weakly controllable non-blocking supervisor, but the resulting controlled behavior is. From the syntheses of system $\mathbf{G}$ (see Fig. 1.14(a)), we can also define the control functions needed to prevent the system from deadlocking. We can define a control function $f_\sigma = M_\sigma(P)$, with $M_\sigma(P)$ the weakest liberal precondition.

If we take, for example, event σ_4, then we can compute that this event is eligible in state $E(\mathbf{G}, \sigma_4) = \{1\}$ and that its target state is $\Delta(1, \sigma_4) = 4$. Since $4 \not\models \{0, 1, 3\}$, $f_{\sigma_4} = \mathit{false}$. In other words, in order to prevent the system to leave the non-blocking state set, we need to block event σ_4 in all its eligible states.

This small example above, gives an overview on the application and use of the basics of non-blocking control of state tree structure. In this example we used the basic synthesis algorithm, which can also be used on flat automata. The next step in our investigation on the supervisory control of STS is to see how the synthesis can be efficiently applied using the structure of STS and a symbolic representation of STS.

1.5 Symbolic synthesis

In the small examples of the previous section, predicates are used to describe state sets. These predicates where defined as a check whether or not a certain state is in this state set or not and the state sets where defined by enumerating all individual states in this set. Another, more efficient, way of describing these sets is to define a characteristic function.

Definition (characteristic functions (predicates)) Let $\mathbf{G}$ be an STS and A a set of basic state trees over $\mathbf{G}$. Then, an n-ary characteristic function P_A is defined that returns *true* iff the input represents a basic state tree in A and *false* otherwise:

$$P_A : Q_1 \times \ldots \times Q_n \to \mathbb{B}$$

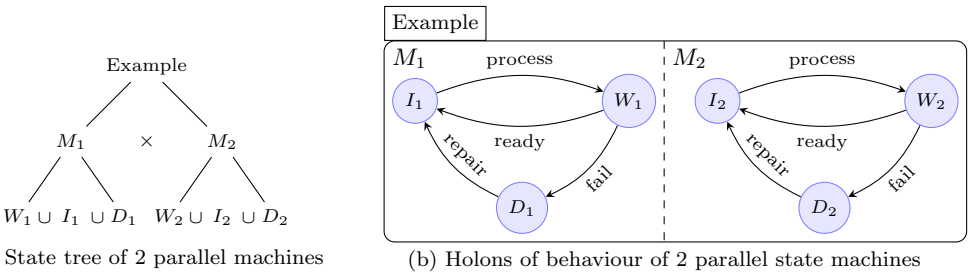
where, Q_1 to Q_n represent the internal state sets of all *or* superstates within $\mathbf{G}$. If, for example, $\mathbf{G}$ contains two parallel machines M_1 and M_2 as shown in Figure 1.15. Then, we define $Q_1 = X^{M_1} = \{I_1, W_1, D_1\}$ and $Q_2 = X^{M_2} = \{I_2, W_2, D_2\}$.

Next, state variables v_{Q_1} to v_{Q_n} are defined. The values of these variables range over sets Q_1 to Q_n respectively. By introducing these variables the predicate can be written as $P_A(v_{Q_1}, \ldots, v_{Q_n})$ and the structure of STS can be exploited to efficiently encode state tree sets as predicates. Going back to the two parallel machines example (Fig. 1.15), then a predicate for all states in which M_1 is down, $A = \{(D_1, W_2), (D_1, I_2), (D_1, D_2)\}$, can easily be defined as:

$$P_A(v_{M_1}, v_{M_2}) = (v_{M_1} = D_1)$$

notice that variable v_{M_2} is not in the function since the system is always in one of the states of M_2 and thus v_{M_2} is redundant.

▪



(a) State tree of 2 parallel machines

(b) Holons of behaviour of 2 parallel state machines

two machines working in parallel

Figure 1.15

This small example shows the effectiveness of using these kind of predicates, i.e. the size of the predicate is *not necessary* proportional to the size of the set it represents. Another advantage of using predicates is that a single predicate can describe all individual states of a set of sub state trees. Notice here that this is *different* to the join of a set of sub state trees. For a single sub state tree, a predicate can be defined recursively using the structure of STS.

Definition (transforming a sub state tree to a predicate $\Theta(\mathrm{ST}_1^x)$)

Let $\mathbf{G}$ be an STS and ST_1^x a sub state tree (rooted at x) within $\mathbf{G}$. Then, the transformation of this sub state tree to a predicate, $\Theta : \mathrm{ST}_* \rightarrow Predicate$, can be defined recursively as:

$$\Theta(\mathrm{ST}_1^x) = \begin{cases} \bigwedge_{y \in \mathcal{E}_1(x)} \Theta(\mathrm{ST}_1^y) & \text{, iff } \mathcal{T}(x) = \textit{and} \\ \bigvee_{y \in \mathcal{E}_1(x)} ((v_x = y) \wedge \Theta(\mathrm{ST}_1^y)) & \text{, iff } \mathcal{T}(x) = \textit{or} \\ 1 & \text{, iff } \mathcal{T}(x) = \textit{simple} \end{cases} \tag{1.34}$$

If *all* children of an *or* superstate are present in the sub state tree, then the variable of this superstate is always *true*. Using this property, the following tautology can be defined:

$$\Big(\bigvee_{\forall y \in \mathcal{E}(x)} (v_x = y) \Big) = 1 \tag{1.35}$$

▪

Lets consider the parallel machine example as illustrated in Figure 1.15. On this state tree, we can define the sub state tree containing all states in which M_1 is down (see Fig. 1.16(a)). Now, the predicate describing this sub state tree can be build recursively:

$$\begin{aligned}
\Theta(\mathrm{ST}_1) &= \Theta(\mathrm{ST}_1^{M_1}) \wedge \Theta(\mathrm{ST}_1^{M_2}) \\
\Theta(\mathrm{ST}_1^{M_1}) &= (v_{M_1} = D_1 \wedge \Theta(\mathrm{ST}_1^{D_1})) \\
&= (v_{M_1} = D_1 \wedge 1) \\
&= (v_{M_1} = D_1) \\
\Theta(\mathrm{ST}_1^{M_2}) &= (v_{M_2} = I_2 \wedge \Theta(\mathrm{ST}_1^{I_2})) \vee (v_{M_2} = D_2 \wedge \Theta(\mathrm{ST}_1^{D_2})) \vee (v_{M_2} = W_2 \wedge \Theta(\mathrm{ST}_1^{W_2})) \\
&= (v_{M_2} = I_2 \wedge 1) \vee (v_{M_2} = W_2 \wedge 1) \vee (v_{M_2} = D_2 \wedge 1) \\
&= (v_{M_2} = I_2) \vee (v_{M_2} = W_2) \vee (v_{M_2} = D_2) \\
&= 1 \\
\Theta(\mathrm{ST}_1) &= (v_{M_1} = D_1)
\end{aligned}$$

The same can be done for the sub state tree as represented in Figure 1.16(b):

$$\Theta(\mathrm{ST}_2) = (v_{M_1} = W_1) \wedge (v_{M_2} = D_1)$$

Notice, that it is also easy to compute the *union*, *intersection* and *complement* of predicates:

$$\begin{aligned}
&\textbf{union} && \Theta(\mathrm{ST}_1) \vee \Theta(\mathrm{ST}_2) &&= (v_{M_1} = D_1) \vee \Big((v_{M_1} = W_1) \wedge (v_{M_2} = D_1) \Big) \\
&\textbf{intersection} && \Theta(\mathrm{ST}_1) \wedge \Theta(\mathrm{ST}_2) &&= (v_{M_1} = D_1) \wedge \Big((v_{M_1} = W_1) \wedge (v_{M_2} = D_1) \Big) = 0 \\
&\textbf{complement} && \neg\Theta(\mathrm{ST}_1) &&= \neg(v_{M_1} = D_1)
\end{aligned}$$

(a) ST_1 (b) ST_2

sub state tree examples

Figure 1.16

Sofar, the structure of state trees have been used to efficiently encode state sets into predicates. The next logical step is to also encode the behaviour (transitions) into predicates. First, the local transition function is considered.

Definition (local transition predicates) Let $\mathbf{G}$ be an STS and $\mathrm{ST}_2^x = \overline{\delta}_I^x(\mathrm{ST}_1^x, \sigma)$ a local transition within *or* superstate x with $x \in \mathbf{G}$. For convenience, we denote the source child state tree ST_1^x as S and the target child state tree ST_2^x as T. Next, for state x we denote v_x as a target variable and v_x' as a source variable. The sets of all source and target variables are denoted $\mathbf{v}$ and $\mathbf{v}'$ respectively. Replacing all source variables v_x with its target variable v_x' is denoted as $P(\mathbf{v}') = P(\mathbf{v})[\mathbf{v} \rightarrow \mathbf{v}']$. Next, the transition $T = \overline{\delta}_I^x(S, \sigma)$, denoted as $\mathbf{t}$, is encoded:

$$\Theta(S)[\mathbf{v_{t,S}} \rightarrow \mathbf{v'_{t,S}}] \wedge \Theta(T)$$

where, $\mathbf{v_{t,S}}$ are the variables appearing in $\Theta(S)$.

▪

Lets, for example, consider the transition *process* in M_1 of the example denoted in Figure 1.15. The source and target (sub) child state tree predicates are $\Theta(S) = (v_{M_1} = I_1)$ and $\Theta(T) = (v_{M_1} = W_1)$ respectively. With these predicates, the local transition *process* can be encoded:

$$\begin{aligned} \Theta(S) &= (v_{M_1} = I_1) \\ \Theta(S)[\mathbf{v_{t,S}} \rightarrow \mathbf{v'_{t,S}}] &= (v'_{M_1} = I_1) \\ \Theta(S)[\mathbf{v_{t,S}} \rightarrow \mathbf{v'_{t,S}}] \wedge \Theta(T) &= (v'_{M_1} = I_1) \wedge (v_{M_1} = W_1) \end{aligned}$$

Next, a formal definition is given for a global transition predicate.

Definition (global transition predicates) Let $\mathbf{G}$ be an STS and $\mathrm{ST}_2^x = \overline{\delta}_I^x(\mathrm{ST}_1^x, \sigma)$ a local transition within *or* superstate x with $x \in \mathbf{G}$. For convenience, we denote the source child state tree ST_1^x as S and the target child state tree ST_2^x as T. Then we define a sub state tree $\hat{T}$ such that:

$$a \in \hat{T} \text{ iff: } \Big((a|x) \vee (a \leq x) \vee (x \leq a)\Big)$$

With $\hat{T}$ we can encode global transitions by:

$$\Theta(S)[\mathbf{v_{t,S}} \rightarrow \mathbf{v'_{t,S}}] \wedge \Theta(\hat{T}) \tag{1.36}$$

The difference with the local transition function is that with the global transition function the states parallel to the state in which the transition takes place are also taken in account. The formal definition for the global transition function stays relative simple because it takes advantage of the local coupling constraint.

▪

Definition (Eligible transition set) With the global transition function as defined above, we can compute the entire (global) set of transitions functions labeled by event σ.

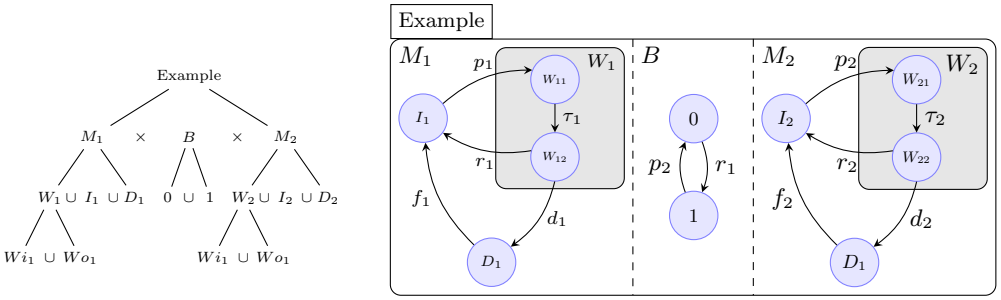
First, we define $D_\sigma = \{x | x \in X, \sigma \in \Sigma_I^x\}$ as the set of OR superstates with σ as internal event. Let T_σ^x be the set of inner transitions in holon H^x labeled by event σ. Then, the set of global transitions labeled with event σ is given in the tuple $< N_\sigma, \mathbf{v}_{\sigma,\mathbf{S}}, \mathbf{v}_{\sigma,\mathbf{T}} >$ where:

$$\begin{array}{rcl} N_\sigma & = & \bigwedge_{x \in D_\sigma} \left(\bigvee_{\mathbf{t} \in T_\sigma^x} N_\mathbf{t} \right) \\ \mathbf{v}_{\sigma,\mathbf{S}} & = & \bigcup_{x \in D_\sigma} \left(\bigcup_{\mathbf{t} \in T_\sigma^x} \mathbf{v}_{\mathbf{t},\mathbf{S}} \right) \\ \mathbf{v}_{\sigma,\mathbf{T}} & = & \bigcup_{x \in D_\sigma} \left(\bigcup_{\mathbf{t} \in T_\sigma^x} \mathbf{v}_{\mathbf{t},\mathbf{T}} \right) \end{array}$$

▪

To illustrate the use of the global transition predicate and the eligible transition set, we replace the W_i simple states of the previous example with *or* superstates and a buffer B is placed in between the two machines, as shown in Figure 1.17. This buffer states that M_2 can only start processing (p_2) when M_1 is ready processing r_1.

two machines working in parallel



(a) State tree of 2 parallel machines (b) Holons of behaviour of 2 parallel state machines

Figure 1.17

Now, lets consider event f_1 and the computation of $< N_{f_1}, \mathbf{v}_{f_1,S}, \mathbf{v}_{f_1,T} >$:

$$\begin{array}{rcl} D_{f_1} & = & \{M_1\} \\ T_{f_1}^{M_1} & = & \{(D_1, I_1)\} \end{array}$$

$$\begin{array}{rcl} \Theta(S) & = & (v_{M_1} = D_1) \\ \Theta(T) & = & (v_{M_1} = I_1); \quad \Theta(\hat{T}) = (v_{M_1} = I_1) \end{array}$$

$$\begin{array}{rcl} N_{f_1} & = & (v'_{M_1} = D_1) \wedge (v_{M_1} = I_1) \\ \mathbf{v}_{f_1,S} & = & \{v_{M_1}\} \\ \mathbf{v}_{f_1,T} & = & \{v_{M_1}\} \end{array}$$

Since f_1 only occurs in M_1 this computation is fairly straight forward. Next, we consider event p_1 which is a boundary transition within superstate W_1 *and* an internal transition within B.

$$
\begin{array}{rcl}
D_{p_1} & = & \{M_1\} \\
T_{p_1}^{M_1} & = & \{(I_1, W_{10})\} \\
T_{p_1}^{B} & = & \{(0,1)\}
\end{array}
$$

$$
\begin{array}{rcl}
\Theta(S_1) & = & (v_{M_1} = I_1) \\
\Theta(T_1) & = & (v_{W_1} = W_{10}); \quad \Theta(\hat{T}) = (v_{M_1} = W_1) \wedge (v_{W_1} = W_{10}) \\
N_{p_1,1} & = & (v'_{M_1} = I_1) \wedge (v_{M_1} = W_1) \wedge (v_{W_1} = W_{10})
\end{array}
$$

$$
\begin{array}{rcl}
\Theta(S_2) & = & (v_B = 0) \\
\Theta(T_2) & = & (v_B = 1); \quad \Theta(\hat{T}) = (v_B = 1) \\
N_{p_1,2} & = & (v'_B = 0) \wedge (v_B = 1)
\end{array}
$$

$$
\begin{array}{rcl}
N_{p_1} & = & N_{p_1,1} \wedge N_{p_1,2} \\
\mathbf{v}_{p_1,S} & = & \{v_{M_1}, v_B\} \\
\mathbf{v}_{p_1,T} & = & \{v_{M_1}, v_{W_1}, v_B\}
\end{array}
$$

With the symbolic representation of both the states and the transitions the synthesis of a non-blocking supervisor can be done symbolically. For this synthesis, there are two important functions namely $[P]$ and $CR(\mathbf{G}, P)$. Both these functions use the global reverse-transition function $\Gamma(P, \sigma)$, with P a state predicate.

Definition (reverse transition function: $\Gamma(P, \sigma)$) Let $\mathbf{G}$ be an STS, P a predicate defined on $\mathbf{G}$ and σ an event in $\mathbf{G}$. Then, with the set of transitions N_σ we can define the reverse transition function as:

$$\Gamma(P, \sigma) = \big(\exists \mathbf{v}_{\sigma,T}(P \wedge N_\sigma)\big)[\mathbf{v}'_{\sigma,S} \rightarrow \mathbf{v}_{\sigma,S}] \tag{1.37}$$

where,
$\exists_{v_x} P = \bigvee_{i=1}^{n} P[y_i/v_x]$ with i ranging over all possible values of v_x.

In words, $(P \wedge N_\sigma)$ is computed first, this results in a predicate that describes the transitions of σ and their target states satisfying P. In the next step $\exists \mathbf{v}_{\sigma,T}$ quantifies out all variables in $\mathbf{v}_{\sigma,T}$ to get the set of all source sub state trees of event σ. Finally, $[\mathbf{v}'_{\sigma,S} \rightarrow \mathbf{v}_{\sigma,S}]$ replaces all prime variables v'_x with the normal variables v_x resulting in the set of source state of event σ

▪

Using the example of Figure 1.17 we explain the reverse transition function in more detail. Let, for example, consider the computation of $\Gamma(P, p_1)$ where $P = (v_{M_1} = W_1) \wedge (v_{M_2} = W_2)$:

$$
\begin{array}{rcl}
P \wedge (N_{p_1,1} \wedge N_{p_1,2}) & = & (v_{M_1} = W_1) \wedge (v_{M_2} = W_2) \\
& & \wedge (v'_{M_1} = I_1) \wedge (v_{M_1} = W_1) \wedge (v_{W_1} = W_{10}) \\
& & \wedge (v'_B = 0) \wedge (v_B = 1)
\end{array}
$$

$$
\begin{array}{rcl}
\exists \mathbf{v}_{\sigma,T}(P \wedge N_{p_1}) & = & (v_{M_2} = W_2) \wedge (v'_{M_1} = I_1) \wedge (v'_B = 0)
\end{array}
$$

$$
\begin{array}{rcl}
\exists \mathbf{v}_{\sigma,T}(P \wedge N_{p_1})[\mathbf{v}'_{\sigma,S} \rightarrow \mathbf{v}_{\sigma,S}] & = & (v_{M_2} = W_2) \wedge (v_{M_1} = I_1) \wedge (v_B = 0)
\end{array}
$$

As shown above, the set of state from which P can be reached by event p_1 are defined by the computation of $\Gamma(P, p_1)$.

Using this reverse transition function and the hierarchical structure of STS the fixpoint $[P]$ can be computed recursively by defining function $[P]^x$.

Definition **($[P]^x$)** Let $\mathbf{G}$ be an STS, $x \in \mathbf{G}, \mathcal{T}(x) \in \{or, and\}$ and P a predicate describing the set of legal states. Then, we define the local fixpoint $[P]^x$ for superstate x as:

```
def [P]^x
    K = P
    if T(x) == or

        while K_i ≠ K
            K_i = K
            K = K ∨ (⋁_{σ_u ∈ Σ^x_{Iu}} Γ(K, σ_u))
            for y in ε(x)
                if T(y) ∈ {or, and}
                    K = [K]^y

    else if T(x) == and

        while K_i ≠ K
            K_i = K
            for y in ε(x)
                if T(y) ∈ {or, and}
                    K = [K]^y

    return K
```

Using this local fixpoint the global fixpoint $[P]^{x_0}$ can be calculated recursively. This algorithm uses a structured search through the systems state space such that the intermediate predicates (representing state sets) stay relatively small. This is because the structured search exploits the following tautology $\bigvee_{y \in \mathcal{E}(x)}(v_x = y) = true$ with $x, y \in \mathcal{E}^*(x_0)$ and v_x the state variable representing the active child state of superstate x.

Simply speaking, if the system is in one child state of *or* superstate x and x has no illegal child states then $[P]^x = true$. By systematically searching through the system we can quantify this tautology immediately, per *or* superstate, instead of building an intermediate predicate containing several parts of different *or* superstates.

▪

Similar to the computation of $[P]^x$, the co-reachable predicate $CR(G, P, R)$ can also be computed recursively using the state trees structured state space.

Definition **($CR^x(G, P, R)$)** Let $\mathbf{G}$ be an STS, $x \in \mathbf{G}, \mathcal{T}(x) \in \{or, and\}$ with P the predicate describing the set of legal states and R the predicate describing the set op legal marker states. Then, we define the local fixpoint $CR^x(G, P, R)$ for superstate x as:

```
def CR^x(G, P, R)
    K = R
    if T(x) == or

        while K_i ≠ K
            K_i = K
            K = K ∨ (P ∧ ⋁_{σ ∈ Σ^x_I} Γ(K, σ))
            for y in ε(x)
                if T(y) ∈ {or, and}
                    K = CR^y(G, P, R)

    else if T(x) == and

        while K_i ≠ K
```

```
15              K_i = K
16              for y in ε(x)
17                  if T(y) ∈ {or, and}
18                      K = CR^y(G, P, R)
19
20      return K
```

▪

The algorithms as defined above exploit the structure of state trees for the efficiency of the supervisor synthesis. A well-formed STS also needs to satisfy the local coupling constraint. This constrained states that events can only be shared among states that have the same *and* super sates as parent. To speed up the synthesis even more, this constraint can also be exploited. This can be done by defining a cluster as a group of *or* super states that share events and calculate the fixpoints sequentially per cluster (in the same manner as the local fixpoint computation per *or* super state. So lines $14 - 18$ of the algorithms will become:

```
14          while K_i ≠ K
15              K_i = K
16              for C in Clusters under x
17                  while K_ii ≠ K
18                      k_ii = K
19                      for y in C
20                          if T(y) ∈ {or, and}
21                              K = CR^y(G, P, R)
```

Bibliography

Cassandras, C. and Lafortune, S. (2004). *Introduction to discrete event systems*. Kluwer Academic Publishers.

Davey, B. and Priestly, H. (2002). *Introduction to lattices and order*. Cambridge University Press.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8*(3), 231–274.

Ma, C. and Wonham, W. (2005). *Nonblocking supervisory control of state tree structures* (1st edition ed.). Springer.

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

Vahidi, A., Fabian, M. and Lennartson, B. (2006). Efficient supervisory sythesis of large systems. *Control Engineering Practice*, *14*, 1157-1167.

Wang, B. (1995). *Top-down design for RW supervisory control theory*. University of Toronto.