UNIVERSITY OF MICHIGAN
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
LECTURE NOTES FOR EECS 661
CHAPTER 3: SUPERVISORY CONTROL OF DISCRETE EVENT SYSTEMS

Stéphane Lafortune

October 2004

3.1: THE FEEDBACK LOOP OF SUPERVISORY CONTROL

References for Chapter 3: Textbook, Chapter 3. For more details, see the references at the end of the chapter, in particular the survey papers by Ramadge & Wonham and by Thistle.

Motivation: *Achieve, by control, the orderly flow of events.*
Formulate and study concepts in the framework of *languages* (thus representation-independent); develop synthesis algorithms in the framework of *finite-state automata* (for the case of regular languages).

3.1: The Feedback Loop of Supervisory Control

Uncontrolled Discrete Event Systems

- Consider a DES modeled by pair of languages, L and L_m, where $L = \overline{L}$ is the set of all traces that the DES can generate and $L_m \subseteq L$ is the language of *marked* traces that is used to represent the completion of some operations or tasks; the definition of L_m is a modeling issue.

  L and L_m are defined over the event set E.

- Without loss of generality and for the sake of convenience, assume that L and L_m are the languages generated and marked by automaton $G = (X, E, f, \Gamma, x_0, X_m)$, where X need not be finite:

$$\mathcal{L}(G) = L \quad \text{and} \quad \mathcal{L}_m(G) = L_m \ .$$

- Thus we will talk of the "DES G".

Legal Behavior

- Control is necessary because the uncontrolled DES G does not satisfy *safety* or *nonblocking* specifications.

  Let us refer to the specifications on safety and nonblocking as the "legal behavior."

- We will in the following describe the legal behavior as a subset of $\mathcal{L}(G)$, usually denoted by L_a where a stands for "admissible".

- In problems where blocking is of concern, then the legal behavior is given as a subset of $\mathcal{L}_m(G)$, usually denoted by L_{am}.

Centralized Control under Full Event Observation

- Let E be partitioned into two disjoint subsets

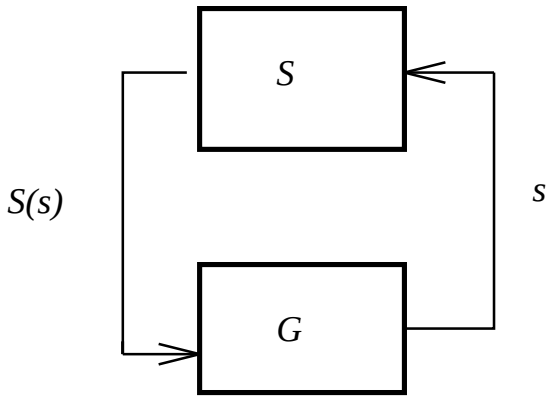
$$E = E_c \cup E_{uc}$$

  where:

  - E_c is the set of *controllable* events: these are the events that can be prevented from happening, or disabled, by control;
  - E_{uc} is the set of *uncontrollable* events: these events cannot be prevented from happening by control.
  - There are many reasons why an event would be modeled as uncontrollable: it is inherently unpreventable (e.g., failure); it models a change of sensor readings not due to a command; it cannot be prevented due to hardware limitations; or it is modeled as uncontrollable by choice, e.g., if the event has high priority and thus should not be disabled or if the event represents the tick of a clock.

- Let us now assume that the transition function of G can be controlled by an external agent in the sense that the controllable events can be enabled or disabled by an external controller. The control paradigm is as follows:

  G, which represents the *uncontrolled behavior* of the DES, is connected in the following feedback loop with a *controller* or *supervisor* S:



- Formally, supervisor (or controller) S is a *function*
$$S : \mathcal{L}(G) \to 2^E \ .$$
- For each $s \in \mathcal{L}(G)$ generated so far by G (under the control of S),
$$S(s) \cap \Gamma(f(x_0, s))$$
is the set of *enabled events* that G can execute at its current state $f(x_0, s)$.
In other words, G cannot execute an event that is in its current active event set if that event is not also contained in $S(s)$.
- A supervisor is *admissible* if for all $s \in \mathcal{L}(G)$,
$$E_{uc} \cap \Gamma(f(x_0, s)) \subseteq S(s)$$
that is, S is not allowed to ever disable a feasible uncontrollable event.
From now on, we will only consider admissible supervisors.
- We call $S(s)$ the *control action* at s.
- S is the *control policy*.
- Observe that this is a case of *dynamic feedback* in the sense that the domain of S is $\mathcal{L}(G)$ and not X; thus the control action may change on subsequent visits to a state.
- We denote the resulting closed-loop system by S/G.

- The *language generated* by S/G is defined recursively as follows:
  1. $\varepsilon \in \mathcal{L}(S/G)$
  2. $[(s \in \mathcal{L}(S/G)) \wedge (s\sigma \in \mathcal{L}(G)) \wedge (\sigma \in S(s))] \Leftrightarrow [s\sigma \in \mathcal{L}(S/G)]$.

  Clearly, $\mathcal{L}(S/G) \subseteq \mathcal{L}(G)$ and it is prefix-closed by definition.
- The *language marked* by S/G is defined as follows:

$$\mathcal{L}_m(S/G) := \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$$

  i.e., it consists exactly of the marked traces of G that survive under the control of S.
- Overall,

$$\emptyset \subseteq \mathcal{L}_m(S/G) \subseteq \overline{\mathcal{L}_m(S/G)} \subseteq \mathcal{L}(S/G) \subseteq \mathcal{L}(G) \ .$$

- At this point, we can think of the (controlled) DES S/G as an automaton that generates $\mathcal{L}(S/G)$ and marks $\mathcal{L}_m(S/G)$.

  We will see later how to build this automaton (in the finite-state case).

- S is said to be *nonblocking* if S/G is nonblocking, i.e.,

$$\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)} \ ;$$

  otherwise, S is said to be *blocking*.

  Interpretation: Since marked traces represent completed tasks or record the completion of some particular operation (by choice at modeling), blocking means that the controlled system cannot terminate the execution of the task at hand. The notions of marked traces and blocking allow to model *deadlock* and *livelock* and thus they are very useful.

Centralized Control under Partial Event Observation

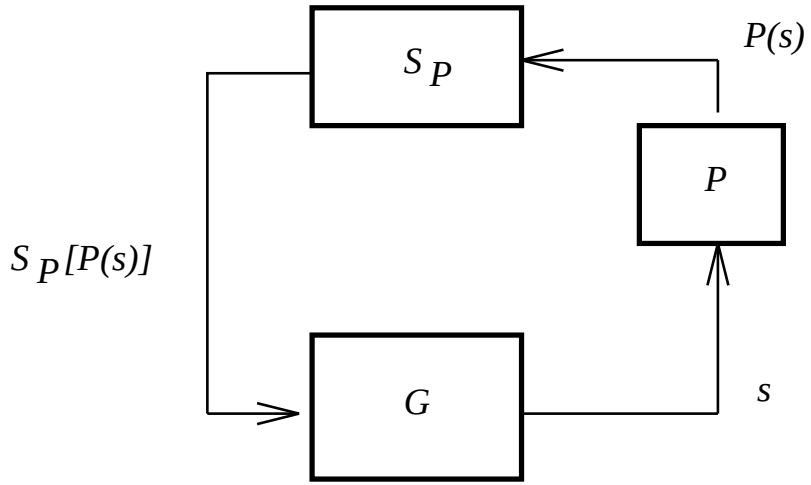
- Now, consider the situation where the supervisor S does not "see" or "observe" all the events that G generates.
- In this regard, let E be partitioned into two disjoint subsets

$$E = E_o \cup E_{uo}$$

where:

  - E_o is the set of *observable* events: these are the events that can be seen by the supervisor;
  - E_{uo} is the set of *unobservable* events: these are the events that in some sense are "internal" to the system and cannot be seen by the supervisor.
  - There are many reasons why an event would be modeled as unobservable: its occurrence is not recorded by the available sensors; it is a remote event; and so forth.

- In this case, the feedback loop is now of the form:



- Here, $P : E^* \rightarrow E_o^*$ is the *natural projection* that is defined as in Chapter 2. It simply "erases" all the unobservable events in a trace. P is extended to languages as described earlier.

- Due to the presence of P, the supervisor cannot distinguish between two traces s_1 and s_2 that have the same projection, i.e., $P(s_1) = P(s_2)$; for such $s_1, s_2 \in \mathcal{L}(G)$, the supervisor will necessarily issue the same control action.
- In order to capture this fact, and as indicated in the preceding figure, we define the action of control under partial observation as the function

$$S_P : P[\mathcal{L}(G)] \to 2^E \ .$$

  and call S_P a *P-supervisor.*

  This means that the control action can change only after the occurrence of an observable event, i.e., when $P(s)$ changes.
- When an (enabled) observable event occurs, the control action is *instantaneously* updated i.e., it is updated before any unobservable event occurs.

  The control action $S_P(t)$, for $t \in P[\mathcal{L}(G)]$, is applied by S_P immediately after the execution by G of the last (observable) event of t and remains in effect until the next observable event is executed by G.
- *Note:* It will sometimes be convenient to view a P-supervisor S_P as a function $S : \mathcal{L}(G) \to 2^E$. This will only make sense if S satisfies the property that:

$$P(s_1) = P(s_2) \Rightarrow S(s_1) = S(s_2) \ .$$

- A P-supervisor is admissible if it never disables a feasible uncontrollable event.

  Let us take $t = t'\sigma$ (where $\sigma \in E_o$). $S_P(t)$ is the control action that applies to all strings in $\mathcal{L}(G)$ that belong to $P^{-1}(t')\{\sigma\}$ as well as to the unobservable continuations of these strings. However, $S_P(t)$ may disable unobservable events and thus prevent some of these unobservable continuations.

  We define

$$L_t = P^{-1}(t')\{\sigma\}(S_P(t) \cap E_{uo})^* \cap \mathcal{L}(G) \ .$$

  L_t contains all the strings in $\mathcal{L}(G)$ that are effectively subject to the control action $S_P(t)$, when S_P controls G.

  Since a supervisor is admissible if it does not disable a feasible uncontrollable event, we conclude that S_P is admissible if for all $t = t'\sigma \in P[\mathcal{L}(G)]$,

$$E_{uc} \cap \Big[\bigcup_{s \in L_t} \Gamma(f(x_0, s)) \Big] \ \subseteq \ S_P(t) \ .$$

  The term in brackets represents all feasible continuations in $\mathcal{L}(G)$ of all strings that $S_P(t)$ applies to.

- The *language generated* by S_P/G is defined recursively as before:
  1. $\varepsilon \in \mathcal{L}(S_P/G)$
  2. $[(s \in \mathcal{L}(S_P/G)) \wedge (s\sigma \in \mathcal{L}(G)) \wedge (\sigma \in S_P[P(s)])] \Leftrightarrow [s\sigma \in \mathcal{L}(S_P/G)]$.
- The *language marked* by S_P/G is defined as before:

$$\mathcal{L}_m(S_P/G) := \mathcal{L}(S_P/G) \cap \mathcal{L}_m(G) \ .$$

- Note that the languages $\mathcal{L}(S_P/G)$ and $\mathcal{L}_m(S_P/G)$ are defined over E, and not E_o, i.e., they correspond to the closed-loop behavior of G before the effect of projection P.
- Finally, observe that we are not making any specific assumptions about the controllability and observability properties of an event: e.g., an unobservable event could be controllable, an uncontrollable event could be observable, and so forth.

Remarks on the Legal Behavior

- One obtains L_a (or L_{am}) after accounting for all the *specifications* (or requirements) that are imposed on the system.
  - These specifications are themselves described by one or more (possibly marked) languages $K_{s,i}, i = 1, \ldots, m$.
  - If a specification language $K_{s,i}$ is not given as a subset of $\mathcal{L}(G)$ (or $\mathcal{L}_m(G)$), then we take
    $$L_{a,i} = \mathcal{L}(G) \cap K_{s,i} \quad \text{or} \quad L_{am,i} = \mathcal{L}_m(G) \cap K_{s,i}$$
    or we take
    $$L_{a,i} = \mathcal{L}(G) \,||\, K_{s,i} \quad \text{or} \quad L_{am,i} = \mathcal{L}_m(G) \,||\, K_{s,i} \; .$$
    The choice of intersection or parallel composition is based on the respective event sets of $\mathcal{L}(G)$ and $K_{s,i}$:
    - If the events that appear $\mathcal{L}(G)$ but not in $K_{s,i}$ are irrelevant to $K_{s,i}$, then we take $||$.
    - On the other hand, if these events are absent from $K_{s,i}$ because they should not happen in the legal behavior, then $\cap$ is the right operation.
  - Overall, the $L_{a,i}$'s (or the $L_{am,i}$'s) are combined to form L_a (L_{am}, resp.); again, intersection or parallel composition is used (same reasoning as above).

- The question that we address in the next section is: under what conditions can given L_a and L_{am} be *exactly* achieved by a supervisor S (or S_P in the case of partial observation)?

Some Techniques for Modifying Automata to Account for "Illegal Behavior"

Illegal States: If a specification identifies certain states of G as illegal, then it suffices to delete these states from G, i.e., remove the state and all the transitions attached to it, and then do the Ac operation.

State Splitting: If a specification requires remembering how a particular state of G was reached (in order to determine what future behavior is legal), then that state has to be split into as many states as necessary. The active event set of each newly introduced state is adjusted according to the respective legal continuations.

Event Alternance: If a specification requires the alternance of two events, then build a two-state automaton that captures this alternance; then take the parallel composition with G to get the generator of the legal language.

Illegal Subtrace: If a specification identifies as illegal all traces of $\mathcal{L}(G)$ that contain subtrace $s_f = \sigma_1 \cdots \sigma_n \in \Sigma^*$ (f for forbidden), then we account for this specification by building automaton

$$H_{spec} = (X, E, f, x_0, X)$$

1. $X = \{\varepsilon, \sigma_1, \sigma_1\sigma_2, \ldots, \sigma_1 \cdots \sigma_{n-1}\}$
   that is, we associate a state of H_{spec} to every proper prefix of s_f.

2. The transition function f is constructed in two steps:
   2.1. $f(\sigma_1 \cdots \sigma_i, \sigma_{i+1}) = \sigma_1 \cdots \sigma_{i+1}$, for $i = 0, \ldots, n-2$.
   2.2. Complete f to E as follows for all the states in X, except for state $\sigma_1 \cdots \sigma_{n-1}$ which is completed to $E \setminus \{\sigma_n\}$ (since that last event is illegal in that state):

$$f(\sigma_1 \cdots \sigma_i, \gamma) = \text{state in } X \text{ corresponding to the longest suffix of } \sigma_1 \cdots \sigma_i \gamma \ .$$

3. Take $x_0 = \varepsilon$.

$$\mathcal{L}(H_{spec}) = \mathcal{L}_m(H_{spec}) = E^* \setminus E^* \{s_f\} E^* \ .$$

Consequently, the desired H_a is obtained by

$$H_a = H_{spec} \times G \ .$$

This procedure can be extended to a finite set of illegal subtraces $s_{f_1}, \ldots, s_{f_m}$ in one step, i.e., a single H_{spec} is constructed as opposed to doing the above procedure once for each illegal s_{f_i}.

3.2: THE MAIN EXISTENCE RESULTS

3.2: The Main Existence Results

The Controllability Theorem

Theorem (CT) : Consider a DES G where $E_{uc} \subseteq E$ is the set of uncontrollable events.
Let $K \subseteq \mathcal{L}(G)$, $K \neq \emptyset$.
There exists S such that

$$\mathcal{L}(S/G) = \overline{K}$$

iff the following condition holds:

$$\text{[controllability]} \qquad \overline{K}E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}.$$

Proof of CT:
[IF] For $s \in \mathcal{L}(G)$, define $S(s)$ according to

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \overline{K}\} \ .$$

This supervisor enables after trace s: (i) all uncontrollable events that are feasible in G after trace s and (ii) all controllable events that extend s inside of $\overline{K}$. Part (i) ensures that S is admissible, i.e., that it never disables a feasible uncontrollable event.
We now prove that with this S, $\mathcal{L}(S/G) = \overline{K}$. The proof is by induction on the length of the traces in the two languages.

- The base case is for traces of length 0. But $\varepsilon \in \mathcal{L}(S/G)$ by definition and $\varepsilon \in \overline{K}$ since $K \neq \emptyset$ by assumption. Thus the base case holds.
- The induction hypothesis is that for all traces s such that $|s| \leq n$, $s \in \mathcal{L}(S/G)$ iff $s \in \overline{K}$. We now prove the same for traces of the form $s\sigma$.

– Let $s\sigma \in \mathcal{L}(S/G)$. By definition of $\mathcal{L}(S/G)$, this implies that

$$[s \in \mathcal{L}(S/G)] \wedge [\sigma \in S(s)] \wedge [s\sigma \in \mathcal{L}(G)]$$

which in turn implies that

$$[s \in \overline{K}] \wedge [\sigma \in S(s)] \wedge [s\sigma \in \mathcal{L}(G)]$$

using the induction hypothesis.

* Now, if $\sigma \in E_{uc}$, then the controllability condition immediately yields $s\sigma \in \overline{K}$.
* On the other hand, if $\sigma \in E_c$, then by the definition of S, we also obtain that $s\sigma \in \overline{K}$.

– For the other direction of the induction step, let $s\sigma \in \overline{K}$. Then $s\sigma \in \mathcal{L}(G)$ since by assumption $\overline{K} \subseteq \mathcal{L}(G)$.

* Now, if $\sigma \in E_{uc}$, then $\sigma \in S(s)$ by definition of $S(s)$.
* On the other hand, if $\sigma \in E_c$, then by the above definition of $S(s)$, we also obtain that $\sigma \in S(s)$.

Overall, we have that

$$[s \in \overline{K}] \wedge [\sigma \in S(s)] \wedge [s\sigma \in \mathcal{L}(G)]$$

which in turns implies that

$$[s \in \mathcal{L}(S/G)] \wedge [\sigma \in S(s)] \wedge [s\sigma \in \mathcal{L}(G)]$$

using the induction hypothesis. It then immediately follows that $s\sigma \in \mathcal{L}(S/G)$.

– This completes the proof of the induction step.

[ONLY IF] Let there exist an admissible S such that $\mathcal{L}(S/G) = \overline{K}$.
Let $s \in \overline{K}$, $\sigma \in E_{uc}$, and $s\sigma \in \mathcal{L}(G)$.

- Then $\sigma \in S(s)$ since any admissible supervisor is not allowed to disable a feasible uncontrollable event.
- But by definition of $\mathcal{L}(S/G)$, we have that

$$[s \in \overline{K} = \mathcal{L}(S/G)] \wedge [s\sigma \in \mathcal{L}(G)] \wedge [\sigma \in S(s)] \Rightarrow s\sigma \in \mathcal{L}(S/G) = \overline{K} \ .$$

- Thus we have shown that

$$[s \in \overline{K}] \wedge [\sigma \in E_{uc}] \wedge [s\sigma \in \mathcal{L}(G)] \Rightarrow s\sigma \in \overline{K}$$

or, in terms of languages,

$$\overline{K}E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}$$

which is the controllability condition. **Q.E.D.**

About the Proof of CT:

- It is important to note that the proof of CT is *constructive* in the sense that if the controllability condition is satisfied, it gives us a supervisor that will achieve the required behavior. That supervisor is:

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \overline{K}\} \ .$$

About Controllability:

- The *controllability* condition in CT is intuitive and a central concept in supervisory control. It can be paraphrased by: *"if you cannot prevent it, it should be legal."*
  We state a general definition of this notion:

  Definiton of controllability: Let K and $M = \overline{M}$ be languages over an event set E. Let E_{uc} be a designated subset of E. K is said to be *controllable* w.r.t. M and E_{uc} if

  $$\overline{K}E_{uc} \cap M \subseteq \overline{K} \ .$$

- Observe that by definition, controllability is a property of the prefix-closure of a language. Thus K is controllable iff $\overline{K}$ is controllable.
- The definition of controllability also provides an implementable test for verifying if this property holds or not, in the case of regular languages.
  It should be apparent that the computational complexity of this test is $O(|E|mn)$, where m is the number of states of the machine that generates $\overline{K}$ and n is the number of states of G.
- **UMDES-LIB:** controllability can be tested using the command `ctrb`.

The Nonblocking Controllability Theorem

Theorem (NCT): Consider a DES G where $E_{uc} \subseteq E$ is the set of uncontrollable events. Consider also the language $K \subseteq \mathcal{L}_m(G)$, where $K \neq \emptyset$.
There exists a *nonblocking* supervisor S for G such that

$$\mathcal{L}_m(S/G) = K \quad (\Rightarrow \mathcal{L}(S/G) = \overline{K})$$

iff the two following conditions hold:

1. [controllability] $\overline{K}E_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}$
2. [$\mathcal{L}_m(G)$-closure] K is $\mathcal{L}_m(G)$-closed, i.e., $K = \overline{K} \cap \mathcal{L}_m(G)$.

Proof of NCT:
[IF] For $s \in \mathcal{L}(G)$, As in the proof of CT, define $S(s)$ according to

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \overline{K}\} \ .$$

The proof of CT already established that with this S, $\mathcal{L}(S/G) = \overline{K}$.
Therefore, $\mathcal{L}_m(S/G) = K$ follows by applying the $\mathcal{L}_m(G)$-closure condition. Thus S is nonblocking.

[ONLY IF] Let there exist an admissible S such that $\mathcal{L}(S/G) = \overline{K}$ and $\mathcal{L}_m(S/G) = K$.
Then by definition of $\mathcal{L}_m(S/G)$, we have that $K = \overline{K} \cap \mathcal{L}_m(G)$, which is the $\mathcal{L}_m(G)$-closure condition.
The remainder of the proof is identical to that of CT.

Q.E.D.

About the Proof of NCT: The proof is constructive, with the same supervisor as in the case of CT:

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \overline{K}\} \ .$$

About $\mathcal{L}_m(G)$-closure:

This condition is of technical nature.

It can be argued that, unlike the controllability condition, $\mathcal{L}_m(G)$-closure will often hold by *construction* of K (interpreted here as legal behavior).

The reasons for that are as follows:

1. In practice, marking is a property of the uncontrolled system G, modeled by proper construction of X_m.
2. Specifications are usually stated in terms of prefix-closed languages, say $\overline{K_s}$.
3. The legal marked language is then obtained by doing $K = \overline{K_s} \cap \mathcal{L}_m(G)$.
4. Such a K is guaranteed to be $\mathcal{L}_m(G)$-closed (Problem 2.4 (b)!).

We now discuss the generalization of the Nonblocking Controllability Theorem (NCT) to the case of control under partial observation.

- It should be clear that this generalization will require another condition beyond the controllability and $\mathcal{L}_m(G)$-closure conditions.

  This extra condition will be called *observability*.

- Intuitively, observability means: *if you cannot differentiate between two traces, then these traces should require the same control action."* Another way to phrase this, from the point of view of event disablement, is: *"if you must disable an event, then you should not lose anything you need in the legal behavior."* This intuition is formalized in the following definition.

- **Observability:** Let K and $M = \overline{M}$ be languages over an event set E. Let E_c be a designated subset of E. Let E_o be another designated subset of E with P as the corresponding natural projection from E^* to E_o^*.
  K is said to be *observable* w.r.t. M, P, and E_c if for all $s \in \overline{K}$ and for all $\sigma \in E_c$,

$$[s\sigma \notin \overline{K}] \wedge [s\sigma \in M] \Rightarrow P^{-1}[P(s)]\sigma \cap \overline{K} = \emptyset \ .$$

  (Note the slight abuse of notation in the definition: $P^{-1}[P(s)]\sigma$ stands for $P^{-1}[P(s)]\{\sigma\}$.)

- **Comments:**

  1. The right-hand side of the implication in the definition identifies all traces in $\overline{K}$ that have the same projection as s and can be continued with event σ. If this set is not empty, this means that $\overline{K}$ contains two traces, s and s', such that $P(s) = P(s')$, and where $s\sigma \notin \overline{K}$ while $s'\sigma \in \overline{K}$. If this happens, then clearly no P-supervisor can exactly achieve language $\overline{K}$.
  2. If the parameter E_c is omitted from the definition, then it will be understood to be equal to E. The parameter E_c in included in the definition in order for the property of observability not to "overlap" with the property of controllability. (See the proof of COT below.) Examining the definition of observability, we can see that when $s \in \overline{K}$, $s\sigma \in \mathcal{L}(G)$, and $\sigma \in E_{uc}$, then controllability implies that $s\sigma \in \overline{K}$, that is, there is no need to "worry" about observability issues for uncontrollable events.
  3. As in the case of controllability, the observability of a language depends only on the prefix-closure of this language. Thus K is observable iff $\overline{K}$ is observable.

The Controllability and Observability Theorem

Theorem (COT): Consider a DES G where $E_{uc} \subseteq E$ is the set of uncontrollable events and $E_o \subseteq E$ is the set of observable events. Let P be the natural projection from E^* to E_o^*. Consider also the language $K \subseteq \mathcal{L}_m(G)$, where $K \neq \emptyset$.
There exists a *nonblocking* P-supervisor S_P for G such that

$$\mathcal{L}_m(S_P/G) = K$$

iff the three following conditions hold:

1. K is controllable w.r.t. $\mathcal{L}(G)$ and E_{uc};
2. K is observable w.r.t. $\mathcal{L}(G)$, P, and E_c;
3. K is $\mathcal{L}_m(G)$-closed.

Proof of COT:
[IF] For $t \in P[\mathcal{L}(G)]$, define $S_P(t)$ according to

$$S_P(t) = E_{uc} \cup \{\sigma \in E_c : \exists s'\sigma \in \overline{K}(P(s') = t)\} \ .$$

This supervisor enables after trace s: (i) all uncontrollable events and (ii) all controllable events that extend any trace s', that projects to t, inside of $\overline{K}$. Part (i) ensures that S_P is admissible, i.e., that it never disables a feasible uncontrollable event.
We now prove that with this S_P, $\mathcal{L}(S_P/G) = \overline{K}$. Then, as before, the $\mathcal{L}_m(G)$-closure condition will imply that $\mathcal{L}_m(S_P/G) = K$ and S_P is nonblocking.
The proof is by induction on the length of the traces in the two languages.

- The base case is for traces of length 0. But $\varepsilon \in \mathcal{L}(S_P/G)$ by definition and $\varepsilon \in \overline{K}$ since $K \neq \emptyset$ by assumption. Thus the base case holds.
- The induction hypothesis is that for all traces s such that $|s| \leq n$, $s \in \mathcal{L}(S_P/G)$ iff $s \in \overline{K}$. We now prove the same for traces of the form $s\sigma$ where $|s| = n$.

– Let $s\sigma \in \mathcal{L}(S_P/G)$. By definition of $\mathcal{L}(S_P/G)$, this implies that

$$[s \in \mathcal{L}(S_P/G)] \wedge [\sigma \in S_P[P(s)]] \wedge [s\sigma \in \mathcal{L}(G)]$$

which in turn implies that

$$[s \in \overline{K}] \wedge [\sigma \in S_P[P(s)]] \wedge [s\sigma \in \mathcal{L}(G)]$$

using the induction hypothesis.

* Now, if $\sigma \in E_{uc}$, then the controllability condition immediately yields $s\sigma \in \overline{K}$.
* On the other hand, if $\sigma \in E_c$, then by the above definition of S_P and with $t = P(s)$, we obtain that there exists $s'\sigma \in \overline{K}$ such that $P(s') = t = P(s)$, that is, we have that

$$P^{-1}[P(s)]\sigma \cap \overline{K} \neq \emptyset \ .$$

But since $s\sigma \in \mathcal{L}(G)$, we must have that $s\sigma \in \overline{K}$, otherwise observability would be contradicted.

This completes the proof that $s\sigma \in \overline{K}$.

– For the other direction of the induction step, let $s\sigma \in \overline{K}$. Then $s\sigma \in \mathcal{L}(G)$ since by assumption $\overline{K} \subseteq \overline{\mathcal{L}_m(G)} \subseteq \mathcal{L}(G)$.

* Now, if $\sigma \in E_{uc}$, then $\sigma \in S_P[P(s)]$ by the admissibility of S_P.
* On the other hand, if $\sigma \in E_c$, then by the above definition of $S_P[P(s)]$, we also obtain that $\sigma \in S_P[P(s)]$.

Overall, we have that

$$[s \in \overline{K}] \wedge [\sigma \in S_P[P(s)]] \wedge [s\sigma \in \mathcal{L}(G)]$$

which in turns implies that

$$[s \in \mathcal{L}(S_P/G)] \wedge [\sigma \in S_P[P(s)]] \wedge [s\sigma \in \mathcal{L}(G)]$$

using the induction hypothesis. It then immediately follows that $s\sigma \in \mathcal{L}(S_P/G)$.

– This completes the proof of the induction step.

[ONLY IF]

- Let S_P be an admissible P-supervisor such that $\mathcal{L}(S_P/G) = \overline{K}$ and $\mathcal{L}_m(S_P/G) = K$. The proof that controllability and $\mathcal{L}_m(G)$-closure hold is the same as in the Nonblocking Controllability Theorem; the only modification is that $S(s)$ there is now replaced by $S_P[P(s)]$.
- To prove that observability must also hold, take $s \in \overline{K}$ and $\sigma \in E_c$, such that $s\sigma \notin \overline{K}$ and $s\sigma \in \mathcal{L}(G)$.
  - From $s \in \overline{K}$, $\sigma \in E_c$, $s\sigma \notin \overline{K}$, and $\mathcal{L}(S_P/G) = \overline{K}$, we must have that $\sigma \notin S_P[P(s)]$, otherwise $\mathcal{L}(S_P/G) \neq \overline{K}$.
  - But this means that there cannot exist $s'\sigma \in \overline{K}$ such that $P(s') = P(s)$, otherwise $\mathcal{L}(S_P/G) \neq \overline{K}$ since S_P does not distinguish between s and s'.

– That is, we must have that

$$P^{-1}[P(s)]\sigma \cap \overline{K} = \emptyset \ .$$

This proves the observability condition. **Q.E.D.**

About the Proof of COT:

- Again, we note that the proof of COT is *constructive* in the sense that if the controllability, observability, and $\mathcal{L}_m(G)$-closure conditions are satisfied, it gives us a supervisor that will achieve the required behavior. That supervisor is:

$$S_P(t) = E_{uc} \cup \{\sigma \in E_c : \exists s'\sigma \in \overline{K}(P(s') = t)\} \ .$$

We present a useful corollary that explicity states a special case of COT when only $\mathcal{L}(S_P/G)$ is of concern:

Corollary of COT: Consider a DES G where $E_{uc} \subseteq E$ is the set of uncontrollable events and $E_o \subseteq E$ is the set of observable events. Let P be the natural projection from E^* to E_o^*. Let $K \subseteq \mathcal{L}(G)$, $K \neq \emptyset$. Then there exists S_P such that $\mathcal{L}(S_P/G) = \overline{K}$ iff K is controllable w.r.t. $\mathcal{L}(G)$ and E_{uc} and observable w.r.t. $\mathcal{L}(G)$, P, and E_c.

About Observability:

- For regular K and M, it can be shown that the worst-case computational complexity of testing observability is $O(m^2 n)$, where m and n are the number of states of the generators of $\overline{K}$ and M, respectively; here, we absorbed $|E|$ in the constants of $O(\cdot)$.
- An intuitive test is to build the observer of $H \times G$ with respect to E_o and see if there are states of that observer where there is a *control conflict*, namely a state of $(H \times G)_{obs}$ where two component states require two different control actions on the same controllable event.
- **UMDES-LIB:** the command `obs` tests observability.

We will see later what can be done when the legal language L_a or L_{am} is not controllable or not observable. It turns out that uncontrollability is in some sense "easier" to deal with than unobservability. But before that, we discuss the issue of building finite realizations of supervisors.

3.3: REALIZATION AND DESIGN OF CONTROLLERS

3.3: Realization and Design of Supervisors

Standard Realization by Automata

- Let there exist a supervisor S such that $\mathcal{L}(S/G) = \overline{K}$. Thus K is a controllable sublanguage of $\mathcal{L}(G)$.
  - It suffices for now to consider $\mathcal{L}(S/G)$; if we are concerned with marked languages, then under the $\mathcal{L}_m(G)$-closure assumption, we get that $\mathcal{L}_m(S/G) = K$ and that S is nonblocking.
  - We rule out the two trivial cases where
    1. $\overline{K} = \emptyset$ (controllable by definition but unachievable by control unless the system is "never turned on") and
    2. $\overline{K} = \mathcal{L}(G)$ (also controllable by definition but S plays no role so need not be there).
  - Observe that the domain of S can be restricted to $\mathcal{L}(S/G)$ without loss of generality.

- The issue here is that for implementation purposes, we need to build a convenient *representation* of the function S other than simply listing $S(s)$ for all $s \in \mathcal{L}(S/G)$, as was done in the proof of the (Nonblocking) Controllability Theorem.
- Given that we are using an automaton to represent the system, let us also use an automaton to represent the supervisor S. Then when we will be dealing with regular languages (here, the languages $\mathcal{L}(G)$, $\mathcal{L}_m(G)$, and K), the required representations will be *finite* and thus implementable.

  $\rightarrow$ We will call an automaton representation of supervisor S a *realization* of S.
- It is important to emphasize that we are now concerned with building *off-line* a complete realization of S for all possible behaviors of the controlled system $\mathcal{L}(S/G)$. This realization will then be stored and at run time it will suffice to "read" the desired control action (i.e., the control action for the trace of events observed up to now).

  (The issue of calculating $S(s)$ *on-line* is also of interest but it will not be discussed here.)

- It turns out that an easy way to build a realization of S is to build an automaton that recognizes the language $\overline{K}$.
  - Let R be such an automaton, i.e., let

$$R = (Y, E, g, \Gamma_R, y_0, Y)$$

    where R is trim and

$$\mathcal{L}_m(R) = \mathcal{L}(R) = \overline{K} \ .$$

  - Now if we "connect" R to G by the product operation, the result $R \times G$ is exactly the behavior that we desire for the closed-loop system S/G:

$$\begin{aligned} \mathcal{L}(R \times G) &= \mathcal{L}(R) \cap \mathcal{L}(G) \\ &= \overline{K} \cap \mathcal{L}(G) \\ &= \overline{K} = \mathcal{L}(S/G) \\ \mathcal{L}_m(R \times G) &= \mathcal{L}_m(R) \cap \mathcal{L}_m(G) \\ &= \overline{K} \cap \mathcal{L}_m(G) \\ &= \mathcal{L}(S/G) \cap \mathcal{L}_m(G) = \mathcal{L}_m(S/G) \ . \end{aligned}$$

  - Note that since R is defined to have the same event set as G (namely E), then $R \,||\, G = R \times G$.

- What the above means is that the control action $S(s)$ is "encoded" in the transition structure of R.
  - Namely,
    $$S(s) = \Gamma_{R\times G}(g \times f((y_0, x_0), s)) = \Gamma_R(g(y_0, s))$$
    where the last equality follows from the fact that $\overline{K} \subseteq \mathcal{L}(G)$ (note that $g \times f$ denotes the state transition function of $R \times G$).
  - Another way to see that $S(s) = \Gamma_R(g(y_0, s))$ is to consider the respective definitions of $S(s)$ and R, which are both based on $\overline{K}$, and to invoke the controllability of $\mathcal{L}(R)$.
- Of course, $R \times G$ is a composition of two automata that is defined without reference to a control mechanism *à la* S/G. The interpretation with our control paradigm is as follows:

  "Let G be in state x and R in state y following the execution of $s \in \mathcal{L}(S/G)$. G generates an event σ that is currently enabled. This means that this event is also present in the active event set of R at y. Thus R also executes the event, as a passive observer of G. Let x' and y' be the new states of G and R after the execution of σ. The set of enabled events of G at $s\sigma$ is now given by the active event set of R at y'."
- Thus we have built a representation of S that in the case of a regular K will only require finite memory.
- We will call the R derived by the above process the *standard realization* of S.

Induced Supervisors

- The standard realization of S by automaton R raises the reverse question:

  Question: If we are given automaton C and form the product $C \times G$, can that be interpreted as controlling G by some supervisor?

  Answer: Not always! It depends on the controllability of $\mathcal{L}(C)$.

- **The notion of induced supervisor:**
  - Let $C = (Y, E, h, \Gamma_C, y_0, Y)$ be a trim automaton.
  - Let us define the *supervisor for G induced by C* as follows:
    for all $s \in \mathcal{L}(G)$,

$$S_i^C(s) = \begin{cases} [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \mathcal{L}(C)\} & \text{if } s \in \mathcal{L}(G) \cap \mathcal{L}(C) \\ E_{uc} & \text{otherwise.} \end{cases}$$

  - Note that we need to add $E_{uc} \cap \Gamma(f(x_0, s))$ to make sure that S_i^C is an admissible supervisor, i.e., that is does not disable a feasible uncontrollable event of G. Precisely for that reason, we get the following fact.

- **Fact:** $\mathcal{L}(S_i^C/G) = \mathcal{L}(C \times G)$ iff $\mathcal{L}(C)$ is controllable w.r.t. $\mathcal{L}(G)$ and E_{uc}.

- The proof of the preceding fact is left as an exercise.

  Intuitive explanation:

  1. If $\mathcal{L}(C)$ is controllable w.r.t. $\mathcal{L}(G)$, then when doing $C \times G$, C can never prevent G from executing an uncontrollable event, because all such transitions will always be defined in C.
  2. Thus the product $C \times G$ can indeed be viewed as the control of G by S_i^C (since only controllable events are "disabled" by the product).
  3. In this case, the resulting closed-loop behavior is

$$\begin{aligned}
\mathcal{L}(S_i^C/G) &= \mathcal{L}(C \times G) \\
&= \mathcal{L}(C) \cap \mathcal{L}(G) \\
\mathcal{L}_m(S_i^C/G) &= \mathcal{L}_m(C \times G) \\
&= \mathcal{L}_m(C) \cap \mathcal{L}_m(G) \\
&= \mathcal{L}(C) \cap \mathcal{L}_m(G) \\
&= \mathcal{L}(C) \cap \mathcal{L}(G) \cap \mathcal{L}_m(G) \\
&= \mathcal{L}(S_i^C/G) \cap \mathcal{L}_m(G) \ .
\end{aligned}$$

Partial Observation Case

- The process of building a realization of a P-supervisor is slightly more involved, due to the presence of unobservable events (which may or may not be controllable).
- Based on the construction of S_P in the proof of the Controllability and Observability Theorem, consider the following initial steps for constructing a realization of S_P:
  1. Build a trim automaton R that generates and marks $\overline{K}$;
  2. Build R_{obs}, the observer for R using the procedure presented earlier for the given set of observable events.
- At this point, we cannot write as before that the active event set of R_{obs} encodes the set of enabled events by the function S_P, since the event set of R_{obs} is E_o and thus contains no information about the desired control action w.r.t. the unobservable events.
  - That information is contained in R.
  - But since each state of R_{obs} is a set of states of R, we can recover the required information.

- Here is how to proceed to recover the control action:

  3. Let t be the current trace of *observable* events and let $x_{obs,current}$ be the state of R_{obs} after the execution of t.

     This means that after the last (observable) event in t, but before the next observable event, automaton R could be in anyone of the states in the set $x_{obs,current}$.

  4. Then we have that

$$S_P^{real}(t) = \bigcup_{x \in x_{obs,current}} [\Gamma_R(x)] \ .$$

- Here, the interpretation of the realization with R and R_{obs} is one where R_{obs} is a "passive observer" that follows the observable (only) transitions of G; the desired control action (namely $S_P(t)$) is then obtained by looking at the current state of R_{obs} (which is a set) and by considering the corresponding active event sets in R of all the states in this set.

- This may seem somewhat messy, yet, when K is regular, all this information requires finite memory, which was our goal.

  Note that we can precompute all the enabled events for each state of R_{obs} so that it is not necessary to store R itself.

- As before, we call this the *standard realization* of S_P.

3.4: DEALING WITH UNCONTROLLABILITY

3.4: Dealing with Uncontrollability

The Property of Controllability

Suppose that a given $K \subseteq M$ (where K is not necessarily prefix-closed) is not controllable w.r.t given $M = \overline{M} \subseteq E^*$ and $E_{uc} \subseteq E$, i.e.,

$$\overline{K} E_{uc} \cap M \not\subseteq \overline{K} \ .$$

(In this section, unless otherwise specified, controllability will always be w.r.t. M and E_{uc}.) We will consider the following two languages derived from K:

- $K^{\uparrow C}$: the *supremal controllable sublanguage of* K;
- $K^{\downarrow C}$: the *infimal prefix-closed and controllable superlanguage of* K.

Overall, we have the following inequalities:

$$\emptyset \subseteq K^{\uparrow C} \subseteq K \subseteq \overline{K} \subseteq K^{\downarrow C} \subseteq M \ .$$

We now prove some properties that guarantee the existence of these two languages.

Proposition: The property of controllability.

1. If K_1 and K_2 are controllable, then $K_1 \cup K_2$ is controllable.
2. If K_1 and K_2 are controllable, then $K_1 \cap K_2$ need not be controllable.
3. If K_1 and K_2 are nonconflicting and both are controllable, then $K_1 \cap K_2$ is controllable. [Reminder: K_1 and K_2 are said to be nonconflicting whenever $\overline{K_1} \cap \overline{K_2} = \overline{(K_1 \cap K_2)}$.]
4. If K_1 and K_2 are prefix-closed and controllable, then $K_1 \cap K_2$ is prefix-closed and controllable.

Proof:

1. The result is proved using the definition of controllability and properties of prefix-closure:

$$\begin{aligned}
\overline{(K_1 \cup K_2)}E_{uc} \cap M &= (\overline{K_1} \cup \overline{K_2})E_{uc} \cap M \\
&= (\overline{K_1}E_{uc} \cap M) \cup (\overline{K_2}E_{uc} \cap M) \\
&\subseteq \overline{K_1} \cup \overline{K_2} \\
&= \overline{(K_1 \cup K_2)} \quad .
\end{aligned}$$

2. Consider the following counter-example:
   $E_{uc} = \{\alpha\}$, $E = \{\alpha, \beta, \gamma\}$
   $M = \{\varepsilon, \alpha, \alpha\beta, \alpha\gamma\}$
   $K_1 = \{\varepsilon, \alpha\beta\}$ and $K_2 = \{\varepsilon, \alpha\gamma\}$.

3. In general, we have that

$$\begin{aligned} \overline{(K_1 \cap K_2)}E_{uc} \cap M &\subseteq (\overline{K_1} \cap \overline{K_2})E_{uc} \cap M \\ &= (\overline{K_1}E_{uc} \cap M) \cap (\overline{K_2}E_{uc} \cap M) \\ &\subseteq \overline{K_1} \cap \overline{K_2} \ . \end{aligned}$$

   But the nonconflicting property says that $\overline{K_1} \cap \overline{K_2} = \overline{(K_1 \cap K_2)}$, from which we obtain the desired result.

4. Immediate from above given that prefix-closed languages are always nonconflicting.

Q.E.D.

Clearly, parts 1 and 4 of the preceding proposition hold for *arbitrary* unions and intersections.

Definition: We define the two classes of languages

$$\begin{aligned} \mathcal{C}_{in}(K) &:= \{L \subseteq K : \overline{L}E_{uc} \cap M \subseteq \overline{L}\} \\ \mathcal{CC}_{out}(K) &:= \{L \subseteq E^* : (K \subseteq L \subseteq M) \wedge (\overline{L} = L) \wedge (\overline{L}E_{uc} \cap M \subseteq \overline{L})\} \ . \end{aligned}$$

About the Controllable Sublanguages of K:

- The class $\mathcal{C}_{in}(K)$ is a partially ordered set (or *poset*) that is closed under arbitrary unions (the partial order is set inclusion).
- Thus $\mathcal{C}_{in}(K)$ possesses a unique *supremal* element. Namely,

$$K^{\uparrow C} := \bigcup_{L \in \mathcal{C}_{in}(K)} L$$

  is a well-defined element of $\mathcal{C}_{in}(K)$.
- We call $K^{\uparrow C}$ the *supremal controllable sublanguage of* K.
  - In the worst case, $K^{\uparrow C} = \emptyset$, since $\emptyset \in \mathcal{C}_{in}(K)$.
  - If K is controllable, then $K^{\uparrow C} = K$.
  - Observe that $K^{\uparrow C}$ need not be prefix-closed in general.

- We will refer to "$\uparrow C$" as the operation of obtaining the supremal controllable sublanguage.
  - It is immediate from the definition of $\uparrow C$ that this operation is *monotone*, i.e.,

$$K_1 \subseteq K_2 \Rightarrow K_1^{\uparrow C} \subseteq K_2^{\uparrow C} \ .$$

  - Algorithms that implement the $\uparrow C$ operation are presented later in this section.
  - The following results deal with closure properties of the $\uparrow C$ operation.

Fact: Closure properties of the $\uparrow C$ operation.

1. If K is prefix-closed, then so is $K^{\uparrow C}$.
2. If $K \subseteq \mathcal{L}_m(G)$ is $\mathcal{L}_m(G)$-closed, then so is $K^{\uparrow C}$. (Here $M = \mathcal{L}(G)$).
3. In general, $\overline{K^{\uparrow C}} \subseteq (\overline{K})^{\uparrow C}$.
4. $(K_1 \cap K_2)^{\uparrow C} \subseteq K_1^{\uparrow C} \cap K_2^{\uparrow C}$
5. $(K_1 \cap K_2)^{\uparrow C} = (K_1^{\uparrow C} \cap K_2^{\uparrow C})^{\uparrow C}$
6. If $K_1^{\uparrow C}$ and $K_2^{\uparrow C}$ are nonconflicting, then $(K_1 \cap K_2)^{\uparrow C} = K_1^{\uparrow C} \cap K_2^{\uparrow C}$.
7. $(K_1 \cup K_2)^{\uparrow C} \supseteq K_1^{\uparrow C} \cup K_2^{\uparrow C}$.

Proof:

1. Since $K^{\uparrow C}$ is controllable, then so is its prefix-closure. But $\overline{K^{\uparrow C}} \subseteq \overline{K} = K$, which implies that $\overline{K^{\uparrow C}} \subseteq K^{\uparrow C}$. This suffices to prove the result.
2. We need to show that $K^{\uparrow C} = \overline{K^{\uparrow C}} \cap \mathcal{L}_m(G)$.

   2.1. ($\subseteq$) is immediate.

   2.2. For ($\supseteq$), define $K' = \overline{K^{\uparrow C}} \cap \mathcal{L}_m(G)$.

   - Then

$$\begin{aligned} K^{\uparrow C} &\subseteq \overline{K^{\uparrow C}} \cap \mathcal{L}_m(G) \\ \Rightarrow \overline{K^{\uparrow C}} &\subseteq \overline{\overline{K^{\uparrow C}} \cap \mathcal{L}_m(G)} = \overline{K'} \ . \end{aligned}$$

   - Also

$$\begin{aligned} & \overline{K^{\uparrow C}} \cap \mathcal{L}_m(G) \subseteq \overline{K^{\uparrow C}} \\ \Rightarrow \quad & \overline{\overline{K^{\uparrow C}} \cap \mathcal{L}_m(G)} \subseteq \overline{K^{\uparrow C}} \\ \Rightarrow \quad & \overline{K'} \subseteq \overline{K^{\uparrow C}} \ . \end{aligned}$$

   - Therefore, $\overline{K'} = \overline{K^{\uparrow C}}$.
     Since $\overline{K^{\uparrow C}}$ in controllable, then so is K', by definition of controllability.

- But $K' \subseteq \overline{K} \cap \mathcal{L}_m(G) = K$ (by hypothesis).
  It follows that $K' \subseteq K^{\uparrow C}$, i.e.,

$$\overline{K^{\uparrow C}} \cap \mathcal{L}_m(G) \subseteq K^{\uparrow C} \ .$$

3. The inclusion is easily proved from the definition and monotonicity of $\uparrow C$. The following example demonstrates that this inclusion can be strict:
   $E_{uc} = \{\beta_1, \beta_2\}$, $E = E_{uc} \cup \{\alpha_1\}$
   $M = \overline{\{\alpha_1\beta_2\alpha_1, \alpha_1\beta_1\alpha_1\beta_1\alpha_1\}}$
   $K = \{\alpha_1\beta_2\alpha_1, \alpha_1\beta_1\alpha_1\}$.

4. The proofs of the remaining properties are left as an exercise.

Q.E.D.

About the Controllable Superlanguages of K:

- The class $\mathcal{CC}_{out}(K)$ is a poset that is closed under arbitrary intersections (and unions).
- Thus $\mathcal{CC}_{out}(K)$ possesses a unique *infimal* element. Namely,

$$K^{\downarrow C} := \bigcap_{L \in \mathcal{CC}_{out}(K)} L$$

  is a well-defined element of $\mathcal{CC}_{out}(K)$.
- We call $K^{\downarrow C}$ the *infimal prefix-closed and controllable superlanguage of K.*
  - In the worst case, $K^{\downarrow C} = M$, since $M \in \mathcal{CC}_{out}(K)$.
  - If K is controllable, then $K^{\downarrow C} = \overline{K}$.

- We will refer to $\downarrow C$ as the operation of obtaining the infimal prefix-closed and controllable superlanguage.
  - It is immediate from the definition of $\downarrow C$ that this operation is *monotone*, i.e.,
    $$K_1 \subseteq K_2 \Rightarrow K_1^{\downarrow C} \subseteq K_2^{\downarrow C} .$$
  - An algorithm that implements the $\downarrow C$ operation is presented later in this section.
- We list some other properties of the $\downarrow C$ operation that follow from its definition.
  1. $(K_1 \cap K_2)^{\downarrow C} \subseteq K_1^{\downarrow C} \cap K_2^{\downarrow C}$
  2. If K_1 and K_2 are nonconflicting, then $(K_1 \cap K_2)^{\downarrow C} = K_1^{\downarrow C} \cap K_2^{\downarrow C}$.
  3. $(K_1 \cup K_2)^{\downarrow C} = K_1^{\downarrow C} \cup K_2^{\downarrow C}$.

Study of the $\uparrow C$ Operation for Regular Languages

In this section, we further characterize and discuss the calculation of $K^{\uparrow C}$ for a given language K that is not controllable (as usual, controllability is w.r.t. (prefix-closed) M and E_{uc}).
We restrict attention to the case of regular languages, namely K and M are regular. We state, without proof, a *key result*:

Theorem: If K and M are regular languages, then $K^{\uparrow C}$ is regular and $||K^{\uparrow C}|| \leq ||K|| \cdot ||M|| + 1$.

Proof: The proof is somewhat technical and is omitted here. See the paper by Wonham & Ramadge in the *SIAM Journal on Control and Optimization*, 1987, pp. 637–659. (This paper is the original reference on the supremal controllable sublanguage.)

- Several algorithms have been proposed in the literature to implement the $\uparrow C$ operation for regular languages.
- These algorithms all have the same worst-case complexity, namely $O(n^2m^2|E|)$, where m is the number of states of G and n that of H, the generator of $\overline{K}$.
- The complexity of calculating $\uparrow C$ can be reduced to $O(nm|E|)$ (in the worst case) in two special cases: (i) K is prefix-closed and (ii) K is *livelock-free* (defined and discussed later in this section).
- We present one algorithm for obtaining a trim automaton that marks $K^{\uparrow C}$, given the two automata G and H.

  (See the literature for other, possibly more "efficient", algorithms.)

  We refer to this algorithm as the "Standard Algorithm for $\uparrow C$".

Standard Algorithm for $\uparrow C$

Step 0: Let $G = (X, E, f, \Gamma, x_0)$ be an automaton that generates M, i.e., $\mathcal{L}(G) = M$.

Let $H = (Y, E, g, \Gamma_H, y_0, Y_m)$ be such that $\mathcal{L}_m(H) = K$ and $\mathcal{L}(H) = \overline{K}$, where it is assumed that $K \subseteq \mathcal{L}(G)$.

Step 1: Let

$$H_0 := (Y_0, E, g_0, \Gamma_{H_0}, (y_0, x_0), Y_{0,m}) = H \times G$$

where $Y_0 \subseteq Y \times X$. Treat all states of G as marked for the purpose of determining $Y_{0,m}$.

By assumption, $\mathcal{L}_m(H_0) = K$ and $\mathcal{L}(H_0) = \overline{K}$.

States of H_0 will be denoted by pairs (y, x).

Set $i = 0$.

Step 2: Calculate

Step 2.1:

$$\begin{aligned} Y_i' &= \{(y,x) \in Y_i : \Gamma(x) \cap E_{uc} \subseteq \Gamma_{H_i}((y,x))\} \\ g_i' &= g_i | Y_i' \\ Y_{i,m}' &= Y_{i,m} \cap Y_i' \end{aligned}$$

where the notation "$|$" stands for "restricted to."

Step 2.2:

$$H_{i+1} = Trim(Y_i', E, g_i', (y_0, x_0), Y_{i,m}') \ .$$

If H_{i+1} is the empty automaton, i.e, (y_0, x_0) is deleted in the above calculation, then $K^{\uparrow C} = \emptyset$ and STOP.

Otherwise, set

$$H_{i+1} =: (Y_{i+1}, E, g_{i+1}, (y_0, x_0), Y_{i+1,m}) \ .$$

Step 3: If $H_{i+1} = H_i$, then

$$\mathcal{L}_m(H_{i+1}) = K^{\uparrow C} \ \text{ and } \ \mathcal{L}(H_{i+1}) = \overline{K^{\uparrow C}}$$

and STOP.

Otherwise, set $i \leftarrow i + 1$ and go to Step 2.

Comments about the Standard Algorithm:

- The condition
$$\Gamma(x) \cap E_{uc} \subseteq \Gamma_{H_i}((y,x))$$
that is tested in Step 2 is called the *active event set constraint.*

- The reason for Step 1 is that we need to be able to map the states of H to those of G when the controllability condition
$$\overline{K}E_{uc} \cap M \subseteq \overline{K} \ ,$$
a trace condition, is to be tested in the form of the active event set constraint in the automata representations of the languages.

- The above-referenced paper by Ramadge & Wonham contains the proof of the correctness of this algorithm. The intuition is clear however.
  - Whenever a trace contains a prefix that violates the controllability condition, all traces in K that contain that prefix need to be removed. This is why we can delete *states* that violate the active event set constraint (Step 2.1).
  - However, upon deleting states, we need to take the trim of the resulting automaton so that it gives the same result as deleting traces from K (Step 2.2).

Remarks on Prefix-Closed and Livelock-Free Languages

- Let us define a regular language to be *livelock-free* if every cycle in an automaton representation of this language contains a marked state.
- When the language K is either prefix-closed or livelock-free, it is possible to modify the "Standard Algorithm for $\uparrow C$" so that it has total worst-case complexity $O(|X_0||E|)$.

  $\rightarrow$ This can be achieved (roughly speaking) by an implementation of Step 2 that performs the verification of the active event set constraint and the trim operation in a "forward search manner" and in doing so avoids iterating between Steps 2 and 3.

  (See the literature for precise details.)
- However, in the general case, linear complexity (in nm) cannot be achieved.

UMDES-LIB

In **UMDES-LIB**, the command `supcon_std` can be used to perform the $\uparrow C$ operation.

The $\downarrow C$ Operation

In this section, we consider the calculation and properties of $K^{\downarrow C}$ for a given language K that is not controllable (again, controllability is w.r.t. M and E_{uc}).

It turns out that $K^{\downarrow C}$ is characterized by a simple formula on languages. This formula immediately suggests an algorithm for computing a generator of $K^{\downarrow C}$ given automata that generate K and M.

Theorem: $K^{\downarrow C} = \overline{K} E_{uc}^* \cap M$.

Proof: Let $K' := \overline{K} E_{uc}^* \cap M$. The inclusion $K^{\downarrow C} \subseteq K'$ follows by observing that $K' \supseteq K$ and that it is prefix-closed and by verifying that K' is controllable:

$$\begin{aligned} K' E_{uc} \cap M &= \overline{K} E_{uc}^* E_{uc} \cap M E_{uc} \cap M \\ &\subseteq \overline{K} E_{uc}^* \cap M \\ &= K' \ . \end{aligned}$$

For the reverse inclusion, take any $L \in \mathcal{CC}_{out}(K)$. Then using the definition of $\mathcal{CC}_{out}$, we have that:

$$\begin{aligned} \overline{K} \cap M &\subseteq L \\ \overline{K} E_{uc} \cap M &\subseteq L E_{uc} \cap M \subseteq L \\ \overline{K} E_{uc}^2 \cap M &\subseteq L E_{uc} \cap M \subseteq L \\ &\cdots \\ \overline{K} E_{uc}^r \cap M &\subseteq L E_{uc} \cap M \subseteq L \quad \text{for all } r \geq 0 \ . \end{aligned}$$

Thus $K' \subseteq L$. Since L was arbitrary, we get that

$$K' \subseteq \bigcap_{L \in \mathcal{CC}_{out}(K)} L =: K^{\downarrow C} \ .$$

Q.E.D.

Corollary: If K and M are regular, then $K^{\downarrow C}$ is regular.

- Observe that there is no need to iterate in the computation of $K^{\downarrow C}$, in contrast to the computation of $K^{\uparrow C}$.
- The calculation of (a generator of) $K^{\downarrow C}$ is performed by implementing the above formula. If the automata that generate $\overline{K}$ and M have n and m states, respectively, then this calculation is of $O(nm|E|)$.

  If automata H and G generate $\overline{K}$ and M, respectively, (here marking is not an issue) then an automaton that generates $K^{\downarrow C}$ can be built as follows:

  1. Build a deterministic automaton that generates $\overline{K}E_{uc}^*$ from H; call this automaton H_{aug}. To do this, add a dead state to H and "partially complete to E_{uc}" the transition function of H by adding all the missing uncontrollable transitions to the dead state.
  2. The intersection in the above formula is then implemented by doing

  $$H_{aug} \times G =: H_{\downarrow C}$$

  3. $\mathcal{L}(H_{\downarrow C}) = K^{\downarrow C}$
- **UMDES-LIB:** The command `infcon` performs the $\downarrow C$ operation.

Some Supervisory Control Problems and Their Solutions

BSCP: Basic Supervisory Control Problem

Given DES G, $E_{uc} \subseteq E$, and legal language $L_a = \overline{L_a} \subseteq \mathcal{L}(G)$, build supervisor S such that:

1. $\mathcal{L}(S/G) \subseteq L_a$
2. $\mathcal{L}(S/G)$ is *as large as possible*, i.e., for any other S' such that $\mathcal{L}(S'/G) \subseteq L_a$,

$$\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G) \ .$$

Solution of BSCP:

- Requirement 2 means that we wish the solution S to be optimal w.r.t. set inclusion. Such a solution is said to be *minimally restrictive* (MRS).
- Using the results of the preceding sections, the solution is to choose S such that

$$\mathcal{L}(S/G) = L_a^{\uparrow C}$$

  as long as $L_a^{\uparrow C} \neq \emptyset$.
- Regular languages: S is realized by the automaton that results from the Standard Algorithm for $\uparrow C$.

BSCP-NB: Nonblocking Version of BSCP

Given DES G, $E_{uc} \subseteq E$, and legal (marked) language $L_{am} \subseteq \mathcal{L}_m(G)$, with L_{am} assumed to be $\mathcal{L}_m(G)$-closed, build *nonblocking* supervisor S such that:

1. $\mathcal{L}_m(S/G) \subseteq L_{am}$
2. $\mathcal{L}_m(S/G)$ is *as large as possible*, i.e., for any other nonblocking S' such that $\mathcal{L}_m(S'/G) \subseteq L_{am}$,

$$\begin{aligned} \mathcal{L}(S'/G) &\subseteq \mathcal{L}(S/G) \\ (\ \Rightarrow \mathcal{L}_m(S'/G) &\subseteq \mathcal{L}_m(S/G)\ ). \end{aligned}$$

Solution of BSCP-NB:

- Due to requirement 2, we call the desired solution S the *minimally restrictive nonblocking solution* (MRNBS).
- Using the results of the preceding sections, the solution is to choose nonblocking S such that

$$\mathcal{L}(S/G) = \overline{L_{am}^{\uparrow C}} \ \text{ and } \ \mathcal{L}_m(S/G) = L_{am}^{\uparrow C}$$

  as long as $L_{am}^{\uparrow C} \neq \emptyset$.

– It is important to note that since L_{am} is assumed to be $\mathcal{L}_m(G)$-closed, then $L_{am}^{\uparrow C}$ is also $\mathcal{L}_m(G)$-closed, which guarantees that $\mathcal{L}_m(S/G) = L_{am}^{\uparrow C}$ whenever $\mathcal{L}(S/G) = \overline{L_{am}^{\uparrow C}}$.

- Regular languages: S is realized by the automaton that results from the Standard Algorithm for $\uparrow C$.
- **Important Observation about Blocking:** Choosing S_{alt} such that

$$\mathcal{L}(S_{alt}/G) = (\overline{L_{am}})^{\uparrow C}$$

will satisfy requirement 1, *but S_{alt} may be blocking.* [Cf. previous fact on closure properties of the $\uparrow C$ operation.]

If $(\overline{L_{am}})^{\uparrow C}$ and $\mathcal{L}_m(G)$ are nonconflicting, then (verify!):

1. S_{alt} is nonblocking;
2. $(\overline{L_{am}})^{\uparrow C} = \overline{L_{am}^{\uparrow C}}$.

DuSCP: "Dual" Version of BSCP

Given DES G, $E_{uc} \subseteq E$, and *minimum* required language $L_{min} \subseteq \mathcal{L}(G)$, build supervisor S such that:

1. $\mathcal{L}(S/G) \supseteq L_{min}$
2. $\mathcal{L}(S/G)$ is *as small as possible*, i.e., for any other S' such that $\mathcal{L}(S'/G) \supseteq L_{min}$,

$$\mathcal{L}(S'/G) \supseteq \mathcal{L}(S/G) \ .$$

(Note that L_{min} need not be prefix-closed and could be given as a subset of $\mathcal{L}_m(G)$.)

Solution of DuSCP:

- Again, from prior results, the desired solution is to take S such that

$$\mathcal{L}(S/G) = L_{min}^{\downarrow C}$$

  which clearly meets requirements 1 and 2.

- Regular languages: S is realized by the automaton that results from the algorithm (formula) for $\downarrow C$.
- Observe that if L_{min} was given as a subset of $\mathcal{L}_m(G)$, we would get

$$\mathcal{L}_m(S/G) = L_{min}^{\downarrow C} \cap \mathcal{L}_m(G) \supseteq \overline{L_{min}} \cap \mathcal{L}_m(G) \supseteq L_{min} \ .$$

- There would no guarantee though that S be nonblocking.
- Indeed, the nonblocking version of **DuSCP** poses technical difficulties as the property of controllability is not preserved under intersection, unless other assumptions are made. We will not discuss this problem.

Finally, consider the following supervisory control problem where instead of a legal language or minimum language, we are given a *desired* language (L_{des}) and a *tolerated* language (L_{tol}).

SCPT: Supervisory Control Problem with Tolerance
Given DES G, $E_{uc} \subseteq E$, *desired* marked language $L_{des} \subseteq \mathcal{L}_m(G)$ and tolerated legal language $L_{tol} = \overline{L_{tol}} \subseteq \mathcal{L}(G)$, where $\overline{L_{des}} \subseteq L_{tol}$, build supervisor S such that:

1. $\mathcal{L}(S/G) \subseteq L_{tol}$
   (this means that we can never exceed the tolerated language);
2. For all prefix-closed and controllable $K \subseteq L_{tol}$,
$$K \cap L_{des} \subseteq \mathcal{L}(S/G) \cap L_{des}$$
   (this means that we want S to achieve as much of the desired language as possible);
3. For all prefix-closed and controllable $K \subseteq L_{tol}$,
$$K \cap L_{des} = \mathcal{L}(S/G) \cap L_{des} \Rightarrow \mathcal{L}(S/G) \subseteq K$$
   (this means that 2 is achieved with the smallest possible solution).

Solution of SCPT:

A little thought (!) shows that the solution is obtained by taking S such that

$$\mathcal{L}(S/G) = (L_{tol}^{\uparrow C} \cap L_{des})^{\downarrow C} \ .$$

This S is not guaranteed to be nonblocking. The nonblocking version of this problem may not have an optimal solution and poses technical difficulties.

3.5: MODULAR CONTROL

3.5: Modular Control

By modular control, we refer to the situation where the control action of supervisor S is given by some combination of the control actions of two or more supervisors, each working under full event observation. For simplicity, we consider the case of two supervisors and discuss their conjunction.

Conjunction of Supervisors

Given S_1 and S_2 each defined for DES G, define the *modular* supervisor denoted by $S_{1\wedge 2} : \mathcal{L}(G) \to 2^E$ and corresponding to the *conjunction* of the two individual supervisors:

$$S_{1\wedge 2}(s) := S_1(s) \cap S_2(s) \ .$$

Then it is straightforward to verify that

$$\begin{aligned} \mathcal{L}(S_{1\wedge 2}/G) &= \mathcal{L}(S_1/G) \cap \mathcal{L}(S_2/G) \\ \mathcal{L}_m(S_{1\wedge 2}/G) &= \mathcal{L}_m(S_1/G) \cap \mathcal{L}_m(S_2/G) \ . \end{aligned}$$

- Given standard realizations R_1 and R_2 of S_1 and S_2, respectively, then a standard realization of $S_{1\wedge 2}$ could be obtained by building $R = R_1 \times R_2$. But the point here is precisely *not* to build this realization, but rather to use the existing R_1 and R_2 and realize the control action $S_{1\wedge 2}(s)$ by taking the *intersection of the active event sets of* R_1 *and* R_2 at their respective states after the execution of s. We call this the *modular realization* of modular supervisor $S_{1\wedge 2}$. This modular approach saves on the size of the realization of $S_{1\wedge 2}$. If R_1 has n_1 states and R_2 has n_2 states, then we need only to store a total of $n_1 + n_2$ states for this modular realization instead of possibly as many as $n_1 n_2$ states if the above R were built.

  Note that we can interpret the supervision of G by $S_{1\wedge 2}$ as the *product* $R_1 \times R_2 \times G$.

- It is a similar complexity argument that motivates the *synthesis* of a supervisor in modular form. If the admissible language L_a for BSCP is given as (or can be decomposed as) the intersection of two prefix-closed languages

$$L_a = L_{a1} \cap L_{a2}$$

then we would like to synthesize S_i for $L_{ai}^{\uparrow C}$, $i = 1, 2$ and use these two supervisors in conjunction instead of doing the full calculation $L_a^{\uparrow C}$. Using this modular approach, the total computational complexity for supervisor synthesis is reduced from $O(n_1 n_2 m)$ to $O(\max(n_1, n_2)m)$. This modular approach does work because in the case of prefix-closed languages,

$$(K_1 \cap K_2)^{\uparrow C} = K_1^{\uparrow C} \cap K_2^{\uparrow C} \ .$$

This discussion is formalized in the following modular version of BCSP, denoted MSCP.

MSCP: Basic Supervisory Control Problem: Modular Version

Given DES G, $E_{uc} \subseteq E$, and legal language $L_a = L_{a1} \cap L_{a2}$ where $L_{ai} = \overline{L_{ai}} \subseteq \mathcal{L}(G)$ for $i = 1, 2$, build a *modular* supervisor S_{mod} such that:

1. $\mathcal{L}(S_{mod}/G) \subseteq L_a$
2. $\mathcal{L}(S_{mod}/G)$ is optimal w.r.t. set inclusion.

Solution of MSCP:

From the above discussion, we build standard realizations R_i of S_i such that

$$\mathcal{L}(S_i/G) = L_{ai}^{\uparrow C}$$

for $i = 1, 2$ and then take S_{mod} to be the modular supervisor $S_{1\wedge 2}$. Then we get that

$$\mathcal{L}(S_{1\wedge 2}/G) = L_{a1}^{\uparrow C} \cap L_{a2}^{\uparrow C} = (L_{a1} \cap L_{a2})^{\uparrow C} = L_a^{\uparrow C}$$

which is the desired optimal solution.

It is unfortunate that this modular approach cannot be extended to the nonblocking version BSCP, BSCP-NB. The problem is that the conjunction of two nonblocking supervisors need not be a nonblocking supervisor. Consider the following fact:

Fact: Let S_i, $i = 1, 2$, be nonblocking supervisors for G. Then $S_{1\wedge 2}$ is nonblocking iff $\mathcal{L}_m(S_1/G)$ and $\mathcal{L}_m(S_2/G)$ are nonconflicting languages.

Proof:

$$\begin{aligned}
\overline{\mathcal{L}_m(S_{1\wedge 2}/G)} &= \overline{\mathcal{L}_m(S_1/G) \cap L_m(S_2/G)} \\
&= \overline{\mathcal{L}_m(S_1/G)} \cap \overline{\mathcal{L}_m(S_2/G)} \ \text{[by nonconflicting assumption]} \\
&= \mathcal{L}(S_1/G) \cap \mathcal{L}(S_2/G) \ \text{[by nonblocking assumptions]} \\
&= \mathcal{L}(S_{1\wedge 2}/G) \ .
\end{aligned}$$

Q.E.D.

This fact has the following implication. If we consider the modular version of BSCP-NB where

$$L_{am} = L_{am1} \cap L_{am2}$$

and where each $L_{ami} \subseteq \mathcal{L}_m(G)$ and both are $\mathcal{L}_m(G)$-closed (which implies that L_{am} itself is $\mathcal{L}_m(G)$-closed), then by synthesizing S_i such that

$$\mathcal{L}(S_i/G) = \overline{L_{ami}^{\uparrow C}}$$

for $i = 1, 2$ and then by forming the modular supervisor $S_{1\wedge 2}$, we get

$$\begin{aligned} \mathcal{L}(S_{1\wedge 2}/G) &= \overline{L_{am1}^{\uparrow C}} \cap \overline{L_{am2}^{\uparrow C}} \\ \mathcal{L}_m(S_{1\wedge 2}/G) &= \overline{L_{am1}^{\uparrow C}} \cap \overline{L_{am2}^{\uparrow C}} \cap \mathcal{L}_m(G) \\ &= L_{am1}^{\uparrow C} \cap L_{am2}^{\uparrow C} \\ &\supseteq (L_{am1} \cap L_{am2})^{\uparrow C} = L_{am}^{\uparrow C} \end{aligned}$$

which means that the *modular supervisor could be blocking*, even though it is legal in the sense that $\mathcal{L}_m(S_{1\wedge 2}/G) \subseteq L_{am}$.

By the above fact, BSCP-NB has a nonblocking modular solution iff $L_{am1}^{\uparrow C}$ and $L_{am2}^{\uparrow C}$ are nonconflicting. The problem is that this condition cannot be verified *before* doing the $\uparrow C$ calculations. Moreover, to verify this condition, we have to examine together both $L_{am1}^{\uparrow C}$ and $L_{am2}^{\uparrow C}$. In contrast, a monolithic (as opposed to modular) approach would require us to form the intersection $L_{am1} \cap L_{am2}$ and then do the $\uparrow C$ operation on the result; this has roughly the same computational complexity than verifying if the modular supervisor is blocking, and in fact it guarantees a nonblocking (monolithic) supervisor (namely, $\mathcal{L}(S/G) = \overline{L_{am}^{\uparrow C}}$, the MRNBS of BSCP-NB). Yet, if the modular solution is indeed nonblocking, then as was mentioned earlier it is still advantageous from an implementation viewpoint.

The conclusion of this discussion is that the issue of blocking is intrinsically a *global* one; it cannot in general be dealt with in a modular manner.

3.7: DEALING WITH UNOBSERVABILITY

3.7: Dealing with Unobservability

The Property of Observability

Observability and Union

The property of *observability* is more difficult to deal with than the property of controllability in the context of supervisory control because it is *not* preserved under *union*, as shown by the following example.

Example: Let $E = E_c = \{\alpha, \beta\}$ and $E_o = \{\beta\}$ and consider the languages

$$\begin{aligned} M &= \{\epsilon, \alpha, \beta, \alpha\beta\} \\ K_1 &= \{\alpha\} \\ K_2 &= \{\beta\} \ . \end{aligned}$$

Then

1. K_1 and K_2 are observable w.r.t. M, E_o, and E_c.

   To see this, consider P-supervisors that enable only α (for K_1) or β (for K_2) at the outset.

2. $K = K_1 \cup K_2$ is not observable.

   To prove this, take $s = \alpha$, $s' = \epsilon$ and $\sigma = \beta$. Then, $s\sigma \notin \overline{K}$, $s\sigma \in M$, $s'\sigma \in \overline{K}$, and $s'\sigma \in P^{-1}[P(s)]\sigma$ since $P(s) = P(s')$. But this is a violation of the definition of observability.

   Intuitively, a P-supervisor should disable β only after α has occurred, but α is not observable so the language $K_1 \cup K_2$ is not achievable by control.

   Note that taking the prefix-closure of K_1 and K_2 would not help here.

Let us denote the partial observation versions of BSCP and BSCP-NB as BSCOP and BSCOP-NB, respectively, meaning Basic Supervisory Control and Observation Problem. The statements of BSCOP and BSCOP-NB are as follows.

BSCOP: Basic Supervisory Control and Observation Problem
Given DES G, $E_{uc} \subseteq E$, $E_o \subseteq E$ with corresponding projection $P : E^* \rightarrow E_o^*$, and legal language $L_a = \overline{L_a} \subseteq \mathcal{L}(G)$, build P-supervisor S_P such that:

1. $\mathcal{L}(S_P/G) \subseteq L_a$
2. $\mathcal{L}(S_P/G)$ is *as large as possible*, i.e., for any other S'_P such that $\mathcal{L}(S'_P/G) \subseteq L_a$,

$$\mathcal{L}(S'_P/G) \subseteq \mathcal{L}(S_P/G) \ .$$

BSCOP-NB: Nonblocking Version of BSCOP

Given DES G, $E_{uc} \subseteq E$, $E_o \subseteq E$ with corresponding projection $P : E^* \to E_o^*$, and legal marked language $L_{am} \subseteq \mathcal{L}_m(G)$, with L_{am} assumed to be $\mathcal{L}_m(G)$-closed, build *nonblocking* P-supervisor S_P such that:

1. $\mathcal{L}_m(S_P/G) \subseteq L_{am}$
2. $\mathcal{L}_m(S_P/G)$ is *as large as possible*, i.e., for any other nonblocking S'_P such that $\mathcal{L}_m(S'_P/G) \subseteq L_{am}$,

$$\begin{aligned} \mathcal{L}(S'_P/G) &\subseteq \mathcal{L}(S_P/G) \\ (\ \Rightarrow \mathcal{L}_m(S'_P/G) &\subseteq \mathcal{L}_m(S_P/G)\ ). \end{aligned}$$

- In view of the preceding example, the supremal observable sublanguage of a given language need not exist.
- Consequently, the supremal observable and controllable sublanguage of a given language need not exist.
- This implies that BSCOP and BSCOP-NB do *not* possess, *in general*, solutions that satisfy requirement 2 in the formulation of these problems.
- Various approaches have been considered to deal with this difficulty.

1. Calculate sublanguages of L_a or L_{am} (as appropriate) that are *maximal*, observable, and controllable (and also $\mathcal{L}_m(G)$-closed, if necessary).

   By maximal we mean that there is no other observable and controllable sublanguage that is stricly larger than the maximal one; on the other hand, there may be many other incomparable maximals.

   This means that requirement 2 in the statement of BSCOP is replaced by the weaker condition:

   2'. $\mathcal{L}(S_P/G)$ is *maximal*, i.e., for any other S'_P,

   $$\mathcal{L}(S'_P/G) \subseteq L_a \Rightarrow \mathcal{L}(S'_P/G) \not\supset \mathcal{L}(S_P/G)$$

   and similarly for BSCOP-NB.

   An effective approach to compute an observable and controllable sublanguage that is maximal is to assign *priorities* to the events during the computation. By varying the priority assignment, different maximals are obtained. This works well for closed languages (i.e., for BSCOP) and in the context of on-line algorithms, where $S_P(s)$ is calculated after s is observed as opposed to pre-computing the whole function S_P. The computation of maximals for non-prefix-closed languages (i.e., for BSCOP-NB) is more difficult and not as well understood.

2. Identify a property of languages that is *stronger* than observability and that *is* closed under union.

   The property of *normality* discussed later satisfies these requirements.

   In this case however, there is no guarantee (in general) that the solution is maximal so 2' above need not hold; only requirement 1 of BSCOP and BSCOP-NB will hold in general.

3. Identify special situations where the supremal observable and controllable sublanguage of a given language does exist.

   One such situation is when $E_c \subseteq E_o$, i.e., when all the controllable events are observable or, stated differently, when all the unobservable events are uncontrollable.

Observability and Intersection

Observability does possess a useful property: similary to controllability, it is closed under *intersection* in the case of *closed* languages.

Fact: If K_1 and K_2 are both prefix-closed and observable w.r.t. M, E_o, and E_c, then so is $K_1 \cap K_2$.

Proof: Clearly, $K := K_1 \cap K_2$ is prefix-closed.
By contradiction, suppose that K is not observable. Then there exists s and s', with $P(s) = P(s')$, and $\sigma \in E_c$ such that

$$(s'\sigma \in \overline{K}) \text{ and } (s \in \overline{K}) \text{ and } (s\sigma \in M \setminus \overline{K}) \ .$$

But

$$s\sigma \in M \setminus \overline{K} \Leftrightarrow (s\sigma \in M) \text{ and } (s\sigma \notin \overline{K_1} \text{ or } s\sigma \notin \overline{K_2}) \ .$$

If $s\sigma \notin \overline{K_1}$, then we have that

$$(s'\sigma \in \overline{K_1}) \text{ and } (s \in \overline{K_1}) \text{ and } (s\sigma \in M \setminus \overline{K_1})$$

which contradicts the assumption that K_1 is observable.
If $s\sigma \notin \overline{K_2}$, then a similar argument contradicts the observability of K_2.
Thus no such s, s', and σ exist and we conclude that K is observable.
Q.E.D.

- Observe where the prefix-closure assumption for K_1 and K_2 is used in this proof.
- This fact holds for arbitrary intersections as well.

  This suggests to proceed as we did for the class $\mathcal{CC}_{out}(K)$ in our study of controllability.

Definition: We define the class of languages

$$\mathcal{CO}_{out}(K) := \{L \subseteq E^* : (K \subseteq L \subseteq M) \wedge (\overline{L} = L) \wedge (L \text{ is observable})\} \ .$$

Here, observability is w.r.t. fixed M, E_o (equivalently, the corresponding projection P), and E_c.

About the Observable Superlanguages of K:

- The class $\mathcal{CO}_{out}(K)$ is a poset that is closed under arbitrary intersections.
- Thus $\mathcal{CO}_{out}(K)$ possesses a unique *infimal* element. Namely,

$$K^{\downarrow O} := \bigcap_{L \in \mathcal{CO}_{out}(K)} L$$

  is a well-defined element of $\mathcal{CO}_{out}(K)$.
- We call $K^{\downarrow O}$ the *infimal closed and observable superlanguage of* K.
  - In the worst case, $K^{\downarrow O} = M$.
  - If K is observable, then $K^{\downarrow O} = \overline{K}$.
- We will refer to $\downarrow O$ as the operation of obtaining the infimal closed and observable superlanguage.
  - It is immediate from the definition of $\downarrow O$ that this operation is *monotone*, i.e.,

$$K_1 \subseteq K_2 \Rightarrow K_1^{\downarrow O} \subseteq K_2^{\downarrow O} \ .$$

  - A key result is that the $\downarrow O$ operation preserves regularity.

Observability, Controllability, and Intersection

We can combine the results about $\downarrow O$ with those about $\downarrow C$ for controllability and conclude that the *infimal prefix-closed observable and controllable superlanguage* of a given language does exist.

To see this, consider the class of languages

$$\mathcal{CCO}_{out}(K) := \mathcal{CC}_{out}(K) \cap \mathcal{CO}_{out}(K) \ .$$

- $\mathcal{CCO}_{out}(K)$ contains the superlanguages of K that are prefix-closed, controllable, and observable.
- $\mathcal{CCO}_{out}(K)$ is closed under arbitrary intersections, and thus its infimal element exists and is the infimal prefix-closed observable and controllable superlanguage of K.
  - Let us denote this infimal language by $K^{\downarrow(CO)}$.
  - In the worst case, $K^{\downarrow(CO)} = M$.
- The existence of $K^{\downarrow(CO)}$ means that the partial observation version of DuSCP does have a solution.

DuSCOP: "Dual" Version of BSCOP

Given DES G, $E_{uc} \subseteq E$, $E_o \subseteq E$ with corresponding projection $P : E^* \to E_o^*$, and *minimum* required language $L_{min} \subseteq \mathcal{L}(G)$, build P-supervisor S_P such that:

1. $\mathcal{L}(S_P/G) \supseteq L_{min}$
2. $\mathcal{L}(S_P/G)$ is *as small as possible*, i.e., for any other S'_P such that $\mathcal{L}(S'_P/G) \supseteq L_{min}$,

$$\mathcal{L}(S'_P/G) \supseteq \mathcal{L}(S_P/G) \ .$$

(Note that L_{min} need not be prefix-closed and could be given as a subset of $\mathcal{L}_m(G)$.)

Solution of DuSCOP:

- The solution is to take S_P such that

$$\mathcal{L}(S_P/G) = L_{min}^{\downarrow(CO)}.$$

  (Here, the $\downarrow(CO)$ operation is w.r.t. $M = \mathcal{L}(G)$.)

- Observe that if L_{min} was given as a subset of $\mathcal{L}_m(G)$, we would get

$$\mathcal{L}_m(S_P/G) = L_{min}^{\downarrow(CO)} \cap \mathcal{L}_m(G) \supseteq \overline{L_{min}} \cap \mathcal{L}_m(G) \supseteq L_{min} \ .$$

  As for DuSCP, there would be no guarantee that S_P be nonblocking.

Some Comments on $\downarrow O$ and $\downarrow (CO)$

About $\downarrow O$: • $K^{\downarrow O}$ can be expressed in terms of a formula on languages involving set operations, concatenation, projection, and inverse projection.

The formula is:

$$K^{\downarrow O} = M \setminus [E^+ \setminus \bigcup_{\sigma \in E_c} [P^{-1}(P(\overline{K}\sigma \cap \overline{K})) \cap E^*\sigma]]E^* \ .$$

- This formula proves that if K and M are regular, then $K^{\downarrow O}$ is regular.
- This formula also suggests an algorithm for computing this language.
- For further details on the infimal closed and observable superlanguage, see the paper by Rudie & Wonham in *Systems & Control Letters*, 1990, pp. 361-371.

About $\downarrow (CO)$: • $K^{\downarrow (CO)}$ can also be expressed in terms of a (slightly different) formula on languages.

The formula is:

$$K^{\downarrow (CO)} = M \setminus [E^*E_c \setminus \bigcup_{\sigma \in E_c} [P^{-1}(P(\overline{K}\sigma \cap \overline{K})) \cap E^*\sigma]]E^* \ .$$

- Again, this formula proves that if K and M are regular, then $K^{\downarrow (CO)}$ is regular. Moreover, the formula suggests an algorithm for computing this language.

The Property of Normality

Defnition: Consider $M = \overline{M} \subseteq E^*$ and projection $P : E^* \rightarrow E_o^*$. A language $K \subseteq M$ is said to be normal w.r.t. P and M if

$$\overline{K} = P^{-1}[P(\overline{K})] \cap M \ .$$

In other words, $\overline{K}$ can be exactly recovered from its projection $P(\overline{K})$ and M.

- Observe that $\emptyset$ and M are both normal, so the property is not vacuous.
- As was done for controllability and observability, the property of normality is defined on the *prefix-closure* of a language.

  Thus K is normal iff $\overline{K}$ is normal.
- Observe that the inequality

  $$\overline{K} \subseteq P^{-1}[P(\overline{K})] \cap M$$

  is always true.

  If furthermore

  $$P^{-1}[P(\overline{K})] \cap M \subseteq \overline{K}$$

  holds, then K is normal (w.r.t. P and M).

The two reasons to study normality are:

Fact 1: If $K \subseteq M$ is normal w.r.t. P and M, then K is observable w.r.t. M, P, and E_c for all $E_c \subseteq E$. However, the converse statement is not true in general.

Fact 2: If K_1 and $K_2 \subseteq M$ are normal w.r.t. P and M, then so is $K_1 \cup K_2$.

Fact 2 holds for arbitrary unions.

Proof of Fact 1: It suffices to prove the result for $E_c = E$.
By contradiction, suppose that K is normal but not observable.
Then, there exists s, σ, and s', such that $s \in \overline{K}$, $\sigma \in E_c = E$, $s\sigma \notin \overline{K}$, $s\sigma \in M$, and $s'\sigma \in \overline{K}$, with $P(s) = P(s')$.
Clearly, $P(s\sigma) = P(s'\sigma) \in P(\overline{K})$. But then $P(s\sigma) \in P(\overline{K})$ and $s\sigma \in M$ imply that

$$s\sigma \in P^{-1}[P(\overline{K})] \cap M \ .$$

Thus, by normality, $s\sigma \in \overline{K}$, a contradiction.

To show that normality is *stronger* than observability, consider the following example. Take $M = \overline{\beta\alpha}$, $K = \{\beta\}$, and $E_o = \{\beta\}$.
Then K is observable for any $E_c \subseteq E$ (verify!) but not normal since

$$P^{-1}[P(\overline{K})] \cap M = M \supset \overline{K} \ .$$

Q.E.D.

Proof of Fact 2:

$$\begin{aligned}
P^{-1}[P(\overline{K_1 \cup K_2})] \cap M &= P^{-1}[P(\overline{K_1} \cup \overline{K_2})] \cap M \\
&= P^{-1}[P(\overline{K_1}) \cup P(\overline{K_2})] \cap M \\
&= (P^{-1}[P(\overline{K_1})] \cup P^{-1}[P(\overline{K_2})]) \cap M \\
&= (P^{-1}[P(\overline{K_1})] \cap M) \cup (P^{-1}[P(\overline{K_2})] \cap M) \\
&= \overline{K_1} \cup \overline{K_2} \\
&= \overline{K_1 \cup K_2}\ .
\end{aligned}$$

Q.E.D.

In view of Fact 2, the class of languages (for $K \subseteq M$)

$$\mathcal{N}_{in}(K) := \{L \subseteq K : \overline{L} = P^{-1}[P(\overline{L})] \cap M\}$$

possesses a unique *supremal* element, the *supremal normal sublanguage of* K:

$$K^{\uparrow N} := \bigcup_{L \in \mathcal{N}_{in}(K)} L \ .$$

- In the worst case, $K^{\uparrow N} = \emptyset$.
- We make two remarks regarding the computation of $K^{\uparrow N}$.
  1. In the case where K is prefix-closed, it can be shown that $K^{\uparrow N}$ is given by the formula

     $$K^{\uparrow N} = K \setminus (P^{-1}[P(M \setminus K)])E^* \ .$$

     [For a proof of this result, see the paper by Brandt *et al.* in the *Systems & Control Letters*, 1990, pp. 111-117.]

     This formula shows that if K and M are regular, then so is $K^{\uparrow N}$.

     The computational complexity however is exponential in the worst case as P results in a nondeterministic automaton (it is assumed here that we wish to have a *deterministic* automaton that generates $K^{\uparrow N}$).

2. In the general case, there exists an iterative procedure for the computation of $K^{\uparrow N}$. Namely,

   2.1. Set $K_0 = K$

   2.2. $K_{i+1} = (\overline{K_i})^{\uparrow N} \cap K$

   This procedure is finitely convergent when K and M are regular, thus showing that $K^{\uparrow N}$ is also regular. The worst-case computational complexity is again exponential.

The following results are analogous to results that we proved earlier about controllability. Their proofs are essentially identical to the proofs for controllability.

Fact:

1. If K is prefix-closed, then so is $K^{\uparrow N}$.
2. If $K \subseteq \mathcal{L}_m(G)$ is $\mathcal{L}_m(G)$-closed, then so is $K^{\uparrow N}$ (here, $M = \mathcal{L}(G)$).
3. If K is controllable, then $K^{\uparrow N}$ need not be controllable.

Proof:

1. Since $K^{\uparrow N}$ is normal, then so is its prefix-closure. But $\overline{K^{\uparrow N}} \subseteq \overline{K} = K$, which implies that $\overline{K^{\uparrow N}} \subseteq K^{\uparrow N}$. This suffices to prove the result.
2. We need to show that $K^{\uparrow N} = \overline{K^{\uparrow N}} \cap \mathcal{L}_m(G)$.

   2.1. ($\subseteq$) is immediate.

2.2. For $(\supseteq)$, define $K' = \overline{K^{\uparrow N}} \cap \mathcal{L}_m(G)$. Then

$$\begin{aligned} K^{\uparrow N} &\subseteq \overline{K^{\uparrow N}} \cap \mathcal{L}_m(G) \\ \Rightarrow \overline{K^{\uparrow N}} &\subseteq \overline{\overline{K^{\uparrow N}} \cap \mathcal{L}_m(G)} = \overline{K'} \ . \end{aligned}$$

Also

$$\begin{aligned} & \overline{K^{\uparrow N}} \cap \mathcal{L}_m(G) \subseteq \overline{K^{\uparrow N}} \\ \Rightarrow \quad & \overline{\overline{K^{\uparrow N}} \cap \mathcal{L}_m(G)} \subseteq \overline{K^{\uparrow N}} \\ \Rightarrow \quad & \overline{K'} \subseteq \overline{K^{\uparrow N}} \ . \end{aligned}$$

Therefore, $\overline{K'} = \overline{K^{\uparrow N}}$.

Since $\overline{K^{\uparrow N}}$ in normal, then so is K', by definition of normality.

But $K' \subseteq \overline{K} \cap \mathcal{L}_m(G) = K$ (by hypothesis).

It follows that $K' \subseteq K^{\uparrow N}$, i.e.,

$$\overline{K^{\uparrow N}} \cap \mathcal{L}_m(G) \subseteq K^{\uparrow N} \ .$$

3. Left as an exercise.

Q.E.D.

We can combine the preceding results about normality and the corresponding ones about controllability and conclude that the *supremal controllable and normal sublanguage* of a given language exists and is well-defined.

- We will denote this language by $K^{\uparrow(CN)}$.
- Clearly, in the worst case, $K^{\uparrow(CN)} = \emptyset$.
- It can be shown that when K and M are regular, then so is $K^{\uparrow(CN)}$.
- Moreover, it is easily verified that if K is $\mathcal{L}_m(G)$-closed, then so is $K^{\uparrow(CN)}$ (proceed in the same way as in the preceding fact).
- In the prefix-closed case, there exists a formula for the computation of $\uparrow (CN)$.
- In the general case, the computation of $\uparrow (CN)$ can be done iteratively as for $\uparrow N$:
  1. Set $K_0 = K$
  2. $K_{i+1} = (\overline{K_i})^{\uparrow CN} \cap K$

Thus $K^{\uparrow(CN)}$ provides a *sub-optimal* solution to BSCOP and BSCOP-NB in the sense that it meets requirement 1 of these problems but not requirement 2.

It should be emphasized though that this solution may not "particularly interesting" in the sense that it need not be maximal, i.e., there may be controllable and observable sublanguages that are stricly larger than the supremal controllable and normal sublanguage.

There is one very interesting feature about the supremal controllable and normal sublanguage and about the relation between normality, observability, and controllability. Consider the following result.

Fact: Let $E_c \subseteq E_o$. If K is controllable (w.r.t. M and E_{uc}) and observable (w.r.t. M, P, and E_c), then K is normal (w.r.t. P and M).

Proof: The result is trivial if $K = \emptyset$, so assume that K is not empty. By contradiction, assume that K is not normal. Then there exists $t \in M$ such that

$$(t \notin \overline{K}) \wedge (P(t) \in P(\overline{K})) \ .$$

But $t \neq \epsilon$ (since $K \neq \emptyset$). Let $s\sigma$ be the *shortest* such t. Then s does not violate normality, but $s\sigma$ does, therefore

$$(s \in \overline{K}) \wedge (s\sigma \in M) \wedge (s\sigma \notin \overline{K}) \wedge (P(s\sigma) \in P(\overline{K})) \ .$$

Since K is controllable, this implies that $\sigma \in E_c \subseteq E_o$. Now,

$$(\sigma \in E_o) \wedge (P(s\sigma) \in P(\overline{K})) \Rightarrow (\exists s'\sigma \in \overline{K}) P(s') = P(s) \ .$$

For this $s'\sigma$, we have that

$$s'\sigma \in P^{-1}[P(s)]\sigma \cap \overline{K})$$

which implies that K is not observable, a contradiction. Thus K is normal. **Q.E.D.**

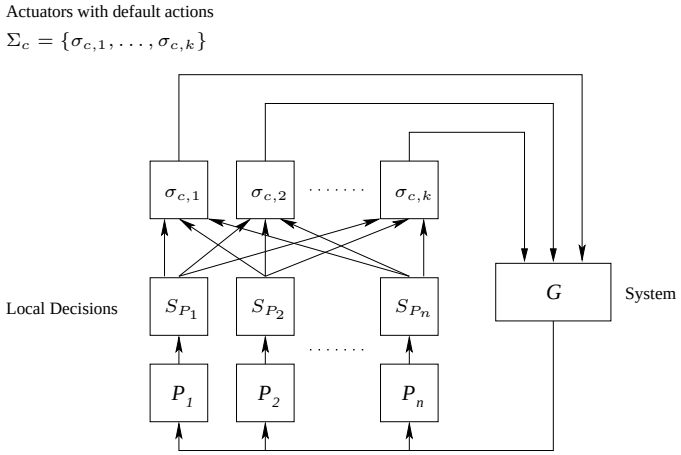
- The preceding fact implies that under the assumption that $E_c \subseteq E_o$, *the supremal observable and controllable sublanguage does exist and consequently BSCOP and BSCOP-NB do have (optimal) solutions.*
- These solutions can be obtained by calculating $L_a^{\uparrow(CN)}$ (or $L_{am}^{\uparrow(CN)}$, as appropriate), or by calculating any maximal controllable and observable sublanguage of these languages (since there can only be one maximal, namely the supremal!).

3.8: DECENTRALIZED CONTROL

3.8: Decentralized Control

The Conjunctive-Permissive Decentralized Architecture

Architecture for decentralized control with a set of partial-observation supervisors and allowing for common controllable events:



- It is convenient to extend the domain of partial-observation supervisor i from $P_i[\mathcal{L}(G)]$ to $\mathcal{L}(G)$.

  Let us denote the extended-domain supervisor by S_i:

$$S_i(s) = S_{P_i}[P_i(s)] \ .$$

- Associated with G are the four usual sets E_c, E_{uc}, E_o, and E_{uo}.
- Corresponding to supervisor S_i at site i, $i = 1, \ldots, n$, we have:
  - the set of controllable events $E_{i,c} \subseteq E_c$, where $\cup_{i=1}^{n} E_{i,c} = E_c$;
  - the set of observable events $E_{i,o} \subseteq E_o$, where $\cup_{i=1}^{n} E_{i,o} = E_o$;
  - the sets $E_{i,c}$ [resp., $E_{i,o}$], $i = 1, \ldots, n$, are not necessarily disjoint;
  - the (natural) projection $P_i : E^* \rightarrow E_{i,o}^*$ corresponding to $E_{i,o}$.
- Role of S_i:
  - make enable/disable decisions on the events in $E_{i,c}$, based on the system model G and on the locally observable events $E_{i,o}$;
  - necessarily enable events in $E \setminus E_{i,c}$ – equivalently, we can assume that S_i does nothing for these events.

- We define the combined control policy $S_{dec}^{con} : \mathcal{L}(G) \to 2^E$ acting on G as follows:

$$S_{dec}^{con}(s) = \bigcap_{i=1}^{n} S_i(s) \ .$$

  This means the fusion rule at each actuator is "AND", i.e., a *common* controllable event is enabled iff all supervisors that have control over it enable the event.

  In other words, if one supervisor disables an event, then that event is necessarily disabled.

- The resulting controlled behavior is described by the languages $\mathcal{L}(S_{dec}^{con}/G)$ and $\mathcal{L}_m(S_{dec}^{con}/G)$.

- One issue hidden in the definition of S_{dec}^{con} is the following:

  If supervisor S_i is not sure about enabling or disabling an event after observing trace $P_i(s)$, then what should it do?

  This will happen if there is a control conflict in the state estimate built from $P^{-1}[P(s)]$.

  Let us assume that in such cases, all S_i's use the following default: ENABLE. In other words, S_i is *permissive* when in doubt.

  This can be paraphrased as "S_i passes the buck" regarding the disablement of the event.

- In view of the above observations, we refer to this decentralized control architecture as the *Conjunctive-Permissive architecture* or CP-architecture.

CP-Coobservability

- We ask the following question for the CP-architecture:

  What is the necessary and sufficient condition on K, beyond controllability, that will ensure the existence of S_i, $i = 1, \ldots, n$, such that $\mathcal{L}(S_{dec}^{con}/G) = K$?

- Intuitively, we know that the required condition will have to be *stronger than*

  K is observable with respect to $\mathcal{L}(G)$, E_o, and E_c

  since a decentralized architecture cannot in general be as powerful as a centralized one.

- However, the required condition should be *weaker than*

  K is observable with respect to $\mathcal{L}(G)$, P_i, and $E_{i,c}$, for each $i = 1, \ldots, n$

  since there may be events that can be controlled by more than one supervisors, and therefore we may not need full "local" observability at all sites as the supervisors may somehow be able to "share the work" on the *common* controllable events.

- Example 1: "Passing the buck" works in the CP-architecture:

$$\mathcal{L}(G) = \overline{\{bg, bbg, ag, abg\}} \qquad E_{1,o} = \{a\} \qquad E_{1,c} = \{a, g\}$$
$$K = \overline{\{bg, bb, ab\}} \qquad E_{2,o} = \{b\} \qquad E_{2,c} = \{b, g\} \ .$$

  K is not observable with respect to $\mathcal{L}(G)$, $E_{i,o}$, and $E_{i,c}$, $i = 1, 2$.

  But:

  1. S_1 can start by enabling events a, b (uncontrollable to S_1), and g, and then enable only b after it sees string a; in other words, S_1 disables event g only after it sees string a;
  2. S_2 can start by enabling events a (uncontrollable to S_2) and b, then enable a, b, and g after it sees string b, and finally enable event a only after it sees string bb (i.e., g is disabled at the beginning and after bb).

- Example 2: "Passing the buck" does not work in the CP-architecture:

$$\mathcal{L}(G) = \overline{\{aag, abg, bag, bbg\}} \qquad E_{1,o} = \{a\} \qquad E_{1,c} = \{a, g\}$$
$$K = \overline{\{aa, bb, ba, abg\}} \qquad E_{2,o} = \{b\} \qquad E_{2,c} = \{b, g\} \ .$$

- **CP-Coobservability:**
  Let K and $M = \overline{M}$ be languages over event set E. Let $E_{i,c}$ and $E_{i,o}$ be sets of controllable and observable events, respectively, for $i = 1, \ldots, n$. Let P_i be the natural projection corresponding to $E_{i,o}$, with $P_i : E^* \rightarrow E_{i,o}^*$.
  K is said to be *CP-coobservable* with respect to M, P_i, and $E_{i,c}$, $i = 1, \ldots, n$, if for all $s \in \overline{K}$ and for all $\sigma \in E_c = \cup_{i=1}^{n} E_{i,c}$,

$$(s\sigma \notin \overline{K}) \text{ and } (s\sigma \in M) \Rightarrow$$
$$\exists i \in \{1, \ldots, n\} \text{ such that } P_i^{-1}[P_i(s)]\sigma \cap \overline{K} = \emptyset \text{ and } \sigma \in E_{i,c} \; .$$

- We can paraphrase CP-coobservability as follows:

  If event σ needs to be disabled, then at least one of the supervisors that can control σ must unambiguously know that it must disable σ, that is, from this supervisor's viewpoint, disabling σ does not prevent any string in $\overline{K}$. Consequently, when in doubt, a supervisor can be permissive and "pass the buck".

- It should by now be clear that the notion of CP-coobservability is tied with the choices made in the decentralized architecture regarding the fusion rule (conjunctive) and the default action of supervisors (permissive).

- If we set $E_{i,o} = E_o$, $E_{i,c} = E_c$, and $E_{j,o} = E_{j,c} = \emptyset$ for $j = 1, \ldots, n$, $j \neq i$, then CP-coobservability reduces to observability, as expected.

- If $E_{i,c} \cap E_{j,c} = \emptyset$ for all $i, j = 1, \ldots, n$, then since each controllable event can only be controlled by one supervisor, the notion of "passing the buck" does not apply. Consequently, each supervisor must unambiguously know when to disable all its controllable events. This means that in this case CP-coobservability of language K is equivalent to

  "K is observable with respect to $\mathcal{L}(G)$, P_i, and $E_{i,c}$, for each $i = 1, \ldots, n$."

The Controllability and Coobservability Theorem

Controllability and Coobservability Theorem (CCoT):
Consider DES $G = (X, E, f, \Gamma, x_0, X_m)$, where $E_{uc} \subseteq E$ is the set of uncontrollable events, $E_c = E \setminus E_{uc}$ is the set of controllable events, and $E_o \subseteq E$ is the set of observable events. For each site i, $i = 1, \ldots, n$, consider the set of controllable events $E_{i,c}$ and the set of observable events $E_{i,o}$; overall, $\cup_{i=1}^{n} E_{i,c} = E_c$ and $\cup_{i=1}^{n} E_{i,o} = E_o$. Let P_i be the natural projection from E^* to $E_{i,o}^*$, $i = 1, \ldots, n$. Consider also the language $K \subseteq \mathcal{L}_m(G)$, where $K \neq \emptyset$. There exists a nonblocking decentralized supervisor S_{dec}^{con} for G for the CP decentralized architecture such that

$$\mathcal{L}_m(S_{dec}^{con}/G) = K \quad \text{and} \quad \mathcal{L}(S_{dec}^{con}/G) = \overline{K}$$

if and only if the three following conditions hold:

1. K is controllable with respect to $\mathcal{L}(G)$ and E_{uc};
2. K is CP-coobservable with respect to $\mathcal{L}(G)$, P_i, and $E_{i,c}$, $i = 1, \ldots, n$;
3. K is $\mathcal{L}_m(G)$-closed.

About the Proof of CCoT:

- The control actions to use are:

$$S_i(s) = S_{P_i}(s_i) = (E_{i,uc} \cup \{\sigma \in E_{i,c} : \exists s'\sigma \in \overline{K}\ (P_i(s') = s_i)\}$$

  where $E_{i,uc} = E \setminus E_{i,c}$

About Coobservability:

- It can be tested in polynomial time

Dealing with Violations of CP-Coobservability

All the difficulties inherent to partial observation in the centralized case (and more) carry to decentralized control architectures. In particular,

- CP- coobservability is not preserved under union.
- However, CP-coobservability is preserved under intersection for prefix-closed languages. So the operations $\downarrow (CP - Co)$ and $\downarrow (CCP - Co)$ are well-defined.
- There is no "useful" generalization of normality in the decentralized case.

Recent research in this area has focused on generalizations of the CP-architecture that allow to achieve larger classes of languages than those that are CP-coobservable.

1. Change the two defaults in CP to get the *DA-architecture*: *Disjunctive* fusion rule, namely OR of enabled events, together with *Antipermissive* default when a supervisor is in doubt, namely, the default action is disable.

   This leads to the notion of *DA-coobservability*.

   DA-coobservability and CP-coobservabililty are *incomparable*.

2. Combine the CP- and DA-architectures into a single one assuming the set of controllable events is partitioned into those the are:

   - *enabled by default* and fused according to AND;
   - *disabled by default* and fused according to OR.

   This new architecture performs "better" than either the CP- or DA-architecture.