Supervisory Control

If we are not satisfied with the behavior of the uncontrolled system, we need supervisory control.

Supervisory control = restriction of the behavior of the plant

\[
S(s) = \text{set of enabled events, the control action, } S' = \text{supervisor.}
\]

\[
S' \text{ typically an automaton.}
\]
Controllable and uncontrollable events

Events are either controllable or uncontrollable.

A controllable event can be stopped or started at will.

An uncontrollable event cannot be stopped or started at will.

The controller may only restrict controllable events.

The controller says what is not desired.

Uncontrollable events are enabled if the plant enables them.

There can be physical reasons for that uncontrollable events cannot take place.
**Controllable and uncontrollable events**

**Example 20.** A machine. The event start is controllable, the event done is not. We can neither say to the machine that ”get ready now”, nor can we stop it from finishing the job when it is done. But done cannot happen before we start the machine. So the idea is to disable start until we can handle done.

![State Diagram](attachment:state_diagram.png)

**Example 21.** Stick picking game. Picks done by us are controllable, the picks done by the opponent are not.

**Example 22.** Man, wolf, goat and cabbage head. The actions involving the man are controllable, the others (the eating) are not.
Supervisory control

The feedback loop is denoted $S/G$, "S controlling G"

$S/G$ is a discrete event system

The language generated by $S/G$ is defined recursively

1. $\varepsilon \in \mathcal{L}(S/G)$

2. $[s \in \mathcal{L}(S/G) \text{ and } (s\sigma \in \mathcal{L}(G)) \text{ and } (\sigma \in S(s))] \iff [s\sigma \in \mathcal{L}(S/G)]$

$\mathcal{L}(S/G)$ are the sequences of events that are allowed by both the supervisor and the plant.

Marking can be introduced in both the plant and the controller, $\mathcal{L}_m(S/G)$ are the strings that takes us to a marked state in $S/G$.

One usually analyze $S/G$ by analyzing $S||G$, $G$ then being an automaton model of the plant, which might not be the same as the true plant.
Blocking

The DES $S/G$ is blocking if

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

and otherwise unblocking.

**Example 23.** Assume that $\mathcal{L}(G) = \{abc\}$ and $\mathcal{L}_m(G) = \{ab\}$. This system is blocking, since $abc \notin \overline{\mathcal{L}_m(G)}$. If put the supervisor $S$, defined as $S(\varepsilon) = \{a\}$, $S(a) = \{b\}$ and $S(ab) = \emptyset$, in feedback with $G$, the controlled system $S/G$ is nonblocking since

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

If event $c$ would be uncontrollable we could not make the system unblocking by control. If we keep the same $S(\varepsilon)$ and $S(a)$, then $S(ab) = \{c\}$ because we cannot prevent uncontrollable events. And $S/G$ would be the same as $G$, and thus blocking. If we set $S(a) = \emptyset$ the controlled system will block after event $a$, and if we set $S(\varepsilon) = \emptyset$ the controlled systems will block at initial state.
Specifications on controlled system

Specifications come in many forms

Might be $\mathcal{L}(S/G)$ or $\mathcal{L}_m(S/G)$, or corresponding automaton.

But it is typically text-based.

Which need to be transformed into something related to the model of the process.

What is relevant to model depends on what we want to achieve, that is the specification.

Important to note that the process itself sets boundaries for what is possible.
Examples of specification

• Marked states.

• Forbidden states.

• First-come first-served.

• Alternating events.

• Specifying order of events (recipe).

• Forbidden sequences of events.

• Allow event $a$ at most $n$ times between two $b$ events.

Specifications can also be expressed with automata.
Example 24. Below a plant $G$ and two specifications $Sp_1$ and $Sp_2$.

$Sp_1$ forbids the string $ab$ and marks the string $ba$. No further $a$ or $b$ events are allowed. $Sp_2$ marks the events $a$ and $b$, and allows no further $a$ or $b$ events.
Partial and total specifications

Total specification = all events are observed

Partial specification = only a subset of the events are observed

Total specification means $S = Sp \times G$ and thus $\mathcal{L}(S) = \mathcal{L}(Sp) \cap \mathcal{L}(G)$, nothing outside the specification is allowed.

The borderline between a specification and a supervisor is vague, as they might in fact be the same. A supervisor is ideally a verified and possibly corrected specification.

Partial: The unobserved events are not restricted, the plant decides how to behave when such an event occurs

$$S = Sp \parallel G$$

Not always necessary or even possible to construct such a supervisor that observes all events.

Note that all states in $G$ are assumed marked in the parallel compositions above. Marking is a specification issue, this way we leave it open for $Sp$ to decide.

$Sp_1$ and $Sp_2$ shown earlier are total specifications.
Example 25. Let us now consider the same plant $G$ but a partial specifications $Sp_3$.

$Sp_3$ allows only a single occurrence of $a$, and marks all states after that, and does not restrict $b$. The total supervisor is obtained by a parallel composition $G \parallel Sp_3$. The execution of $b:s$ is restricted by the plant when execution of $a$ is restricted by the specification.

Note that $G \parallel Sp_3$ is incorrectly done in the Supremica-book, the authors haven’t checked it with Supremica!
Static and dynamic specifications

Static: Can be expressed with marked and forbidden states

Dynamic: Cannot be expressed with marked and forbidden states, needs an automaton.

A static controller is a sub-automaton of the plant-automaton, which might help the implementation.

Example 26. The specifications $Sp_1$, $Sp_2$ and $Sp_3$ given earlier are all dynamic specifications. None of them can be expressed with marking and/or forbidding states in $G$.

$Sp_1$ marks $ba$ and forbids $ab$, but both strings takes us to the same state $p0$. $Sp_2$ marks $a$ and $b$, and marking $p2$ would mark $a$ but also all other strings reaching $p2$, $aba$, $baa$, $ababa$, ...

In $G||Sp_3$ you can see that you would have to mark all states, but actually not before event $a$...? So $Sp_3$ is dynamic.
Example 27. Let us now consider the automata $G_1$ and $G_2$ a specification $Sp$ given below

$G_1$

$G_2$

$Sp$

$G_1$ and $G_2$ can be viewed as two different models of the same plant. $G_2$ is more detailed than $G_1$, you keep track in which direction you are heading in $G_2$, and thus $q_1$ is split up into two different states.

Sp is dynamic in relation to $G_1$ but static in relation to $G_2$. It specifies that you have to keep track of direction, separate between $a$ and $abc$, which is already done in $G_2$. So the only new thing for $G_2$ is the marking of the initial state.

Note that both $G_1 || Sp = Sp$ and $G_2 || Sp = Sp$, same supervisor in both cases.
Examples of specification: Forbidden states

If we want to forbid a certain state in an automaton $G$, we simply remove the state and transitions associated with it, and take the Ac operation to remove non-accessible states and Trim if also want it to be nonblocking.

Examples of specification: State splitting

If a specification requires remembering how a certain state state is reached in order to determine what future behavior is admissible, then the state must be split up in as many states as necessary. The transitions are also modified accordingly. We had an example of state splitting already in Example 27, but here is another one.
Example 28. Consider two transactions in a database system

\[ T_1 = a_1 b_1 \quad \text{and} \quad T_2 = a_2 b_2 \]

where \( a \) and \( b \) are records (e.g., bank accounts) and 1 and 2 are transaction index. The automaton \( T_1 \parallel T_2 \) given below models concurrent execution of \( T_1 \) (e.g., balance of \( a \) and \( b \)) and \( T_2 \) (e.g., move money from \( a \) to \( b \)). In databases one require that

\[ a_1 \text{ precedes } a_2 \text{ if and only if } b_1 \text{ precedes } b_2. \]

So we need to know at state \( q_1.q_1 \) if we did \( a_1 \) or \( a_2 \) first. So we have to split up \( q_1.q_1 \) into two states \( q_1.q_1_1 \) and \( q_1.q_1_2 \). An automaton \( S_p \) that has the desired behavior is also given below.
Examples of specification: Event alternance

If we want that two events $a$ and $b$ to alternate, and $a$ to occur first, we can build a two state automaton $S_{P_{alt}}$ to capture this.

The desired supervisor is obtained by parallel composition with the plant automaton. The other events of the plant are not affected since they do not appear in $S_{P_{alt}}$. We mark both states in $S_{P_{alt}}$ and specify marking in the plant (or another specification automaton) this time.
Examples of specification: Illegal substring

This is very similar to the PIN-code example, where we had one substring that lead to opening of the door. Now we explicitly want to forbid one substring, and allow all other strings. This is best illustrated with an example:

**Example 29.** Consider the event set $\Sigma = \{a, b, c\}$ and the illegal substring $s_f = abc$. An automaton that forbids $s_f$ is depicted below:

![Automaton diagram]

All events are enabled at all states except at $q_2$, where we have registered $ab$ and cannot allow $c$.
All states are marked, the marking is again best specified elsewhere.
Examples of specification: Recipe

This is analogous to event alternance, but it may involve more events. We might want to specify that the sequence of events \( abc \) should occur indefinitely. For example, \( a = \) fill container with raw material, \( b = \) mix, \( c = \) drain container. This can be modeled with the automaton below:

Now it is natural to specify marking here, and mark all states in the plant automaton before the parallel composition. It is now also possible that there are other events in the plant. If they are not included in the specification, they are only limited by the plant.
Example 30. Consider the following tank process:

![Tank Diagram]

Specification: We want fill the tank with water using valve $F$, heat it using $E$, and drain it using valve $D$. It is ok to turn off heating after the target temperature has been reached, and you do not have to care about the temperature after that, before you start filling again that is. You can sense the water level with sensors $H$ and $L$, and you should not turn on the heat before sensor $L$ is covered with water.
Events: F1 = opening of F, F0 = closing of F; D1 and D0 same for drain; H1 = level has risen to H, H0 = level fell below H; L1 and L0 same for L; E1 = turns E on, E0 = turns E off; T1 = temperature rose to desired level, T0 = temperature fell below desired level.

Temperature model

Heater model

Note that we can model that temperature cannot rise unless heat is on (T1 on state "on") or fall unless heater is off (T0 on state "off").

Level model

Fill model

Drain model

Similarly level cannot rise unless we fill, or fall unless we drain.
Specification 1 for heating. No heating before level has reached a certain level, and no draining before heating is done.

Specification 2 for filling and draining. First fill, then drain.

Specification 3 for that we do not want to turn on heat before it is too cold, we need event alternance:

Might not be completely clear why we need such a specification at this point. Later it will be tested whether it is necessary or not.
Supervisor candidate for tank example,
Temp || Heater || Level || Fill || Drain || Spec1 || Spec2 || Spec3

Note that the supervisor gets two marked states and gets much larger for one reason: We do not demand that the temperature must fall below desired level before we start filling again.

Marking was specified in the specifications, all the states in all plant automata are marked before parallel composition.

Note also that this is a supervisor candidate, we are not yet sure that it always works; we have not checked if we have blocking, and we have not checked if it is controllable (the latter to be defined).
Alternative solution to Example 30

The assumption made in the heater model that the temperature cannot fall when heating does not necessarily hold, for example when filling the tank:

![Diagram of heater model]

The temperature specification need to be revised, as T0 can take place also when heating:

![Diagram of temperature specification]

We cannot only add a T1 selfloop at q2, as we need to disable E0 when it becomes too cold. This is the reason we need T0 in the above specification. Note that we have also included the event alternance between T0 and E1, so we no longer need that specification.
The resulting supervisor actually becomes smaller, only 25 states: