Controllability

We have already mentioned that events can be controllable or uncontrollable.

Now we are going to define controllability for supervisors and states.

A supervisor \( S \) is controllable if there are no uncontrollable events that cause transitions to forbidden states.

States can be forbidden for a number of reasons:

1. We have specified it as forbidden.
2. We have detected that it is blocking.
3. We have detected that the state is uncontrollable, to be defined.

We can make the supervisor controllable by forbidding the states where we have uncontrollable transitions to forbidden states.
Example 32. Consider the two-user supervisor from example 31, where we have forbidden the blocking states (indicated with red squares):

We have two blocking paths, two dead ends

Requests r1 and r2 are uncontrollable.

Both marked states must be forbidden.!

⇒ All the other states become blocking as we do not have any marked states.

The problem comes from that First-come first-served specification and user alternance specification is in conflict with each other. We have to choose one:
Both are controllable and nonblocking. From the latter we can remove 6 states, the two forbidden ones and four that can only be reached from forbidden states.
Example 33. Consider the tank supervisor from example 28.

This is both controllable and non-blocking.

Let us assume that we would have forgot specification 3, that heat should not be switched on before temperature is below target temperature:

Now if we switch on heating before temperature has fallen below target (event T0), T0 can no longer take place (according to the model, not realistic), and event T1 (temperature has risen to target) cannot either take place. So we can no longer turn off heat (E0), we have a deadlock (indicated with red squares in the figure above).

But the event E1 turning on heating is controllable, so the supervisor is controllable. All we have to do is to remove the red squares to make the system non-blocking.
Uncontrollable states

Uncontrollable states comes from a specification trying to restrict an uncontrollable event:

At uncontrollable states we have uncontrollable events that are allowed by the plant but disabled by the specification

Uncontrollable states can best be found from the parallel composition $G \parallel Sp$.

Specifications cannot stop uncontrollable events, so the uncontrollable events can take us outside the specifications, to a state not recognized by the supervisor, with unpredictable results!

Not the same as controllability, where we detect if uncontrollable events can take us to a forbidden state.

All uncontrollable states can be detected directly based on $G \parallel Sp$, while controllability of a supervisor depends on the forbidden states, which can be increased when searching for blocking states.
Finding uncontrollable states with Compare in Supremica

In Supremica the Compare operation finds and forbids uncontrollable states.

Synchronize (directly) = Synch + Compare (under Workbench)

Compare does not have to be repeated, and if it is done it should be done before NonBlocking (Coreachability) and Controllability, as it forbids states.

Compare detects a mismatch between the specification and the plant, the error can be in either or both.

Compare can sometimes "auto-correct" the specifications, when it is possible and desirable to simply avoid the uncontrollable states. This is the case in the example on next slide.

But it can also lead to a supervisor with only forbidden states (directly or after the synthesis), especially if there are uncontrollable events that are always allowed.
Example 34. Consider a robot and a machine modeled by the following automata:

The robot can take a workpiece from storage and put it available for the machine. The machine can load it and process it, and put it afterwards on either of two conveyor belts A and B. The put operation is uncontrollable (maybe the robot cannot be stopped after it has been started).

Let us first consider the following specification

We do not want the robot to overfill the machine with workpieces (there is nothing that stops it in the plant model).
If we Synchronize (Synch + Compare) it with the plant we get the following result:

Why are three states forbidden? Only one of them is a deadlock state (and we have not tested for that yet).
Because the states are uncontrollable, robot is at r1 and can execute put, which takes us outside the specs, not good.

Let us try to allow put then:

If we Synchronize we get the following result:

Now it is controllable, but is it ok? No, we do not stop overfilling of the machine.

So we actually have to forbid take before the machine has unloaded, which was the result of the compare-operation.
We can also put that directly into the specification (previously we left take for the plant to decide):

If we Synchronize we get the following result:

Here the only thing that is left to do is to forbid unloadB, causing a deadlock, with a NonBlock (Coreachability) operation.
What if we don consider put at all?

If we Synthesize we get the following result:

Again we do not stop overfilling the machines.
Controllability using languages

A supervisor $S$ is controllable with respect to a plant $G$ and a set of uncontrollable events $\Sigma_u$ if

$$(\mathcal{L}(S)\Sigma_u) \cap \mathcal{L}(G) \subseteq \mathcal{L}(S)$$

The uncontrollable events should not take the plant outside the language of the supervisor.

We see here clearly that controllability for $S$ depends on the plant $G$ and the set of uncontrollable events.

Note that this is the classical definition of controllability, which do not distinguish between uncontrollable states and supervisors.

From a practical point of view it is two different things, in that sense that we use two different operations for detection of these.