4 Controller checks

Before implementing a supervisor, checks for confluence, finite response and nonblocking under control are to be performed, as is discussed in Section VI. Folder 4 contains all the required files for performing these checks.

4.1 Folder *Confluence*

A supervisory controller can sometimes 'choose' between two controllable events. In a confluent supervisory controller the choice made between these events does not influence the future behavior of the system. [2] defines confluence as follows. Whenever a choice between two controllable events exists, each event can be extended by a sequence of controllable events such that both paths end in the same state.

CIF3 contains a tool to check if a supervisor is confluent. This tool checks if all combinations of controllable events meet the sufficient requirements for confluence proposed in [1]. If all combinations of controllable events are independent, mutually exclusive, update equivalent, skippable, or reversible, the supervisor is said to be confluent.

The folder named *Confluence*, contains the local supervisors in the files *loc1.cif* and *loc2.cif*, as well as a *tooldef2* file for each local supervisors, which is used to perform the confluence check. The tool cannot proof confluence for both local supervisors, and a list of event combinations that do not meet the requirements is listed. These event combinations are manually checked to proof confluence, two example cases are treated below.

The first event combination which is undefined, i.e. does not meet the sufficient requirements, is the combination of events *Pump1_c_on* and *Mode_c_store* of the middle pumpcellar. The set of states, in which both of these events are enabled, can be partitioned into three subsets for three cases, as shown in Figure 4.1.

- Case A: the water level in the middle pumpcellar is above level 5.
- Case B: the water level in the middle pumpcellar is below level 4.
- Case C: the water level in the middle pumpcellar is between level 4 and level 5.

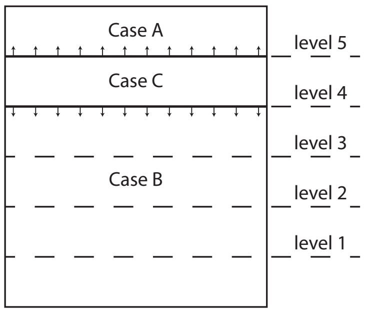

Figure 4.1: Cases A, B, and C

Note that in all cases event *Mode_c_store* is allowed after event *Pump1_c_on*, as the requirements for event *Mode_c_store* do not refer to component *Pump1* and *Pump1_c_on* never disables *Mode_c_store*.

In case A, the event *Pump1_c_on* is still allowed after event *Mode_c_store*, as the water level is above level 5. Therefore, the events are independent and reach the same state. In Case B, the event *Pump1_c_on* is a reverse event for event *Pump1_c_on* is enabled. Therefore, the event combination is reversible.

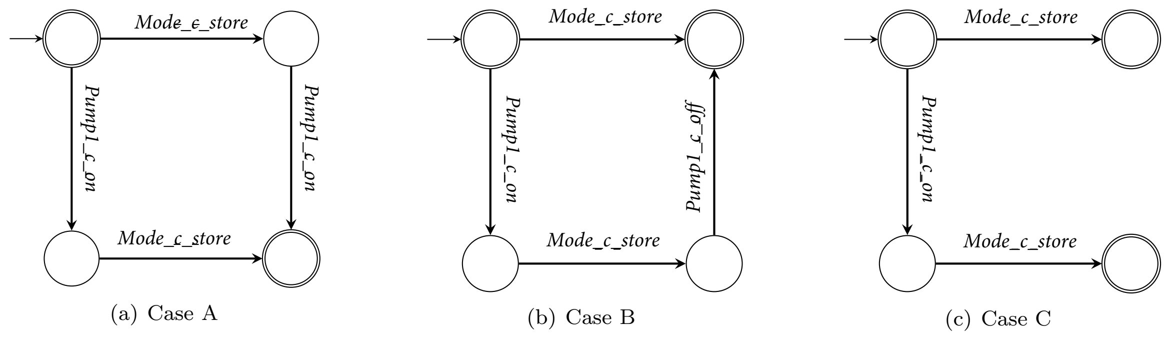

Figure 4.2: Three cases for *Pump1_c_on* and *Mode_c_store*

In case C, the event combination does not meet the requirements. However, after event *Pump1_c_on* and *Mode_c_store*, the water level will lower, until the water level comes below level 4. This will enable event *Pump1_c_off*. The system will therefore eventually reach the same state, through reverse event *Pump1_c_off*. In this project this is considered to be sufficient for confluence.

The second event combination, that is labeled undefined by the controller check, consists of events *Regime_c_bergen* of the second main pumpcellar and event *c_operational* of the first traffic tube. Again, three cases are identified.

- Case A: the *Mode* automaton of main pumpcellar is in state *Manual* and the *button observer* is in state *store*.
- Case B: the *Control Mode* is in state *auto* and the *Mode* is in state *drain*.
- Case C: the *Control Mode* is in state *auto* and the *Mode* is in state *off*.

In all three of these cases, the traffic tube is in state *emergency* or *recovery*, otherwise event *c_operational* is not enabled. Outside of these three cases, the event combination is mutually exclusive, as these are the only cases in which both events are enabled simultaneously. In case A, the events are independent, *c_operational* does not disable *c_store*, as the system is in manual mode. In case B, event *c_store* is reversible by event *c_drain*. Finally, the situation in case C is not recognized by the confluence check, but is sufficient for confluence. In this case, there are two possible paths of controllable events, which lead to the same state. Event combination *Mode_c_store* and *c_operational* is not identified by the confluence check, but does not cause the system to not have confluence.

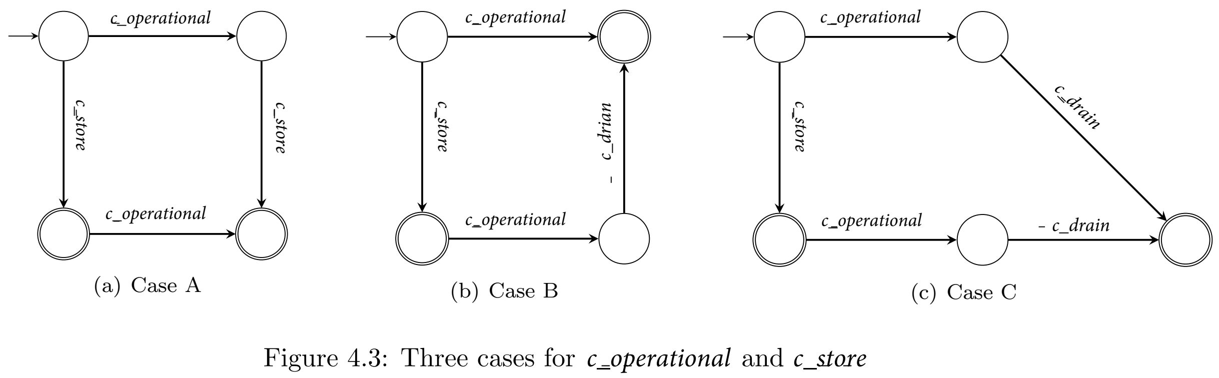

Figure 4.3: Three cases for *c_operational* and *c_store*

There are in total 24 event combinations that are unidentified by the confluence check, they are manually checked in a similar manner as the previously discussed event combinations. The remaining 22 event combinations however, are not discussed. The manual check concludes that the system is confluent.

4.2 Folder *Finite Response*

A PLC cycle is executed as follows: the PLC will first read the input variables, then execute controllable events until no events are enabled, and finally write to the output variables. If a supervisory controller contains a loop of controllable events, the PLC can infinitely execute this loop of events and therefore never start updating output variables. This will result in a cycle time exceeded error on the PLC. If a supervisory controller has finite response, no such loop of controllable events exists within the controller. This folder contains the local supervisors and two *.tooldef2* files, which run the finite response check of CIF3 for the local supervisors. Both local supervisors have finite response.

4.3 Folder *Nonblocking under control*

A system can become blocking under control, if a marked state can only be reached by an uncontrollable event from a state in which a controllable event is also enabled. A PLC will execute controllable events until no controllable events are enabled, before updating output and input variables. Therefore the PLC will always prioritize controllable events over uncontrollable events, which is discussed in [2].

To check if a system is nonblocking under control in CIF3, first, a boolean *ctrlEnabled* is defined. This boolean is true if a controllable event is enabled and false if no controllable event is enabled. Next, the automaton of Figure 4.4 is added. By adding this automaton to the model, uncontrollable events are disabled until no controllable events are enabled. Next, synthesis is performed. If the resulting supervisor automaton does not contain any guards, the system is nonblocking even with the addition of the automaton of Figure 4.4.

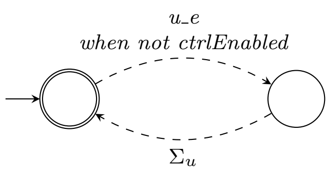

Figure 4.4: Restriction automaton

This folder contains two files, one for each local supervisor. In these files the supervisor automata are removed. Next, the restriction automaton is added. The databased synthesis tool of CIF3 results in supervisor automata without any guards. It can, therefore, be concluded that both local systems are nonblocking under control.