Landing Gear System: An ASM-based Solution for the ABZ’14 Case Study
Technical Report SCCH-TR-1401 with the complete model and proofs

Felix Kossak
Software Competence Center Hagenberg GmbH,
Softwarepark 21, 4232 Hagenberg, Austria
felix.kossak@scch.at
http://www.scch.at

Abstract. We present an ASM model for the case study given as a challenge for the ABZ’14 conference, which specifies the digital part of a landing gear system for aircraft. We strove to make the formal model well understandable for humans. We present manual proofs and one refutation of a selection of given requirements. We note inconsistencies, ambiguities and gaps in the case study.

Keywords: Formal specifications, Rigorous specifications, Abstract state machines, ASMs, Safety-critical software

1 Introduction

In this work, we present a solution to the case study by Boniol and Wiels given as a challenge for the ABZ’14 conference [1], which specifies the digital part of a landing gear system for aircraft. Our model is based on abstract state machines (ASMs) as presented in [2]. Throughout this paper, we will refer to the cited case study document by Boniol and Wiels as the “requirements document”, and by default, page numbers refer to this requirements document.

We strived to make the formal specification well understandable for humans. We use the same terms as given in the case study document whenever possible, and in general, we use long and telling identifiers (rather than single letters or fancy abbreviations). However, expecting all stakeholders to have a technical background, we assume them to be familiar with e.g. the usual set notation.

During our work on the model and the proofs, we have noted several inconsistencies or ambiguities, which we state in Section 2 on Specification Issues.

In Section 3, we present an ASM ground model for the software, i.e. the digital part of the landing gear system. The signature of the model is given at the end of Section 3.

In Section 4, we present several manual proofs and one refutation of a selection of both normal mode requirements and failure mode requirements as stated on pp. 18–19 of the requirements document. Amongst others, we prove that at
least one requirement \((R_{11})\) cannot be fulfilled given the times stated in the requirements document.

A summary of experiences gained by the case study will be published in the proceedings of the ABZ 2014 conference.

2 Specification Issues

We found a few issues in the requirements document which would, in a real-life scenario, require discussions with representatives of the customer. These regard ambiguities or confusing wording as well as obvious errors. We summarise these issues in this section.

2.1 Normal and Emergency Mode

We found the requirements document confusing regarding “normal” (or “nominal”?) mode and “emergency mode”. And while the document states that “In this case study, we do not consider the emergency mode” (p. 1), there is an output variable “anomaly” to be set and the requirements require to set an output variable “normal\_mode” to false in certain cases (see p. 19, “Failure mode requirements”).

Regarding notation, several different names are used for modes or variables which are apparently associated:

- On p. 1, “nominal mode” and “emergency mode” are mentioned.
- On p. 7, a boolean state variable “anomaly” is mentioned, parametrised by the computing module.
- On p. 19, a boolean output variable “normal\_mode” is mentioned (without mention of a parameter).
- No input variable is mentioned to indicate whether the whole system is actually in “normal/nominal mode” or “emergency mode”.

We assume that “normal\_mode” will be set to false whenever some computing module sets “anomaly” to true. We interpret “normal\_mode” as a global state variable which is set independently of the digital part, probably by the physical combination of the “anomaly” outputs of each computing module (cf. p. 7 of the requirements document); in the digital part, we need it as an input variable (see below). We furthermore assume that “nominal mode” is given as long as “normal\_mode” is true.

We interpret the remark on p. 3 (top), “… the green light … must be on”, such that the monitoring part of the specified system should still work even in case of “failure” (if possible). On the other hand, one can assume that the controlling part should not do anything anymore, i.e. should not send any commands to any valves anymore. In order to continue operating in principle while not sending any further commands, however, it is necessary to know whether the emergency mode is active or not, for which we use the variable “normal\_mode”, as mentioned. We interpret “failure” as synonymous with “anomaly”.
2.2 Synchronous Parallelism

Two computing modules shall run “in parallel” (p. 5). However, it is not explicitly stated whether they shall be executed synchronously or asynchronously. We assume that they shall be executed *asynchronously*.

2.3 Obvious Errors in Monitoring Specification

We found two obvious errors in Section 4.3 of the requirements document on health monitoring, regarding *gears motion monitoring* (p. 17):

- In the first list item, it surely must read, “if the control software does not see the value $gear_{\text{extended}}[x] = \text{false}$ [...] after stimulation of the retraction electro-valve [...]”, instead of $gear_{\text{retracted}}$.
- In the third list item, it must likewise read, “if the control software does not see the value $gear_{\text{retracted}}[x] = \text{false}$ [...] after stimulation of the extension electro-valve [...]”, instead of $gear_{\text{extended}}$.

We allowed ourselves to correct this in the formal specification.

Furthermore, under *Gears motion monitoring* (p. 17), in the second, third, and fourth item, it is stated that “the doors are considered as blocked” when obviously it must read, “the gears are considered as blocked”.

2.4 Inconsistencies in Timing

On p. 9 of the requirements document, it is stated that the analogical switch can take up to 0.8s (i.e. 800ms) to close. However, according to p. 16, an anomaly shall be detected already 160ms after the handle position has changed. We consider Section 4 of the requirements document to be authoritative (and Section 3 to be primarily informative), thus we assume 160ms.

2.5 Miscellaneous

In Section 5 (Requirements / Properties) of the requirements document (pp. 18–19), we encounter a “command button” which can be pushed “DOWN” or “UP”. We assume that this is synonymous to the “handle” as mentioned e.g. on p. 2 or on p. 6.

In Section 3 of the requirements document, on p. 11, it is stated that “door cylinders are locked [...] only in closed position.” This is corroborated on p. 5 (Section 2). However, on p. 6, we read that “$door_{\text{open}}[x]$ is true if and only if the corresponding door is locked open”. Likewise, in Section 5 (Requirements), e.g. in $(R_{31})$ (p. 18), there is talk of “when the three doors are locked open”. In this paper, we consider “open” and “locked open” as synonymous in the context of doors.
3 A Ground Model for the Software

We now present an ASM ground model for the software for the landing gear system. We assume knowledge of the requirements document [1]; quoted page numbers refer to this document by default.

A summary of the signature of the ASM can be found at the end of this section.

We largely use the ASM notation given in [2], but we make use of the flexibility of ASM notation to improve general understandability. We have experienced that lack of understandability for lay people is a major deterrent for the use of formal methods in practice. Therefore we also use long names for rules, functions, and local variables rather than single letters or short abbreviations – except from abbreviations used consistently in the requirements document as well (such as “EV” for “electro-valve”). However, as a compromise with the need for brevity and a clear structure, we use common set and set operator notation, assuming that the major stakeholders in this case have a technical background.

3.1 Main rules

Two computing modules run “in parallel” (p. 5). It is not explicitly stated whether they shall be executed synchronously or asynchronously, but we think that it is more plausible to execute them asynchronously, and we assume this.

For the sake of later generalisation, we define a set computingModules which for now shall contain two such modules. Furthermore, we define a function moduleNumber which maps each computing module to a unique number by which the respective output variables are parametrised. Each module runs the rule LandingGearComputingModule independently.

According to the requirements document, the outputs of the different computing modules are physically combined (p. 7); consequently, we do not combine those values in the digital part.

main rule LandingGearSystem =
sequentialblock
  Initialise
    foreach module ∈ computingModules do
      program(module) := LandingGearComputingModule(
        moduleNumber(module))
  endsequentialblock

abstract rule Initialise

Each module computes commands for the gears and status information for the pilot, which is specified in the rule ControlSystem, and also monitors the system to detect anomalies, which is specified in the rule MonitorSystem (cf. p. 13).

rule LandingGearComputingModule(moduleNumber) =
parallelblock
ControlSystem(moduleNumber)
MonitorSystem(moduleNumber)
endparallelblock

3.2 Control

The controlling part, in normal mode, consists of an outgoing sequence (p. 14), which we specify in the rule OutgoingSequence, and a retraction sequence (pp. 14–15), which we specify in the rule RetractionSequence. (Regarding the shared function normal_mode, see p. 1, p. 3, p. 13 (“Expected scenarios in normal mode”), and p. 19 of the requirements document; see also our section on Specification Issues. Regarding timing constraints, see pp. 15-16 of the requirements document.)

In each of the sequences (for extending or retracting), we used explicit state variables which are named in the following way: Each name starts with either “extend” or “retract”, depending on the subrule which caused the state; this is followed by the step number according to the requirements document (pp. 14–15), a possible sub-step number if we deemed it necessary (e.g. for steps which take time between being initialised and finished or for transition from the outgoing to the retraction sequence or the other way around), and a short natural-language expression for the state for better orientation. Exceptions are the states “lockedRetracted” and “lockedExtended”, each of which marks the end state of one full sequence as well as the (normal) starting sequence of the other full sequence.

In each of the mentioned subrules, we must regard the requirement that each sequence “should be interruptible by counter orders”. Consequently, steps in OutgoingSequence can also start from a retract_... state and vice versa. Note that we need to be able to deal with every possible state in both OutgoingSequence and RetractionSequence, thus we must also make sure that certain steps which cannot be aborted are completed; so e.g. the step between retract_5_1_stopKeepingDoorsOpen and retract_5_2_stoppedKeepingDoorsOpen, where we consider the minimum time between contrary orders, must be completed in any case, including within OutgoingSequence, before anything else is done.

The control part must also set the boolean state variables to the cockpit, gears_locked_down and gears_maneuvering (see p. 7 and p. 15 of the requirements document). (gears_maneuvering is set to true in the subrule OpenDoors; likewise, gears_locked_down is set to false in the subrule RetractGears.)

We use the rule EvaluateSensor to compute a single value from triple sensors (see p. 5 and p. 16 of the requirements document). It is a rule rather than a derived function because it needs to update the validity status of each channel and sensor if deviations are detected (see p. 16). Note that each computing module assesses the status of the sensors and their channels separately (as valid or not), thus we need the parameter moduleNumber for EvaluateSensor and similar subrules. For more complex state assessments, where we need to combine values for all three gears, we use custom derived functions (such as gearsLockedEx-
tended). For the same reason as for EvaluateSensor, we also need the parameter moduleNumber for the state.

For the health monitoring part (see MonitorSystem further below) to be able to measure how much time has elapsed since a command was given by the control part, we need to set a start\(Time\) for each command, parametrised by the module number, the electro-valve to which the command was given, and the kind of the command (true or false). Once we detect that an action has been finished, we have to reset the start\(Time\) to undef (undefined). Note that we need to check whether an action has completed also in the control part because only then can we start the next action.

```
rule ControlSystem(moduleNumber) =
    if normal_mode then
        parallelblock
            OutgoingSequence(moduleNumber)
            RetractionSequence(moduleNumber)
        endparallelblock
```

The rule OutgoingSequence specifies the outgoing sequence according to p. 14 of the requirements document.

```
rule OutgoingSequence(moduleNumber) =
    if EvaluateSensor(moduleNumber, handle) = down and
        EvaluateSensor(moduleNumber, analogical_switch) = closed then
        if state(moduleNumber) ∈ {lockedRetracted, retract, generalEValveOpening} then
            parallelblock
                CloseGeneralEV(moduleNumber)
                state(moduleNumber) := extend_1_1_generalEValveClosing
            endparallelblock
        else if state(moduleNumber) = extend_1_1_generalEValveClosing then
            if EvaluateSensor(moduleNumber, circuit_pressurized) = true and
                (now - startTime(moduleNumber, general_EV, true)) ≥
                minTimeBetweenGeneralEVAAndManeuvering then
                parallelblock
                    startTime(moduleNumber, general_EV, true) := undef
                    state(moduleNumber) := extend_1_2_hydraulicCircuitPressurized
                endparallelblock
        else if state(moduleNumber) = retract_6_doorsClosing then
            parallelblock
                StopDoorsClosing(moduleNumber)
                state(moduleNumber) := extend_2_0_1_stopDoorsClosing
            endparallelblock
    else if state(moduleNumber) = extend_2_0_1_stopDoorsClosing and
        (now - startTime(moduleNumber, close_EV, false)) ≥
minTimeBetweenContraryOrders then
state(moduleNumber) := extend_2.0.2_stoppedDoorsClosing
else if state(moduleNumber) =
    retract_5.1_stopKeepingDoorsOpen and
    (now - startTime(moduleNumber, open_EV, false)) ≥
    minTimeBetweenContraryOrders then
    // NB: this is a “retract” action which has to be completed
    state(moduleNumber) := retract_5.2_stoppedKeepingDoorsOpen
else if state(moduleNumber) ∈ {extend_1.2_hydraulicCircuitPressurized,
    extend_2.0.2_stoppedDoorsClosing,
    retract_7_doorsClosed,
    retract_5.2_stoppedKeepingDoorsOpen} then
    parallelblock
    OpenDoors(moduleNumber)
    state(moduleNumber) := extend_2.1_doorsOpening
    endparallelblock
else if state(moduleNumber) = extend_2.1_doorsOpening then
    if DoorsAreOpen(moduleNumber) then
        parallelblock
        startTime(moduleNumber, open_EV, true) := undef
        state(moduleNumber) := extend_2.2_doorsOpen
        endparallelblock
    else if state = retract_3.1_gearsRetracting then
        parallelblock
        StopGearsRetracting(moduleNumber)
        state(moduleNumber) := extend_3.0.1_stopGearsRetracting
        endparallelblock
else if state(moduleNumber) =
    extend_3.0.1_stopGearsRetracting and
    (now - startTime(moduleNumber, retract_EV, false)) ≥
    minTimeBetweenContraryOrders then
    state(moduleNumber) := extend_3.0.2_stoppedGearsRetracting
else if state(moduleNumber) =
    retract_3.0.1_stopGearsExtending and
    (now - startTime(moduleNumber, extend_EV, false)) ≥
    minTimeBetweenContraryOrders then
    state(moduleNumber) := retract_3.0.2_stoppedGearsExtending
else if state(moduleNumber) ∈ {extend_2.2_doorsOpen,
    extend_3.0.2_stoppedGearsRetracting,
    retract_4_gearsRetracted,
    retract_3.0.2_stoppedGearsExtending} then
    parallelblock
    ExtendGears(moduleNumber)
    state(moduleNumber) := extend_3.1_gearsExtending
    endparallelblock
else if state(moduleNumber) = extend_3_1_gearsExtending then
  if GearsAreLockedExtended(moduleNumber) then
    parallelblock
      extend_EV(moduleNumber) := false
      startTime(moduleNumber, extend_EV, true) := undef
      gears_lockdown(moduleNumber) := true
      state(moduleNumber) := extend_4_gearsExtended
    endparallelblock
  else if state(moduleNumber) = {extend_4_gearsExtended, retract_3_2_gearRetractionAborted, retract_2_2_doorsOpen, retract_2_1_doorsOpening} then
    parallelblock
      StopKeepingDoorsOpen(moduleNumber)
      state(moduleNumber) := extend_5_1_stopKeepingDoorsOpen
    endparallelblock
  else if state(moduleNumber) = {extend_5_1_stopKeepingDoorsOpen} and
       (now - startTime(moduleNumber, open_EV, false)) ≥
       minTimeBetweenContraryOrders then
    state(moduleNumber) := extend_5_2_stoppedKeepingDoorsOpen
  else if state(moduleNumber) = {extend_5_2_stoppedKeepingDoorsOpen, retract_2_0_stopDoorsClosing} and
       (now - startTime(moduleNumber, close_EV, false)) ≥
       minTimeBetweenContraryOrders then
    state(moduleNumber) := retract_2_0_2_stoppedDoorsClosing
  else if state(moduleNumber) ∈ {extend_5_2_stoppedKeepingDoorsOpen, retract_2_0_2_stoppedDoorsClosing} then
    parallelblock
      CloseDoors(moduleNumber)
      state(moduleNumber) := extend_6_doorsClosing
    endparallelblock
  else if state(moduleNumber) = extend_6_doorsClosing then
    if DoorsAreLockedClosed(moduleNumber) then
      parallelblock
        gears_maneuvering(moduleNumber) := false
        close_EV(moduleNumber) := false
        startTime(moduleNumber, close_EV, true) := undef
        state(moduleNumber) := extend_7_doorsClosed
      endparallelblock
    else if state(moduleNumber) ∈ {extend_7_doorsClosed, retract_1_2_hydraulicCircuitPressurized, retract_1_1_generalEValveClosing} and
         (now - startTime(moduleNumber, close_EV, true)) ≥
         minTimeBetweenManeuveringAndStopOfGeneralEV then
      parallelblock
        OpenGeneralERV(moduleNumber)
state(moduleName) := extend_8_generalEValveOpening
endparallelblock
else if state(moduleNumber) =
    extend_8_generalEValveOpening then
    if EvaluateSensor(moduleNumber, circuit_pressurized) = false then
        parallelblock
        startTime(moduleNumber, general_EV, false) := undef
        state(moduleNumber) := lockedExtended
        endparallelblock

The rule RetractionSequence specifies the retraction sequence according to pp. 14–15 of the requirements document. Note that RetractionSequence is not exactly the reverse of OutgoingSequence – see especially the possibility that gear retraction may be aborted if the shock absorbers are not relaxed.

rule RetractionSequence(moduleNumber) =
  if EvaluateSensor(moduleNumber, handle) = up and
    EvaluateSensor(moduleNumber, analogical_switch) = closed then
  if state(moduleNumber) ∈ {lockedExtended, extend_8_generalEValveOpening} then
    parallelblock
    CloseGeneralEV(moduleNumber)
    state(moduleNumber) := retract_1_1_generalEValveClosing
    endparallelblock
  else if state(moduleNumber) =
    retract_1_1_generalEValveClosing then
    if EvaluateSensor(moduleNumber, circuit_pressurized) = true and
      (now - startTime(moduleNumber, general_EV, true)) ≥
      minTimeBetweenGeneralEVAndManeuvering then
      parallelblock
      startTime(moduleNumber, general_EV, true) := undef
      state(moduleNumber) := retract_1_2_hydraulicCircuitPressurized
      endparallelblock
  else if state(moduleNumber) = extend_6_doorsClosing then
    parallelblock
    StopDoorsClosing(moduleNumber)
    state(moduleNumber) := retract_2_0_1_stopDoorsClosing
    endparallelblock
  else if state(moduleNumber) = retract_2_0_1_stopDoorsClosing and
    (now - startTime(moduleNumber, close_EV, false)) ≥
    minTimeBetweenContraryOrders then
    state(moduleNumber) := retract_2_0_2_stoppedDoorsClosing
  else if state(moduleNumber) =
    extend_5_1_stopKeepingDoorsOpen and
    (now - startTime(moduleNumber, close_EV, false)) ≥
minTimeBetweenContraryOrders then
// NB: this is an "extend" action to be completed
state(moduleNumber) := extend_5_2_stoppedKeepingDoorsOpen
else if state(moduleNumber) ∈ { retract_1_2_hydraulicCircuitPressurized,
                                 retract_2_0_2_stoppedDoorsClosing,
                                 extend_7_doorsClosed,
                                 extend_5_2_stoppedKeepingDoorsOpen} then
  parallelblock
    OpenDoors(moduleNumber)
    state(moduleNumber) := retract_2_1_doorsOpening
  endparallelblock
else if state(moduleNumber) = retract_2_1_doorsOpening then
  if DoorsAreOpen(moduleNumber) then
    parallelblock
      startTime(moduleNumber, open_EV, true) := undef
      state(moduleNumber) := retract_2_2_doorsOpen
    endparallelblock
  endparallelblock
else if state(moduleNumber) = extend_3_1_gearsExtending then
  parallelblock
    StopGearsExtending(moduleNumber)
    state(moduleNumber) := retract_3_0_1_stopGearsExtending
  endparallelblock
else if state(moduleNumber) = retract_3_0_1_stopGearsExtending and
  (now - startTime(moduleNumber, extend_EV, false)) ≥
  minTimeBetweenContraryOrders then
    state(moduleNumber) := retract_3_0_2_stoppedGearsExtending
else if state(moduleNumber) = extend_3_0_1_stopGearsRetracting and
  (now - startTime(moduleNumber, retract_EV, false)) ≥
  minTimeBetweenContraryOrders then
    state(moduleNumber) := extend_3_0_2_stoppedGearsRetracting
else if state(moduleNumber) ∈ {retract_2_2_doorsOpen,
                                 retract_3_0_2_stoppedGearsExtending,
                                 extend_4_gearsExtended,
                                 extend_3_0_2_stoppedGearsRetracting} then
  if ShockAbsorbersAreRelaxed(moduleNumber) then
    parallelblock
      RetractGears(moduleNumber)
      state(moduleNumber) := retract_3_1_gearsRetracting
    endparallelblock
  else
    state(moduleNumber) := retract_3_2_gearRetractionAborted
  endelse if state(moduleNumber) = retract_3_1_gearsRetracting then
if GearsAreLockedRetracted(moduleNumber) then
    parallelblock
        retract_EV(moduleNumber) := false
        startTime(moduleNumber, retract_EV, true) := undef
        state(moduleNumber) := retract_4_gearsRetracted
    endparallelblock
else if state(moduleNumber) ∈ {retract_4_gearsRetracted,
                                retract_3_2_gearRetractionAborted,
                                extend_2_2_doorsOpen, extend_2_1_doorsOpening} then
    parallelblock
        StopKeepingDoorsOpen(moduleNumber)
        state(moduleNumber) := retract_5_1_stopKeepingDoorsOpen
    endparallelblock
else if state(moduleNumber) = retract_5_1_stopKeepingDoorsOpen and
       (now - startTime(moduleNumber, open_EV, false)) ≥
       minTimeBetweenContraryOrders then
    state(moduleNumber) := retract_5_2_stoppedKeepingDoorsOpen
else if state(moduleNumber) = extend_2_0_1_stopDoorsClosing and
       (now - startTime(moduleNumber, close_EV, false)) ≥
       minTimeBetweenContraryOrders then
    state(moduleNumber) := extend_2_0_2_stoppedDoorsClosing
else if state(moduleNumber) ∈ {retract_5_2_stoppedKeepingDoorsOpen,
                                extend_2_0_2_stoppedDoorsClosing} then
    parallelblock
        CloseDoors(moduleNumber)
        state(moduleNumber) := retract_6_doorsClosing
    endparallelblock
else if state(moduleNumber) = retract_6_doorsClosing then
    if DoorsAreLockedClosed(moduleNumber) then
        parallelblock
            gears_maneuvering(moduleNumber) := false
            close_EV(moduleNumber) := false
            startTime(moduleNumber, close_EV, true) := undef
            state(moduleNumber) := retract_7_doorsClosed
        endparallelblock
else if state(moduleNumber) ∈ {retract_7_doorsClosed,
                                extend_1_2_hydraulicCircuitPressurized,
                                extend_1_1_generalEValveClosing} and
       (now - startTime(moduleNumber, close_EV, true)) ≥
       minTimeBetweenManeuveringAndStopOfGeneralEV then
    parallelblock
        OpenGeneralEV(moduleNumber)
        state(moduleNumber) := retract_8_generalEValveOpening
    endparallelblock
else if state(moduleNumber) = retract_generalEVValveOpening then
    if EvaluateSensor(moduleNumber, circuit_pressurized) = false then
        parallelblock
        startTime(moduleNumber, general_EV, false) := undef
        state(moduleNumber) := lockedRetracted
        endparallelblock

The following subrules, apart from starting the action indicated by their names, also set the startTime for the respective action as well as reset the startTime for the inverse action. This is required for health monitoring (see the next subsection).

rule CloseGeneralEV(moduleNumber) =
parallelblock
    general_EV(moduleNumber) := true
    startTime(moduleNumber, general_EV, true) := now
    startTime(moduleNumber, general_EV, false) := undef
    startTime(moduleNumber, analogical_switch, true) := undef
endparallelblock

rule StopDoorsClosing(moduleNumber) =
parallelblock
    close_EV(moduleNumber) := false
    startTime(moduleNumber, close_EV, false) := now
    startTime(moduleNumber, close_EV, true) := undef
endparallelblock

When we start to open the doors, the state variable gears_maneuvering for the cockpit must be set to true (see p. 15 of the requirements document).

rule OpenDoors(moduleNumber) =
parallelblock
    gears_maneuvering(moduleNumber) := true
    open_EV(moduleNumber) := true
    startTime(moduleNumber, open_EV, true) := now
    startTime(moduleNumber, open_EV, false) := undef
endparallelblock

rule StopGearsRetracting(moduleNumber) =
parallelblock
    retract_EV(moduleNumber) := false
    startTime(moduleNumber, retract_EV, false) := now
    startTime(moduleNumber, retract_EV, true) := undef
endparallelblock
When we start to retract the gears, the state variable `gears_locked_down` for the cockpit must be set to false (see p. 15 of the requirements document).

```plaintext
rule RetractGears(moduleNumber) =
parallelblock
    gears_locked_down(moduleNumber) := false
    retract_EV(moduleNumber) := true
    startTime(moduleNumber, retract_EV, true) := now
    startTime(moduleNumber, retract_EV, false) := undef
endparallelblock
```

For the specification of less specific subrules, some of which are also used in the monitoring part, see subsection “Auxiliary Subrules” further below.
3.3 Monitoring

The rule \textit{MonitorSystem} follows pp. 16–17 of the requirements document. The purpose is to detect an \textit{anomaly}. The required checks of five different kinds of system components can be performed in parallel. Note that each subrule can only set \textit{anomaly} to true (if applicable) but never to false, thus writing access to \textit{anomaly} as a shared variable cannot lead to a conflict.

\begin{verbatim}
rule MonitorSystem(moduleNumber) =
parallelblock
  CheckSensors(moduleNumber)
  CheckAnalogicalSwitch(moduleNumber)
  CheckPressureSensor(moduleNumber)
  CheckDoorsMotion(moduleNumber)
  CheckGearsMotion(moduleNumber)
endparallelblock
\end{verbatim}

The rule \textit{CheckSensors} checks every sensor whether it is still \textit{valid}. The validity of each sensor channel and of each sensor as a whole is checked every time a sensor is evaluated (in \textit{ControlSystem} or \textit{MonitorSystem}) – see the rule \textit{EvaluateSensor} in subsection \textit{Auxiliary Subrules} further below, which may set the controlled function \textit{isValid} for a sensor to false. See p. 16 of the requirements document.

\begin{verbatim}
rule CheckSensors(moduleNumber) =
  if forsome sensor ∈ Sensors holds
    isValid(moduleNumber, sensor) = false then
    anomaly(moduleNumber) := true
\end{verbatim}

In the rule \textit{CheckAnalogicalSwitch}, we check whether the analogical switch reacts to movements of the handle. Therefore we need to monitor whether the states of the handle and of the analogical switch have changed. To this end, we need to keep track of the last states of the handle as well as the analogical switch so that at any time, we can compare the previous states with the present states (given by the sensors). Therefore we maintain the controlled function \textit{lastValue}, parametrised with arbitrary sensors (in addition to the module number). An \textit{anomaly} is given when the analogical switch does not react to a handle movement within a certain period of time. For details, see p. 16 of the requirements document.

\begin{verbatim}
rule CheckAnalogicalSwitch(moduleNumber) =
  if HandleWasMoved(moduleNumber) then
    startTime(moduleNumber, analogical_switch, true) := now
  else if startTime(moduleNumber, analogical_switch, true) ≠ undef then
    if EvaluateSensor(moduleNumber, analogical_switch) = open then
      parallelblock
        if (now - startTime(moduleNumber, analogical_switch, true) ≥

\end{verbatim}
maxAnalogicalSwitchClosingTime \textbf{then} \\
\text{anomaly}(\text{moduleNumber}) \textbf{:= true} \\
\textbf{if} \text{lastValue}(\text{moduleNumber, analogical\_switch}) = \text{closed} \textbf{then} \\
\text{parallelblock} \\
\text{startTime}(\text{moduleNumber, analogical\_switch, true}) \textbf{:= undef} \\
\text{lastValue}(\text{moduleNumber, analogical\_switch}) \textbf{:= open} \\
\text{endparallelblock} \\
\text{endparallelblock} \\
\textbf{else if} \text{EvaluateSensor}(\text{moduleNumber, analogical\_switch}) = \text{closed} \\
\textbf{then} \\
\text{parallelblock} \\
\textbf{if} \text{lastValue}(\text{moduleNumber, analogical\_switch}) = \text{open} \textbf{then} \\
\text{lastValue}(\text{moduleNumber, analogical\_switch}) \textbf{:= closed} \\
\textbf{if} (\text{now} - \text{startTime}(\text{moduleNumber, analogical\_switch, true})) \geq \text{maxTimeAnalogicalSwitchAllowedClosed} + \text{maxAnalogicalSwitchOpeningTime} \textbf{then} \\
\text{anomaly}(\text{moduleName}) \textbf{:= true} \\
\text{endparallelblock} \\
\text{endparallelblock} \\

The auxiliary rule \textit{HandleWasMoved} checks whether the state of the \textit{handle} sensor has changed since the last iteration. It is a rule rather than a derived function because it uses \textit{EvaluateSensor} which is itself a rule as it will update the controlled function \textit{isValid} if necessary.

\textbf{rule} \textit{HandleWasMoved}(\text{moduleNumber}) = \\
\textbf{return} \textit{result} \textbf{in} \\
\textbf{if} \text{lastValue}(\text{moduleNumber, handle}) \neq \text{EvaluateSensor}(\text{moduleNumber, handle}) \textbf{then} \\
\text{parallelblock} \\
\text{result} \textbf{:= true} \\
\text{lastValue}(\text{moduleNumber, handle}) \textbf{:= EvaluateSensor}(\text{moduleNumber, handle}) \\
\text{endparallelblock} \\
\textbf{else} \\
\text{result} \textbf{:= false}

In the rule \textit{CheckPressureSensor}, we check whether the hydraulic pressure reacts to commands by the \textit{general\_EV} within a certain time (see p. 16 of the requirements document). Again, we use the controlled function \textit{lastValue} to detect changes of a sensor value. Note that the respective \textit{startTime} is set in \textit{ControlSystem} whenever a command is given, and reset to \textit{undef} each time the completion of the respective action has been detected. Thus if \textit{startTime} is \textit{undef} for some kind of action, we need not monitor anything with respect to this kind of action.

\textbf{rule} \textit{CheckPressureSensor}(\text{moduleNumber}) = \\
\textbf{parallelblock}
if startTime(moduleNumber, general_EV, true) ≠ undef then
    if EvaluateSensor(moduleNumber, circuit_pressurized) = false and
        (now - startTime(moduleNumber, general_EV, true)) ≥ maxCircuitPressuringTime then
        anomaly(moduleNumber) := true
    endif
endif
if startTime(moduleNumber, general_EV, false) ≠ undef then
    if EvaluateSensor(moduleNumber, circuit_pressurized) = true and
        (now - startTime(moduleNumber, general_EV, false)) ≥ maxCircuitDepressuringTime then
        anomaly(moduleNumber) := true
    endif
endif
endparallelblock

In the rule \textit{CheckDoorsMotion}, we check whether the doors actually react to commands within a certain time (see p. 17 of the requirements document). Again, the respective \texttt{startTime} is set in \texttt{ControlSystem} whenever a command is given, and reset to \texttt{undef} each time the completion of the respective action has been detected.

\texttt{rule CheckDoorsMotion(moduleNumber) = parallelblock
    if startTime(moduleNumber, open_EV, true) ≠ undef then
        if \texttt{not} NoDoorIsClosed(moduleNumber) and
            (now - startTime(moduleNumber, open_EV, true)) ≥ maxStartDoorsOpeningTime then
            anomaly(moduleNumber) := true
        endif
    else if \texttt{not} DoorsAreOpen(moduleNumber) and
        (now - startTime(moduleNumber, open_EV, true)) ≥ maxDoorOpeningTime then
            anomaly(moduleNumber) := true
    endif
endif
    if startTime(moduleNumber, close_EV, true) ≠ undef then
        if \texttt{not} NoDoorIsOpen(moduleNumber) and
            (now - startTime(moduleNumber, close_EV, true)) ≥ maxStartDoorsClosingTime then
                anomaly(moduleNumber) := true
        endif
    else if \texttt{not} DoorsAreLockedClosed(moduleNumber) and
        (now - startTime(moduleNumber, close_EV, true)) ≥ maxDoorClosingTime then
            anomaly(moduleNumber) := true
    endif
endparallelblock

Regarding gears motion monitoring, we allowed ourselves to correct two obvious errors in the specification document (p. 17), as already mentioned in our section on \textit{Specification Issues}:

- In the first list item, it must read, “if the control software does not see the value \texttt{gear\_extended}[x] = false [...] after stimulation of the retraction electro-valve [...]”, instead of \texttt{gear\_retracted}.
In the third list item, it must read, “if the control software does not see the value \( gear_{\text{retracted}}[x] = false \) [...] after stimulation of the extension electro-valve [...],” instead of \( gear_{\text{extended}} \).

In the rule \textit{CheckGearsMotion}, we check whether the gears actually react to commands within a certain time (see p. 17 of the requirements document). Again, the respective \textit{startTime} is set in \textit{ControlSystem} whenever a command is given, and reset to \textit{undef} each time the completion of the respective action has been detected.

\[
\text{rule CheckGearsMotion(moduleNumber) = par}
\]
\[
\text{allelblock}
\]
\[
\text{if startTime(moduleNumber, retract\_EV, true) \neq undef then}
\]
\[
\text{if not NoGearIsExtended(moduleNumber) and}
\]
\[
(now - startTime(moduleNumber, retract\_EV, true)) \geq max\text{StartGearsRetractingTime then}
\]
\[
\text{anomaly(moduleNumber) := true}
\]
\[
\text{else if not GearsAreLockedRetracted(moduleNumber) and}
\]
\[
(now - startTime(moduleNumber, retract\_EV, true)) \geq max\text{GearRetractingTime then}
\]
\[
\text{anomaly(moduleNumber) := true}
\]
\[
\text{if startTime(moduleNumber, extend\_EV, true) \neq undef then}
\]
\[
\text{if not NoGearIsRetracted(moduleNumber) and}
\]
\[
(now - startTime(moduleNumber, extend\_EV, true)) \geq max\text{StartGearsExtendingTime then}
\]
\[
\text{anomaly(moduleNumber) := true}
\]
\[
\text{else if not GearsAreLockedExtended(moduleNumber) and}
\]
\[
(now - startTime(moduleNumber, extend\_EV, true)) \geq max\text{GearExtendingTime then}
\]
\[
\text{anomaly(moduleNumber) := true}
\]
\[
\text{endparallelblock}
\]

3.4 Auxiliary Subrules

This subsection contains auxiliary subrules, some of which are used by both \textit{ControlSystem} and \textit{MonitorSystem}.

The rule \textit{EvaluateSensor} checks all channels of a given sensor. If all channels are considered as valid and one channel deviates from the other two, the deviating channel’s validity is set to false. If only two channels remain valid and they deviate from each other, the whole sensor is invalidated (see p. 16). Because \textit{EvaluateSensor} must be able to set the validity of channels and sensors, it is a rule rather than a derived function, yet with a (Boolean) return value.

\[
\text{rule EvaluateSensor(moduleNumber, sensor) = return sensorValue in}
\]
\[
\text{if isValid(moduleNumber, sensor) then}
\]


if forsome val ∈ SensorValues and 
  foreach channel ∈ SensorChannels holds 
    isValid(moduleNumber, sensor, channel) and
    sensor(channel) = val then
  sensorValue := val
else if forsome val ∈ SensorValues and 
  forsome channel_fault ∈ SensorChannels and 
  foreach channel_valid ∈ (SensorChannels \ { channel_fault }) holds 
    isValid(moduleNumber, sensor, channel_valid) and
    sensor(channel_valid) = val and
    sensor(channel_fault) ≠ val then
  parallelblock
    sensorValue := val
    isValid(moduleNumber, sensor, channel_fault) := false
  endparallelblock
else
  parallelblock
    sensorValue := undef
    isValid(moduleNumber, sensor) := false
  endparallelblock
else
  sensorValue := undef
end

The rule \textit{DoorsAreOpen} checks whether \textit{all} doors are open (for all landing sets). It uses the rule \textit{EvaluateSensor}, which can set the validity status of sensor channels and sensors, and therefore also \textit{DoorsAreOpen} is a rule rather than a derived function.

\textbf{rule} DoorsAreOpen(moduleNumber) =
  foreach landingSet ∈ landingSets holds
    EvaluateSensor(moduleNumber, door_open(landingSet)) = true

Likewise, the rule \textit{DoorsAreLockedClosed} checks whether all doors are locked closed.

\textbf{rule} DoorsAreLockedClosed(moduleNumber) =
  foreach landingSet ∈ landingSets holds
    EvaluateSensor(moduleNumber, door_closed(landingSet)) = true

The rule \textit{NoDoorIsClosed} checks whether none of the doors is locked closed. Likewise, the rule \textit{NoDoorIsOpen} checks whether none of the doors is (fully) open.

\textbf{rule} NoDoorIsClosed(moduleNumber) =
  foreach landingSet ∈ landingSets holds
    EvaluateSensor(moduleNumber, door_closed(landingSet)) = false

\textbf{rule} NoDoorIsOpen(moduleNumber) =
  foreach landingSet ∈ landingSets holds
    EvaluateSensor(moduleNumber, door_open(landingSet)) = false
The rule $\text{NoDoorIsOpen}(\text{moduleNumber})$ checks whether all gears are locked in the extended position.

$$\text{rule } \text{NoDoorIsOpen}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{door\_open(landingSet)}) = \text{false}$$

The rule $\text{GearsAreLockedExtended}(\text{moduleNumber})$ checks whether all gears are locked in the extended position.

$$\text{rule } \text{GearsAreLockedExtended}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{gear\_extended(landingSet)}) = \text{true}$$

The rule $\text{GearsAreLockedRetracted}(\text{moduleNumber})$ checks whether all gears are locked in the retracted position.

$$\text{rule } \text{GearsAreLockedRetracted}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{gear\_retracted(landingSet)}) = \text{true}$$

The rule $\text{NoGearIsExtended}(\text{moduleNumber})$ checks whether none of the gears is locked extended. Likewise, the rule $\text{NoGearIsRetracted}(\text{moduleNumber})$ checks whether none of the gears is locked retracted.

$$\text{rule } \text{NoGearIsExtended}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{gear\_extended(landingSet)}) = \text{false}$$

$$\text{rule } \text{NoGearIsRetracted}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{gear\_retracted(landingSet)}) = \text{false}$$

The rule $\text{ShockAbsorbersAreRelaxed}(\text{moduleNumber})$ checks whether all shock absorbers are relaxed (i.e. the gears are obviously not touching the ground).

$$\text{rule } \text{ShockAbsorbersAreRelaxed}(\text{moduleNumber}) =$$
$$\text{foreach } \text{landingSet} \in \text{landingSets} \text{ holds }$$
$$\text{EvaluateSensor}(\text{moduleNumber}, \text{gear\_shock\_absorber(landingSet)}) = \text{flight}$$

### 3.5 Signature

We give the signature of the ASM in the following order:

- Universes
- Constant functions
- Monitored (input) functions
- Output functions
- Controlled functions

Note that we have modelled a few sets as constant functions rather than as universes (such as the $\text{landingSets}$). This is because those could be configured without changing the rest of the specification (in particular, the algorithmic part); so, for instance, landing sets could be added or renamed.
Universes Additionally to the custom universes specified below, we assume the “standard” universes BOOLEAN and TIME (and implicitly also SET). We give times in milliseconds, represented by integers. We further assume that the difference between points in time, as given by the monitored function now (see further below), also gives times in milliseconds.

Universe Sensors consists of a set of monitored functions (or input functions), each of which is specified under Monitored functions further below (see also p. 6 of the requirements document). Some sensors can be directly addressed by a name, as those stated below, while others are addressed by a sensor group parametrised by a landing set – see universe SensorGroups below.

Sensors $\supseteq \{\text{handle, analogical switch, circuit pressurized}\}$

In a sensor group, there is one respective sensor for each landing set. Thus we model a sensor group as a constant function from landingSets (see below) to Sensors; each such function is specified under Monitored functions (see also our notes to universe Sensors).

SensorGroups = \{\text{gear extended, gear retracted, gear shock absorber, door closed, door open}\}

Each sensor has three channels (p. 5):

SensorChannels := \{1, 2, 3\}

(Note that we cannot make the sensor channels configurable due to the algorithm for evaluating the validity of a sensor; see the rule EvaluateSensor.)

The following possible states of certain sensors are taken from p. 6 of the requirements document.

HandleStates = \{up, down\}
SwitchStates = \{open, closed\}
ShockAbsorberStates = \{ground, flight\}
SensorValues = \bigcup\{BOOLEAN, HandleStates, SwitchStates, ShockAbsorberStates\}

Universe ElectroValves consists of a set of output functions, each of which is specified under Output functions further below (see also pp. 6–7 of the requirements document).

ElectroValves = \{general EV, close EV, open EV, retract EV, extend EV\}

The following, possible system states are basically derived from the steps given on pp. 14–15 of the requirements document, but also include intermediate steps as deemed necessary, including transition steps from the middle of the outgoing sequence to the retraction sequence or the other way around. Each state name starts with either “extend” or “retract”, depending on the rule which caused the
state (see OutgoingSequence (for “extend”) and RetractionSequence); this is followed by the step number according to the requirements document (pp. 14–15), a possible sub-step number if we deemed it necessary (e.g. for steps which take time between being initialised and finished or for transition from the outgoing to the retraction sequence or the other way around), and a short natural-language expression for the state for better orientation. Exceptions are the states “lockedRetracted” and “lockedExtended”, each of which marks the end state of one full sequence as well as the (normal) starting sequence of the other full sequence.

\[
\text{SystemStates} = \{ \text{lockedRetracted, lockedExtended, extend}_{1.1} \text{generalEValveClosing,}
\]
\[
\text{extend}_{1.2} \text{hydraulicCircuitPressurized, extend}_{2.0} \text{stopDoorsClosing, extend}_{2.0.2} \text{stoppedDoorsClosing, extend}_{2.1} \text{doorsOpening, extend}_{2.2} \text{doorsOpen, extend}_{3.0} \text{stopGearsRetracting, extend}_{3.0.2} \text{stoppedGearsRetracting, extend}_{3.1} \text{gearsExtending, extend}_{4.2} \text{gearsExtended, extend}_{5.1} \text{stopKeepingDoorsOpen, extend}_{5.2} \text{stoppedKeepingDoorsOpen, extend}_{6} \text{doorsClosing, extend}_{7} \text{doorsClosed, extend}_{8} \text{generalEValveOpening, retract}_{1.1} \text{generalEValveClosing,}
\]
\[
\text{retract}_{1.2} \text{hydraulicCircuitPressurized, retract}_{2.0} \text{stopDoorsClosing, retract}_{2.0.2} \text{stoppedDoorsClosing, retract}_{2.1} \text{doorsOpening, retract}_{2.2} \text{doorsOpen, retract}_{3.0} \text{stopGearsExtending, retract}_{3.0.2} \text{stoppedGearsExtending, retract}_{3.1} \text{gearsRetracting, retract}_{3.2} \text{gearRetractionAborted, retract}_{4} \text{gearsRetracted, retract}_{5.1} \text{stopKeepingDoorsOpen, retract}_{5.2} \text{stoppedKeepingDoorsOpen, retract}_{6} \text{doorsClosing, retract}_{7} \text{doorsClosed, retract}_{8} \text{generalEValveOpening} \}
\]

**Constant functions** Some of the following constant functions could alternatively have been declared as universes. However, we want to emphasise that these could be easily configured, e.g. by adding or renaming landing sets, as opposed to e.g. sensors, whose extension would require to change the algorithmic part of the specification.

Each computing module is identified by a natural number in order to address the respective output channels (cf. pp. 6–7 of the requirements document) as well as to maintain its own state, and the constant function \textit{moduleNumber} assigns this identification number to each module (see the main rule, \textit{LandingGearSystem}).

\[
\text{computingModules} := \{ \text{Module1, Module2} \}
computingModuleNumbers := \{1, 2\}
moduleNumber : \{Module1 \mapsto 1, Module2 \mapsto 2\}

The landing gear consists of three landing sets (see p. 3 and p. 6 of the requirements document):

landingSets := \{front, right, left\}

The following constant functions refer to the temporal behaviour of the system. Times (more specifically, periods of time) are given in milliseconds.

The first set of times concern timing constraints for the control part, as given in Section 4.2 of the requirements document (pp. 15–16).

\begin{align*}
\text{minTimeBetweenGeneralEVAndManeuvering} &= 200 \\
\text{minTimeBetweenManeuveringAndStopOfGeneralEV} &= 1000 \\
\text{minTimeBetweenContraryOrders} &= 100
\end{align*}

The following times are needed in health monitoring (see rule MonitorSystem). Times regarding the analogical switch and the general EV (circuit pressurising) are taken from p. 16 of the requirements document, times regarding doors and gears from p. 17.

\begin{align*}
\text{maxAnalogicalSwitchClosingTime} &= 160 \\
\text{maxAnalogicalSwitchOpeningTime} &= 280 \\
\text{maxTimeAnalogicalSwitchAllowedClosed} &= 40000 \\
\text{maxCircuitPressuringTime} &= 2000 \\
\text{maxCircuitDepressuringTime} &= 10000 \\
\text{maxStartDoorsOpeningTime} &= 500 \\
\text{maxDoorOpeningTime} &= 2000 \\
\text{maxStartDoorsClosingTime} &= 500 \\
\text{maxDoorClosingTime} &= 2000 \\
\text{maxStartGearsRetractingTime} &= 500 \\
\text{maxGearRetractingTime} &= 10000 \\
\text{maxStartGearsExtendingTime} &= 500 \\
\text{maxGearExtendingTime} &= 10000
\end{align*}

**Monitored functions** Monitored functions represent system input.

We assume a monitored function \texttt{now} which gives the current system time.

\texttt{now} : \rightarrow \text{TIME}

The custom monitored functions correspond to the inputs listed in the requirements document on p. 6. Each monitored function corresponds to a sensor (cf. universe \texttt{Sensors}) or a set of sensors (one sensor for each landing set, see universe \texttt{SensorGroups} and controlled function \texttt{landingSets}), and each sensor consists of three channels (see universe \texttt{SensorChannels}). They shall all be initialised as \texttt{undef}. 
handle : SensorChannels → HandleStates
analogical_switch : SensorChannels → SwitchStates
circuit_pressurized : SensorChannels → BOOLEAN

In the case of sensor groups, we define, for each sensorGroup in SensorGroups,

sensorGroup : landingSets → Sensors

and for each landingSet in landingSets,

gear_extended(landingSet) : SensorChannels → BOOLEAN
gear_retracted(landingSet) : SensorChannels → BOOLEAN
gear_shock_absorber(landingSet) : SensorChannels → ShockAbsorberStates
door_closed(landingSet) : SensorChannels → BOOLEAN
door_open(landingSet) : SensorChannels → BOOLEAN

As discussed in our section on Specification Issues, we further assume that the shared function normal_mode is given as an input (probably as a physical combination of the anomaly outputs of each computing module).

normal_mode : → BOOLEAN

Output functions Output functions correspond to the outputs as listed in the requirements document on pp. 6–7. The first group consists of commands to the electro-valves, all of which shall be initialised as false. The second group consists of state variables for the cockpit; by default, they shall be initialised as undef, as we do not know the initial states at start-up. (We might assume gears_locked_down = true, gears_maneuvering = false, and anomaly = false, but for the sake of safety, such assumptions must be discussed with the customer.)

general_EV : computingModuleNumbers → BOOLEAN
close_EV : computingModuleNumbers → BOOLEAN
open_EV : computingModuleNumbers → BOOLEAN
retract_EV : computingModuleNumbers → BOOLEAN
extend_EV : computingModuleNumbers → BOOLEAN
gears_locked_down : computingModuleNumbers → BOOLEAN
gears_maneuvering : computingModuleNumbers → BOOLEAN
anomaly : computingModuleNumbers → BOOLEAN

Controlled functions Controlled functions are controlled by the system and not visible from outside. They describe the internal state of the ASM at each iteration.
The controlled function \emph{state} describes the explicit states of the control sequences specified in \emph{OutgoingSequence} and \emph{RetractionSequence}. (It does not describe the overall state of the ASM, for which all controlled as well as all monitored and output functions are relevant!) The \emph{state} shall be initialised by \emph{undef} by default as we do not explicitly know the initial state of the hardware (though \emph{lockedExtended} could be guessed).

\[
\text{state} : \text{computingModuleNumbers} \rightarrow \text{SystemStates}
\]

The controlled function \emph{isValid} stores for each sensor and for each sensor channel whether it is considered to be valid or not; see the rule \emph{EvaluateSensor} and \emph{Generic monitoring} at p. 16 of the requirements document. It is to be initialised by \textit{true} for each location.

\[
\text{isValid} : \text{computingModuleNumbers} \times \text{Sensors} \rightarrow \text{BOOLEAN}
\]

\[
\text{isValid} : \text{computingModuleNumbers} \times \text{Sensors} \times \text{sensorChannels} \rightarrow \text{BOOLEAN}
\]

The controlled function \emph{startTime} models the time at which a particular command (represented by \textit{true} or \textit{false}) was given to an electro-valve by a given computing module. Once it is detected that the command achieved the desired effect (e.g. all doors are fully open), \emph{startTime} is reset to \emph{undef} (which is also the initial value for each location).

\[
\text{startTime} : \text{computingModuleNumbers} \times \text{ElectroValves} \times \text{BOOLEAN} \rightarrow \text{TIME}
\]

The controlled function \emph{lastValue} is used to detect changes of sensor values: it allows to compare the last sensor value (e.g. the last state of the handle) with the current value. This is used in health monitoring (cf. pp. 16-17 of the requirements document). The initial value for all locations is \emph{undef}.

\[
\text{lastValue} : \text{computingModuleNumbers} \times \text{Sensors} \rightarrow \text{SensorValues}
\]

\section{Proof of Requirements}

In this section we prove or refute fulfilment of some of the requirements stated in Section 5 (pp. 17–19) of the requirements document. We restricted ourselves to the refutation of \((R_{11})\) and the proofs of \((R_{11\text{bis}}), (R_{21}), (R_{22}), (R_{31})\), and \((R_{41})\), all from the \emph{normal mode requirements}, as well as the proofs of \((R_{61})\) and \((R_{71})\) from the \emph{failure mode requirements}.

We assume that computation time is negligible. We show each theorem for an arbitrary but fixed \emph{moduleNumber}. We state summed-up maximum times up to a given state with labels identifying the requirement number plus a running index. By default, times are given in milliseconds (ms).

We use the times given in Section 4 of the requirements document (pp. 16–17) rather than those given in Section 3 because an anomaly will only be detected after the times given in Section 4 are reached without success. (Compare our note on activation time of the analogical switch in our section on \emph{Specification Issues}.)
4.1 Normal Mode Requirements

(R11) We show that requirement \((R_{11})\) cannot be guaranteed. With the relevant times stated in the requirements document, extending the gears can take up to 16260 milliseconds from the movement of the handle to “down” (assuming that it is kept to “down” all the time), that is, more than 16 seconds, rather than the required maximum of less than 15 seconds.

If normal_mode is true, then OutgoingSequence is called in every iteration (see ControlSystem).

If the handle is pushed down, then the analogical switch will be activated (outside of the digital part). According to p. 16, we can assume that this takes less than 160ms, for otherwise an anomaly shall be detected. (According to p. 9, it can take up to 800ms, but as noted in our section on Specification Issues, we consider 160ms to be authoritative.)

(R11-1) We consider a maximum of 160ms until the analogical switch is closed after the handle was moved.

If EvaluateSensor(moduleNumber, handle) = down and EvaluateSensor(moduleNumber, analogical_switch) = closed, then OutgoingSequence makes a step which depends on the system state.

The worst case with respect to time is if we start in one of the states in \{lockedRetracted, retract \& generalEvalveOpening\} because

- then we must first pressurise the circuit, which takes at least 200ms (p. 15), while switching from any command to the opposite command, as may be required when we start from some other state of the retraction sequence, must only wait for at least 100ms (p. 16), and
- every step which must be performed after a switch from another state of the retraction sequence must also be performed after pressurising, plus in the latter case we must also wait until the circuit has actually been pressurised.

So we assume that the state is either lockedRetracted or retract \& generalEValveOpening.

Consequently, in the next iteration (the next step of the ASM), CloseGeneralEV is called. In the following state, extend_1 \_ generalEValveClosing, we have to wait until the general electro-valve has closed, which takes at most 2 seconds (2000ms) (p. 16). Thus, in the worst normal-mode case, we reach the state \(extend_1 \_ 2 \_ hydraulicCircuitPressurized\) in 160ms (R11-1) + 2000ms:

(R11-2) We reach the state \(extend_1 \_ 2 \_ hydraulicCircuitPressurized\) in a maximum of 2160ms.

When state = \(extend_1 \_ 2 \_ hydraulicCircuitPressurized\), OpenDoors is called and we reach the state \(extend_2 \_ 1 \_ doorsOpening\). According to p. 17, opening the doors is not supposed to take longer than 2 seconds, so we reach the state \(extend_2 \_ 2 \_ doorsOpen\) in a maximum of 2160ms + 2000ms:
(R11-3) We reach the state $extend_{2\_2\_doorsOpen}$ in a maximum of
4160ms.

When $state = extend_{2\_2\_doorsOpen}$, $ExtendGears$ is called and we reach the
state $extend_{3\_1\_gearsExtending}$. According to p. 17, extending the gears is not
supposed to take longer than 10 seconds, so we reach the state
$extend_{4\_gearsExtended}$ in a maximum of $4160ms + 10000ms$:

(R11-4) We reach the state $extend_{4\_gearsExtended}$ in a maximum of
14160ms.

When $state = extend_{4\_gearsExtended}$, $StopKeepingDoorsOpen$ is called and
we reach the state $extend_{5\_1\_stopKeepingDoorsOpen}$. Before we can switch to state $extend_{5\_2\_stoppedKeepingDoorsOpen}$, according to p. 16, we have to wait
at least 100ms before we can start the contrary order to close the doors. We can
assume that we reach the state $extend_{5\_2\_stoppedKeepingDoorsOpen}$ in $14160ms + 100ms$:

(R11-5) We reach the state $extend_{5\_2\_stoppedKeepingDoorsOpen}$ in a
maximum of 14260ms.

When $state = extend_{5\_2\_stoppedKeepingDoorsOpen}$, $CloseDoors$ is called
and we reach the state $extend_{7\_doorsClosing}$. According to p. 17, closing the
doors can take up to 2 seconds, so we reach the state $extend_{7\_doorsClosed}$ in a
maximum of $14260ms + 2000ms$:

(R11-6) We reach the state $extend_{7\_doorsClosed}$ in a maximum of 16260ms.

16260ms is more than 16 seconds and thus more than 15 seconds, so require-
ment $(R_{11})$ cannot be fulfilled.

$(R_{11}\text{bis})$ Above, we have shown that the gears will be locked down and the doors
will be closed after a maximum of 16260 milliseconds from the time when the
handle is moved to “down” (provided it is kept in that position). Consequently,
we have shown that the gears will eventually be locked down and the doors
closed.

Thus requirement $(R_{11}\text{bis})$ is fulfilled by the given ASM model.

$(R_{21})$ We prove that $(R_{21})$ is fulfilled by the given ASM model.

If $normal\_mode$ is true, then $RetractionSequence$ is called in every iteration
(see $ControlSystem$).

However, $RetractionSequence$ has an overall guard which is a conjunction
including the expression,

$$\text{EvaluateSensor(moduleNumber, handle)} = \text{up}$$

When the handle is $\text{down}$, it cannot be $\text{up}$ (cf. p. 6). Thus if the handle is
$\text{down}$, no action is taken in $RetractionSequence$.

Consequently, if the “gear command button” (i.e. handle) remains in the
DOWN position, the retraction sequence is not observed, which was to be proven.
(R22) We prove that (R22) is fulfilled by the given ASM model.

If normal_mode is true, then OutgoingSequence is called in every iteration (see ControlSystem).

However, OutgoingSequence has an overall guard which is a conjunction including the expression,

\[ \text{EvaluateSensor}(moduleNumber, \text{handle}) = \text{down} \]

When the handle is up, it cannot be down. Thus if the handle is up, no action is taken in OutgoingSequence.

Consequently, if the “gear command button” (i.e. handle) remains in the UP position, then the outgoing sequence is not observed, which was to be proven.

(R31) We prove that (R31) is fulfilled by the given ASM model.

Note that we consider “open” and “locked open” as synonymous in the context of doors (see our remark in our section on Specification Issues).

If normal_mode is true, then both OutgoingSequence and RetractionSequence are called. We consider the outgoing sequence and the retraction sequence separately.

We start with OutgoingSequence, where only gear extending is done.

The stimulation of “the gears outgoing” electro-valve, i.e. setting \( \text{extend} \_EV \) to true, is done via ExtendGears in one of the four states, extend_2_2_doorsOpen, extend_3_0_2_stoppedGearsRetracting, retract_4_gearsRetracted, or retract_3_0_2_stoppedGearsExtending (all of which lead to the state extend_3_1_gearsExtending).

We consider these four states separately.

1. extend_2_2_doorsOpen is only reached from state extend_2_1_doorsOpening and only if DoorsAreOpen, which checks whether all doors are open.
2. extend_3_0_2_stoppedGearsRetracting is only reached from extend_3_0_1_stopGearsRetracting, which in turn is only reached from retract_3_1_gearsRetracting. The latter state can be reached from one of the states, retract_2_2_doorsOpen, retract_3_0_2_stoppedGearsExtending, extend_4_gearsExtended, or extend_3_0_2_stoppedGearsRetracting. We consider all four states separately.
   (a) retract_2_2_doorsOpen is only reached from retract_2_1_doorsOpening and only if DoorsAreOpen.
   (b) retract_3_0_2_stoppedGearsExtending is only reached from retract_3_0_1_stopGearsExtending, which in turn is only reached from extend_3_1_gearsExtending, which in turn is the state reached when ExtendGears is called, which is the main case we are currently considering. So if we can show that all doors must be open in all other cases, we have also shown this case.
   (c) extend_4_gearsExtended is only reached from extend_3_1_gearsExtending – see the previous case (2b).
   (d) extend_3_0_2_stoppedGearsRetracting is the case we are currently considering (2), so if we can show that all doors must be open in all other cases, we have also shown this case.
So from considering the four cases above, we conclude that the state $extend_{3.0.2.stoppedGearsRetracting}$ can only be reached when all doors are open.

3. $retract_{4.gearsRetracted}$ is only reached from $retract_{3.1.gearsRetracting}$, which in turn can be reached from one of the states, $retract_{2.2.doorsOpen}$, $retract_{3.0.2.stoppedGearsExtending}$, $extend_{4.gearsExtended}$, or $extend_{3.0.2.stoppedGearsRetracting}$. We consider all four states separately.

   (a) $retract_{2.2.doorsOpen}$ has already been considered above (2a).

   (b) $retract_{3.0.2.stoppedGearsExtending}$ has already been considered above (2b).

   (c) $extend_{4.gearsExtended}$ has already been considered above (2c).

   (d) $extend_{3.0.2.stoppedGearsRetracting}$ has already been considered above (2d).

So we conclude that the state $retract_{4.gearsRetracted}$ can only be reached when all doors are open.

4. $retract_{3.0.2.stoppedGearsExtending}$ has already been considered above (2b).

So from considering the four (outer) cases above, we conclude that $ExtendGears$ can only be called when all doors are open. Consequently:

(R31-1) We have shown for the outgoing sequence that the stimulation of the “gears outgoing” electro-valve can only happen when the three doors are open.

Now we consider RetractionSequence, where only gear retraction is done.

The stimulation of the “retraction” electro-valve, i.e. setting $retract_{EV}$ to true, is done via $RetractGears$ in one of the four states, $retract_{2.2.doorsOpen}$, $retract_{3.0.2.stoppedGearsExtending}$, $extend_{4.gearsExtended}$, or $extend_{3.0.2.stoppedGearsRetracting}$. All four cases have already been considered in the context of the outgoing sequence above:

- $retract_{2.2.doorsOpen}$ in (2a),
- $retract_{3.0.2.stoppedGearsExtending}$ in (2b),
- $extend_{4.gearsExtended}$ in (2c), and
- $extend_{3.0.2.stoppedGearsRetracting}$ in (2).

So we conclude that also $RetractGears$ can only be called when all doors are open. Consequently:

(R31-2) We have shown for the retraction sequence that the stimulation of the retraction electro-valve can only happen when the three doors are open.

From (R31-1) and (R31-2), having considered both the outgoing and the retraction sequence, we follow for the whole model that the stimulation of the gears outgoing can only happen when the three doors are open, which was to be proven.
(R_{41}) We prove that (R_{41}) is fulfilled by the given ASM model.

We prove two properties separately:

\begin{itemize}
  \item (R41-1) Opening and closure doors electro-valves are not stimulated simultaneously.
  \item (R41-2) Outgoing and retraction gears electro-valves are not stimulated simultaneously.
\end{itemize}

If \textit{normal\_mode} is true, then both \textit{OutgoingSequence} and \textit{RetractionSequence} will be called. We consider both sequences separately for both (R41-1) and (R41-2).

(R41-1) We have to show that \texttt{open\_EV} and \texttt{close\_EV} are never true at the same time. We start with the outgoing sequence (R41-1-Out) and then prove the property for the retraction sequence (R41-1-Ret).

(R41-1-Out) In \textit{OutgoingSequence}, we first show that whenever \texttt{open\_EV} is stimulated, then \texttt{close\_EV} is false (R41-1-Out-Open). Then we show that whenever \texttt{close\_EV} is stimulated, then \texttt{open\_EV} is false (R41-1-Out-Close).

(R41-1-Out-Open) We show that whenever \texttt{open\_EV} is stimulated, then \texttt{close\_EV} is false. The opening doors electro-valve (\texttt{open\_EV}) is only stimulated in the subrule \textit{OpenDoors}, which is only called in one of the states \texttt{extend\_1\_2\_hydraulicCircuitPressurized}, \texttt{extend\_2\_0\_2\_stoppedDoorsClosing}, \texttt{retract\_7\_doorsClosed}, or \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen}. We consider all four cases separately.

(R41-1-Out-Open-1) \texttt{extend\_1\_2\_hydraulicCircuitPressurized} can only be reached from \texttt{extend\_1\_1\_generalEValveClosing}, in which state we wait for the general electro-valve to close. Before the general electro-valve has not closed, no other electro-valve can be successfully stimulated nor is one stimulated. Thus, in particular, also \texttt{close\_EV} cannot be and is not stimulated. So case (R41-1-Out-Open-1) is proven.

(R41-1-Out-Open-2) \texttt{extend\_2\_0\_2\_stoppedDoorsClosing} can only be reached from \texttt{extend\_2\_0\_1\_stopDoorsClosing}, which in turn can only be reached in a step (from \texttt{retract\_6\_doorsClosing}) in which the subrule \textit{StopDoorsClosing} is called. In \textit{StopDoorsClosing}, \texttt{close\_EV} is set to \texttt{false}, thus when \texttt{OpenDoors} is called, \texttt{close\_EV} is \texttt{false}. So case (R41-1-Out-Open-2) is proven.

(R41-1-Out-Open-3) \texttt{retract\_7\_doorsClosed} can only be reached in a step (from \texttt{retract\_6\_doorsClosing}) in which \texttt{close\_EV} is explicitly set to \texttt{false}, thus when \texttt{OpenDoors} is called, \texttt{close\_EV} is \texttt{false}. So case (R41-1-Out-Open-3) is proven.

(R41-1-Out-Open-4) \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen} can only be reached from \texttt{retract\_5\_1\_stopKeepingDoorsOpen}, which in turn can only be reached from one of the states \texttt{retract\_4\_gearsRetracted}, \texttt{retract\_3\_2\_gearRetractionAborted}, \texttt{extend\_2\_2\_doorsOpen}, or \texttt{extend\_2\_1\_doorsOpening}. We consider all four cases separately.

(R41-1-Out-Open-4-1) \texttt{retract\_4\_gearsRetracted} can only be reached from \texttt{retract\_3\_1\_gearsRetracting}, which in turn can be reached from one of the states \texttt{retract\_2\_2\_doorsOpen}, \texttt{retract\_3\_0\_2\_stoppedGearsExtending}, \texttt{extend\_4\_gearsExten-
ded, or extend$_3.0.2$. We consider all four cases separately.

(R41-1-Out-Open-4-1-1) retract$_2.2$.doorsOpen can only be reached from retract$_2.1$.doorsOpening, which in turn can be reached from one of the states retract$_1.2$.hydraulicCircuitPressurized, retract$_2.0.2$.stoppedDoorsClosing, extend$_7$.doorsClosed, or extend$_5.2$.stoppedKeepingDoorsOpen. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-1) retract$_1.2$.hydraulicCircuitPressurized can only be reached from retract$_1.1$.generalEVValveClosing, in which state we wait for the general electro-valve to close. Before the general electro-valve has not closed, no other electro-valve can be successfully stimulated nor is one stimulated. Thus, in particular, also close$_EV$ cannot be and is not stimulated. So case (R41-1-Out-Open-4-1-1-1) is proven.

(R41-1-Out-Open-4-1-1-2) retract$_2.0.2$.stoppedDoorsClosing can only be reached from retract$_2.0.1$.stopDoorsClosing, which in turn can only be reached in a step (from extend$_6$.doorsClosing in which the subrule StopDoorsClosing is called. In StopDoorsClosing, close$_EV$ is set to false, thus when OpenDoors is called, close$_EV$ is false. So case (R41-1-Out-Open-1-1-2) is proven.

(R41-1-Out-Open-4-1-1-3) extend$_7$.doorsClosed can only be reached in a step (from extend$_6$.doorsClosing) in which close$_EV$ is explicitly set to false, thus when OpenDoors is called, close$_EV$ is false. So case (R41-1-Out-Open-4-1-1-3) is proven.

(R41-1-Out-Open-4-1-1-4) extend$_5.2$.stoppedKeepingDoorsOpen can only be reached from extend$_5.1$.stopKeepingDoorsOpen, which in turn can only be reached from one of the states extend$_4$.gearsExtended, retract$_3.2$.gearRetraction-Aborted, retract$_2.2$.doorsOpen, or retract$_2.1$.doorsOpening. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-4-1) extend$_4$.gearsExtended can only be reached from extend$_3.1$.gearsExtending, which in turn can be reached from one of the states extend$_2.2$.doorsOpen, extend$_3.0.2$.stoppedGearsRetracting, retract$_4$.gearsRetracted, or retract$_3.0.2$.stoppedGearsExtending. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-4-1-1) extend$_2.2$.doorsOpen can only be reached from extend$_2.1$.doorsOpening, which in turn can be reached from one of the states extend$_1.2$.hydraulicCircuitPressurized, extend$_2.0.2$.stoppedDoorsClosing, retract$_7$.doorsClosed, or retract$_5.2$.stoppedKeepingDoorsOpen. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-4-1-1-1) The case of state extend$_1.2$.hydraulicCircuitPressurized has already been proven in (R41-1-Out-Open-1).

(R41-1-Out-Open-4-1-1-4-1-1-2) The case of state extend$_2.0.2$.stoppedDoorsClosing has already been proven in (R41-1-Out-Open-2).

(R41-1-Out-Open-4-1-1-4-1-1-3) The case of state retract$_7$.doorsClosed has already been proven in (R41-1-Out-Open-3).

(R41-1-Out-Open-4-1-1-4-1-1-4) retract$_5.2$.stoppedKeepingDoorsOpen is the case considered in (R41-1-Out-Open-4). Thus, if all other subcases of (R41-1-
Out-Open-4) can be proven, also (R41-1-Out-Open-4-1-1-4-1-1-4) is proven (by recursion). Thus, having considered all four cases:

(R41-1-Out-Open-4-1-1-4-1-1) is proven if all the remaining cases of (R41-1-Out-Open-4) can be proven, due to case (R41-1-Out-Open-4-1-1-4-1-1-4). The three other possible cases have been proven.

(R41-1-Out-Open-4-1-1-4-1-2) extend\_3\_0\_2\_stoppedGearsRetracting can only be reached from extend\_3\_0\_1\_stopGearsRetracting, which in turn can only be reached from retract\_3\_1\_gearsRetracting, which in turn can be reached from one of the states retract\_2\_2\_doorsOpen, retract\_3\_0\_2\_stoppedGearsExtending, extend\_4\_gearsExtended, or extend\_3\_0\_2\_stoppedGearsRetracting. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-4-1-2-1) retract\_2\_2\_doorsOpen is the case considered in (R41-1-Out-Open-4-1-1). Thus, if all other subcases of (R41-1-Out-Open-4-1-1) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-1) is proven (by recursion).

(R41-1-Out-Open-4-1-1-4-1-2-2) retract\_3\_0\_2\_stoppedGearsExtending can only be reached from retract\_3\_0\_1\_stopGearsExtending, which in turn can only be reached from extend\_3\_1\_gearsExtending, which in turn can be reached from one of the states extend\_2\_2\_doorsOpen, extend\_3\_0\_2\_stoppedGearsRetracting, retract\_4\_gearsRetracted, or retract\_3\_0\_2\_stoppedGearsExtending. We consider all four cases separately.

(R41-1-Out-Open-4-1-1-4-1-2-2-1) The case of state extend\_2\_2\_doorsOpen has already been proven in (R41-1-Out-Open-4-1-1-4-1-1).

(R41-1-Out-Open-4-1-1-4-1-2-2-2) extend\_3\_0\_2\_stoppedGearsRetracting is the case considered in (R41-1-Out-Open-4-1-1-4-1-2). Thus, if all other subcases of (R41-1-Out-Open-4-1-1-4-1-2) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-2-2) is proven (by recursion).

(R41-1-Out-Open-4-1-1-4-1-2-2-3) retract\_4\_gearsRetracted is the case considered in (R41-1-Out-Open-4-1). Thus, if all other subcases of (R41-1-Out-Open-4-1) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-2-3) is proven (by recursion).

(R41-1-Out-Open-4-1-1-4-1-2-2-4) retract\_3\_0\_2\_stoppedGearsExtending is the case considered in (R41-1-Out-Open-4-1-1-4-1-2-2). Thus, if all other subcases of (R41-1-Out-Open-4-1-1-4-1-2-2) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-2-4) is proven (by recursion). Thus, having considered all four cases:

(R41-1-Out-Open-4-1-1-4-1-1-4-1-2-2) is proven as soon as all other subcases of and (R41-1-Out-Open-4-1) have been proven (subcases of (R41-1-Out-Open-4-1-1-4-1-2-2) being subcases of (R41-1-Out-Open-4-1) as well). (The first subcase has already been proven, while the last subcase is identical with (R41-1-Out-Open-4-1-1-4-1-2-2).)

(R41-1-Out-Open-4-1-1-4-1-2-3) extend\_4\_gearsExtended is the case considered in (R41-1-Out-Open-4-1-1-4-1-1). Thus, if all other subcases of (R41-1-Out-Open-4-1-1-4-1) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-3) is proven (by recursion).

(R41-1-Out-Open-4-1-1-4-1-2-4) extend\_3\_0\_2\_stoppedGearsRetracting is the case considered in (R41-1-Out-Open-4-1-1-4-1-2). Thus, if all other subcases of
(R41-1-Out-Open-4-1-1-4-1-2) can be proven, also (R41-1-Out-Open-4-1-1-4-1-2-4) is proven (by recursion). Thus, having considered all four cases:

- (R41-1-Out-Open-4-1-1-4-1-2) is proven as soon as all other subcases of (R41-1-Out-Open-4-1-1) have been proven.

- (R41-1-Out-Open-4-1-1-4-1-3) *retract*$_4$._gearsRetracted is the case considered in (R41-1-Out-Open-4-1). Thus, if all other subcases of (R41-1-Out-Open-4-1) can be proven, also (R41-1-Out-Open-4-1-1-4-1-3) is proven (by recursion).

- (R41-1-Out-Open-4-1-1-4-1-4) The case of state *retract*$_3$._0$._2$._stoppedGearsExtending has already been considered in (R41-1-Out-Open-4-1-1-4-1-2-2). Thus, having considered all four cases:

  - (R41-1-Out-Open-4-1-1-4-1-1) is proven as soon as all other subcases of (R41-1-Out-Open-4-1-1) have been proven.

- (R41-1-Out-Open-4-1-1-4-2) *retract*$_3$._2$._gearRetractionAborted can be reached from one of the states *retract*$_2$.doorsOpen, *retract*$_3$.0$._2$._stoppedGearsExtending, extend$_4$.gearsExtended, or extend$_3$.0$._2$._stoppedGearsRetracting. We consider all four cases separately.

- (R41-1-Out-Open-4-1-1-4-2-1) *retract*$_2$.doorsOpen is the case considered in (R41-1-Out-Open-4-1-1). Thus, if all other subcases of (R41-1-Out-Open-4-1-1) can be proven, also (R41-1-Out-Open-4-1-1-4-2-1) is proven (by recursion).

- (R41-1-Out-Open-4-1-1-4-2-2) The case of *retract*$_3$.0$._2$._stoppedGearsExtending has already been considered in (R41-1-Out-Open-4-1-1-4-1-2-2).

- (R41-1-Out-Open-4-1-1-4-2-3) The case of extend$_4$.gearsExtended has already been considered in (R41-1-Out-Open-4-1-1-4-1).

- (R41-1-Out-Open-4-1-1-4-2-4) extend$_3$.0$._2$._stoppedGearsRetracting has already been considered in (R41-1-Out-Open-4-1-1-4-1-2). Thus, having considered all four cases,

  - (R41-1-Out-Open-4-1-1-4-2) is proven as soon as all other subcases of (R41-1-Out-Open-4-1-1) have been proven.

- (R41-1-Out-Open-4-1-1-4-3) *retract*$_2$.doorsOpen is the case considered in (R41-1-Out-Open-4-1-1). Thus, if all other subcases of (R41-1-Out-Open-4-1-1) can be proven, also (R41-1-Out-Open-4-1-1-4-3) is proven (by recursion).

- (R41-1-Out-Open-4-1-1-4-4) The case of state *retract*$_2$.doorsOpening is also considered in (R41-1-Out-Open-4-1-1). Thus, if all other subcases of (R41-1-Out-Open-4-1-1) can be proven, also (R41-1-Out-Open-4-1-1-4-4) is proven (by recursion). Thus, having considered all four cases:

  - (R41-1-Out-Open-4-1-1-4) is proven as soon as all other subcases of (R41-1-Out-Open-4-1) have been proven. Thus, having considered all four cases:

  - (R41-1-Out-Open-4-1-1-4-1) is proven as soon as all other subcases of (R41-1-Out-Open-4-1-1) have been proven.

- (R41-1-Out-Open-4-1-1-2) The case of state *retract*$_3$.0$._2$._stoppedGearsExtending has already been considered in (R41-1-Out-Open-4-1-1-4-1-2-2).

- (R41-1-Out-Open-4-1-1-3) The case of state extend$_4$.gearsExtended has already been considered in (R41-1-Out-Open-4-1-1-4-1).
(R41-1-Out-Open-4-1-4) The case of state $\text{extend}_3.0.2\_\text{stopped}GearsRetracting$ has already been considered in (R41-1-Out-Open-4-1-1-4-1-2). Thus, having considered all four cases:

(R41-1-Out-Open-4-1) is proven.
(R41-1-Out-Open-4-2) The case of state $\text{retract}_3.2\_\text{gearRetractionAborted}$ has already been proven in (R41-1-Out-Open-4-1-1-4-2).
(R41-1-Out-Open-4-3) The case of state $\text{extend}_3.2\_\text{doorsOpen}$ has already been proven in (R41-1-Out-Open-4-1-1-4-1-1).
(R41-1-Out-Open-4-4) The case of state $\text{extend}_2.1\_\text{doorsOpening}$ has already been considered in the course of (R41-1-Out-Open-4-1-1-4-1-1). Thus, having considered all four cases:

(R41-1-Out-Open-4) is proven. Thus, having considered all four cases:

(R41-1-Out-Open) is proven. This means that $\text{open}_\text{EV}$ is never stimulated when $\text{close}_\text{EV}$ is true within the outgoing sequence.

We now show that whenever $\text{close}_\text{EV}$ is stimulated, then $\text{open}_\text{EV}$ is false within the outgoing sequence.

(R41-1-Out-Close) The closure doors electro-valve ($\text{close}_\text{EV}$) is only stimulated in the subrule $\text{CloseDoors}$. In $\text{OutgoingSequence}$, $\text{CloseDoors}$ is only called in one of the states $\text{extend}_5.2\_\text{stoppedKeepingDoorsOpen}$ or $\text{retract}_2.0.2\_\text{stoppedDoorsClosing}$. We consider both cases independently.

(R41-1-Out-Close-1) $\text{extend}_5.2\_\text{stoppedKeepingDoorsOpen}$ can only be reached from $\text{extend}_5.1\_\text{stopKeepingDoorsOpen}$, which in turn can only be reached in a step in which the subrule $\text{StopKeepingDoorsOpen}$ is called. In $\text{StopKeepingDoorsOpen}$, $\text{open}_\text{EV}$ is set to false. Thus, in this case, when $\text{close}_\text{EV}$ is stimulated, $\text{open}_\text{EV}$ is false. So (R41-1-Out-Close-1) is proven.

(R41-1-Out-Close-2) $\text{retract}_2.0.2\_\text{stoppedDoorsClosing}$ can only be reached from $\text{retract}_2.0.1\_\text{stopDoorsClosing}$, which in turn can only be reached from $\text{extend}_6.\text{doorsClosing}$, which in turn can be reached from one of the states $\text{extend}_5.2\_\text{stoppedKeepingDoorsOpen}$ or $\text{retract}_2.0.2\_\text{stoppedDoorsClosing}$. We consider both cases independently.

(R41-1-Out-Close-2-1) The case of state $\text{extend}_5.2\_\text{stoppedKeepingDoorsOpen}$ has already been proven in (R41-1-Out-Close-1).

(R41-1-Out-Close-2-2) $\text{retract}_2.0.2\_\text{stoppedDoorsClosing}$ is the case considered in (R41-1-Out-Close-2). Thus, as the only other case of (R41-1-Out-Close-2) has already been proven, also (R41-1-Out-Close-2-2) is proven (by recursion). Thus, having considered both cases:

(R41-1-Out-Close-2) is proven. Thus, having considered both cases:

(R41-1-Out-Close) is proven. This means that $\text{close}_\text{EV}$ is never stimulated when $\text{open}_\text{EV}$ is true within the outgoing sequence. Thus, having considered both cases:

(R41-1-Out) is proven. This means that opening and closure doors electro-valves are not stimulated simultaneously within the outgoing sequence.

We now consider the retraction sequence.

(R41-1-Ret) We first show that whenever $\text{open}_\text{EV}$ is stimulated, then $\text{close}_\text{EV}$ is false in $\text{RetractionSequence}$ (R41-1-Ret-Open). Then we show that when-
ever close_{EV} is stimulated, then open_{EV} is \textbf{false} in RetractionSequence (R41-1-Ret-Close).

(R41-1-Ret-Open) We show that whenever open_{EV} is stimulated, then close_{EV} is \textbf{false} within the retraction sequence. open_{EV} is only stimulated in the subrule \texttt{OpenDoors}, which is only called in one of the states \texttt{retract\_1\_2\_hydraulicCircuitPressurized}, \texttt{retract\_2\_0\_2\_stoppedDoorsClosing}, \texttt{extend\_7\_doorsClosed}, or \texttt{extend\_5\_2\_stoppedKeepingDoorsOpen}. We consider all four cases separately. Thereby we can re-use cases already proven in (R41-1-Out-Open), because there the problem was the same, just that we considered the call of \texttt{OpenDoors} in OutgoingSequence, though we also encountered many states of the retraction sequence there.

(R41-1-Ret-Open-1) The case of state \texttt{retract\_1\_2\_hydraulicCircuitPressurized} was already proven in (R41-1-Out-Open-4-1-1-1).
(R41-1-Ret-Open-2) The case of state \texttt{retract\_2\_0\_2\_stoppedDoorsClosing} was already proven in (R41-1-Out-Open-4-1-1-2).
(R41-1-Ret-Open-3) The case of state \texttt{extend\_7\_doorsClosed} was already proven in (R41-1-Out-Open-4-1-1-3).
(R41-1-Ret-Open-4) The case of state \texttt{extend\_5\_2\_stoppedKeepingDoorsOpen} was already proven in (R41-1-Out-Open-4-1-1-4). Thus, having considered all four cases:
(R41-1-Ret-Open) is proven. This means that open_{EV} is never stimulated when close_{EV} is \textbf{true} within the retraction sequence.

(R41-1-Ret-Close) We now show that whenever close_{EV} is stimulated, then open_{EV} is \textbf{false} within the retraction sequence. close_{EV} is only stimulated in the subrule \texttt{CloseDoors}, which is only called in one of the states \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen} or \texttt{extend\_2\_0\_2\_stoppedDoorsClosing}. We consider both cases separately.

(R41-1-Ret-Close-1) \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen} can only be reached from \texttt{extend\_5\_1\_stopKeepingDoorsOpen}, which in turn can only be reached in a step in which the subrule \texttt{StopKeepingDoorsOpen} is called. In \texttt{StopKeepingDoorsOpen}, open_{EV} is set to \textbf{false}. Thus, in this case, when close_{EV} is stimulated, open_{EV} is \textbf{false}. So (R41-1-Ret-Close-1) is proven.
(R41-1-Ret-Close-2) \texttt{extend\_2\_0\_2\_stoppedDoorsClosing} can only be reached from \texttt{extend\_2\_0\_1\_stopDoorsClosing}, which in turn can only be reached from \texttt{retract\_6\_doorsClosing}, which in turn can be reached from one of the states \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen} or \texttt{extend\_2\_0\_2\_stoppedDoorsClosing}. We consider both cases separately.
(R41-1-Ret-Close-2-1) The case of \texttt{retract\_5\_2\_stoppedKeepingDoorsOpen} has already been proven in (R41-1-Ret-Close-1).
(R41-1-Ret-Close-2-2) \texttt{extend\_2\_0\_2\_stoppedDoorsClosing} is the state considered in (R41-1-Ret-Close-2). Thus, as the only other case of (R41-1-Ret-Close-2) has already been proven, also (R41-1-Ret-Close-2-2) is proven (by recursion). Thus, having considered both cases:
(R41-1-Ret-Close-2) is proven. Thus, having considered both cases:
(R41-1-Ret-Close) is proven. This means that close\textsubscript{EV} is never stimulated when open\textsubscript{EV} is true within the retraction sequence. Thus, having considered both cases,

(R41-1-Ret) is proven. This means that opening and closure doors electro-valves are not stimulated simultaneously within the retraction sequence. Thus, having considered both cases,

(R41-1) is proven. This means that opening and closure doors electro-valves are never stimulated simultaneously.

(R41-2) We now prove that outgoing and retraction gears electro-valves are not stimulated simultaneously. This means that extend\textsubscript{EV} and retract\textsubscript{EV} are never true at the same time. We start with the outgoing sequence (R41-2-Out) and then prove the property for the retraction sequence (R41-2-Ret).

(R41-2-Out) In OutgoingSequence, we only have to show that whenever extend\textsubscript{EV} is stimulated, then retract\textsubscript{EV} is false, because retract\textsubscript{EV} is never stimulated within OutgoingSequence. extend\textsubscript{EV} is only stimulated in the subrule ExtendGear, which is only called in one of the states extend\textsubscript{2}_2_doorsOpen, extend\textsubscript{3}_0_2_stoppedGearsRetracting, retract\textsubscript{4}_gearsRetracted, or retract\textsubscript{3}_0_2_stoppedGearsExtending. We consider all four cases separately.

(R41-2-Out-1) extend\textsubscript{2}_2_doorsOpen can only be reached from extend\textsubscript{2}_1_doorsOpening, which in turn can be reached from one of the states extend\textsubscript{1}_2_hydraulicCircuitPressurized, extend\textsubscript{2}_0_2_stoppedDoorsClosing, retract\textsubscript{7}_doorsClosed, or retract\textsubscript{5}_2_stoppedKeepingDoorsOpen. We consider all four cases separately.

(R41-2-Out-1-1) extend\textsubscript{1}_2_hydraulicCircuitPressurized can only be reached from extend\textsubscript{1}_1_generalEValveClosing, in which state we wait for the general electro-valve to close. Before the general electro-valve has not closed, no other electro-valve can be successfully stimulated nor is one stimulated. Thus, in particular, also retract\textsubscript{EV} cannot be and is not stimulated. So (R41-2-Out-1-1) is proven.

(R41-2-Out-1-2) extend\textsubscript{2}_0_2_stoppedDoorsClosing can only be reached from extend\textsubscript{2}_0_1_stopDoorsClosing, which in turn can only be reached from retract\textsubscript{6}_doorsClosing, which in turn can be reached from one of the states retract\textsubscript{5}_2_stoppedKeepingDoorsOpen or extend\textsubscript{2}_0_2_stoppedDoorsClosing. We consider both cases separately.

(R41-2-Out-1-2-1) retract\textsubscript{5}_2_stoppedKeepingDoorsOpen can only be reached from retract\textsubscript{5}_1_stopKeepingDoorsOpen, which in turn can be reached from one of the states retract\textsubscript{4}_gearsRetracted, retract\textsubscript{3}_2_gearRetractionAborted, extend\textsubscript{2}_2_doorsOpen, or extend\textsubscript{2}_1_doorsOpening. We consider all four cases separately.

(R41-2-Out-1-2-1-1) retract\textsubscript{4}_gearsRetracted can only be reached in a step in which retract\textsubscript{EV} is explicitly set to false. Thus, in this case, when extend\textsubscript{EV} is stimulated, retract\textsubscript{EV} is false. So (R41-2-Out-1-2-1-1) is proven.

(R41-2-Out-1-2-1-2) retract\textsubscript{3}_2_gearRetractionAborted can be reached from one of the states retract\textsubscript{2}_2_doorsOpen, retract\textsubscript{3}_0_2_stoppedGearsExtending,
extend_4_gearsExtended, or extend_3_0_2_stoppedGearsRetracting. We consider all four cases separately.

(R41-2-Out-1-2-1-2-1) retract_2_2_doorsOpen can only be reached from retract_2_1_doorsOpening, which in turn can be reached from one of the states retract_1_2_hydraulicCircuitPressurized, retract_2_0_2_stoppedDoorsClosing, extend_7_doorsClosed, or extend_5_2_stoppedKeepingDoorsOpen. We consider all four cases separately.

(R41-2-Out-1-2-1-2-1-1) retract_1_2_hydraulicCircuitPressurized can only be reached from retract_1_1_generalEValveClosing, in which state we wait for the general electro-valve to close. Before the general electro-valve has not closed, no other electro-valve can be successfully stimulated nor is one stimulated. Thus, in particular, also retract_EV cannot be and is not stimulated. So (R41-2-Out-1-2-1-2-1-1) is proven.

(R41-2-Out-1-2-1-2-1-2) retract_2_0_2_stoppedDoorsClosing can only be reached from retract_2_0_1_stopDoorsClosing, which in turn can only be reached from extend_6_doorsClosing, which in turn can be reached from one of the states extend_5_2_stoppedKeepingDoorsOpen or retract_2_0_2_stoppedDoorsClosing. We consider both cases separately.

(R41-2-Out-1-2-1-2-1-2-1) extend_5_2_stoppedKeepingDoorsOpen can only be reached from extend_5_1_stopKeepingDoorsOpen, which in turn can be reached from one of the states extend_4_gearsExtended, retract_3_2_gearRetractionAborted, retract_2_2_doorsOpen, or retract_2_1_doorsOpening. We consider all four cases separately.

(R41-2-Out-1-2-1-2-1-2-1-1) extend_4_gearsExtended can only be reached from extend_3_1_gearsExtending, which in turn is only reached by the step in which ExtendGear is called, which is the case considered in (R41-2-Out). Thus, if all other subcases of (R41-2-Out) can be proven, also (R41-2-Out-1-2-1-2-1-1-1) is proven (by recursion).

(R41-2-Out-1-2-1-2-1-2-1-2) retract_3_2_gearRetractionAborted is the case considered in (R41-2-Out-1-2-1-2-1-2). Thus, if all other subcases of (R41-2-Out-1-2-1-2-1-2) can be proven, also (R41-2-Out-1-2-1-2-1-2-1-2) is proven (by recursion).

(R41-2-Out-1-2-1-2-1-2-1-3) retract_2_2_doorsOpen is the case considered in (R41-2-Out-1-2-1-2-1-2-1). Thus, if all other subcases of (R41-2-Out-1-2-1-2-1-2-1) can be proven, also (R41-2-Out-1-2-1-2-1-2-1-2-1-3) is proven (by recursion).

(R41-2-Out-1-2-1-2-1-2-1-4) The case of state retract_2_1_doorsOpening is also considered in (R41-2-Out-1-2-1-2-1). Thus, if all other subcases of (R41-2-Out-1-2-1-2-1-2-1-4) can be proven, also (R41-2-Out-1-2-1-2-1-2-1-2-1-4) is proven (by recursion). Thus, having considered all four cases:

(R41-2-Out-1-2-1-2-1-2-1-1) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-1-2-2) retract_2_0_2_stoppedDoorsClosing is the case considered in (R41-2-Out-1-2-1-2-1-2-2). Thus, if the only other subcase of (R41-2-Out-1-2-1-2-1-2-2) can be proven, also (R41-2-Out-1-2-1-2-1-2-1-2-2) is proven (by recursion). Thus, having considered both cases:
(R41-2-Out-1-2-1-2-1-2) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-1-3) extend\_doorsClosed can only be reached from extend\_doorsClosing, which in turn can be reached from one of the states extend\_stoppedKeepingDoorsOpen or retract\_stoppedDoorsClosing. We consider both cases separately.

(R41-2-Out-1-2-1-2-1-3-1) The case of state extend\_stoppedKeepingDoorsOpen has already been considered in (R41-2-Out-1-2-1-2-1-2-1). It is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-1-3-2) The case of state retract\_stoppedDoorsClosing has already been considered in (R41-2-Out-1-2-1-2-1-2). It is proven if all other subcases of (R41-2-Out) can be proven. Thus, having considered both cases:

(R41-2-Out-1-2-1-2-1-3) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-1-4) extend\_stoppedKeepingDoorsOpen has already been considered in (R41-2-Out-1-2-1-2-1-2-1). It is proven if all other subcases of (R41-2-Out) can be proven. Thus, having considered all four cases:

(R41-2-Out-1-2-1-2-1) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-2) retract\_stoppedGearsExtending can only be reached from retract\_stopGearsExtending, which in turn can only be reached from extend\_gearsExtending, which has already been considered in (R41-2-Out-1-2-1-2-1-1). This case is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-3) extend\_gearsExtended is the case considered in (R41-2-Out-1-2-1-2-1-2-1-1). It is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-2-4) extend\_stoppedGearsRetracting can only be reached from extend\_stopGearsRetracting, which in turn can only be reached in a step in which StopGearsRetracting is called. In the subrule StopGearsRetracting, retract\_EV is set to false. Thus, in this case, when extend\_EV is stimulated, then retract\_EV is false. So (R41-2-Out-1-2-1-2-4) is proven. Thus, having considered all four cases:

(R41-2-Out-1-2-1-2) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-1-3) extend\_doorsOpen is the case considered in (R41-2-Out-1). Thus, if all other subcases of (R41-2-Out-1) can be proven, also (R41-2-Out-1-2-1-3) is proven (by recursion).

(R41-2-Out-1-2-1-4) The case of state extend\_doorsOpening is also considered in (R41-2-Out-1). Thus, if all other subcases of (R41-2-Out-1) can be proven, also (R41-2-Out-1-2-1-3) is proven (by recursion). Thus, having considered all four cases:

(R41-2-Out-1-2-1) is proven if all other subcases of (R41-2-Out) can be proven.
(R41-2-Out-1-2-2) \( \text{extend}_{2.0.2}. \text{stoppedDoorsClosing} \) is the case considered in (R41-2-Out-1-2). Thus, if all other subcases of (R41-2-Out-1-2) can be proven, also (R41-2-Out-1-2-2) is proven (by recursion), which is the case if all other subcases of (R41-2-Out) can be proven. Thus, having considered both cases:

(R41-2-Out-1-2) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-3) \( \text{retract}_{7}. \text{doorsClosed} \) can only be reached from \( \text{retract}_{6}. \text{doorsClosing} \), which in turn can be reached from one of the states \( \text{retract}_{5.2}. \text{stoppedKeepingDoorsOpen} \) or \( \text{extend}_{2.0}. \text{stoppedDoorsClosing} \). We consider both cases separately.

(R41-2-Out-1-2-2-1) The case of state \( \text{retract}_{5.2}. \text{stoppedKeepingDoorsOpen} \) has already been considered in (R41-2-Out-1-2-2). It is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-2-2-2) The case of state \( \text{extend}_{2.0}. \text{stoppedDoorsClosing} \) has already been considered in (R41-2-Out-1-2). It is proven if all other subcases of (R41-2-Out) can be proven. Thus, having considered both cases:

(R41-2-Out-1-2) is proven if all other subcases of (R41-2-Out) can be proven.

(R41-2-Out-1-3) \( \text{extend}_{3.0}. \text{stoppedGearsRetracting} \) has already been proven in (R41-2-Out-1-2-1-2-4).

(R41-2-Out-2) The case of state \( \text{retract}_{3}. \text{stoppedGearsExtending} \) has already been considered in (R41-2-Out-1-2-1-2-2). It is proven if all other subcases of (R41-2-Out) can be proven, which has now been achieved. Thus, having considered all four cases:

(R41-2-Out) is proven. This means that whenever \( \text{extend}. \text{EV} \) is stimulated, then \( \text{retract}. \text{EV} \) is false.

(R41-2-Ret) In RetractionSequence, we only have to show that whenever \( \text{retract}. \text{EV} \) is stimulated, then \( \text{extend}. \text{EV} \) is false, because \( \text{extend}. \text{EV} \) is never stimulated within RetractionSequence. \( \text{retract}. \text{EV} \) is only stimulated in the subrule RetractGears, which is only called in one of the states \( \text{retract}_{2}. \text{doorsOpen} \), \( \text{retract}_{3}. \text{stoppedGearsExtending} \), \( \text{extend}. \text{gearsExtended} \), or \( \text{extend}_{3.0}. \text{stoppedGearsRetracting} \). We consider all four cases separately.

(R41-2-Ret-1) \( \text{retract}_{2}. \text{doorsOpen} \) can only be reached from \( \text{retract}_{2.1}. \text{doorsOpening} \), which in turn can be reached from one of the states \( \text{retract}_{1.2}. \text{hydraulicCircuitPressurized} \), \( \text{retract}_{2}. \text{stoppedDoorsClosing} \), \( \text{extend}. \text{doorsClosed} \), or \( \text{extend}_{3.2}. \text{stoppedKeepingDoorsOpen} \). We consider all four cases separately.

(R41-2-Ret-1-1) \( \text{retract}_{1}. \text{hydraulicCircuitPressurized} \) can only be reached from \( \text{retract}_{1.1}. \text{generalEVValveClosing} \), in which state we wait for the general electro-valve to close. Before the general electro-valve has not closed, no other electro-valve can be successfully stimulated nor one stimulated. Thus, no other
electro-valve can be successfully stimulated nor is one stimulated in this state. Thus, in particular, also extend\_EV cannot be and is not stimulated. So case (R41-2-Ret-1-1) is proven.

(R41-2-Ret-1-2) retract\_2\_0\_2\_stoppedDoorsClosing can only be reached from retract\_2\_0\_1\_stopDoorsClosing, which in turn can only be reached from extend\_6\_doorsClosing, which in turn can be reached from one of the states extend\_5\_2\_stoppedKeepingDoorsOpen or retract\_2\_0\_2\_stoppedDoorsClosing. We consider both cases separately.

(R41-2-Ret-1-2-1) extend\_5\_2\_stoppedKeepingDoorsOpen can only be reached from extend\_5\_1\_stopKeepingDoorsOpen, which in turn can be reached from one of the states extend\_4\_gearsExended, retract\_3\_2\_gearRetractionAborted, retract\_2\_2\_doorsOpen, or retract\_2\_1\_doorsOpening. We consider all four cases separately.

(R41-2-Ret-1-2-1-1) extend\_4\_gearsExended can only be reached in a step in which extend\_EV is explicitly set to \textbf{false}. Thus, in this case, when retract\_EV is stimulated, extend\_EV is \textbf{false}. So case (R41-2-Ret-1-2-1-1) is proven.

(R41-2-Ret-1-2-1-2) retract\_3\_2\_gearRetractionAborted can only be reached in the step in which RetractGears is called (under certain conditions), which is the case considered in (R41-2-Ret). Note that this argument does not depend on whether RetractGears is actually called or not (in this case, it is actually not called). Thus, if all other subcases of (R41-2-Ret) can be proven, then also (R41-2-Ret-1-2-1-2) is proven (by recursion).

(R41-2-Ret-1-2-1-3) retract\_2\_2\_doorsOpen is the case considered in (R41-2-Ret-1). Thus, if all other subcases of (R41-2-Ret-1) can be proven, also (R41-2-Ret-1-2-1-3) is proven (by recursion).

(R41-2-Ret-1-2-1-4) The case of state retract\_2\_1\_doorsOpening is also considered in (R41-2-Ret-1). Thus, if all other subcases of (R41-2-Ret-1-2) can be proven, also (R41-2-Ret-1-2-1-4) is proven (by recursion). Thus, having considered all four cases:

(R41-2-Ret-1-2-1) is proven if all other subcases of (R41-2-Ret) can be proven.

(R41-2-Ret-1-2-2) retract\_2\_0\_2\_stoppedDoorsClosing is the case considered in (R41-2-Ret-1-2). Thus, if all other subcases of (R41-2-Ret-1-2) can be proven, also (R41-2-Ret-1-2-2) is proven, which is the case if all other subcases of (R41-2-Ret) can be proven. Thus, having considered all four cases:

(R41-2-Ret-1-2) is proven if all other subcases of (R41-2-Ret) can be proven.

(R41-2-Ret-1-3) extend\_7\_doorsClosed can only be reached from extend\_6\_doorsClosing, which has already been considered in (R41-2-Ret-1-2): it is proven if all other subcases of (R41-2-Ret) can be proven.

(R41-2-Ret-1-4) The case of state extend\_5\_2\_stoppedKeepingDoorsOpen has already been considered in (R41-2-Ret-1-2-1): it is proven if all other subcases of (R41-2-Ret) can be proven. Thus, having considered all four cases:

(R41-2-Ret-1) is proven if all other subcases of (R41-2-Ret) can be proven.

(R41-2-Ret-2) retract\_3\_0\_2\_stoppedGearsExtending can only be reached from retract\_3\_0\_1\_stopGearsExtending, which in turn can only be reached in a step in which StopGearsExtending is called. In the subrule StopGearsExtending, ex-
$tend_{EV}$ is set to false. Thus, in this case, when $retract_{EV}$ is stimulated, $extend_{EV}$ is false. So case (R41-2-Ret-2) is proven.

(R41-2-Ret-3) $extend_{4\_gearsExtended}$ can only be reached in a step in which $extend_{EV}$ is explicitly set to false. Thus, in this case, when $retract_{EV}$ is stimulated, $extend_{EV}$ is false. So case (R41-2-Ret-3) is proven.

(R41-2-Ret-4) $extend_{3\_0\_2\_stoppedGearsRetracting}$ can only be reached from $extend_{3\_0\_1\_stopGearsRetracting}$, which in turn can only be reached from $retract_{3\_1\_gearsRetracting}$, which is only reached in the step in which $RetractGears$ is called, which is the case considered in (R41-2-Ret). Thus, as all other subcases of (R41-2-Ret) have already been proven, also (R41-2-Ret-4) is proven (by recursion). Thus, having considered all four cases:

(R41-2-Ret) is proven. This means that whenever $retract_{EV}$ is stimulated, then $extend_{EV}$ is false. Thus, having considered both the outgoing and the retraction sequence:

(R41-2) is proven. This means that outgoing and retraction gears electro-valves are never stimulated simultaneously.

(R41) is proven by (R41-1), where we show that opening and closure doors electro-valves are never stimulated simultaneously, and (R41-2), where we show that outgoing and retraction gears electro-valves are never stimulated simultaneously.

4.2 Failure Mode Requirements

(R61) We prove that (R61) is fulfilled by the given ASM model.

We assume that the handle position is not changed during the opening of the doors (although this is not explicitly stated, but else the requirement could obviously not be met).

$MonitorSystem$, and consequently also $CheckDoorsMotion$, are called in every iteration without any preconditions, for each $moduleName$.

The opening electro-valve is stimulated in the rule $OpenDoors$, which can be called from both $OutgoingSequence$ and $RetractionSequence$. In $OpenDoors$, $startTime(moduleNumber, open\_EV, true)$ is set to $now$, i.e. the time at which the iteration (step of the ASM) takes place. This is done in parallel with stimulating the opening electro-valve.

startTime(moduleNumber, open\_EV, true) is reset to $undef$ in both $OutgoingSequence$ and $RetractionSequence$ with the guard, if $DoorsAreOpen(moduleNumber)$.

startTime(moduleNumber, open\_EV, true) is also reset to $undef$ in $StopKeepingDoorsOpen$, but this is only (redundantly) called after the doors were actually open, so this is irrelevant in this place. startTime(moduleNumber, open\_EV, true) is not reset to $undef$ in any other place.

$DoorsAreOpen(moduleNumber)$ checks the sensor $door\_open$ for each landing set, i.e. it becomes true when all doors are locked open.

The following lemma will be used in a later proof as well. As a consequence of what we have stated so far,
(Lemma R61-1) \( \text{startTime}(\text{moduleNumber}, \text{open}_E \text{V}, \text{true}) \neq \text{undef} \) from the time of the stimulation of \( \text{open}_E \text{V} \) to the time when all doors are seen locked open.

In \textit{CheckDoorsMotion}, when \( \text{startTime}(\text{moduleNumber}, \text{open}_E \text{V}, \text{true}) \neq \text{undef} \), we have two cases:

– either \( \text{NoDoorIsClosed}(\text{moduleNumber}) \) is false, i.e. at least one door is still locked closed (\( \text{NoDoorIsClosed} \) checks whether \( \text{door\_closed} \) is false for each landing set);
– or \( \text{NoDoorIsClosed}(\text{moduleNumber}) \) is true; this case is irrelevant for \( \text{(R61)} \).

Now, in \textit{CheckDoorsMotion},

– If \( \text{NoDoorIsClosed}(\text{moduleNumber}) \) is false and
– the difference between
  – \( \text{now} \) (i.e. the time of the check) and
  – \( \text{startTime}(\text{moduleNumber}, \text{open}_E \text{V}, \text{true}) \) (i.e. the time when \( \text{open}_E \text{V} \) was last stimulated)

is greater than or equal to \( \text{maxStartDoorsOpeningTime} = 500 \text{ milliseconds} \) (see \textit{Signature / Constant functions}),

then \( \text{anomaly}(\text{moduleNumber}) \) is set to \text{true}. As mentioned in our section on \textit{Specification Issues}, we assume that this is equivalent to \( \text{normal\_mode} = \text{false} \) (note that \( \text{normal\_mode} \) is not mentioned in the specification of outputs on p. 15, nor on pp. 16–17, of the requirements document).

Consequently, if one of the three doors is still seen locked in the closed position more than 0.5 seconds (= 500ms) after stimulating the opening electro-valve, then an anomaly is detected, which was to be proven (as we claim).

\textbf{(R71)} We prove that \( \text{(R71)} \) is fulfilled by the given ASM model.

We follow the proof of \( \text{(R61)} \) up to and including (Lemma R61-1).

In this case, we have to consider three cases:

– either \( \text{NoDoorIsClosed}(\text{moduleNumber}) \) is false, i.e. at least one door is still closed, then obviously the doors cannot all be open;
– or \( \text{NoDoorIsClosed}(\text{moduleNumber}) \) is true but \( \text{DoorsAreOpen}(\text{moduleNumber}) \) is false, i.e. at least one door is not open;
– or \( \text{DoorsAreOpen}(\text{moduleNumber}) \) is true, this case is irrelevant for \( \text{(R71)} \).

In the first case, as shown in the proof of \( \text{(R61)} \), an anomaly will already be detected after 0.5 seconds. An anomaly cannot be reset, it is “maintained forever” (p. 17). Thus setting \( \text{anomaly} \) to true once again is irrelevant.

So all that remains to be considered is the second case. In \textit{CheckDoorsMotion},

– If \( \text{DoorsAreOpen}(\text{moduleNumber}) \) is false and
– the difference between
• now (i.e. the time of the check) and
• \texttt{startTime(moduleNumber, open\_EV, true)} (i.e. the time when open\_EV was last stimulated)

is greater than or equal to \textit{maxDoorOpeningTime} = 2000 milliseconds,

then \texttt{anomaly(moduleNumber)} is set to \texttt{true}.

Consequently, if one of the three doors is not seen locked in the open position more than 2 seconds (= 2000ms) after stimulating the opening electro-valve, then an anomaly is detected, which was to be proven (as we claim).

5 Experiences

A summary of experiences gained by the case study will be published in the proceedings of the ABZ 2014 conference.

Acknowledgement. This publication has been written within the project “Vertical Model Integration”. The project Vertical Model Integration is supported within the program “Regionale Wettbewerbsfähigkeit OÖ 2007-2013” by the European Fund for Regional Development as well as the State of Upper Austria.

References