# Analysis and Verification of an Automatic Document Feeder

Bas Ploeger Lou Somers

Department of Mathematics and Computer Science
*Technische Universiteit Eindhoven*
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{S.C.W.Ploeger, L.J.A.M.Somers}@tue.nl

**Abstract**

Modern copying machines are versatile and complex systems in which embedded software plays an essential role. The progress towards faster and more stable machines that can satisfy ever growing customers' needs, places strict requirements on the efficiency and quality of such software. In order to meet these requirements, the software should be well-designed and this design should be free of errors. Using modern formal verification techniques, software designs can be checked for errors and deadlocks so that their quality can be assessed and improved at an early stage of the development process.

In this report, we analyze the embedded software of an Automatic Document Feeder (ADF). ADFs are important components of many copier machines. The ADF studied here is a prototype developed by Océ–Technologies B.V., a company that develops professional printing systems. We construct a model of the ADF in $\mu$CRL, a process algebra-based specification language, and express the system's requirements in the modal $\mu$-calculus. Next, we use the $\mu$CRL and CADP toolsets to check whether the system meets its requirements. This analysis reveals important errors in the ADF and we propose solutions to these problems. Also, we show that some requirements that engineers assumed to be valid, are too strict. We present slightly weaker versions of these requirements and show that these do hold. In this sense, in addition to finding errors in the ADF, our analysis also led to a better understanding of the behaviour the system.

## 1 Introduction

At the Research & Development (R&D) department of Océ–Technologies B.V. in Venlo, the Netherlands, various professional document systems are developed. Océ's product line ranges from wide format to cut sheet and from full colour to black-and-white printing systems. In all of these machines, embedded software is used to control the hardware and provide functionalities to the user. The progress towards faster and more stable machines that can satisfy ever growing customers' needs, places increasingly higher requirements on the efficiency and quality of that software. In order to meet these requirements, it is important that the software is well-designed and that this design is free of errors.

Typically, the embedded software is a distributed system: it consists of a number of components that are running in parallel. The behaviour of such systems tends to become very complex as more components are added. Consequently, trying to understand or describe all situations that may possibly occur in the system, in order to check whether errors can occur, becomes a formidable task.

In this report, we apply formal verification techniques to find errors in the embedded software of a copier machine. In particular, we study a prototype of an Automatic Document Feeder (ADF). The ADF automatically feeds sheets of paper to the scanner of the copying machine. It moves the sheets in its input tray over the scanner one at a time and places every sheet into an output tray after it has been scanned[1]. We construct a model of the ADF's embedded control software by specifying the behaviour of each of its components in a specification language called $\mu$CRL. Next, we formulate the system's requirements in a formal logic called the $\mu$-calculus and verify whether the specified system meets its requirements using model checking techniques.

The $\mu$CRL language and the $\mu$-calculus are introduced shortly in Section 2. In Section 3 we describe the ADF and we translate its behaviour to $\mu$CRL in Section 4. Section 5 contains the requirements that the ADF should meet, along with their translations to the $\mu$-calculus. We also report the results of model checking the requirements on the specification of Section 4 and propose solutions to the problems that are found. We conclude in Section 6.

## 2 Languages and tools

### 2.1 $\mu$CRL

For the specification of the ADF behaviour we use $\mu$CRL [7] which is based on the process algebra ACP [1], with extensions to support abstract data types. The language and its accompanying toolset [2] have been used successfully for the analysis of various real-life distributed systems and protocols [4, 6, 9, 11]. For a more thorough treatment of the language we refer to [7]. A $\mu$CRL specification mainly consists of two parts: a data type specification and a process specification.

The data type specification defines sorts (abstract data types) that can be used in the specification of the processes. Functions on sorts can be declared and their meanings can be defined using equations. For example, the following sort defines the natural numbers 0, $S(0)$, $S(S(0))$, …:

**sort** $\quad Nat$
**func** $\quad 0 :\to Nat$
$\qquad\quad S : Nat \to Nat$

The process specification defines the processes that model the behaviour of the system under scrutiny. Processes are defined using process terms that can be built from actions and other process terms using operators. Given processes $p$ and $q$, operators include sequential composition $p \cdot q$ (first $p$ then $q$), alternative composition $p+q$ (either $p$ or $q$), summation over a sort $D$ $\sum_{d:D} p(d)$ (that is $p(d_0)+\ldots+p(d_n)$ if $D=\{d_0,\ldots,d_n\}$) and conditional $p \lhd b \rhd q$ (if $b$ then $p$ else $q$). The action $\delta$ denotes deadlock. The parallel composition $p \parallel q$ of $p$ and $q$ denotes all possible interleavings of the action sequences of $p$ with those of $q$. In addition, actions from $p$ and $q$ may synchronize to a communication action if a communication rule has been defined for them. Actions may carry data parameters and two actions can only synchronize if all of their data parameters have the same values. To prevent the separate actions of a communication

[1]Note that it is not the responsibility of the ADF to actually scan the sheet.

action from occurring in isolation, they can be blocked using the encapsulation $\partial_H(p)$ which replaces every occurrence of an action from set $H$ in $p$ by $\delta$.

For example, the following is a specification of two buffers in sequence. Buffer $B_1$ repeatedly receives a natural number over channel 1 and sends it to buffer $B_2$ over channel 2. On receipt of a number over channel 2, $B_2$ sends the number over channel 3 to the outside world.

| | |
|---|---|
| **act** | $rcv_1, snd_2, rcv_2, cmm_2, snd_3 : Nat$ |
| **comm** | $snd_2 \mid rcv_2 = cmm_2$ |
| **proc** | $B_1 = \sum_{n:Nat} rcv_1(n) \cdot snd_2(n) \cdot B_1$ |
| | $B_2 = \sum_{n:Nat} rcv_2(n) \cdot snd_3(n) \cdot B_2$ |
| **init** | $\partial_{\{snd_2, rcv_2\}}(B_1 \parallel B_2)$ |

The $\mu$CRL toolset includes tools with which a labelled transition system (LTS) can be generated from a $\mu$CRL specification. An LTS contains all states that the modelled system can reach via any number of transitions, which correspond to actions in the $\mu$CRL specification. This LTS can be used for model checking of temporal logic formulas.

## 2.2 The $\mu$-calculus

For expressing the requirements of the system we use the $\mu$-calculus of Kozen [8], although we do not use propositional constants in our formulas. Given $\mu$-calculus formulas $f$ and $g$, action $a$ and propositional variable $X$, this logic includes:

- constants $\mathsf{T}$ (true) and $\mathsf{F}$ (false);
- boolean negation $\neg f$, disjunction $f \vee g$ and conjunction $f \wedge g$;
- diamond modality $\langle a \rangle f$ (there is an $a$-transition to a state in which $f$ holds) and box modality $[a]f$ ($f$ holds in every state that is reachable via an $a$-transition);
- least fixpoint $\mu X.f$ and greatest fixpoint $\nu X.f$ in which cases $f$ may contain free occurrences of $X$ and every occurrence of $X$ in $f$ falls under an even number of negations in $f$.

It is possible to check whether the system satisfies or violates its requirements using an automatic model checker. In this report, we use the EVALUATOR model checker of the CADP toolset [3, 5].

# 3 System Description

## 3.1 Hardware components

An overview of the ADF is given in Figure 1 (this figure is not to scale). The ADF consists of pinches (K1, K2, ...), a shoe pressure (S1), sensors (O1, O2, ...), clutches (C1, C2, ...), motors (M1, M2, ...), belts and trays. A pinch can hold a sheet of paper and can be connected to a motor. Pinch K1 is accompanied by shoe pressure S1 that can provide extra pressure when grabbing a sheet of paper. A sensor can be either covered or uncovered, which is indicated by its high or low electronic output signal, respectively. A clutch can be set to either on or off. If on, the motor on one side of the clutch is connected to the pinch on the other side. If off, the motor and the pinch are disconnected, meaning that no transmission between them is possible. There are two types of motor: motors that can run at one speed and in one direction only (M4 and M5) and motors that can run at various speeds and in two directions (M1, M2 and M3).

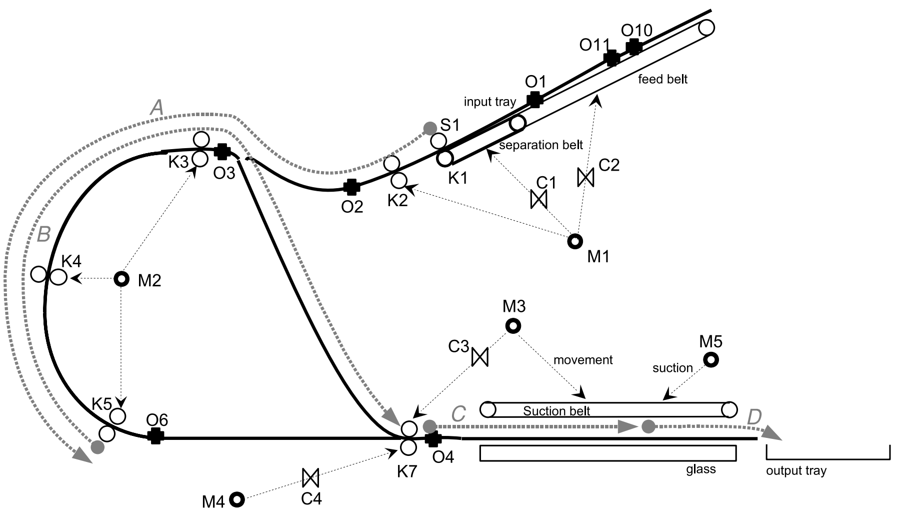


Figure 1: An overview of the ADF

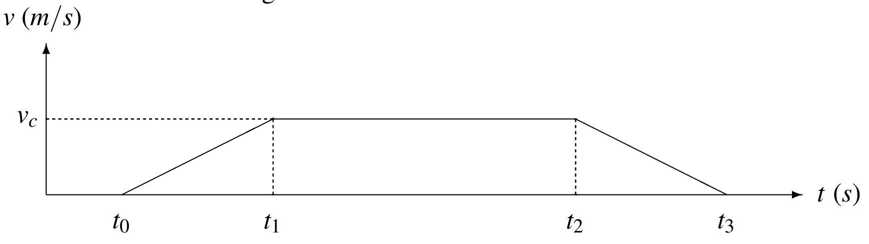


Figure 2: An example of a motor profile

Motors of the former type can be switched either on or off. Every motor of the latter type has a *controller* (indicated by suffix 'C') and an *encoder* (indicated by suffix 'E') attached to it. The encoder monitors the distance that the motor has covered thus far and can be asked to return this value. The controller can be in one of two states: Controlled and Idle. In the Controlled state the motor can receive instructions to execute *profiles*; in the Idle state it cannot. A profile (from here on indicated by X1, X2, ...) is a sequence of commands that instruct the motor to start/stop accelerating or decelerating at a certain rate. These commands are separated by specific time intervals. An example of a profile is depicted in Figure 2. Commands are given to the motor at time stamps $t_0$, $t_1$, $t_2$ and $t_3$ to accelerate, stop accelerating, decelerate and stop decelerating, respectively. Between $t_1$ and $t_2$ the motor runs at a constant speed $v_c$. The turn direction of the motor is indicated by the sign of the speed (positive or negative).

## 3.2 Embedded software

At any point in time, three instances of the control software are running in parallel. Every instance is responsible for guiding one sheet through the ADF. Hence, at most three sheets can be inside the ADF at the same time. Apart from the hardware components, the instances use a global boolean variable FirstSheet, semaphores sem1, sem2 and sem3 to ensure that some sections of the paper path contain at most one sheet at a time and a semaphore semN that is used to register the number of sheets inside the machine.

The behaviour of the instances is described in an *execution sheet*, which is a table of state–trigger–response combinations. Intuitively, the table specifies that if the control software is in a certain state and it receives a certain trigger, it responds by executing a series of actions, after which it goes to the next state which usually is the state of the next row of the table. The comment column is used to provide additional information and can specify that the control software should go to another state than the one on the next row. Unless indicated otherwise by the comments, the software traverses the execution sheet from top to bottom. The execution sheet is included in Appendix A. A global description of the behaviour is included in the next section.

### 3.2.1 Global description

**Initialization and aligning:** Every instance starts in a deactivated state, except for instance I0 which is already activated initially. If activated, the instance waits for a high signal from O1 (any low signals from this sensor are received but ignored). On receipt of a high signal, the instance continues with separation immediately if the FirstSheet flag is *false*. Otherwise, the set of sheets needs to be aligned and the instance waits for a fixed time interval. If O1 is still high after that interval, M5 is switched on, M1C, M2C and M3C are brought to the Controlled state and FirstSheet is set to *false*. The set of sheets is aligned by switching on C2 and S1, switching off C1 and executing profile X1 on motor M1. When M1 has finished executing the profile, the instance continues with separation.

**Separation:** The instance first reserves semaphore sem1 and increases semN by one, after which a short move to the right is executed on M1 (profile X2), followed by a move to the left (profile X3). This separates the sheet guided by the instance from the set. The sheet continues to move to the left until it covers O2. In the meanwhile, the instance monitors the value of M1E to determine when C1 should be switched on, S1 can be switched off and an error situation has occurred: if M1E exceeds some error value before O2 has become covered, M1 is stopped and the process halts.

**Turning and blousing:** The instance reserves semaphore sem2 and moves the sheet towards O3 and pinch K3: M1 and M2 start running at corresponding speeds and directions (profiles X4 and X5 respectively). An error check is performed to stop the motors at any time when O2 is covered, O3 is uncovered and M2E's value indicates that O3 should have been covered. When O3 becomes covered, the sheet is about to be moved by two motors at the same time: M1 and M2. When O2 becomes uncovered, M1 is stopped and the next instance is activated (the next sheet can be separated from the set in the input tray). The sheet proceeds until O3 becomes uncovered, upon which M2 is ordered to change its direction (profile X6). The sheet moves over O3 towards pinch K7. Every time the instance waits for O3 to become covered or uncovered, error checks are performed similar to those described earlier. As the sheet approaches K7, the speed of M2 is lowered (profile X7) and C4 and M4 are switched on, which makes K7 run in opposite direction to that of K3. This makes the sheet *blouse* (bulge and tremble) as soon as it hits K7. This ensures that the sheet is straightened before it is placed onto the scanner, as it may have become skewed on its way to K7. The instance stops the blousing process after a certain time period.

**Scanning and finalization:** The instance reserves semaphore sem3 and it moves the sheet towards O4 and the glass plate: M2 and M3 start running at corresponding speeds and directions (profiles X8 and X9 respectively). Error checks are performed to stop the motors at any time when:

- O3 is covered and M2E's value indicates that O3 should have been uncovered, or
- O4 is uncovered and M3E's value indicates that O4 should have been covered.

When O4 becomes covered (which happens before O3 becomes uncovered), we know that the sheet is being moved by two motors (M2 and M3). When O3 becomes uncovered, M2 is stopped and sem2 is released. When O4 becomes uncovered, the sheet is moved a bit further (so it is completely on top of the glass plate), after which M3 is stopped and the scanning process begins. Upon termination of the scanning process (modelled by a timeout that occurs after 500 ms), semN is decreased by one and sem3 is released. The instance now proceeds as follows, depending on the value of semN:

- If semN $> 0$, it does not take any further action: the sheet will be removed from the scanner by the next instance that executes profile X9 on M3.
- Otherwise (semN $= 0$), the sheet is moved to the right (profile X9 is executed on M3) in order to remove it from the ADF, after which M3 is stopped, FirstSheet is set to *true*, M5 is switched off and M1C, M2C and M3C are brought to the Idle state.

Finally, the instance returns to its initial state (deactivated), ready to guide another sheet through the ADF.

### 3.2.2 Constraints.

The following constraints and assumptions apply:

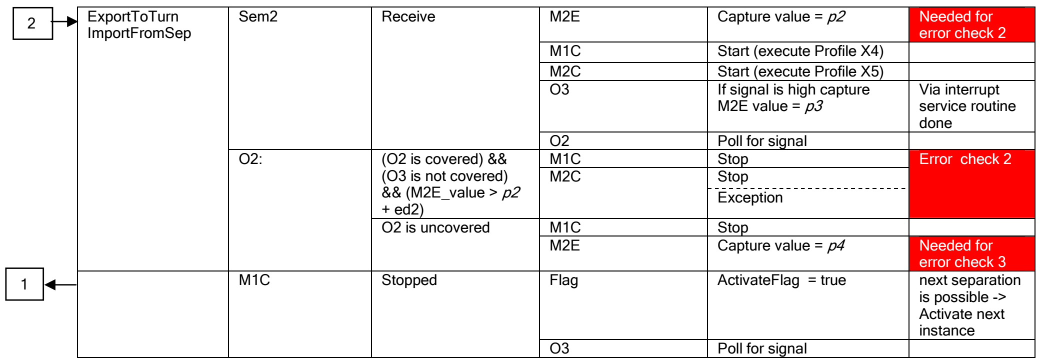
- The real-time behaviour is not taken into account. By this we mean that we only consider the sequences of events an actions that can occur and their relative order, not the time they take or the time in-between them.
- Only simplex (one-sided) scanning is analyzed.
- Measuring the sheets before they enter the ADF is not taken into account. (It is assumed that all sheets are of size A4.)

# 4 Specification

The $\mu$CRL model of the ADF system contains processes that model the global variables, semaphores and three instances of the control software. The system as a whole consists of the parallel composition of these processes. We do not translate the full execution sheet of Appendix A to $\mu$CRL here. Instead, we translate a part of the execution sheet (see Figure 3) to illustrate the general idea. The full $\mu$CRL specification is included in Appendix B.

## 4.1 Data types

**Booleans and natural numbers** We define a sort *Bool* representing booleans, along with the $\wedge$, $\vee$ and $=$ relations and a sort *Nat* representing natural numbers, along with the $>$ and $=$ relations and the *dec* function which decreases a natural number by 1.



| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | ExportToTurn ImportFromSep | Sem2 | Receive | M2E | Capture value = *p2* | 🟥 Needed for error check 2 |
| | | | | M1C | Start (execute Profile X4) | |
| | | | | M2C | Start (execute Profile X5) | |
| | | | | O3 | If signal is high capture M2E value = *p3* | Via interrupt service routine done |
| | | | | O2 | Poll for signal | |
| | | O2: | (O2 is covered) && (O3 is not covered) && (M2E_value > *p2* + ed2) | M1C | Stop | 🟥 Error check 2 |
| | | | | M2C | Stop | 🟥 |
| | | | | | Exception | 🟥 |
| | | | O2 is uncovered | M1C | Stop | |
| | | | | M2E | Capture value = *p4* | 🟥 Needed for error check 3 |
| 1 | | M1C | Stopped | Flag | ActivateFlag = true | next separation is possible -> Activate next instance |
| | | | | O3 | Poll for signal | |

Figure 3: Part of the execution sheet of Appendix A

**Identifiers** For every type of component and for the motor profiles, we introduce a sort that contains the names of the components of that type and of the profiles, respectively. As the shoe pressure, clutches and motors M4 and M5 are all on/off components, we introduce one sort for all of these components. The sorts are called *SensorID*, *OnOffID*, *ControllerID*, *EncoderID* and *ProfileID*.

The sorts containing the identifiers of the instances and semaphores (*InstanceID* and *SemaphoreID* respectively) are defined similarly, yet in combination with an equality relation $=$, because they are used in the types of some synchronizing actions. In addition, functions $next, prev : InstanceID \to InstanceID$ are defined as:

$$next(I0) = I1 \quad next(I1) = I2 \quad next(I2) = I0$$
$$prev(I0) = I2 \quad prev(I1) = I0 \quad prev(I2) = I1$$

Sorts *TimePeriod* and *Distance* contain the time periods and distances respectively. The specific values have been replaced by dummy identifiers in the execution sheet of Appendix A for confidentiality reasons and because the specific values are irrelevant to the analysis.

## 4.2 Processes

### 4.2.1 Global variables

The value of the FirstSheet boolean variable can be set and read. This behaviour is specified by the *FirstSheet* process as follows:

**act** $setFirstSheet, SetFirstSheet, getFirstSheet, GetFirstSheet : Bool \times InstanceID$
**comm** $setFirstSheet \mid setFirstSheet = SetFirstSheet$
$getFirstSheet \mid getFirstSheet = GetFirstSheet$

**proc** $FirstSheet(b : Bool) =$
$\sum_{i:InstanceID}(\sum_{c:Bool}(setFirstSheet(c,i) \cdot FirstSheet(c)) +$
$getFirstSheet(b,i) \cdot FirstSheet(b))$

A semaphore can be reserved, released and checked for zero-valuedness (needed for semN). The behaviour is specified in the *Semaphore* process:

**act** $\mathit{reserve}, \mathit{Reserve}, \mathit{release}, \mathit{Release} : \mathit{SemaphoreID} \times \mathit{InstanceID}$
$\mathit{semValueZero}, \mathit{SemValueZero} : \mathit{SemaphoreID} \times \mathit{Bool} \times \mathit{InstanceID}$

**comm** $\mathit{reserve} \mid \mathit{reserve} = \mathit{Reserve}$
$\mathit{release} \mid \mathit{release} = \mathit{Release}$
$\mathit{semValueZero} \mid \mathit{semValueZero} = \mathit{SemValueZero}$

**proc** $\mathit{Semaphore}(s : \mathit{SemaphoreID}, n : \mathit{Nat}) =$
$\sum_{i:\mathit{InstanceID}}(\mathit{reserve}(s,i) \cdot \mathit{Semaphore}(s, \mathit{dec}(n)) \lhd n > 0 \rhd \delta +$
$\mathit{release}(s,i) \cdot \mathit{Semaphore}(s, S(n)) +$
$\mathit{semValueZero}(s, n = 0, i) \cdot \mathit{Semaphore}(s,n))$

### 4.2.2 Instances

The behaviour of an instance is specified by 37 processes named *Instance*, *Instance0*, ..., *Instance35*. The following parts of the behaviour are modelled by the following $\mu$CRL processes:

- Initialization and aligning: *Instance*, *Instance0*, ..., *Instance3*;
- Separation: *Instance4*, ..., *Instance13*;
- Turning and blousing: *Instance14*, ..., *Instance23*;
- Scanning and finalization: *Instance24*, ..., *Instance35*.

In addition to the actions that correspond to actions or triggers in the execution sheet, we define the following actions that we will need for the verification of some of the requirements of Section 5:

**act** *LEAVE_K2, ENTER_K3, LEAVE_K3, ENTER_K7, LEAVE_K7, ENTER_M1, LEAVE_M1, ENTER_M2, LEAVE_M2, ENTER_M3, LEAVE_M3* : *InstanceID*

The actions represent the following events, for any $i$ : *InstanceID*:

*ENTER_Mx*($i$): $i$ enters the part of its behaviour in which it needs motor M$x$;

*LEAVE_Mx*($i$): $i$ leaves the part of its behaviour in which it needs motor M$x$;

*ENTER_Kx*($i$): $i$'s sheet enters pinch K$x$;

*LEAVE_Kx*($i$): $i$'s sheet leaves pinch K$x$.

In order to translate the execution sheet of Figure 3 to $\mu$CRL, we have to *interpret* the behaviour described in the execution sheet. This is because some parts of the control software behaviour may be described by comments or other constructs that are less suitable for translation by automatic procedures or algorithms. For the greater part, the translation is straightforward, but we have to pay special attention to actions such as the fourth action from the top of the execution sheet. This action says that as soon as sensor O3 becomes covered (its signal becomes high) the value of encoder M2E should be captured. All this is done "via an interrupt service routine", as noted by the comment. This means that while the control software is waiting for sensor O2 to become uncovered, it should also check whether O3 has become covered after which the value of M2E should be captured. Furthermore, we note that after O3 has become covered, the condition for "Error check 2" can no longer be *true* as the second conjunct will always be *false*. The execution sheet of Figure 3 is now modelled by processes *Instance14*, ..., *Instance17*:

**proc** $\mathit{Instance14}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{reserve}(\mathit{sem}2, \mathit{id}) \cdot \mathit{Enter}(M2C, \mathit{id}) \cdot \mathit{CaptureValue}(M2E, \mathit{id})$
$\quad \cdot \mathit{ExecuteProfile}(X4, M1C, \mathit{id}) \cdot \mathit{ExecuteProfile}(X5, M2C, \mathit{id})$
$\quad \cdot \mathit{Instance15}(\mathit{id})$

**proc** $\mathit{Instance15}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{SignalHigh}(O2, \mathit{id}) \cdot$
$\quad (\ \mathit{SignalHigh}(O3, \mathit{id}) \cdot \mathit{ENTER\_K3}(\mathit{id}) \cdot \mathit{CaptureValue}(M2E, \mathit{id})$
$\quad\ \cdot \mathit{Instance16}(\mathit{id}) +$
$\qquad \mathit{SignalLow}(O3, \mathit{id}) \cdot$
$\qquad (\mathit{NotTooHigh}(M2E, \mathit{id}) \cdot \mathit{Instance15}(\mathit{id})$
$\qquad + \mathit{TooHigh}(M2E, \mathit{id}) \cdot \mathit{Stop}(M1C, \mathit{id}) \cdot \mathit{Stop}(M2C, \mathit{id}) \cdot \delta))$
$\quad + \mathit{SignalHigh}(O3, \mathit{id}) \cdot \mathit{ENTER\_K3}(\mathit{id}) \cdot \mathit{CaptureValue}(M2E, \mathit{id})$
$\quad \cdot \mathit{Instance16}(\mathit{id})$

**proc** $\mathit{Instance16}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{SignalHigh}(O2, \mathit{id}) \cdot \mathit{Instance16}(\mathit{id}) +$
$\quad \mathit{SignalLow}(O2, \mathit{id}) \cdot \mathit{LEAVE\_K2}(\mathit{id}) \cdot \mathit{Stop}(M1C, \mathit{id})$
$\quad \cdot \mathit{CaptureValue}(M2E, \mathit{id}) \cdot \mathit{Instance17}(\mathit{id})$

**proc** $\mathit{Instance17}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{Stopped}(M1C, \mathit{id}) \cdot \mathit{Leave}(M1C, \mathit{id}) \cdot \mathit{activate}(\mathit{next}(\mathit{id}), \mathit{id})$
$\quad \cdot \mathit{release}(\mathit{sem}1, \mathit{id}) \cdot \mathit{Instance18}(\mathit{id})$

## 5 Verification

In this section we list the requirements that the ADF should meet and report on the verification of these requirements on the model discussed in Section 4. We propose solutions to any problems that are uncovered by the verification process, and show that these solutions indeed solve all problems.

### 5.1 Requirements

The following requirements have to be verified for the ADF. These requirements are made more explicit and formulated in the $\mu$-calculus in the next section.

R1: Whenever a sheet approaches the scanner, motor M5 is on.

R2: As long as a sheet is in pinch K7, motor M4 is off.

R3: Two motors transporting the same sheet at the same time are not in conflict.

R4: Every sheet that enters the ADF, eventually passes sensor O4.

R5: Clutches C3 and C4 are not on at the same time.

R6: A motor does not get conflicting commands from two instances.

R7: Every sensor is always uncovered between two sheets.

### 5.2 Model checking

Using the INSTANTIATOR tool of the $\mu$CRL toolset, an LTS was generated from the specification of Appendix B on a Pentium 4 (3 GHz) machine with 1 GB of RAM. Generation took about two minutes and produced an LTS of 358.153 states and 1.101.648 transitions. The requirements were translated to the $\mu$-calculus. These formulas were then checked on the LTS using the EVALUATOR tool of the CADP toolset. For each requirement we give a $\mu$-calculus formula that expresses it and comment on the result of model checking the requirement on the LTS.

**Notation** In the $\mu$-calculus formulas below, we allow ourselves to use sets of actions between the brackets of the diamond and box modality operators. For instance, for any set of actions $A$ and formula $\varphi$, $[A]\varphi$ denotes the property that $\varphi$ holds in every state that is reachable via an action that is in $A$.

Let $\mathsf{Act}$ denote the set of all parameterised actions, $a \in \mathsf{Act}$ and $A, B \subseteq \mathsf{Act}$. We shall use the following shorthands to denote sets of actions: $\mathsf{T}$ denotes $\mathsf{Act}$, $a$ denotes $\{a\}$, $\neg A$ denotes $\mathsf{Act} - A$, $A \wedge B$ denotes $A \cap B$ and $A \vee B$ denotes $A \cup B$. Moreover, we shall use the notation $\mathsf{Act}_{x_1,\ldots,x_n}$ to denote "any action in which components $x_1, \ldots, x_n$ are involved" or, more formally, "any action $a(p_1, \ldots, p_m)$ for which there are $i_1, \ldots, i_n$ such that $p_{i_1} = x_1 \wedge \ldots \wedge p_{i_n} = x_n$". The asterisk $*$ is used as a parameter to actions to denote "any value of type $D$" in places where a specific value of type $D$ is expected.

### 5.2.1 R1

R1 is formulated by the following property: always after M5 is switched off, O4 is not covered before M5 is switched on and M5 is always switched on at least once before O4 becomes covered:

$$(1.1)\quad \nu X.[\mathsf{T}]X \;\wedge\; [\mathit{Off}(M5,*)]\,\nu Y.([\neg \mathit{On}(M5,*)]Y \;\wedge\; [\mathit{SignalHigh}(O4,*)]\mathsf{F})$$
$$\wedge\;\; \nu X.[\neg \mathit{On}(M5,*)]X \;\wedge\; [\mathit{SignalHigh}(O4,*)]\mathsf{F}$$

**Result:** Using the EVALUATOR model checker, we have proven that formula (1.1) does not hold. Consider the following processes from the $\mu$CRL specification:

**proc** $\mathit{Instance0}(id : \mathit{InstanceID}) =$
$\quad \mathit{SignalLow}(O1, id) \cdot \mathit{Instance0}(id) +$
$\quad \mathit{SignalHigh}(O1, id) \cdot$
$\quad (\mathit{getFirstSheet}(\mathtt{T}, id) \cdot \mathit{InformIn}(1s, id) \cdot \mathit{Instance1}(id) +$
$\quad \mathit{getFirstSheet}(\mathtt{F}, id) \cdot \mathit{ENTER\_M1}(id) \cdot \mathit{Instance4}(id))$

**proc** $\mathit{Instance33}(id : \mathit{InstanceID}) =$
$\quad \mathit{semValueZero}(\mathit{semN}, \mathtt{F}, id) \cdot \mathit{Leave}(M3C, id) \cdot \mathit{Instance}(id) +$
$\quad \mathit{semValueZero}(\mathit{semN}, \mathtt{T}, id) \cdot \mathit{ExecuteProfile}(X9, M3C, id)$
$\quad \cdot \mathit{InformIn}(500ms, id) \cdot \mathit{Instance34}(id)$

**proc** $\mathit{Instance34}(id : \mathit{InstanceID}) =$
$\quad \mathit{Timeout}(id) \cdot \mathit{Stop}(M3C, id) \cdot \mathit{setFirstSheet}(\mathtt{T}, id) \cdot \mathit{Off}(M5, id)$
$\quad \cdot \mathit{Instance35}(id)$

**proc** $\mathit{Instance35}(id : \mathit{InstanceID}) =$
$\quad \mathit{Stopped}(M3C, id) \cdot \mathit{ActivateIdleState}(M3C, id)$
$\quad \cdot \mathit{ActivateIdleState}(M1C, id) \cdot \mathit{ActivateIdleState}(M2C, id)$
$\quad \cdot \mathit{Leave}(M3C, id) \cdot \mathit{Instance}(id)$

Suppose instance $i$ has executed $\mathit{semValueZero}(\mathit{semN}, \mathtt{T}, i)$ in $\mathit{Instance33}$ but has not yet executed $\mathit{setFirstSheet}(\mathtt{T}, i)$ in $\mathit{Instance34}$ and instance $j$ executes $\mathit{getFirstSheet}(\mathtt{F}, j)$ in

*Instance0*. Next, $j$ can execute an arbitrary number of actions of its normal behaviour (process *Instance4* and beyond), before $i$ switches off M5 in *Instance34* and executes the actions in *Instance35*. When $j$'s sheet approaches the scanner, M5 is still switched off which violates requirement R1.

This corresponds to the situation in which instance $i$ has decided to move its sheet into the output tray (as it assumes its sheet is the last of the set) and right after that decision has been made, a new sheet is put into the input tray. Instance $j$ guides the new sheet into the machine, assuming it belongs to the same set as $i$'s sheet. When $j$'s sheet reaches the scanner, $i$ has already switched off M5 which complicates movement of the sheet over the glass plate as the sheet is not sucked against the transport belt.

However, $j$'s sheet may not even reach the scanner: when $i$'s sheet has reached the output tray, $i$ brings M1C, M2C and M3C to the Idle state in which these motors are unable to respond to commands, in particular those issued by instance $j$. Hence, $j$'s sheet gets stuck halfway through the ADF. Obviously, this is an situation that should be avoided at all times. This counter example and violation of R1 thus indicates a serious error in the system. We fix this error in Section 5.3.

### 5.2.2 R2

R2 is formulated by the following property: for any $i : \mathit{InstanceID}$, always after M4 is switched on, $i$'s sheet does not enter pinch K7 before M4 is switched off and always after $i$'s sheet enters pinch K7, M4 is not switched on before $i$'s sheet leaves K7:

$$\begin{aligned} (2.1) \qquad & \nu X.[\mathsf{T}]X \wedge [\mathit{On}(M4,*)]\,\nu Y.([\neg \mathit{Off}(M4,*)]Y \wedge [\mathit{ENTER\_K7}(i)]\mathsf{F}) \\ \wedge \quad & \nu X.[\mathsf{T}]X \wedge [\mathit{ENTER\_K7}(i)]\,\nu Y.([\neg \mathit{LEAVE\_K7}(i)]Y \wedge [\mathit{On}(M4,*)]\mathsf{F}) \end{aligned}$$

**Result:** Formula (2.1) does not hold, because the second conjunct evaluates to *false*. Consider the following processes from the $\mu$CRL specification:

**proc** $\mathit{Instance14}(id : \mathit{InstanceID}) =$
$\quad \mathit{reserve}(sem2, id) \cdot \mathit{ENTER\_M2}(id) \cdot \mathit{CaptureValue}(M2E, id)$
$\quad \cdot \mathit{ExecuteProfile}(X4, M1C, id) \cdot \mathit{ExecuteProfile}(X5, M2C, id)$
$\quad \cdot \mathit{Instance15}(id)$

**proc** $\mathit{Instance21}(id : \mathit{InstanceID}) =$
$\quad \mathit{PosReached}(M2E, id) \cdot \mathit{Stop}(M2C, id) \cdot \mathit{On}(M4, id) \cdot \mathit{InformIn}(50ms, id)$
$\quad \cdot \mathit{Instance22}(id)$

**proc** $\mathit{Instance28}(id : \mathit{InstanceID}) =$
$\quad \mathit{Stopped}(M2C, id) \cdot \mathit{Leave}(M2C, id) \cdot \mathit{release}(sem2, id) \cdot \mathit{LEAVE\_K7}(id)$
$\quad \cdot \mathit{Instance29}(id)$

Suppose instance $i$ has released semaphore *sem2* in process *Instance28*. Then an instance $j$ can reserve semaphore *sem2* (*Instance14*) and execute all actions in subsequent processes up to and including $\mathit{On}(M4, j)$ in *Instance21*, before $i$ executes $\mathit{LEAVE\_K7}(i)$. This is a counter example to (2.1) and a violation of R2. It corresponds to the situation in which $i$'s sheet is in pinch K7 (moving towards the scanner) and has just stopped covering sensor O3. Instance $j$ can have its sheet move over O3 towards K5, turn back, move towards pinch K7 and start blousing, which involves switching off C3 and switching on C4 and M4. This happens while $i$'s sheet is still in pinch K7 and moving onto the glass plate. Under influence of M4, K7 starts turning in opposite direction making $i$'s sheet move backward towards O3. This situation is clearly undesired and indicates an error in the system. We fix this error in Section 5.3.

### 5.2.3 R3

R3 is formulated by the following properties:

- For any $i : \mathit{InstanceID}$, whenever $i$'s sheet enters pinch K3, M1 is executing profile X4 and M2 is executing X5 and whenever $i$'s sheet enters pinch K7, M2 is executing profile X8 and M3 is executing X9:

$$
\begin{aligned}
(3.1)\quad & \nu X.[\mathsf{T}]X \wedge [\mathsf{Act}_{M1C} \wedge \neg \mathit{ExecuteProfile}(X4, M1C, *)] \\
& \qquad \nu Y.(\,[\neg \mathit{ExecuteProfile}(X4, M1C, *)]Y \wedge [\mathit{ENTER\_K3}(i)]\mathsf{F}\,) \\
\wedge\; & \nu X.[\mathsf{T}]X \wedge [\mathsf{Act}_{M2C} \wedge \neg \mathit{ExecuteProfile}(X5, M2C, *)] \\
& \qquad \nu Y.(\,[\neg \mathit{ExecuteProfile}(X5, M2C, *)]Y \wedge [\mathit{ENTER\_K3}(i)]\mathsf{F}\,) \\
\wedge\; & \nu X.[\mathsf{T}]X \wedge [\mathsf{Act}_{M2C} \wedge \neg \mathit{ExecuteProfile}(X8, M2C, *)] \\
& \qquad \nu Y.(\,[\neg \mathit{ExecuteProfile}(X8, M2C, *)]Y \wedge [\mathit{ENTER\_K7}(i)]\mathsf{F}\,) \\
\wedge\; & \nu X.[\mathsf{T}]X \wedge [\mathsf{Act}_{M3C} \wedge \neg \mathit{ExecuteProfile}(X9, M3C, *)] \\
& \qquad \nu Y.(\,[\neg \mathit{ExecuteProfile}(X9, M3C, *)]Y \wedge [\mathit{ENTER\_K7}(i)]\mathsf{F}\,)
\end{aligned}
$$

- For any $i : \mathit{InstanceID}$, always after $i$'s sheet enters pinch K3, no command is sent to M1C or M2C before $i$'s sheet leaves pinch K2 and always after $i$'s sheet enters pinch K7, no command is sent to M2C or M3C before $i$'s sheet leaves pinch K3:

$$(3.2)\; \nu X.[\mathsf{T}]X \wedge [\mathit{ENTER\_K3}(i)]\; \nu Y.(\,[\neg \mathit{LEAVE\_K2}(i)]Y \wedge [\mathsf{Act}_{M1C} \vee \mathsf{Act}_{M2C}]\mathsf{F}\,)$$

$$(3.3)\; \nu X.[\mathsf{T}]X \wedge [\mathit{ENTER\_K7}(i)]\; \nu Y.(\,[\neg \mathit{LEAVE\_K3}(i)]Y \wedge [\mathsf{Act}_{M2C} \vee \mathsf{Act}_{M3C}]\mathsf{F}\,)$$

**Result:** Formulas (3.1) and (3.2) hold, but (3.3) does not hold. Consider the following processes from the $\mu$CRL specification:

$$
\begin{aligned}
\textbf{proc}\quad & \mathit{Instance25}(id : \mathit{InstanceID}) = \\
& \mathit{Timeout}(id) \cdot \mathit{Enter}(M3C, id) \cdot \mathit{ExecuteProfile}(X8, M3C, id) \\
& \cdot \mathit{ExecuteProfile}(X9, M3C, id) \cdot \mathit{ENTER\_K7}(id) \cdot \mathit{CaptureValue}(M3E, id) \\
& \cdot \mathit{Instance26}(id)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{proc}\quad & \mathit{Instance26}(id : \mathit{InstanceID}) = \\
& \mathit{SignalLow}(O4, id) \cdot \\
& (\quad \mathit{NotTooHigh}(M3E, id) \cdot \mathit{Instance26}(id) \\
& \quad + \mathit{TooHigh}(M3E, id) \cdot \mathit{Stop}(M2C, id) \cdot \mathit{Stop}(M3C, id) \cdot \delta \quad ) \\
& + \mathit{SignalHigh}(O4, id) \cdot \mathit{Instance27}(id)
\end{aligned}
$$

An instance $i$ can first execute $\mathit{ENTER\_K7}(i)$ in *Instance25* and later on send stop commands to M2C and M3C (in *Instance26*) without executing $\mathit{LEAVE\_K3}(i)$ first. This is a counter example to formula (3.3) and a violation of requirement R3. It corresponds to the situation in which a sheet has entered pinch K7 and is moving towards the glass plate, driven by motor M3. If M3 has covered a certain distance and O4 has not yet detected the sheet, the instance assumes something went wrong, stops M2 and M3 and deadlocks. This is desired behaviour: if an error occurs, stop all running motors and bail out. Hence, the requirement is too strict and needs to be changed. The following formula expresses the requirement that for any $i : \mathit{InstanceID}$, always after $i$'s sheet enters pinch K7, no command is sent to M2C or M3C before $i$'s sheet leaves pinch K3 *unless* i *detects an error*. We have verified that this weaker version of (3.3) holds.

$$
\begin{aligned}
(3.4)\quad & \nu X.[\mathsf{T}]X \wedge [\mathit{ENTER\_K7}(i)] \\
& \qquad \nu Y.(\,[\neg(\mathit{LEAVE\_K3}(i) \vee \mathit{TooHigh}(*, i))]Y \wedge [\mathsf{Act}_{M2C} \vee \mathsf{Act}_{M3C}]\mathsf{F}\,)
\end{aligned}
$$

### 5.2.4 R4

R4 is a liveness property which can only be satisfied if the following fairness condition holds: if an action $a$ is enabled infinitely often, then $a$ is executed infinitely often (see also [10]). R4 is formulated by the following property, for all $i : \mathit{InstanceID}$:

$$(4.1)\qquad \nu X.[\mathsf{T}]X \wedge [\mathit{ENTER\_M1}(i)]\, \nu Y.(\,[\neg \mathit{SignalLow}(O4,i)]Y \wedge \mu Z.(\,\langle \neg \mathit{SignalLow}(O4,i)\rangle Z \vee \langle \mathit{SignalLow}(O4,i)\rangle \mathsf{T}\,))$$

**Result:** Formula (4.1) does not hold. After an instance has detected an error (like the one in the counter example for R3 above) it will no longer be able to guide its sheet further towards sensor O4, because it always reaches a deadlock state. For the same reason as for R3, this is desired behaviour, so the requirement is too strict.

To obtain a weaker version of R4, we reason as follows. If an instance *i*'s sheet enters the ADF when another instance *j* has just detected an error, *i*'s sheet will not be able to reach O4. This is because *i* will at some point be waiting to reserve a semaphore that *j* will never release. We have verified this using the CADP toolset. For the same reason, if *i*'s sheet has entered the ADF and some instance (either *i* or another instance) detects an error, *i*'s sheet may not reach O4. We have also verified this using the CADP toolset. Based on these observations, we formulate the following weaker version of (4.1): for all $i : \mathit{InstanceID}$, if *i*'s sheet enters the ADF *and no error has occurred*, *i*'s sheet will eventually pass sensor O4 *unless an error occurs*. We have verified that this formula holds for all $i : \mathit{InstanceID}$:

$$(4.2)\qquad \nu X.[\mathit{TooHigh}(*,*)]X \wedge [\mathit{ENTER\_M1}(i)]\, \nu Y.(\,[\neg(\mathit{SignalLow}(O4,i) \vee \mathit{TooHigh}(*,*))]Y \wedge \mu Z.(\,\langle \neg(\mathit{SignalLow}(O4,i) \vee \mathit{TooHigh}(*,*))\rangle Z \vee \langle \mathit{SignalLow}(O4,i) \vee \mathit{TooHigh}(*,*)\rangle \mathsf{T}\,))$$

### 5.2.5 R5

Initially both C3 and C4 are switched off. Then R5 is formulated by the following property: always after C3 is switched on, C4 is never switched on before C3 is switched off and always after C4 is switched on, C3 is never switched on before C4 is switched off:

$$(5.1)\qquad \nu X.[\mathsf{T}]X \wedge [\mathit{On}(C3,*)]\, \nu Y.(\,[\neg \mathit{Off}(C3,*)]Y \wedge [\mathit{On}(C4,*)]\mathsf{F}\,) \wedge \nu X.[\mathsf{T}]X \wedge [\mathit{On}(C4,*)]\, \nu Y.(\,[\neg \mathit{Off}(C4,*)]Y \wedge [\mathit{On}(C3,*)]\mathsf{F}\,)$$

**Result:** Formula (5.1) holds.

### 5.2.6 R6

R6 is formulated by the following properties:

- For any $i : \mathit{InstanceID}$, always after *i* commands M1 to execute profile X1 or X2, M1 does not receive another command (from any instance) before it has finished executing the profile and always after *i* commands M2 to execute profile X7, M2 does not receive another command (from any instance) before $i$ commands M2 to stop:

$$\begin{aligned}
(6.1)\quad & \nu X.[\mathsf{T}]X \wedge [\mathit{ExecuteProfile}(X1,M1C,i) \vee \mathit{ExecuteProfile}(X2,M1C,i)] \\
& \qquad \nu Y.(\,[\neg \mathit{PosReached}(M1C,i)]Y \wedge [\mathsf{Act}_{M1C} \wedge \neg \mathit{PosReached}(M1C,i)]\mathsf{F}\,) \\
\wedge\quad & \nu X.[\mathsf{T}]X \wedge [\mathit{ExecuteProfile}(X7,M2C,i)] \\
& \qquad \nu Y.(\,[\neg \mathit{Stop}(M2C,i)]Y \wedge [\mathsf{Act}_{M2C} \wedge \neg \mathit{Stop}(M2C,i)]\mathsf{F}\,)
\end{aligned}$$

- For any $i,j : \mathit{InstanceID}, i \neq j$, and $c : \mathit{ControllerID}$, as long as $i$ needs $c$, $j$ does not send a command to $c$:

$$(6.2) \qquad \nu X.[\mathsf{T}]X \wedge [\mathit{Enter}(c,i)]\, \nu Y.(\,[\neg \mathit{Leave}(c,i)]Y \wedge [\mathsf{Act}_{c,j}]\mathsf{F}\,)$$

**Result:** Formulas (6.1) and (6.2) hold.

### 5.2.7 R7

For any $i : \mathit{InstanceID}$ and $s : \mathit{SensorID}, s \in \{O2, O3, O4\}$, always after $i$'s sheet starts covering $s$, $\mathit{next}(i)$'s sheet does not cover $s$ before $i$'s sheet has stopped covering $s$:

$$\begin{aligned}
(7.1)\quad & \nu X.[\mathsf{T}]X \wedge [\mathit{SignalHigh}(s,i)] \\
& \qquad \nu Y.(\,[\neg \mathit{SignalLow}(s,i)]Y \wedge [\mathit{SignalHigh}(s,\mathit{next}(i))]\mathsf{F}\,)
\end{aligned}$$

**Result:** Formula (7.1) holds.

## 5.3 Solutions

The problem revealed by the violation of R1 stems from the fact that it is possible for one instance to assume that the set contains no more sheets, while another instance assumes that a newly inserted sheet is still part of that set. In essence, the problem is that instances do not have exclusive access to the initialization and finalization parts of their behaviour. These parts contain behaviour (like accessing global variables) that should not be interleaved as a result of parallelism. Therefore we introduce a new semaphore sem4, which initially has value 1 and grants instances exclusive access to either the initialization or the finalization part. Initialization starts when the instance has detected a sheet in the input tray and ends when either the sheet is no longer detected after time period *t1* (if FirstSheet is *true*) or the SemN counter has been increased (if FirstSheet is *false*). Finalization starts when the instance is about to check whether the value of SemN equals zero and ends when either SemN is found to be unequal to zero or M1, M2 and M3 have been brought to Idle (if SemN equals zero).

R2 is violated because sheets do not have exclusive access to pinch K7. We introduce a new semaphore sem5 which initially has value 1 and grants instances exclusive access to K7. The first time an instance influences K7 is when it executes the $\mathit{Off}(C3,\mathit{id})$ action in process *Instance18*, which is right after the sheet has turned and starts moving towards K7. Semaphore sem5 should be reserved before the sheet turns. Because it is moving in the other direction at that time, the sheet should also be stopped before sem5 can be reserved. We introduce a new process *Instance18A* for this purpose. Naturally, sem5 can be released after the sheet has left K7. Action $\mathit{LEAVE\_K7}(\mathit{id})$ in *Instance28* would be an obvious choice for marking this event, but as this is not an action of the real system, we take $\mathit{SignalLow}(O4,\mathit{id})$ in *Instance29* instead.

## 5.4 Revised specification

The solutions described above involve the following additions to the $\mu$CRL process definitions (additions listed in **bold**):

**proc** $\mathit{Instance0}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \ldots + \mathit{SignalHigh}(O1, \mathit{id}) \cdot \mathbf{reserve}(\mathbf{sem4}, \mathbf{id}) \cdot \ldots$

**proc** $\mathit{Instance1}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{Timeout}(\mathit{id}) \cdot$
$\quad (\mathit{SignalLow}(O1, \mathit{id}) \cdot \mathbf{release}(\mathbf{sem4}, \mathbf{id}) \cdot \mathit{Instance0}(\mathit{id}) + \ldots)$

**proc** $\mathit{Instance4}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{reserve}(\mathit{sem}1, \mathit{id}) \cdot \mathit{Off}(S1, \mathit{id}) \cdot \mathit{On}(C2, \mathit{id}) \cdot \mathit{On}(C1, \mathit{id}) \cdot \mathit{release}(\mathit{semN}, \mathit{id})$
$\quad \cdot \mathbf{release}(\mathbf{sem4}, \mathbf{id}) \cdot \ldots$

**proc** $\mathit{Instance18}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \ldots + \mathit{SignalLow}(O3, \mathit{id}) \cdot \mathbf{Stop}(\mathbf{M2C}, \mathbf{id}) \cdot \mathbf{Instance18A}(\mathbf{id})$

**proc** $\mathbf{Instance18A}(\mathbf{id} : \mathbf{InstanceID}) =$
$\quad \mathbf{Stopped}(\mathbf{M2C}, \mathbf{id}) \cdot \mathbf{reserve}(\mathbf{sem5}, \mathbf{id}) \cdot \mathit{CaptureValue}(\mathit{M2E}, \mathit{id})$
$\quad \cdot \mathit{ExecuteProfile}(X6, \mathit{M2C}, \mathit{id}) \cdot \mathit{Off}(C3, \mathit{id}) \cdot \mathit{Instance19}(\mathit{id})$

**proc** $\mathit{Instance29}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \ldots + \mathrm{SignalLow}(\mathrm{O4}, \mathit{id}) \cdot \mathbf{release}(\mathbf{sem5}, \mathbf{id}) \cdot \ldots$

**proc** $\mathit{Instance32}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{Timeout}(\mathit{id}) \cdot \mathit{reserve}(\mathit{semN}, \mathit{id}) \cdot \mathit{release}(\mathit{sem}3, \mathit{id}) \cdot \mathbf{reserve}(\mathbf{sem4}, \mathbf{id})$
$\quad \cdot \mathit{Instance33}(\mathit{id})$

**proc** $\mathit{Instance33}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{semValueZero}(\mathit{semN}, \mathtt{F}, \mathit{id}) \cdot \mathbf{release}(\mathbf{sem4}, \mathbf{id}) \cdot \mathit{Instance}(\mathit{id}) + \ldots$

**proc** $\mathit{Instance35}(\mathit{id} : \mathit{InstanceID}) =$
$\quad \mathit{Stopped}(\mathit{M3C}, \mathit{id}) \cdot \mathit{ActivateIdleState}(\mathit{M3C}, \mathit{id})$
$\quad \cdot \mathit{ActivateIdleState}(\mathit{M1C}, \mathit{id}) \cdot \mathit{ActivateIdleState}(\mathit{M2C}, \mathit{id})$
$\quad \cdot \mathit{Leave}(\mathit{M3C}, \mathit{id}) \cdot \mathbf{release}(\mathbf{sem4}, \mathbf{id}) \cdot \mathit{Instance}(\mathit{id})$

Other changes include additions to the data specification and the initial process specification. The full revised $\mu$CRL specification is included in Appendix C. We have model checked all formulas of Section 5.2 against the LTS of the revised specification, which has 78.751 states and 231.456 transitions. The model checker returns *false* only on formulas (3.2) and (4.1), for the same reasons as described in Sections 5.2.3 and 5.2.4. The weaker versions of these properties (formulas (3.4) and (4.2), respectively) do hold.

# 6 Conclusions

We have analyzed the embedded software of an Automatic Document Feeder by applying formal verification techniques. We have given a system description, requirements, a specification in $\mu$CRL and the results of model checking the requirements using the CADP model checker. If requirement violations indicated errors in the system, we suggested solutions to these problems and showed that a system that incorporates these solutions meets all of the requirements. We formulated weaker versions of two requirements that were found to be too strict. Our analysis revealed errors in the ADF prototype that would have been difficult to find otherwise. This indicates that formal specification and verification is a very powerful and effective method for finding errors in systems.

Regarding the problem found by the violation of requirement R2, we note that it may very well be that this situation cannot occur if real time is taken into account, as the amount of time needed for a sheet to move from O2 to K7 (paths *A* and *B*) may exceed the amount of time needed for another sheet to move onto the glass plate (path *C*). As we did not consider real time in the analysis, we have not been able to verify this claim and cannot guarantee that R2 is never violated. To be 100% sure, we have solved the problem for either case (whether the timed behaviour allows R2 to be violated or not).

We have shown that both problems that were found, can be resolved very easily and efficiently by adding only two semaphores to the system and applying minor modifications to the behaviour. As can be seen from the fact that the LTS belonging to the revised system is more than four times as small as that of the original system, these solutions simplify the behaviour of the system as a whole.

For the verification of untimed properties on the ADF, $\mu$CRL and CADP are very well suited: generation of the LTS and model checking of the modal properties was easily done on a contemporary desktop computer. The fact that timed behaviour could not be taken into account, meant that no timed properties could be checked for the ADF. On the other hand, it allowed us to abstract away from irrelevant details and to focus on those parts of the behaviour that were relevant to the requirements we wanted to verify.

An important issue is that the execution sheet in which the behaviour of the embedded software is specified, is at some points unclear or ambiguous. A more strict and formal specification style or language can aid engineers in describing the behaviour of the system more precisely and unambiguously. It helps in the communication of system designs between engineers as it prevents misinterpretations and requires less additional explanation. Indeed, one of the greatest difficulties in this analysis was to gain a precise and correct understanding of the behaviour of the system.

Formal behavioural descriptions may also be better suited for the automatic translation of system descriptions to $\mu$CRL specifications. Regarding this issue we note however, that the real challenge and difficulties stem from the fact that the specifications are aimed at *modelling* the real behaviour and, hence, are usually at a higher level of abstraction. This often requires model design issues regarding the appropriate level of abstraction to be properly resolved, which is arguably a task at which humans perform better than machines. An example of such a model design issue is the decision to abstract away from specific time periods and distances if they are irrelevant to the requirements that will be checked.

## Acknowledgements

We would like to thank the engineers at Océ, Jan Friso Groote, Aad Mathijssen, Hannes Pretorius and Muck van Weerdenburg for valuable discussions and comments.

## References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge*

*tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1990.

[2] S. Blom, W. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. $\mu$CRL: A toolset for analysing algebraic specifications. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th Conference on Computer-Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254. Springer, July 2001.

[3] J.-C. Fernandez, H. Garavel, A. Kerbrat, A. Mounier, R. Mateescu, and M. Sighireanu. CADP: a protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[4] W.J. Fokkink, J.F. Groote, J. Pang, B. Badban, and J.C. van de Pol. Verifying a sliding window protocol in $\mu$CRL. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Proceedings 10th Conference on Algebraic Methodology and Software Technology – AMAST'04*, Lecture Notes in Computer Science 3116, pages 148–163. Springer, July 2004.

[5] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. Technical Report RT-0254, INRI, France, 2001.

[6] Jan Friso Groote, Jun Pang, and Arno G. Wouters. Analysis of a distributed system for lifting trucks. *Journal of Logic and Algebraic Programming*, 55(1-2):21–56, 2003.

[7] J.F. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. Report CS-R9076, CWI, Amsterdam, 1990.

[8] D. Kozen. Results on the propositional $\mu$-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[9] J. van de Pol and M. Valero Espada. Verification of JavaSpaces (TM) Parallel Programs. In J. Lilius, F. Balarin, and Machado R.J., editors, *Proceedings 3rd International Conference on Application of Concurrency to System Design (ACSD)*, Guimaraes, Portugal, pages 196–205. IEEE, June 2003.

[10] J.P. Queille and J. Sifakis. Fairness and related properties in transition systems: a temporal logic to deal with fairness. *Acta Informatica*, 19(3):195–220, 1983.

[11] C. Shankland and M.B. van der Zwaag. The tree identify protocol of IEEE 1394 in $\mu$CRL. *Formal Aspects of Computing*, 10:509–531, 1998.

## A Execution sheet

Reservation of a semaphore sem$S$ is denoted by $\boxed{S}\!\rightarrow$ and release by $\boxed{S}\!\leftarrow$ in the margin on the lefthand side of the execution sheet. Some of the behaviour is not specified explicitly or clearly. For example, in the "alignOriginalSet" state, if O1's signal is high and the value of FirstSheet is *true* then the instance continues at the line where sem1 is reserved; if FirstSheet is *false* it continues as normal (switch on M5, etc.).

|  | state | trigger |  | Action |  | comment |
|---|---|---|---|---|---|---|
|  |  | Device/Timer | Result/Reply | Device/Timer | action |  |
|  |  | "Framework" | Initialisation | O1 | Poll for high signal | start-up (first sheet) |
|  |  | O1 | Sensor is high | Timer | InformIn(t1) | t1 delay (avoid signal chattering) |
|  |  | Timer | Timeout | O1 | Check if still high |  |
|  | alignOriginalSet | O1 | Sensor is low | O1 | Poll for high signal |  |
|  |  |  | Sensor is high | M5 | On |  |
|  |  |  |  | C2 | On |  |
|  |  |  |  | C1 | Off |  |
|  |  |  |  | S1 | Off |  |
|  |  |  |  | M1C | Activate control state |  |
|  |  |  |  | M2C | Activate control state |  |
|  |  |  |  | M3C | Activate control state |  |
|  |  |  |  | Flag | FirstSheet=false |  |
|  |  |  |  | Timer | InformIn(t2) |  |
|  |  | Timer | Timeout | M1C | Move left (execute Profile X1) |  |
|  |  | M1C | Pos. reached |  |  |  |
|  | If the set is already aligned, the next separation starts from here (realised with flag FirstSheet) and as precondition Activated & O1 true |  |  |  |  |  |
| 1 → |  |  |  | S1 | Off |  |
|  |  |  |  | C2 | On |  |
|  |  |  |  | C1 | On |  |
|  |  |  |  | SemN | Increase |  |
|  |  |  |  | Timer | InformIn(t3) |  |
|  | ShakeOriginalSet | Timer | Timeout | M1C | Move right (execute Profile X2) |  |
|  |  | M1C | Pos. reached | C2 | Off |  |
|  |  |  |  | Timer | InformIn(t4) |  |
|  | separateSet | Timer | Timeout | S1 | On |  |
|  |  |  |  | C2 | On |  |
|  |  |  |  | C1 | Off |  |
|  |  |  |  | Timer | InformIn(t5) |  |
|  |  | Timer | Timeout | M1C | Move left (execute Profile X3) |  |
|  |  |  |  | M1E | Capture value = *p1* | Needed for error check 1 |
|  |  |  |  | M1E | InformAfter(d1) |  |
|  |  | M1E | Pos. reached | C1 | On |  |
|  |  |  |  | M1E | InformAfter(d2) |  |
|  |  | M1E | Pos. reached | S1 | Off |  |
|  |  |  |  | O2 | Poll for signal |  |
|  |  | O2 | sensor is not covered && (M1E_value > *p1* + ed1) | M1C | Stop<br>Exception | Error check 1 |
|  |  |  | Sensor is covered | C1 | Off |  |
|  |  |  |  | Timer | InformIn(t6) |  |
|  |  | Timer | Timeout | C2 | On |  |
|  |  |  |  | Timer | InformIn(t7) |  |
|  |  |  |  | M1C | Stop |  |
|  |  | M1C & Timer | Stopped & Timeout | Sem2 | Request |  |
| 2 → | ExportToTurn ImportFromSep | Sem2 | Receive | M2E | Capture value = *p2* | Needed for error check 2 |
|  |  |  |  | M1C | Start (execute Profile X4) |  |
|  |  |  |  | M2C | Start (execute Profile X5) |  |
|  |  |  |  | O3 | If signal is high capture M2E value = *p3* | Via interrupt service routine done |
|  |  |  |  | O2 | Poll for signal |  |
|  |  | O2: | (O2 is covered) && (O3 is not covered) && (M2E_value > *p2* + ed2) | M1C | Stop | Error check 2 |
|  |  |  |  | M2C | Stop<br>Exception |  |
|  |  |  | O2 is uncovered | M1C | Stop |  |
|  |  |  |  | M2E | Capture value = *p4* | Needed for error check 3 |
| 1 ← |  | M1C | Stopped | Flag | ActivateFlag = true | next separation is possible -> Activate next instance |
|  |  |  |  | O3 | Poll for signal |  |

| | | | | | | |
|---|---|---|---|---|---|---|
| | TurnCW | O3: | (sensor is covered) && (M2E_value > *p4* + ed3) | M2C | Stop<br>Exception | Error check 3 |
| | | | Sensor is not covered | M2E | Capture value = *p5* | This value is used to calculate the actual paper length |
| | | | | M2C | Change turn direction (execute Profile X6) | |
| | | | | O3 | Poll for low to high | |
| | | | | C3 | Off | |
| | | | M2E_value > *p5* + ed4 | M2C | Stop<br>Exception | Error check 4 |
| | | O3: | Sensor is high | M2E | Capture value = *p6*<br>Calculate paper length = *pl* | Needed for error check 5 |
| | | | | M2E | InformAfter(d3) | Each time in exe.: check >= *p6* + d3 |
| | | M2E | Pos. reached | M2C | Change speed (execute Profile X7) | |
| | | | | C4 | On | |
| | | | | M2E | InformAfter(d4) | |
| | | M2E | Pos. reached | M2C | Stop | |
| | | | | M4 | On | |
| | | | | Timer | InformIn(t8) | |
| | | Timer | Timeout | C4 | Off | |
| | | | | M4 | Off | |
| | | | | Timer | InformIn(t9) | |
| | | Timer | Timeout | Sem3 | Request | |
| $\boxed{3} \rightarrow$ | | Sem3 | Receive | C3 | On | |
| | | | | Timer | InformIn(t10) | |
| | ExportToFeed Feed | Timer | Timeout | M2C | Start (execute Profile X8) | |
| | | | | M3C | Start (execute Profile X9) | |
| | | | | O3 | Poll for signal | |
| | | | | M3E | Capture value = *p7* | Needed for error check 6 |
| | | O3: | (O3 is covered) && (M2E_value > *pl* + ed5) | M2C | Stop | Error check 5 |
| | | | | M3C | Stop<br>Exception | |
| | | | (O4 is not covered) && (M3E_value >= *p7* + ed6) | M2C | Stop | Error check 6 |
| | | | | M3C | Stop<br>Exception | |
| | | | Sensor is not covered | M2C | Stop | |
| $\boxed{2} \leftarrow$ | | M2C | Stopped | Sem2 | Release | Next ExportToTurn is possible |
| | | | | O4 | Poll for high to low | |
| | | O4: | (O4 is covered) && (M3E_value > *p7* + *pl* + ed7) | M3C | Stop<br>Exception | Error check 7 |
| | | | Sensor is not covered | M3E | InformAfter(d5) | |
| | | M3E | Pos. reached | M3C | Stop | |
| | | M3C | stopped | Timer | InformIn(t11) | Just a delay to simulate the scan |
| | Scan | Timer | Timeout | SemN | Decrease | |
| $\boxed{3} \leftarrow$ | | | | Sem3 | Release | Next scan is possible |
| | | | | SemN | Get value | Sheet transport is finished |
| | | SemN | SemN ≠ 0 | - | - | restart instance next sheet is waiting for transport |
| | Remove original | | SemN = 0 | M3C | Start (execute Profile X9) | |
| | | | | Timer | InformIn(t12) | |
| | | Timer | Timeout | M3C | Stop | Last sheet restart instance no following sheet, just wait for next job |
| | | | | Flag | FirstSheet=true | |
| | | | | M5 | Off | |
| | | M3C | stopped | M3C | Go to idle | |
| | | | | M1C | Go to idle | |
| | | | | M2C | Go to idle | |

## B Original $\mu$CRL specification

This is the $\mu$CRL specification of the original ADF system as specified by the execution sheet of Appendix A.

```
sort  Bool
func  T,F: ->Bool
map   not:Bool->Bool
      and,or,eq:Bool#Bool->Bool
var   b:Bool
rew   not(F)=T    not(T)=F
      and(b,T)=b  and(b,F)=F
      or(b,T)=T   or(b,F)=b
      eq(b,b)=T   eq(T,F)=F   eq(F,T)=F

sort  Nat
func  0:->Nat
      S:Nat->Nat
map   gt,eq:Nat#Nat->Bool
      dec:Nat->Nat
var   m,n:Nat
rew   gt(0,n)=F gt(S(m),0)=T gt(S(m),S(n))=gt(m,n)
      eq(0,0)=T eq(0,S(n))=F eq(S(n),0)=F eq(S(m),S(n))=eq(m,n)
      dec(S(n))=n

sort  SensorID
func  O1,O2,O3,O4:->SensorID

sort  OnOffID
func  S1,C1,C2,C3,C4,M4,M5:->OnOffID

sort  ControllerID
func  M1C,M2C,M3C:->ControllerID

sort  EncoderID
func  M1E,M2E,M3E:->EncoderID

sort  ProfileID
func  X1,X2,X3,X4,X5,X6,X7,X8,X9:->ProfileID

sort  InstanceID
func  I0,I1,I2: -> InstanceID
map   eq:InstanceID#InstanceID->Bool
      next,prev:InstanceID->InstanceID
rew   eq(I0,I0)=T eq(I0,I1)=F eq(I0,I2)=F
      eq(I1,I0)=F eq(I1,I1)=T eq(I1,I2)=F
      eq(I2,I0)=F eq(I2,I1)=F eq(I2,I2)=T
      next(I0)=I1 prev(I0)=I2
      next(I1)=I2 prev(I1)=I0
      next(I2)=I0 prev(I2)=I1

sort  SemaphoreID
func  sem1,sem2,sem3,semN:->SemaphoreID
map   eq:SemaphoreID#SemaphoreID->Bool
rew   eq(sem1,sem1)=T eq(sem1,sem2)=F eq(sem1,sem3)=F eq(sem1,semN)=F
      eq(sem2,sem1)=F eq(sem2,sem2)=T eq(sem2,sem3)=F eq(sem2,semN)=F
      eq(sem3,sem1)=F eq(sem3,sem2)=F eq(sem3,sem3)=T eq(sem3,semN)=F
      eq(semN,sem1)=F eq(semN,sem2)=F eq(semN,sem3)=F eq(semN,semN)=T

sort  TimePeriod
func  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12:->TimePeriod

sort  Distance
func  d1,d2,d3,d4,d5:->Distance

act   ENTER_K3, LEAVE_K3, LEAVE_K2, ENTER_K7, LEAVE_K7, ENTER_M1, LEAVE_M1,
      ENTER_M2, LEAVE_M2, ENTER_M3, LEAVE_M3, Timeout: InstanceID
      setFirstSheet, SetFirstSheet, getFirstSheet,GetFirstSheet:Bool#InstanceID
      reserve,Reserve,release,Release:SemaphoreID#InstanceID
      semValueZero,SemValueZero:SemaphoreID#Bool#InstanceID
      activate,Activate:InstanceID#InstanceID
      SignalLow,SignalHigh:SensorID#InstanceID
      On,Off:OnOffID#InstanceID
      ActivateControlState,ActivateIdleState,PosReached,Stop,Stopped:
      ControllerID#InstanceID
      CaptureValue,PosReached,NotTooHigh,TooHigh:EncoderID#InstanceID
      ExecuteProfile:ProfileID#ControllerID#InstanceID
      InformIn:TimePeriod#InstanceID
      InformAfter:EncoderID#Distance#InstanceID

comm  setFirstSheet | setFirstSheet = SetFirstSheet
      getFirstSheet | getFirstSheet = GetFirstSheet
      reserve | reserve = Reserve
      release | release = Release
      semValueZero | semValueZero = SemValueZero
      activate | activate = Activate

proc  FirstSheet(b:Bool) =
        sum(i:InstanceID,
          sum(b1:Bool, setFirstSheet(b1,i).FirstSheet(b1))
          +getFirstSheet(b,i).FirstSheet(b) )

      Semaphore(sid:SemaphoreID,n:Nat) =
        sum(i:InstanceID,
          reserve(sid,i).Semaphore(sid,dec(n)) <| gt(n,0) |> delta
          +release(sid,i).Semaphore(sid,S(n))
          +semValueZero(sid,eq(n,0),i).Semaphore(sid,n) )

      %% ----- Initialization and aligning ----- %%

      Instance(id:InstanceID) =
        activate(id,prev(id)).Instance0(id)

      Instance0(id:InstanceID) =
        SignalLow(O1,id).Instance0(id)
        + SignalHigh(O1,id).
          ( getFirstSheet(T,id).InformIn(t1,id).Instance1(id)
           +getFirstSheet(F,id).ENTER_M1(id).Instance4(id) )

      Instance1(id:InstanceID) =
        Timeout(id).
        (SignalLow(O1,id).Instance0(id)
        +
        SignalHigh(O1,id).ENTER_M1(id).On(M5,id).On(C2,id).Off(C1,id).On(S1,id)
        .ActivateControlState(M1C,id).ActivateControlState(M2C,id)
        .ActivateControlState(M3C,id).setFirstSheet(F,id).InformIn(t2,id)
        .Instance2(id))

      Instance2(id:InstanceID) =
        Timeout(id).ExecuteProfile(X1,M1C,id).Instance3(id)

      Instance3(id:InstanceID) =
        PosReached(M1C,id).Instance4(id)

      %% ----- Separation ----- %%

      Instance4(id:InstanceID) =
        reserve(sem1,id).Off(S1,id).On(C2,id).On(C1,id).release(semN,id)
        .InformIn(t3,id).Instance5(id)

      Instance5(id:InstanceID) =
        Timeout(id).ExecuteProfile(X2,M1C,id).Instance6(id)

      Instance6(id:InstanceID) =
        PosReached(M1C,id).Off(C2,id).InformIn(t4,id).Instance7(id)

      Instance7(id:InstanceID) =
        Timeout(id).On(S1,id).On(C2,id).Off(C1,id).InformIn(t5,id)
        .Instance8(id)

      Instance8(id:InstanceID) =
        Timeout(id).ExecuteProfile(X3,M1C,id).CaptureValue(M1E,id)
        .InformAfter(M1E,d1,id).Instance9(id)

      Instance9(id:InstanceID) =
        PosReached(M1E,id).On(C1,id).InformAfter(M1E,d2,id).Instance10(id)
```

```
Instance10(id:InstanceID) =
  PosReached(M1E,id).Off(S1,id).Instance11(id)

Instance11(id:InstanceID) =
  SignalLow(O2,id).
  (NotTooHigh(M1E,id).Instance11(id) + TooHigh(M1E,id).Stop(M1C,id).delta)
  +SignalHigh(O2,id).Off(C1,id).InformIn(t6,id).Instance12(id)

Instance12(id:InstanceID) =
  Timeout(id).On(C2,id).InformIn(t7,id).Stop(M1C,id).Instance13(id)

Instance13(id:InstanceID) =
  (Timeout(id).Stopped(M1C,id)+Stopped(M1C,id).Timeout(id)).Instance14(id)

%% ----- Turning and blousing ----- %%

Instance14(id:InstanceID) =
  reserve(sem2,id).ENTER_M2(id).CaptureValue(M2E,id)
  .ExecuteProfile(X4,M1C,id).ExecuteProfile(X5,M2C,id).Instance15(id)

Instance15(id:InstanceID) =
  SignalHigh(O2,id).
    (SignalHigh(O3,id).ENTER_K3(id).CaptureValue(M2E,id)
     .Instance16(id)
     +SignalLow(O3,id).
      (NotTooHigh(M2E,id).Instance15(id)
       +TooHigh(M2E,id).Stop(M1C,id).Stop(M2C,id).delta))
  +SignalHigh(O3,id).ENTER_K3(id).CaptureValue(M2E,id)
  .Instance16(id)

Instance16(id:InstanceID) =
  SignalHigh(O2,id).Instance16(id)
  +SignalLow(O2,id).LEAVE_K2(id).Stop(M1C,id).CaptureValue(M2E,id)
   .Instance17(id)

Instance17(id:InstanceID) =
  Stopped(M1C,id).LEAVE_M1(id).activate(next(id),id).release(sem1,id)
  .Instance18(id)

Instance18(id:InstanceID) =
  SignalHigh(O3,id).
  (NotTooHigh(M2E,id).Instance18(id) + TooHigh(M2E,id).Stop(M2C,id).delta)
  +SignalLow(O3,id).CaptureValue(M2E,id).ExecuteProfile(X6,M2C,id)
   .Off(C3,id).Instance19(id)

Instance19(id:InstanceID) =
  SignalLow(O3,id).
  (NotTooHigh(M2E,id).Instance19(id) + TooHigh(M2E,id).Stop(M2C,id).delta)
  +SignalHigh(O3,id).CaptureValue(M2E,id).InformAfter(M2E,d3,id)
   .Instance20(id)

Instance20(id:InstanceID) =
  PosReached(M2E,id).ExecuteProfile(X7,M2C,id).On(C4,id)
  .InformAfter(M2E,d4,id).Instance21(id)

Instance21(id:InstanceID) =
  PosReached(M2E,id).Stop(M2C,id).On(M4,id).InformIn(t8,id)
  .Instance22(id)

Instance22(id:InstanceID) =
  Timeout(id).Off(C4,id).Off(M4,id).InformIn(t9,id).Instance23(id)

Instance23(id:InstanceID) =
  Timeout(id).Instance24(id)

%% ----- Scanning and finalization ----- %%

Instance24(id:InstanceID) =
  reserve(sem3,id).On(C3,id).InformIn(t10,id).Instance25(id)

Instance25(id:InstanceID) =
  Timeout(id).ENTER_M3(id).ExecuteProfile(X8,M2C,id)
  .ExecuteProfile(X9,M3C,id).ENTER_K7(id).CaptureValue(M3E,id)
  .Instance26(id)
```

```
      Instance26(id:InstanceID) =
        SignalLow(O4,id).
        ( NotTooHigh(M3E,id).Instance26(id)
          +TooHigh(M3E,id).Stop(M2C,id).Stop(M3C,id).delta )
        +SignalHigh(O4,id).Instance27(id)

      Instance27(id:InstanceID) =
        SignalHigh(O3,id).
        ( NotTooHigh(M2E,id).Instance27(id)
          +TooHigh(M2E,id).Stop(M2C,id).Stop(M3C,id).delta )
        +SignalLow(O3,id).LEAVE_K3(id).Stop(M2C,id).Instance28(id)

      Instance28(id:InstanceID) =
        Stopped(M2C,id).LEAVE_M2(id).release(sem2,id).LEAVE_K7(id)
        .Instance29(id)

      Instance29(id:InstanceID) =
        SignalHigh(O4,id).
        ( NotTooHigh(M3E,id).Instance29(id)
          +TooHigh(M3E,id).Stop(M3C,id).delta )
        +SignalLow(O4,id).InformAfter(M3E,d5,id).Instance30(id)

      Instance30(id:InstanceID) =
        PosReached(M3E,id).Stop(M3C,id).Instance31(id)

      Instance31(id:InstanceID) =
        Stopped(M3C,id).LEAVE_M3(id).InformIn(t11,id).Instance32(id)

      Instance32(id:InstanceID) =
        Timeout(id).reserve(semN,id).release(sem3,id).Instance33(id)

      Instance33(id:InstanceID) =
        semValueZero(semN,F,id).Instance(id)
        +semValueZero(semN,T,id).ENTER_M3(id).ExecuteProfile(X9,M3C,id)
         .InformIn(t12,id).Instance34(id)

      Instance34(id:InstanceID) =
        Timeout(id).Stop(M3C,id).setFirstSheet(T,id).Off(M5,id).Instance35(id)

      Instance35(id:InstanceID) =
        Stopped(M3C,id).ActivateIdleState(M3C,id).ActivateIdleState(M1C,id)
        .ActivateIdleState(M2C,id).LEAVE_M3(id).Instance(id)

init
      encap( {setFirstSheet,getFirstSheet,reserve,release,activate,semValueZero},

        FirstSheet(T) || Instance0(I0) || Instance(I1) || Instance(I2)
        || Semaphore(sem1,S(0)) || Semaphore(sem2,S(0)) || Semaphore(sem3,S(0))
        || Semaphore(semN,0)
      )
```

## C Revised μCRL specification

This is the μCRL specification of the ADF system in which all errors that were found in the verification process, have been fixed.

```
sort  Bool
func  T,F: ->Bool
map   not:Bool->Bool
      and,or,eq:Bool#Bool->Bool
var   b:Bool
rew   not(F)=T    not(T)=F
      and(b,T)=b  and(b,F)=F
      or(b,T)=T   or(b,F)=b
      eq(b,b)=T   eq(T,F)=F   eq(F,T)=F

sort  Nat
func  0:->Nat
      S:Nat->Nat
map   gt,eq:Nat#Nat->Bool
```

```
      dec:Nat->Nat
var   m,n:Nat
rew   gt(0,n)=F gt(S(m),0)=T gt(S(m),S(n))=gt(m,n)
      eq(0,0)=T eq(0,S(n))=F eq(S(n),0)=F eq(S(m),S(n))=eq(m,n)
      dec(S(n))=n

sort  SensorID
func  O1,O2,O3,O4:->SensorID

sort  OnOffID
func  S1,C1,C2,C3,C4,M4,M5:->OnOffID

sort  ControllerID
func  M1C,M2C,M3C:->ControllerID

sort  EncoderID
func  M1E,M2E,M3E:->EncoderID

sort  ProfileID
func  X1,X2,X3,X4,X5,X6,X7,X8,X9:->ProfileID

sort  InstanceID
func  I0,I1,I2: -> InstanceID
map   eq:InstanceID#InstanceID->Bool
      next,prev:InstanceID->InstanceID
rew   eq(I0,I0)=T eq(I0,I1)=F eq(I0,I2)=F
      eq(I1,I0)=F eq(I1,I1)=T eq(I1,I2)=F
      eq(I2,I0)=F eq(I2,I1)=F eq(I2,I2)=T
      next(I0)=I1 prev(I0)=I2
      next(I1)=I2 prev(I1)=I0
      next(I2)=I0 prev(I2)=I1

sort  SemaphoreID
func  sem1,sem2,sem3,sem4,sem5,semN:->SemaphoreID
map   eq:SemaphoreID#SemaphoreID->Bool
rew   eq(sem1,sem1)=T eq(sem1,sem2)=F eq(sem1,sem3)=F eq(sem1,sem4)=F
      eq(sem1,sem5)=F eq(sem1,semN)=F
      eq(sem2,sem1)=F eq(sem2,sem2)=T eq(sem2,sem3)=F eq(sem2,sem4)=F
      eq(sem2,sem5)=F eq(sem2,semN)=F
      eq(sem3,sem1)=F eq(sem3,sem2)=F eq(sem3,sem3)=T eq(sem3,sem4)=F
      eq(sem3,sem5)=F eq(sem3,semN)=F
      eq(sem4,sem1)=F eq(sem4,sem2)=F eq(sem4,sem3)=F eq(sem4,sem4)=T
      eq(sem4,sem5)=F eq(sem4,semN)=F
      eq(sem5,sem1)=F eq(sem5,sem2)=F eq(sem5,sem3)=F eq(sem5,sem4)=F
      eq(sem5,sem5)=T eq(sem5,semN)=F
      eq(semN,sem1)=F eq(semN,sem2)=F eq(semN,sem3)=F eq(semN,sem4)=F
      eq(semN,sem5)=F eq(semN,semN)=T

sort  TimePeriod
func  t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12:->TimePeriod

sort  Distance
func  d1,d2,d3,d4,d5:->Distance

act   ENTER_K3, LEAVE_K3, LEAVE_K2, ENTER_K7, LEAVE_K7, ENTER_M1, LEAVE_M1,
      ENTER_M2, LEAVE_M2, ENTER_M3, LEAVE_M3, Timeout: InstanceID
      setFirstSheet, SetFirstSheet, getFirstSheet,GetFirstSheet:Bool#InstanceID
      reserve,Reserve,release,Release:SemaphoreID#InstanceID
      semValueZero,SemValueZero:SemaphoreID#Bool#InstanceID
      activate,Activate:InstanceID#InstanceID
      SignalLow,SignalHigh:SensorID#InstanceID
      On,Off:OnOffID#InstanceID
      ActivateControlState,ActivateIdleState,PosReached,Stop,Stopped:
      ControllerID#InstanceID
      CaptureValue,PosReached,NotTooHigh,TooHigh:EncoderID#InstanceID
      ExecuteProfile:ProfileID#ControllerID#InstanceID
      InformIn:TimePeriod#InstanceID
      InformAfter:EncoderID#Distance#InstanceID

comm  setFirstSheet | setFirstSheet = SetFirstSheet
      getFirstSheet | getFirstSheet = GetFirstSheet
      reserve | reserve = Reserve
      release | release = Release
      semValueZero | semValueZero = SemValueZero
      activate | activate = Activate

proc  FirstSheet(b:Bool) =
        sum(i:InstanceID,
          sum(b1:Bool, setFirstSheet(b1,i).FirstSheet(b1))
          +getFirstSheet(b,i).FirstSheet(b) )

      Semaphore(sid:SemaphoreID,n:Nat) =
        sum(i:InstanceID,
          reserve(sid,i).Semaphore(sid,dec(n)) <| gt(n,0) |> delta
          +release(sid,i).Semaphore(sid,S(n))
          +semValueZero(sid,eq(n,0),i).Semaphore(sid,n) )

      %% ----- Initialization and aligning ----- %%

      Instance(id:InstanceID) =
        activate(id,prev(id)).Instance0(id)

      Instance0(id:InstanceID) =
        SignalLow(O1,id).Instance0(id)
        + SignalHigh(O1,id).reserve(sem4,id).
          ( getFirstSheet(T,id).InformIn(t1,id).Instance1(id)
           +getFirstSheet(F,id).ENTER_M1(id).Instance4(id) )

      Instance1(id:InstanceID) =
        Timeout(id).
        (SignalLow(O1,id).release(sem4,id).Instance0(id)
        +
        SignalHigh(O1,id).ENTER_M1(id).On(M5,id).On(C2,id).Off(C1,id).On(S1,id)
        .ActivateControlState(M1C,id).ActivateControlState(M2C,id)
        .ActivateControlState(M3C,id).setFirstSheet(F,id).InformIn(t2,id)
        .Instance2(id))

      Instance2(id:InstanceID) =
        Timeout(id).ExecuteProfile(X1,M1C,id).Instance3(id)

      Instance3(id:InstanceID) =
        PosReached(M1C,id).Instance4(id)

      %% ----- Separation ----- %%

      Instance4(id:InstanceID) =
        reserve(sem1,id).Off(S1,id).On(C2,id).On(C1,id).release(semN,id)
        .release(sem4,id).InformIn(t3,id).Instance5(id)

      Instance5(id:InstanceID) =
        Timeout(id).ExecuteProfile(X2,M1C,id).Instance6(id)

      Instance6(id:InstanceID) =
        PosReached(M1C,id).Off(C2,id).InformIn(t4,id).Instance7(id)

      Instance7(id:InstanceID) =
        Timeout(id).On(S1,id).On(C2,id).Off(C1,id).InformIn(t5,id)
        .Instance8(id)

      Instance8(id:InstanceID) =
        Timeout(id).ExecuteProfile(X3,M1C,id).CaptureValue(M1E,id)
        .InformAfter(M1E,d1,id).Instance9(id)

      Instance9(id:InstanceID) =
        PosReached(M1E,id).On(C1,id).InformAfter(M1E,d2,id).Instance10(id)

      Instance10(id:InstanceID) =
        PosReached(M1E,id).Off(S1,id).Instance11(id)

      Instance11(id:InstanceID) =
        SignalLow(O2,id).
        (NotTooHigh(M1E,id).Instance11(id) + TooHigh(M1E,id).Stop(M1C,id).delta)
        +SignalHigh(O2,id).Off(C1,id).InformIn(t6,id).Instance12(id)

      Instance12(id:InstanceID) =
        Timeout(id).On(C2,id).InformIn(t7,id).Stop(M1C,id).Instance13(id)

      Instance13(id:InstanceID) =
        (Timeout(id).Stopped(M1C,id)+Stopped(M1C,id).Timeout(id)).Instance14(id)

      %% ----- Turning and blousing ----- %%

      Instance14(id:InstanceID) =
        reserve(sem2,id).ENTER_M2(id).CaptureValue(M2E,id)
        .ExecuteProfile(X4,M1C,id).ExecuteProfile(X5,M2C,id).Instance15(id)

      Instance15(id:InstanceID) =
        SignalHigh(O2,id).
          (SignalHigh(O3,id).ENTER_K3(id).CaptureValue(M2E,id)
           .Instance16(id)
           +SignalLow(O3,id).
            (NotTooHigh(M2E,id).Instance15(id)
             +TooHigh(M2E,id).Stop(M1C,id).Stop(M2C,id).delta))
        +SignalHigh(O3,id).ENTER_K3(id).CaptureValue(M2E,id)
        .Instance16(id)

      Instance16(id:InstanceID) =
        SignalHigh(O2,id).Instance16(id)
        +SignalLow(O2,id).LEAVE_K2(id).Stop(M1C,id).CaptureValue(M2E,id)
         .Instance17(id)

      Instance17(id:InstanceID) =
        Stopped(M1C,id).LEAVE_M1(id).activate(next(id),id).release(sem1,id)
        .Instance18(id)

      Instance18(id:InstanceID) =
        SignalHigh(O3,id).
        (NotTooHigh(M2E,id).Instance18(id) + TooHigh(M2E,id).Stop(M2C,id).delta)
        +SignalLow(O3,id).Stop(M2C,id).Instance18A(id)

      Instance18A(id:InstanceID) =
        Stopped(M2C,id).reserve(sem5,id).CaptureValue(M2E,id)
        .ExecuteProfile(X6,M2C,id).Off(C3,id).Instance19(id)

      Instance19(id:InstanceID) =
        SignalLow(O3,id).
        (NotTooHigh(M2E,id).Instance19(id) + TooHigh(M2E,id).Stop(M2C,id).delta)
        +SignalHigh(O3,id).CaptureValue(M2E,id).InformAfter(M2E,d3,id)
         .Instance20(id)

      Instance20(id:InstanceID) =
        PosReached(M2E,id).ExecuteProfile(X7,M2C,id).On(C4,id)
        .InformAfter(M2E,d4,id).Instance21(id)

      Instance21(id:InstanceID) =
        PosReached(M2E,id).Stop(M2C,id).On(M4,id).InformIn(t8,id)
        .Instance22(id)

      Instance22(id:InstanceID) =
        Timeout(id).Off(C4,id).Off(M4,id).InformIn(t9,id).Instance23(id)

      Instance23(id:InstanceID) =
        Timeout(id).Instance24(id)

      %% ----- Scanning and finalization ----- %%

      Instance24(id:InstanceID) =
        reserve(sem3,id).On(C3,id).InformIn(t10,id).Instance25(id)

      Instance25(id:InstanceID) =
        Timeout(id).ENTER_M3(id).ExecuteProfile(X8,M2C,id)
        .ExecuteProfile(X9,M3C,id).ENTER_K7(id).CaptureValue(M3E,id)
        .Instance26(id)

      Instance26(id:InstanceID) =
        SignalLow(O4,id).
        ( NotTooHigh(M3E,id).Instance26(id)
          +TooHigh(M3E,id).Stop(M2C,id).Stop(M3C,id).delta )
        +SignalHigh(O4,id).Instance27(id)

      Instance27(id:InstanceID) =
        SignalHigh(O3,id).
```

```
        ( NotTooHigh(M2E,id).Instance27(id)
          +TooHigh(M2E,id).Stop(M2C,id).Stop(M3C,id).delta )
        +SignalLow(O3,id).LEAVE_K3(id).Stop(M2C,id).Instance28(id)

      Instance28(id:InstanceID) =
        Stopped(M2C,id).LEAVE_M2(id).release(sem2,id).LEAVE_K7(id)
        .Instance29(id)

      Instance29(id:InstanceID) =
        SignalHigh(O4,id).
        ( NotTooHigh(M3E,id).Instance29(id)
          +TooHigh(M3E,id).Stop(M3C,id).delta )
        +SignalLow(O4,id).release(sem5,id).InformAfter(M3E,d5,id).Instance30(id)

      Instance30(id:InstanceID) =
        PosReached(M3E,id).Stop(M3C,id).Instance31(id)

      Instance31(id:InstanceID) =
        Stopped(M3C,id).LEAVE_M3(id).InformIn(t11,id).Instance32(id)

      Instance32(id:InstanceID) =
        Timeout(id).reserve(semN,id).release(sem3,id).reserve(sem4,id)
        .Instance33(id)

      Instance33(id:InstanceID) =
        semValueZero(semN,F,id).release(sem4,id).Instance(id)
        +semValueZero(semN,T,id).ENTER_M3(id).ExecuteProfile(X9,M3C,id)
         .InformIn(t12,id).Instance34(id)

      Instance34(id:InstanceID) =
        Timeout(id).Stop(M3C,id).setFirstSheet(T,id).Off(M5,id).Instance35(id)

      Instance35(id:InstanceID) =
        Stopped(M3C,id).ActivateIdleState(M3C,id).ActivateIdleState(M1C,id)
        .ActivateIdleState(M2C,id).LEAVE_M3(id).release(sem4,id).Instance(id)

init
      encap( {setFirstSheet,getFirstSheet,reserve,release,activate,semValueZero},

        FirstSheet(T) || Instance0(I0) || Instance(I1) || Instance(I2)
        || Semaphore(sem1,S(0)) || Semaphore(sem2,S(0)) || Semaphore(sem3,S(0))
        || Semaphore(sem4,S(0)) || Semaphore(sem5,S(0)) || Semaphore(semN,0)
      )
```