Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2008-11

Supervisor for toner error handling: a case study in supervisory control of Océ printers

M. Petreczky D.A. van Beek
J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2008-11
Eindhoven, December 2008
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

The purpose of this report is to demonstrate the viability of supervisory control synthesis approach by presenting the formulation and a preliminary solution of a real-life control problem in Océ printers. In a nutshell, supervisory control synthesis is a procedure for automatic generation of control algorithms based on the formal model of the underlying system (plant) and of the requirements the controlled system has to satisfy. The underlying theory guarantees that the generated control algorithm will indeed force the system to meet the specified requirements, provided that the model of the system and of the requirements are accurate enough. The viability of the approach is demonstrated by applying it to a particular use-case of Océ, the toner error-handling problem.

The presence of continuous-time behavior in the current use-case compelled us to use supervisory control theory in a novel way. Usually, when applying supervisory control theory, the plant is modeled as an automaton and this automaton is constructed manually. Instead, here we generate the finite-state automaton model by a computer program, which takes as inputs the value of a number of physical parameters. The reason for choosing to generate the model by a program is that the model is obtained by discretizing a hybrid model in time. In turn, the time-step used in the discretization of the hybrid model is one of the parameters of the computer program. By generating the model automatically, we are able to experiment with supervisors for different discretization time steps and different values of physical parameters. [1]

[1] This work was done as part of the ITEA project Twins 05004. The authors thank Ronald Fabel and Lou Somers from Océ R&D, Venlo, The Netherlands, for their help and cooperation.

Contents

1 Introduction

The goal of the paper is to present the application of supervisory control theory to the problem of toner error-handling in Océ printers. This is intended as a proof that supervisory control can be used for designing high-level control for complex machines. The emphasis is more on the "proof of concept" aspect rather than on actually deploying the obtained supervisor. In fact, the *use-case of the paper has already been successfully solved by Océ*. What we would like to demonstrate in this paper that the same use-case can also be solved using supervisory control theory. The potential reason for using supervisory control as opposed to manual design is that it allows for automatic generation of a solution for various combination of the problem parameters. Hence, coming up with a solution when the design parameters changes can be done much faster if supervisory control is used. In addition, it provides a multitude of solution, from which the solution matching certain additional criteria can be extracted. We hope that this will provide a sufficient motivation to investigate the applicability of supervisory control for other uses cases.

Control theory has been around for more than a half-century. It has been actively applied in almost every branch of industry, from aerospace engineering to medical technology. Control theory/engineering methods are also applied for the design of complex high-tech systems such as printers, copy machines, chip manufacturing lines. However, when it comes to complex machines, control is usually used locally, to ensure correct functioning of some small physical component. Engines which drive the paper path or conveyor belt represent an apt example of components of complex machines to which control theory is extensively applied.

The current paper is driven by a different vision on the scope of application of control theory. Namely, instead of assigning control theory a relatively modest role in controlling small hardware components, we would like to elevate it to one of the central paradigms for system design. More precisely, *we propose to view the problem of designing the software controlling the global behavior of the machine as the problem of designing a controller which achieves certain control objectives.* In addition, *we propose to generate the control algorithm and the control software automatically from the formal model of the plant and of the requirements.* The argument in favor of this approach is that it should considerably reduce the length of the developmental cycle of the product, as it facilitates reduction in the number of design-test-redesign loops.

On a more detailed level, the proposed approach can be summarized as follows. We distinguish between the uncontrolled physical system (referred to as *plant*) and the control software. Our goal is to generate the control software such that if the generated control software is coupled with the physical system, the resulting entity exhibits the desired behavior. There are many ways to formulate what is meant by the desired behavior. One possibility is to state it as conditions on the input-output map. More precisely, the dynamic behavior of the machines is usually determined by two kinds of input signals: *user input* and *machine control input*. The former is essentially the input provided by the user or external environment. For example, a possible user input for printers is sending an order to the printer to print a certain document on a certain paper format. The machine control inputs are the ones which can be used by the controller to change the behavior of the system. For example, setting the speed of the engines driving the paper path to a certain value is a machine control input. The machine also produces outputs. An output can be for example the presence/absence of a paper sheet in the finisher, or a signal from a sensor in the paper path. The schematic picture can be seen on Figure 1 for the general case and on Figure 2 for the case of printers. The desired behavior can be formulated as a relation describing what kind of output the machine is supposed to produce for a

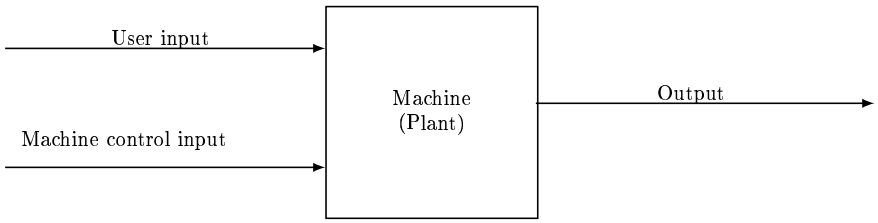

Figure 1: Machines as abstract input-output systems

specific user input. For example, we could demand that five seconds after the arrival of the order to print a document, the first sheet of this document should lie in the finisher.

In order to achieve the desired user input/output behavior, we need to design a controller. In the vast majority of cases, the controller is implemented as a piece of software. The controller is allowed to use information on the user inputs, the output produced by the system and perhaps the states of the system. Its task is to choose the machine control inputs in such a way, that the resulting output meets the requirements. The interconnection is best illustrated by the picture in Figure 3.

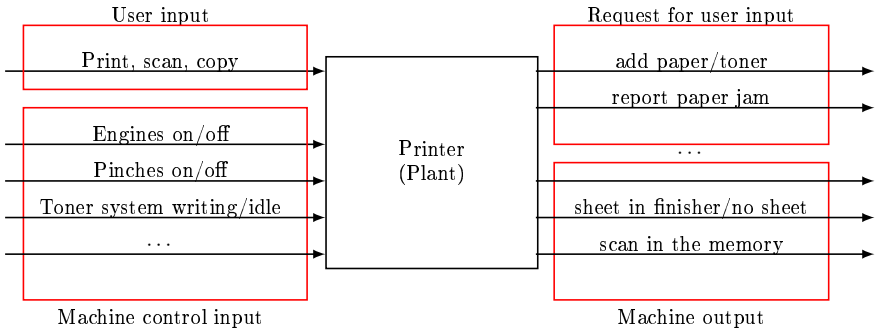

Figure 2: Machines as abstract input-output systems: examples of printers

With the point view explained above in mind, the main problem of software design for machines can be reformulated as follows.

Problem 1 (Design as control-problem). Assume that a model $\mathcal{P}$ of the plant is given and assume that a formal specification $\mathcal{S}$ of the requirements is given. Furthermore, assume that the requirements are formulated as properties which should be satisfied by user input/machine output pairs. The task is to generate a controller $\mathcal{C}$ such that the closed-loop system on Figure 3 satisfies the requirements S.

The problem formulated above indeed looks similar to a classical control problem. However, in the case of complex machines the inputs and the outputs can be both discrete and continuous. This calls for methods and techniques which can deal with control systems exhibiting both continuous and discrete behavior. Such methods and techniques exist and they are known under the name of control theory for discrete-event systems and hybrid systems [10, 13, 12, 11, 2].

Remark 1 (Controlled synthesis as design procedure). Notice that the formulation of Problem 1 implies that the generated controller $\mathcal{C}$ should be *correct by design*. That

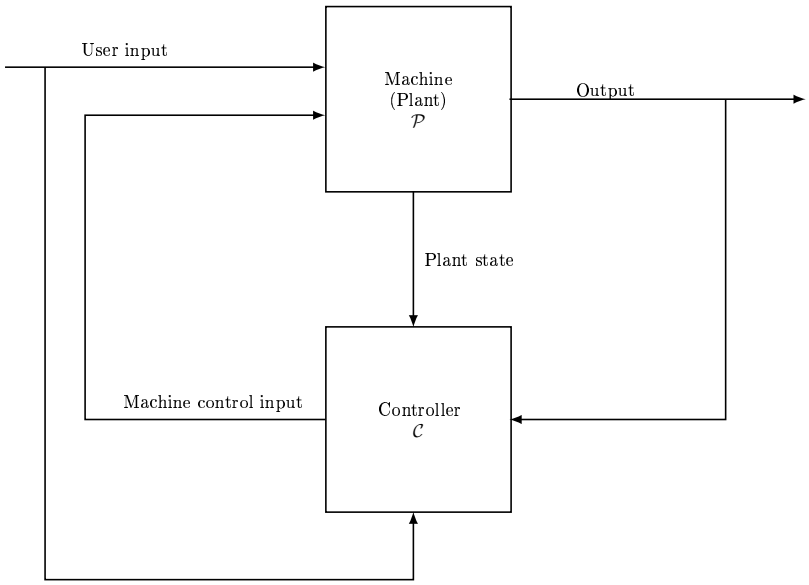

Figure 3: Control of complex machines

is, if the mathematical model of the plant and of the requirements is accurate enough, then the controller will enforce the desired behavior of the system. This means that in contrast to the case when the controller is designed by hand, no testing and debugging of the controller is required. The only purpose of testing is to find out whether the plant model is realistic enough and whether the model of the requirements formalizes all the relevant features. The overall design procedure is then as follows.

1. Build a mathematical model $\mathcal{P}$ of the plant.
2. Build a mathematical model $\mathcal{S}$ of the requirements.
3. Generate a controller $\mathcal{C}$ which is correct by construction.
4. Test the closed-loop system (see Figure 3).
5. If the test results indicate incorrect functionality, then there can be only two causes for this; either the model $\mathcal{P}$ of the plant is inadequate, or the model $\mathcal{S}$ of the requirements is inadequate. Improve the model of the plant or the model of the requirements and repeat the steps above again.
6. If the outcome of the tests is satisfactory, then the design is ready.

The proposed approach also fits well into the more general paradigm of *model-based engineering*. The main idea of model-based engineering can be formulated as follows. Instead of making prototypes of the system, the engineers are expected to make mathematical models of the system behavior and to test/verify their design choices by analyzing the impact of the choices on the behavior of the models. One is expected to built and test the prototype only after the the design is proved to work on the level of models.. The main rational behind this approach is that it is expected to reduce the costs and length of the developmental cycle by facilitating discovery of design faults on an early stage of development, before actual prototypes are built.

In this report we will mainly use the classical supervisory control theory for discrete-event systems, pioneered by Ramadge and Wonham, [10, 13, 12, 2] for generating a controller which is correct by construction. The main feature of this theory is that it is based on the well-known theory of finite-state automata and formal languages. There are several algorithms and tools available which can solve control problem described in Problem 1, if the requirements and the plant model are both formulated in the framework of finite-state automata. Please note that *according to the terminology of supervisory control theory, the controller is referred to as supervisor. In this paper we will use the two terms interchangeably.*

Although the main tool for generating the supervisor is the classical theory by Ramadge and Wonham [10, 13, 12, 2], the approach we took differs significantly from the usual approach for applying supervisory control theory. Namely, instead of building a finite-state automaton model of the plant manually, we generate the finite-state automaton model of the plant by a computer program. The reason for this is that the plant exhibits both discrete and continuous behavior, and hence it cannot be modeled directly as a discrete-even system in the sense of [10, 13, 12, 2]. In fact, the most appropriate modeling formalism for the uncontrolled system seems to be the framework of hybrid systems [11]. We solve the resulting challenge by modeling the behavior of the system only on sample times, up to a certain number of time steps. However, the resulting finite-state automaton depends on the number of time steps and the choice of the sampling interval. Hence, it is not at all practical to construct the automaton manually. Instead, an algorithm was devised to generate the finite-state automaton model of the plant. This algorithm takes as input the number of time steps, the sampling times, and the physical parameters of the plant and produces a finite-state automaton model. Subsequently, the thus obtained model is used for supervisory control synthesis. The details of the approach are presented in Section 6.

The outline of the paper is the following. Section 2 presents some general background information on how printers work. Section 3 contains the informal description of the use case treated in the paper. Section 4 presents a brief overview of supervisory control theory. Section 5 presents the main modeling assumptions informally. Section 6 presents the formal finite-state automaton model of the plant, which in this case is going to be the toner system. Section 7 contains the formal model of the requirements. Section 8 describes the procedure for generating the supervisor which is expected to solve the use case. Finally, in Section 9 some conclusions are formulated and plans for future works are sketched.

2 Functioning of a printer

As it is well-know from everyday experience, a printer accepts commands for printing. After a command is accepted the printer fetches a sheet of paper from the paper tray, prints the required image on it and puts it into the so-called *finisher*. This cycle is repeated several times, depending on the command. The subsequent cycles are executed in a concurrent way; the next cycle may start even before the previous one is completed.

During the cycle, the paper sheet moves through a number of stages, each stage being responsible for a certain activity. The whole route of the paper is referred to as the *paper path*. The transportation of the sheets is realized by *pinches*. Each pinch consists of two rollers, One of the rollers is responsible for moving the sheet and we call it the dynamic roller. The other roller (we call it the static roller) is responsible for keeping the sheet on the paper path, i.e. to make sure that the paper would not slide from the dynamic roller.

The dynamic roller revolves around its axis and it is driven by an engine. In contrast, the static roller is unable to move independently, it can just revolve around its axis when it is driven by the movement of the sheet. In this way the paper is "pushed" forward by the movement of the dynamic roller. Several rollers may be driven by the same engine through mechanical transmission mechanisms. The distance between the pinches should not be greater than the length of the smallest sheet which the printer is ready to handle, otherwise the paper will fall off the paper path.

The paper is transported from the paper tray to the *fuse pinch* and subsequently it is transported from the fuse pinch to the *finisher*. The fuse pinch is a pinch on the paper path where the image to be printed is put on the sheet. More precisely, this is the point where the tape containing the image meets the paper sheet. The tape which contains the image is called the *toner belt* or *TTF belt*. After being treated in the fuse, the paper gets transferred to the *finisher*, which is the place it can be picked up from.

In this paper we will focus on the toner system, that is the part of the printer which is responsible for creating the toner image of what is to be printed. As it was pointed out above, the TTF belt transports the toner to the paper sheet. The picture gets on the TTF belt as follows. There is a *OPC belt* , on which the picture is put by a writing unit. This unit is controlled by the imaging software. The OPC belt revolves around its axis and the image gets written on its surface. From the revolving OPC belt , the image gets on the TTF belt as follows. The TTF belt forms a closed loop which revolves around its center. That is, each point of the TTF belt makes a full cycle. The stages of this cycle are the following;

fuse that is where the TTF belt is pressed against the paper

cleaner that is where the toner belt is cleaned.

transfer pinch that is where the image gets from the OPC belt onto the TTF belt. The image is first written onto the surface of the OPC belt , and then as the OPC belt revolves, it gets to the point where the OPC belt meets the TTF belt, the so called *transfer pinch*.

The TTF belt and the OPC belt are both powered by separate engines, hence they can be moved independently of each other. The fuse pinch can be lifted from the TTF belt and the transfer pinch can be lifted from the TTF belt.

There are certain general considerations which should be kept in mind when designing control algorithms for printers. These considerations are the following.

High throughput The throughput (sheets per second) of the paper should be as high as possible.

Low number of service calls The amount of service calls should be as low as possible. Service calls might be caused by those paper jams for which the paper gets stuck in a section of the paper path which is unavailable for the user, or by TTF belt problems (polluted OPC belt , etc.).

Trade-off between costs and performance of the controller Since controlling the system is essential for its correct functioning, the design choices can be altered if it improves the performance of the control algorithms. However, the altered design choices might increase the cost of the machine. Hence, there is a clear trade-off between control performance and costs.

3 The 'Sheet too late scenario' use case

The goal of this section is to describe the use case in more detail but in an informal manner. The structure of the section is the following. In Subsection 3.1 we present a detailed but informal description of the problem constituting the use-case. In Subsection 3.2 we will state the requirements which should be met by the controlled system. That is, we will describe what should be achieved by a potential control algorithm. In Subsection 3.3 we will describe the sensor information which is available for the potential controller to accomplish the task. In Subsection 3.4 we will present the list of possible control actions which are available to the controller. In Subsection 3.5 we will explain the relevance of the described problem. Finally, in Subsection 3.6 we will briefly outline the proposed solution to the problem formulated in the use-case.

3.1 Description of the scenario

As mentioned above, the image gets on the sheet when the sheet is in the fuse. By the time the portion of the TTF belt which contains the image arrives at the fuse, the sheet should be in the right position. The necessary synchronization is achieved by starting the process of writing onto the TTF long before the sheet is near the fuse. The time when the writing process starts is determined by the *estimated arrival time of the sheet* into the fuse. The estimated arrival time depends on a number of factors. For example, it can depend on the size of the sheet, the speed of the paper tray and in general on the speed of transportation of the section of the paper path which precedes the fuse. However, sometimes the sheet does not arrive at the estimated time. This happens when there is a paper jam.

If the sheet does not arrive on time to the fuse and the TTF belt proceeds as planned, then there will be no sheet at the fuse when the section of the TTF belt containing the image arrives to the fuse. If this is not dealt with, then the image on the TTF belt is be pressed against the pinch of the fuse, instead of the paper. The pinch gets polluted and in turn it pollutes that portion of the TTF belt which passes along the pinch. This kind of pollution can not be cleaned by the cleaner. From the TTF belt the pollution gets onto the OPC belt . The polluted OPC belt will result in a bad quality printout (stripes on the printout). The only solution for the latter one is to call the service and replace the OPC belt . Moreover, the polluted fuse pinch might pollute the paper directly as well.

3.2 Control Objectives

The task is to prevent the portion of the TTF belt containing the image from coming into contact with the fuse pinch, if there is no paper in the fuse.

3.3 Relevant sensors

The only relevant sensor is the so called *X-fine sensor* . This sensor is situated before the fuse. The sensor works in the following manner. The sensor start scanning the environment for presence of a sheet some time before the expected arrival time. If no sheet is detected within some time range after the expected arrival time, then the control software of the printer sends a signal indicating that no sheet arrived. This signal will be referred to as the **Sheet too late** signal . This is the only sensor-based information available for solving the use case.

3.4 Possible control actions and constraints

There are several control actions which can be performed in order to prevent contamination of the fuse pinch. Below we will list some of them and describe the constraints on the execution of each of these actions.

Lift the fuse pinch from the TTF belt. It takes some time to accomplish the action. Moreover, if the speed of the TTF belt prior to lifting the fuse pinch is too low, then the fuse pinch has the tendency to stick to the surface of the TTF belt. In general, the faster the TTF belt, the less chance there is that the TTF belt and the fuse pinch will stick. There is a safe interval, such that if the speed of the TTF belt is within this interval then it is highly unlikely that the TTF belt and the fuse pinch would stick. There is a pinch after the fuse. If there is a sheet in the fuse, and the front end of it has not reached the pinch after the fuse, then the pinch at the fuse cannot be lifted, because then the sheet will not stay on the paper path. In addition, if the fuse pinch is lifted, then it is impossible to slow down the TTF tape properly, as the friction caused by the pinch significantly contributes to slowing down the TTF tape.

Lift the OPC belt from the TTF belt. It takes time to remove the OPC belt from the TTF belt.

Stop writing on the OPC belt . It takes time to stop writing on the OPC belt .

Slow down the TTF belt. If the fuse pinch is lifted when the TTF belt is too slow, then the pinch fuse has the tendency to stick to the TTF belt. See *Lift the fuse pinch from the TTF belt* for more information. If the TTF belt is too slow, then the cleaner will not clean properly. A safe speed interval exists.

3.5 Relevance of the problem

The main rational behind trying to prevent the contamination of the fuse pinch is the following. If the fuse pinch gets contaminated, then it leads to a service call, that is, service department has to send out someone to fix the problem. This increases the costs for Océ and it costs the user time and printing capacity.

3.6 Proposed solution

Based on the specification and constraints we propose to generate the error-handling algorithm automatically. That is, we would like to design an algorithm such that the algorithm takes as inputs the concrete values for the speed and time constraints specified above. The output of the algorithm is the error-handling algorithm. The error-handling algorithm produces a timed sequence of control actions which have to be carried out in order to avoid fuse pinch contamination. It can also happen that under the specific constraints it is impossible to avoid fuse pinch contamination in all the possible cases. The algorithm generating the error-handling algorithm should be able to recognize such a situation. The generated error-handling algorithm will be correct by construction, i.e. if the formulated constraints are accurate enough, then the error-handling algorithm will *always* prevent fuse pinch contamination. The generation of the error-handling algorithm will be based on *supervisory control theory.*

4 Review of supervisory control theory

The purpose of this section is to present the necessary background knowledge on supervisory control theory. This is necessary for understanding the technical details of the solution of the use case. For a more detailed account on supervisory control see [10, 13, 12, 2], and on automata theory and formal languages [3, 4, 5].

The theory is centered around the solution Problem 1. However, in this case the model of the plant and of the requirements is assumed to be a finite-state automaton. In addition, in this case the controller will be called *supervisor*. The system to be controlled is assumed to react to two kinds of events: *uncontrollable events and controllable events*. Uncontrollable events are events which are beyond our control. The most typical examples are error conditions. Using the framework outlined in Section 1 as a reference point, controllable events correspond to machine control, and uncontrollable events correspond to user inputs and machine outputs. For example, whether the sheet arrives within the expected time range to the sensor or not is an uncontrollable event as we cannot directly prevent it. Controllable events are events which can be triggered deliberately. In our example, lifting the pinch, lifting the OPC belt , or slowing down the TTF belt are controllable events. We can trigger them at will.

In order to present the problem formulation and the theory rigorously, we need to recall the notion of finite-state automata.

Definition 1 *(Finite-state automata,[12, 10, 12, 2, 5]).* A finite-state automaton is a tuple

$$\mathcal{A} = (Q, E, \delta, F, q_0) \tag{1}$$

where

Q is the finite state-space of $\mathcal{A}$.

E is the set of discrete events of $\mathcal{A}$. As it is customary in supervisory control theory, it is assumed that E is the disjoint union of the set of controllable events E_c and uncontrollable events E_{uc}, i.e. $E = E_c \cup E_{uc}$ and $E_c \cap E_{uc} = \emptyset$.

$\delta : Q \times E \to Q$ is a partial map, the so called *state-transition* map of $\mathcal{A}$.

$F \subseteq Q$ is the set of accepting states.

$q_0 \in Q$ is the initial state.

Remark 2 (Terminology). In the supervisory control literature the accepting states of $\mathcal{A}$ are often referred to as the *marked states*.

Intuitively, one can think of $\mathcal{A}$ as a state-machine, which at every step receives or generates a discrete event $e \in E$ and depending on this discrete event, it changes its current state q to the state $\delta(q, e)$ defined by the state-transition map. Note that it may happen that for some $e \in E$, $\delta(q, e)$ is not defined, i.e. if the machine is in state q, then it cannot receive or generate the discrete event $e \in E$. The initial state q_0 is the state in which the machine resides when it starts operating. The set of accepting states F has the following intuitive interpretation; if the machine is in state $q \in F$, then it means that the machine has just accomplished some task, for example, the assembly of some product has just been finished.

For systems described by finite-state automata, the input-output behavior is represented by the *language generated/accepted by the automaton*. Before presenting the formal definition of these concepts, additional notation is needed. For the finite set E of discrete

events denote by E^* the set of all finite sequences of elements of E, including the empty sequence. This is a standard notation in automata theory, see [5, 3, 4]. That is, a typical element of E^* is a sequence $w = e_1e_2\cdots e_k$, where $e_1, e_2, \ldots, e_k \in E$, $k \geq 0$. If $k = 0$, then w is the empty sequence and it is denoted by ϵ. . If $v = f_1f_2\cdots f_l \in E^*$, $f_1, f_2, \ldots, f_l \in E$, $l \geq 0$ is another sequence, then wv denotes the *concatenation of* w *and* v and it is defined as follows; $wv = e_1e_2\cdots e_kf_1f_2\cdots f_l \in E^*$. Notice that if $v = \epsilon$, then $wv = w$ and if $w = \epsilon$, then $wv = v$.

We can extend the definition of δ to act on sequences of discrete events. More precisely, define the partial map

$$\hat{\delta} : Q \times E^* \to Q$$

as follows.

- For all $q \in Q$, $\hat{\delta}(q, \epsilon) = q$.
- For all $q \in Q$ and $w = e_1e_2\cdots e_k$, $e_1, e_2, \ldots, e_k \in E$, $k \geq 0$, $\hat{\delta}(q, w) = \hat{q}$ if and only if there exist discrete states $q_0 = q, q_1, q_2, \ldots, q_k = \hat{q}$, such that for each $i = 0, 1, \ldots, k$, $\delta(q_i, e_{i+1})$ is defined and $\delta(q_i, e_{i+1}) = q_i$.

With the abuse of notation, in the sequel we will denote $\hat{\delta}$ simply by δ. Following the convention adopted in [3, 4, 12, 5], we will denote the fact that $\delta(q, w)$ is defined for some state $q \in Q$ and sequence $w \in E^*$ by $\delta(q, w)!$. Now we are ready to define the notion of a language generated/accepted by a finite-state automaton.

Definition 2 *(Language generated/accepted by an automaton, [5, 4, 3, 2, 12]).* Consider an automaton $\mathcal{A}$ of the form (1). The *language* $L(\mathcal{A})$ *generated by the automaton* $\mathcal{A}$ is the following subset of E^*

$$L(\mathcal{A}) = \{w \in E^* \mid \delta(q_0, w)!\}$$

The *language* $L_m(\mathcal{A})$ *accepted by* $\mathcal{A}$ is the following subset of E^*

$$L_m(\mathcal{A}) = \{w \in E^* \mid \delta(q_0, w)! \text{ and } \delta(q_0, w) \in F\}$$

Remark 3 (Marked language). In the supervisory control literature the language $L_m(\mathcal{A})$ is often called the *marked language of* $\mathcal{A}$.

Recall the following terminology from automata theory.

Definition 3 *(Regular languages,[5, 4, 3, 2, 12]).* A set of sequences $K \subseteq E^*$ is called a *regular language*, if there exists a finite-state automaton $\mathcal{A}$ such that the language accepted by $\mathcal{A}$ equals K, i.e. $L_m(\mathcal{A}) = K$. In this case we say that the automaton $\mathcal{A}$ *recognizes* the language K.

One remarkable property of a regular language is that it is completely determined by the automaton which recognizes it. In fact, for the purposes of this paper one can identify regular languages with finite-state automata.

We proceed with recalling the notion of a supervisor for a system (plant) described by a finite state automaton $\mathcal{A}$ of the form (1).

Definition 4 *(Supervisor for* $\mathcal{A}$*, [2, 12, 10, 13]).* A supervisor S for $\mathcal{A}$ is a map of the form

$$S : L(\mathcal{A}) \to 2^{E_c} \tag{2}$$

where 2^{E_c} denotes the set of all subsets of E_c.

In other words, for each sequence $w \in L(\mathcal{A})$, the supervisor chooses a (possibly) empty set of controllable events which the system (plant) may execute. As it was noted earlier, supervisors play the same role as controllers introduced in Section 1. The definition above just expresses the fact that the controller bases its choice of the control input (controllable event) on the past user inputs, outputs and past control inputs.

Obviously, not all supervisors can be implemented in practice. A particular class of supervisors which can be implemented are the supervisors which are implementable by a finite-state automaton.

Definition 5 *(Supervisors implementable by a finite-state automaton).* A supervisor S of the form (2) is *implementable by a finite-state automaton*, if there exists a finite-state automaton $\mathcal{B} = (Q_{sup}, E, \delta_{sup}, F_{sup}, q_{0,sup})$ such that the following holds. For each sequence $w \in L(\mathcal{A})$,

$$S(w) = \{e \in E_c \mid \delta_{sup}(q_{0,sup}, we)!\}$$

and for each uncontrollable event $e \in E_{uc}$, $\delta_{sup}(q_{0,sup}, we)$ is defined. Here Q_{sup} denotes the state-space of $\mathcal{B}$, δ_{sup} denotes the state-transition map of $\mathcal{B}$, F_{sup} is the set of accepting states and $q_{0,sup}$ is the initial state of $\mathcal{B}$. The automaton $\mathcal{B}$ itself is said to be an *implementation* of S.

It is left to define what is the behavior of the interconnection of the plant $\mathcal{A}$ with the supervisor S. The behavior will be defined in terms of languages, i.e. sets of sequences of discrete events generated/accepted by the closed-loop system.

Definition 6 *(Closed loop languages).* Let S be a supervisor of the form (2) and let $\mathcal{A}$ be a finite automaton of the form (1). The *closed-loop language $L(\mathcal{A}/S)$ generated by the interconnection of $\mathcal{A}$ with S* is the smallest set (with respect to set inclusion) of sequences from E^* satisfying the following properties

- $\epsilon \in L(\mathcal{A}/S)$.
- For any $e \in E_{uc}$, if $w \in L(\mathcal{A}/S)$ and $we \in L(\mathcal{A})$, then $we \in L(\mathcal{A}/S)$.
- For any $e \in E_c$, if $w \in L(\mathcal{A}/S)$ and $e \in S(w)$, and $we \in L(\mathcal{A})$, then $we \in L(\mathcal{A}/S)$.

The *marked closed-loop language $L_m(\mathcal{A}/S)$ generated by the interconnection of $\mathcal{A}$ and S* is defined as

$$L_m(\mathcal{A}/S) = L(\mathcal{A}/S) \cap L_m(\mathcal{A})$$

Recall from [12] that if $\mathcal{B}$ is a finite-state automaton which is an implementation of S, then $L(\mathcal{A}/S)$ is simply the language generated by the synchronous parallel product of $\mathcal{B}$ and $\mathcal{A}$.

An important property of supervisors is *non-blockingness*.

Definition 7 *(Non-blocking supervisors,[10, 13, 12, 2]).* A supervisor S of the form (2) is said to be non-blocking, if the following condition holds. For any $w \in L(\mathcal{A}/S)$ there exists $v \in E^*$ such that $wv \in L_m(\mathcal{A}/S)$.

The intuition behind the definition of non-blockingness is the following. The supervisor is non-blocking if any sequence belonging to the closed-loop language can be extended to a sequence in the marked language. That is, the supervisor never leads the plant "astray"; it never drives the plant to a state from which it is impossible to reach an accepting state. In case of manufacturing machines, this would mean that the machine is never driven to a state from which it cannot finish the production process of a particular product.

Now we are ready to state the supervisory control problem formally.

Problem 2 (Supervisory control problem). Let $K \subseteq E^*$ be a regular language which models the *requirements*. Let $\mathcal{A}$ be a finite-state automaton modeling the *plant*. The task is to find the supervisor S such that

- The closed-loop marked language is a subset of K, i.e. $L_m(\mathcal{A}/S) \subseteq K$.
- The supervisor S is non-blocking.
- If V is any other supervisor for $\mathcal{A}$ such that $L_m(\mathcal{A}/V) \subseteq K$ and V is non-blocking, then $L_m(\mathcal{A}/V) \subseteq L_m(\mathcal{A}/S)$.

A supervisor S satisfying the above properties is called the *least restrictive non-blocking supervisor* enforcing the specification K.

The following is a standard result of supervisory control theory [10, 13, 12, 2].

Theorem 1 *(Least restrictive non-blocking supervisors, [10, 13, 12, 2]).* With the assumptions of Problem 2 the following holds. A least restrictive non-blocking supervisor always exists, it is unique and it can be computed from $\mathcal{A}$ and from an arbitrary automaton $\mathcal{B}$ recognizing the language K. Moreover, the least restrictive non-blocking supervisor can be implemented by a finite-state automaton. The algorithm computing the least restrictive supervisor also computes an implementation of this supervisor.

The rational behind computing the least restrictive supervisor is the following. Consider the least restrictive non-blocking supervisor S. It can be shown that essentially any supervisor of interest can be obtained from S by restricting the set of controllable events generated by S. Hence, if we want to obtain a non-blocking supervisor which in addition to enforcing the specification has some other desirable properties, then we can proceed as follows. We find first the least restrictive non-blocking supervisor S and then we try to extract from S a supervisor which has these desirable properties.

One of the potential criteria for the supervisor which is to be extracted is that it should be deterministic. By a *deterministic supervisor* we mean a supervisor V such that for any sequence w, $V(w)$ is either empty or it contains precisely one controllable event. This becomes an issue if the controllable events represent control actions and the supervisor is expected to function as a controller, i.e. it is expected to produce control inputs for the underlying plant. In this case, supervisors which yield several controllable events for some sequence of events are not meaningful, as only one controllable event can actually be executed.

The problem of extracting a deterministic supervisor a supervisor is still largely open. Below we will present one potential extraction scheme. Note that the approach sketched below is only one of the many possible solutions.

Remark 4 (Deterministic supervisors as controllers). One way to extract an implementable deterministic supervisor from the least restrictive supervisor is the following. For each state of the automaton implementing the supervisor S a shortest path to a marked state is computed. One could then define a supervisor V as follows. If for some w the automaton of S is in state q and the shortest path leading from q to a target states starts with a controllable event e, then $V(w)$ allows only e. If the shortest path starts with an uncontrollable event, then $V(w)$ allows no controllable event at all. The resulting supervisor is then deterministic and non-blocking V and it satisfies $L_m(\mathcal{T}/V) \subseteq K$.

Even if a deterministic supervisor is obtained, the implementation of such a supervisor as a controller is not a straightforward task. In fact, the interpretation of a deterministic supervisor as a controller raises several questions. In particular, the role of uncontrollable events and the choice between executing a controllable or an uncontrollable event is not uniquely determined by the existing theory.

The computation of S can be carried out using one of the several tools, for example TCT. The TCT tool is a text-based interactive program for manipulating finite-state automata and computing supervisors. For computing the least restrictive supervisor S, one needs to provide an ASCII text file with the definition of the automaton $\mathcal{A}$ modeling the plant, and another ASCII text file with the definition of the automaton recognizing the language K which models the requirements. Then TCT can be used to compute an automaton which implements the least restrictive non-blocking supervisor and write it to a ASCII text file. For more on TCT and other tools on supervisory control see [12].

5 Main modeling considerations

The main underlying idea of the proposed plant model is the following. We model the behavior of the toner system from the point of view of the TTF belt. That is, we start looking at the system from the moment when the system starts transferring the image from the OPC belt onto the TTF belt. Subsequently we keep track of the point of the TTF belt which was at the transfer fuse (OPC belt) when the transfer of the image from the OPC belt onto the TTF belt started.

Notice the following simple facts about the point of TTF described above. First, as long as this point has not reached the fuse pinch, it must contain some toner. Second, the fuse contamination scenario takes place precisely when this point reached the fuse and the paper sheet is not yet at the fuse and the fuse is on the paper path. Third, the toner system becomes safe, i.e. no fuse pinch contamination can occur, if the above mentioned point passes the cleaner.

Since the the goal of modeling the toner subsystem is to provide a model for the control algorithm which would prevent fuse pinch contamination, it is reasonable to restrict our modeling effort to those components of the toner subsystem which can potentially influence whether fuse pinch contamination takes place. We claim that the movement of the point of the TTF belt contains all the information we need to know about the toner system in order to determine if fuse pinch contamination can occur.

Of course, the above statement is true only with some discretion. First, the length of the image is also important for the following reason. In addition to preventing fuse pinch contamination, we also have to make sure the the fuse pinch is put back on the paper path at the end of the error-handling algorithm. But if we put back the fuse pinch when the section of the TTF belt containing the image is still at the the fuse pinch, then we again get a fuse pinch contamination. Second, we should also make sure that we stop writing the image on the TTF belt after we received the **Sheet too late** signal. Otherwise, we get a continuous section of the TTF belt covered with toner, and we can not prevent the contamination of the fuse pinch. There is also a whole range of other constraints such as the presence or absence of paper immediately after the fuse pinch, etc.

We will incorporate all the other constraints on a later stage. First, we will concentrate on the most essential aspects. That is, we neglect the length of an image, i.e. we assume that the image is simply a point. Moreover, we do not take into account the potential

presence of other sheets in the section of the paper path situated right after the fuse pinch. We disregard the time needed to put back the fuse pinch onto the paper path. However, we will take into account the minimal and maximal speed limitations of the TTF tape and the time needed to lift the fuse pinch.

Assumption 1. The modeling assumptions are the following.

1. Each image is identified with a point on the TTF belt, i.e. the length of the image is assumed to be a point.
2. We assume that the fuse pinch can be lifted any time, but it takes a certain amount of time to complete the procedure of lifting it.
3. We assume that the speed of the TTF tape should belong to a certain interval.

6 Formal model of the plant

We will call the point of the TTF belt which stood at the transfer pinch when the writing began the *image point*. Since the belt moves, the *image point* moves as well. As it was already remarked above, the change in time of the relative position of the image point with respect to the transfer pinch will be central to our model.

In order to faciliate use of supervisor control theory, the toner system is modeled as the following finite state-automaton

$$\mathcal{T} = (Q, E, \delta, F, q_0) \tag{3}$$

Here Q is the finite state-space, E is the set of events, $\delta : Q \times E \to Q$ is the state-transition map and F is the set of accepting states, and q_0 is the initial state. The automaton above models the behavior of the toner system only up to a finite time instance $\mathbf{T}_{max}$.

As it was already mentioned earlier, the finite-state automaton is not constructed manually, but it is generated by an algorithm. Intuitively, what the algorithm does is the following. It takes a hybrid model of the plant (toner system), it samples it with a sampling interval Δ and outputs the thus generated automaton. By a hybrid model we mean a model of the system written using the formalism of hybrid systems [11]. Hybrid systems are mathematical objects used for describing phenomena exhibiting both continuous and discrete behavior.

Unfortunately, we do not have a discretization algorithm capable of handling general hybrid systems yet. However, the authors are actively working on a generic algorithm for converting a hybrid system to a finite-state automaton. In order to bypass the lack of theoretical results, we will present a particular version of the algorithm, which is specific to the model of the toner system at hand. That is, we will not formally separate the hybrid system model from the algorithm transforming it to a finite-state automaton. Instead, we will present both the algorithm and the hybrid system as one unit.

In the future we plan formulate the algorithm in its full generality. After this has been done, the steps for generating a supervisors would be as follows

1. Construct a hybrid model of the plant from the values of the physical parameters describing the plant.

2. Feed in the hybrid model to the algorithm and generate a finite-state automaton model of the plant.
3. Formulate the requirements as a finite-state automaton.
4. Using the finite-state automaton models of the plant and the requirements generate a supervisor.

The procedure above is depicted in Figure 4. As it was noted above, in this report we will present steps 1 and 2 as one joint step.

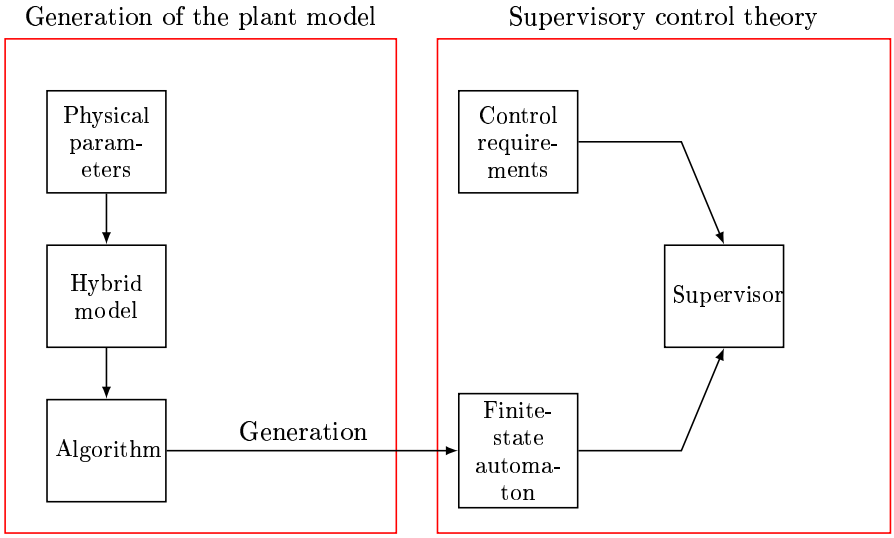

Figure 4: Generation of a supervisor

The algorithm mentioned above takes as inputs a number of parameters which are described in Subsection 6.1. In Subsection 6.2 the set E of controllable and uncontrollable events will be described. Finally, Subsection 6.3 presents the finite-state automaton generating algorithm, along with the definition of the initial state and the set of accepting states. When this algorithm is executed with a set of parameter values, its output is the state-space Q and the state-transition map δ of the finite-state automaton $\mathcal{T}$ from (3).

6.1 Model parameters

When defining the model, we will use the following parameters of the toner system.

Fp The numerical value in meters of the relative position of the fuse pinch with respect to the transfer pinch along the TTF tape. More precisely, this is the length of the section of the TTF tape which is spanned between the fuse pinch and the transfer pinch.

Cp The numerical value in meters of the relative position of the cleaner with respect to transfer pinch along the TTF tape. In other words, this is the length of the section of the TTF tape which is spanned between the cleaner and the transfer pinch and passes the fuse pinch.

$\mathbf{V}_{max}$ The maximal allowed speed in m/s of the TTF belt.

$\mathbf{V}_{min}$ The minimal allowed speed in m/s of the TTF belt.

$\mathbf{N}_{max}$ The number of time steps during which the behavior of the system is studied.

$\mathbf{T}_{fo}$ The time which is needed to open the fuse pinch.

$\mathbf{T}_{pl,max}$ The latest time instance in seconds at which a **Sheet too late** signal can be generated.

$\mathbf{T}_{pl,min}$ The earliest time instance in seconds at which a **Sheet too late** signal can be generated.

$\mathbf{A}$ The maximal value of acceleration in m/s^2 with which the TTF can be accelerated.

$\mathbf{D}$ The maximal value of deceleration in m/s^2 with which the TTF tape can be slowed down.

$\mathbf{T}_{max}$ The value in seconds of the maximal time instance up to which the behavior of the TTF tape is modeled.

Δ The length of the sampling interval in seconds. The finite state-automaton model to be presented below models the behavior of the TTF tape by sampling it on integer multiples of Δ.

The parameters above are not entirely independent. More precisely, if $\mathbf{N}_{max}$ is specified, then Δ and $\mathbf{T}_{max}$ are determined as follows

$$\Delta = \mathbf{Cp}/\mathbf{V}_{min}(\mathbf{N}_{max} - 1)$$
$$\mathbf{T}_{max} = \mathbf{N}_{max}\Delta$$

Conversely, if Δ and $\mathbf{T}_{max}$ are specified, then $\mathbf{N}_{max}$ is the smallest integer such that

$$(\mathbf{N}_{max} - 1)\Delta < \mathbf{T}_{max} \leq \mathbf{N}_{max}\Delta$$

6.2 The set of events E

The set of events E is the union of controllable events E_c and uncontrollable events E_{uc}. Below we will describe the set of controllable and uncontrollable events separately.

6.2.1 Controllable events

We model the input actions we are allowed to execute as controllable events. In addition, we model the passage of time as a controllable event as well. Formally, the set of controllable events E_c is

$$E_c = \{\mathbf{c}_{FU}, \mathbf{c}_{FD}, \mathbf{c}_A, \mathbf{c}_D, \mathbf{tick}\}$$

where the elements of E_c denote the following control actions

$\mathbf{c}_{FU}$ – lift the fuse pinch.

$\mathbf{c}_{FD}$ – put back the fuse pinch.

$\mathbf{c}_A$ – accelerate the TTF tape with a constant acceleration $\mathbf{A}$. The action takes Δ time units. The TTF tape is accelerated either untill Δ has elapsed, or until the TTF tape has reached the maximal speed $\mathbf{V}_{max}$. In the latter case, after the maximal speed $\mathbf{V}_{max}$ has been reached, we do nothing and just wait until Δ elapses, counting from the moment when $\mathbf{c}_A$ was issued.

$\mathbf{c}_D$ – slow down the TTF tape with a constant deceleration **D**. The TTF tape is slowed down either until Δ has elapsed, or until the TTF tape has reached the minimal speed $\mathbf{V}_{min}$. In the latter case, after the minimal speed $\mathbf{V}_{min}$ has been reached, we do nothing and just wait until Δ elapses, counting from the moment when $\mathbf{c}_D$ was issued.

tick – let time Δ pass and update the state variables accordingly.

6.2.2 Uncontrollable events

The set of uncontrollable events E_{uc} is defined as

$$E_{uc} = \{\mathbf{e}_{PL}, \mathbf{e}_{FUc}, \mathbf{e}_{NI}, \mathbf{e}_{NPIF}\}$$

where

$\mathbf{e}_{PL}$ The event $\mathbf{e}_{PL}$ stands for the **Sheet too late** signal notifying us the the sheet did not arrive on time.

$\mathbf{e}_{FUc}$ The event $\mathbf{e}_{FUc}$ denotes the situation when the lifting of the fuse pinch has been completed.

$\mathbf{e}_{NPIF}$ The event $\mathbf{e}_{NPIF}$ is generated when the toner image reaches the fuse when the fuse pinch is on the TTF tape, but there is no sheet on the paper path at the fuse pinch. In short, $\mathbf{e}_{NPIF}$ is generated if fuse pinch contamination is taking place. This is the event which should be avoided.

$\mathbf{e}_{NI}$ The event $\mathbf{e}_{NI}$ is generated when the point of the TTF tape, which before reaching the fuse pinch contained the toner image, reaches the cleaner, after having passed the fuse. That is, $\mathbf{e}_{NI}$ indicates that from our point of view the TTF tape has reached the end of "production cycle".

The event $\mathbf{e}_{PL}$ is the only one which is physically generated by a sensor. All the other uncontrollable events listed above are "virtual events", in a sense that they just represent modeling abstraction. There is no physical sensor to detect them. A potential supervisor will have to detect these events by simulating the movement of the image point on the TTF tape.

6.3 State-space and state-transition map

It is convenient to present the definition of the state-space and the state-transition map together. The reason for that is that the state-space Q of the automaton $\mathcal{T}$ is a finite subset of a potentially infinite bigger space. More precisely, the elements of Q are those elements of the bigger set which can be reached from the initial state q_0 using the state-transition map. Hence, defining the state-space and the state-transition relation separately leads to a chicken-and-egg problem. In fact, we will present an algorithm for generating both the state-space Q and the state-transition map δ.

More precisely, the state-space Q will be a subset of possible valuations of variables, some of them are discrete and some of them are numerical. We will start with describing these variables in more detail in Subsection 6.3.1 and Subsection 6.3.2. In Subsection 6.3.3 we will describe the algorithm for generating the state-space Q and the state-transition map δ. Finaly, Subsection 6.3.4 presents the definition of the initial state q_0 of $\mathcal{T}$, and Subsection 6.3.5 describes the set F of accepting states of $\mathcal{T}$.

6.3.1 Real-valued variables

Below we will describe the numerical variables of the model. For a variable X, we will denote by $\mathbf{Val}[X]$ the range of values of X.

$\mathbf{P}$ The variable $\mathbf{P}$ denotes the current position of the point of the TTF belt which stood at the transfer pinch when the transfer of the image began. The value of $\mathbf{P}$ is a real number from the interval $\mathbf{Val}[\mathbf{P}] = [0, \mathbf{Fp}]$.

$\mathbf{P}_{old}$ The variable $\mathbf{P}_{old}$ is an auxiliary variable, it is used to store the value of the variable $\mathbf{P}$ at the time instance which is the current time minus the sampling time Δ. The range of values $\mathbf{Val}[\mathbf{P}_{old}]$ of $\mathbf{P}_{old}$ is the same as that of $\mathbf{P}$, i.e. $\mathbf{Val}[\mathbf{P}_{old}] = \mathbf{Val}[\mathbf{P}]$.

$\mathbf{V}$ The variable $\mathbf{V}$ denotes the current speed of the TTF tape. The value of $\mathbf{V}$ is a real number from the interval $\mathbf{Val}[\mathbf{V}] = [\mathbf{V}_{min}, \mathbf{V}_{max}]$

$\mathbf{C}_{fu}$ The variable $\mathbf{C}_{fu}$ is active only if the pinch is being lifted. It denotes the time which has passed since the command to lift the pinch was issued. The value of $\mathbf{C}_{fu}$ is a real number from the interval $\mathbf{Val}[\mathbf{C}_{fu}] = [0, \mathbf{T}_{fo}]$.

$\mathbf{C}_{fu,old}$ The variable $\mathbf{C}_{fu,old}$ is an auxiliary variable, it is used to store the value of the variable $\mathbf{C}_{fu}$ at the time instance which is the current time minus the sampling time Δ. The range of values of $\mathbf{C}_{fu,old}$ is the same as that of $\mathbf{C}_{fu}$, i.e. $\mathbf{Val}[\mathbf{C}_{fu,old}] = \mathbf{Val}[\mathbf{C}_{fu}]$.

$\mathbf{T}$ Denotes the time which has passed since the start of the TTF tape. The value of $\mathbf{T}$ is a real number from the range $\mathbf{Val}[\mathbf{T}] = [0, \mathbf{N}_{max}\Delta]$.

$\mathbf{T}_{old}$ Stores the value of $\mathbf{T} - \Delta$.

Denote by $\mathbf{Var}_c$ the set of all continuous variables, i.e.

$$\mathbf{Var}_c = \{\mathbf{P}, \mathbf{T}, \mathbf{T}_{old}, \mathbf{C}_{fu}, \mathbf{V}, \mathbf{P}_{old}, \mathbf{C}_{fu,old}\}$$

For each numerical variable $X \in \mathbf{Var}_c$, it holds that the values of X belong to the subset $\mathbf{Val}[X]$ of real numbers.

6.3.2 Discrete variables

The set of discrete variables $\mathbf{Var}_d$ is the set

$$\mathbf{Var}_d = \{\mathbf{S}_{PL}, \mathbf{S}_{NI}, \mathbf{S}_{FU}, \mathbf{S}_{FU}, \mathbf{S}_{FD}, \mathbf{S}_{NI}\}$$

Each discrete variable from the set $\mathbf{Var}_d$ can have value which is either 0 or 1. The physical interpretation of each variable is the following.

$\mathbf{S}_{PL}$ is true if a **Sheet too late** signal has been detected.

$\mathbf{S}_{NI}$ is true if the image point has reached the cleaner.

$\mathbf{S}_{FU}$ is true if an order to lift the fuse pinch has been received.

$\mathbf{S}_{FU}$ is true if the lifting of the fuse pinch has been completed.

$\mathbf{S}_{FD}$ is true is the fuse pinch is on the TTF belt.

6.3.3 Algorithm for generating the state-space and the state-transition map

We start with defining the big set, of which Q is going to be a subset. As it was noted earlier, Q is simply a subset of the set of all valuations of the numerical and discrete variables defined above. Below we will define the concept of variable valuations more precisely. Denote by **Var** the set of all model variables, discrete and numerical, i.e.

$$\mathbf{Var} = \mathbf{Var}_c \cup \mathbf{Var}_d$$

Define the set Val as the set of all valuations of of the variables from **Var**. Formally, Val is the set of all functions (maps) S of the form

$$S : \mathbf{Var} \to \bigcup_{X \in \mathbf{Var}} \mathbf{Val}[X]$$

such that $S(X) \in \mathbf{Val}[X]$. That is, S represents simply a collection of assignments of values to the variables in **Var**. The state-space Q is finite subset of Val consisting of all those elements of Val which are reachable by some finite sequence of events using the transition relation δ to be defined below.

We will define the state-transition map δ and the state-space Q by a recursive algorithm. However, in order to do this, we will have to introduce some additional notation. If $S \in$ Val is an valuation of the variables in **Var** then for any collection of variables $X_1, X_2, \ldots, X_n$ and any collection of values $a_1 \in \mathbf{Val}[X_1], a_2 \in \mathbf{Val}[X_2], \ldots, a_n \in \mathbf{Val}[X_n]$ denote by $S[X_1 \leftarrow a_1, X_2 \leftarrow a_2, \ldots, X_n \leftarrow a_n]$ the valuation obtained from S by setting the value of each variable X_i to a_i for $i = 1, 2, \ldots, n$ and leaving the values of the rest of the variables unchanged. That is,

$$S[X_1 \leftarrow a_1, X_2 \leftarrow a_2, \ldots, X_n \leftarrow a_n](X) = \begin{cases} a_i & \text{if } X = X_i, \\ & \text{for some } i = 1, 2, \ldots, n \\ S(X) & \text{otherwise} \end{cases}$$

With the notation above, we are ready to present the algorithm generating the state-space Q and the state-transition map δ of $\mathcal{T}$. However, before proceeding to the algorithm, let us remark that the map δ can naturally be identified with the set of triples $(q, e, \hat{q})$ such that $q \in Q$ and $\delta(q, e)$ is defined and $\hat{q} = \delta(q, e)$. With this identification in mind, the algorithm generating δ and Q can be described as in Algorithm 6.3.1.

Algorithm 6.3.1 GenStatTrans

1: The initial state $q_0 \in$ Val is a valuation defined as follows;

$$q_0[\mathbf{P}] = q_0[\mathbf{P}_{old}] = 0$$
$$q_0[\mathbf{V}] = \mathbf{V}_{max}$$
$$q_0[\mathbf{C}_{fu}] = q_0[\mathbf{C}_{fu,old}] = 0$$
$$q_0[X] = 0 \text{ for all } X \in \mathbf{Var}_d$$

Set $Q := \{q_0\}$.
Set $\delta = \emptyset$.

2: **GenTransMap**(δ, Q).
3: return Q and δ.

In Algorithm 6.3.1 the auxiliary algorithm **GenTransMap** is used. The description of the latter can be found in Algorithm 6.3.2.

The intuition behind Algorithm 6.3.1 and Algorithm 6.3.2 is the following. A natural state-space model of the toner is an infinite-state automaton, state space of which is

Algorithm 6.3.2 GenTransMap(δ, Q)

1: For each $q \in Q$ and $e \in E$.

If $e = \mathbf{e}_{PL}$ and $q[\mathbf{T}] \in [\mathbf{T}_{pl,min}, \mathbf{T}_{pl,max}]$, then set $\hat{q} = q[\mathbf{S}_{PL} \leftarrow 1]$.

If $e = \mathbf{c}_{FU}$ and $q(\mathbf{S}_{FU}) = 0$ and $q(\mathbf{S}_{FU}) = 0$, then set $\hat{q} = q[\mathbf{S}_{FU} \leftarrow 0, \mathbf{S}_{FU} \leftarrow 1, \mathbf{C}_{fu} \leftarrow 0, \mathbf{C}_{fu,old} \leftarrow 0]$.

If $e = \mathbf{e}_{FUc}$ and $q(\mathbf{S}_{FU}) = 1$ and $q(\mathbf{S}_{FU}) = 0$, and $q(\mathbf{C}_{fu}) \geq \mathbf{T}_{fo}$ and $q(\mathbf{C}_{fu,old}) < \mathbf{T}_{fo}$, then set $\hat{q} = q[\mathbf{S}_{FU} \leftarrow 0, \mathbf{S}_{FU} \leftarrow 1]$.

If $e = \mathbf{c}_{FD}$ and $q(\mathbf{S}_{FU}) = 1$, then set $\hat{q} = q[\mathbf{S}_{FD} \leftarrow 1, \mathbf{S}_{FU} \leftarrow 0]$.

If $e = \mathbf{c}_A$ and $q(\mathbf{T}) < \mathbf{T}_{max}$, then let $\tau = \min\{\Delta, (\mathbf{V}_{max} - q(\mathbf{V}))/\mathbf{A}\}$ and set

$$\hat{q} = \begin{cases} q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{V} \leftarrow q(\mathbf{V}) + \mathbf{A}\tau, \\ \mathbf{C}_{fu} \leftarrow q(\mathbf{C}_{fu}) + \Delta, \mathbf{C}_{fu,old} \leftarrow q(\mathbf{C}_{fu}), \mathbf{P}_{old} \leftarrow q(\mathbf{P}), & \text{if } q(\mathbf{S}_{FU}) = 1 \\ \mathbf{P} \leftarrow q(\mathbf{P}) + \tau q(\mathbf{V}) + \tau^2 \mathbf{A} + \mathbf{V}_{max}(\Delta - \tau)] \\ \\ q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{P}_{old} \leftarrow q(\mathbf{P}), \mathbf{V} \leftarrow q(\mathbf{V}) + \mathbf{A}\tau \\ \mathbf{P} \leftarrow q(\mathbf{P}) + \tau q(\mathbf{V}) + \tau^2 \mathbf{A} + \mathbf{V}_{max}(\Delta - \tau)] & \text{otherwise} \end{cases}$$

If $e = \mathbf{c}_D$ and $q(\mathbf{T}) < \mathbf{T}_{max}$, then let $\tau = \min\{\Delta, (q(\mathbf{V}) - \mathbf{V}_{min})/\mathbf{D}\}$ and set

$$\hat{q} = \begin{cases} q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{V} \leftarrow q(\mathbf{V}) - \mathbf{D}\tau \\ \mathbf{C}_{fu} \leftarrow q(\mathbf{C}_{fu}) + \Delta, \mathbf{C}_{fu,old} \leftarrow q(\mathbf{C}_{fu}), \mathbf{P}_{old} \leftarrow q(\mathbf{P}) & \text{if } q(\mathbf{S}_{FU}) = 1 \\ \mathbf{P} \leftarrow q(\mathbf{P}) + \tau q(\mathbf{V}) - \tau^2 \mathbf{D} + \mathbf{V}_{min}(\Delta - \tau)] \\ \\ q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{V} \leftarrow q(\mathbf{V}) - \mathbf{D}\tau], \mathbf{P}_{old} \leftarrow q(\mathbf{P}) \\ \mathbf{P} \leftarrow q(\mathbf{P}) + \tau q(\mathbf{V}) - \tau^2 \mathbf{D} + \mathbf{V}_{min}(\Delta - \tau)] & \text{otherwise} \end{cases}$$

If $e = \mathbf{tick}$ and $q(\mathbf{T}) < \mathbf{T}_{max}$, then set

$$\hat{q} = \begin{cases} q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{C}_{fu} \leftarrow q(\mathbf{C}_{fu}) + \Delta, \\ \mathbf{C}_{fu,old} \leftarrow q(\mathbf{C}_{fu}), \mathbf{P}_{old} \leftarrow q(\mathbf{P}), \mathbf{P} \leftarrow q(\mathbf{P}) + \Delta q(\mathbf{V})] & \text{if } q(\mathbf{S}_{FU}) = 1 \\ \\ q[\mathbf{T} \leftarrow q(\mathbf{T}) + \Delta, \mathbf{T}_{old} \leftarrow \mathbf{T}, \mathbf{P}_{old} \leftarrow q(\mathbf{P}), \\ \mathbf{P} \leftarrow q(\mathbf{P}) + \Delta q(\mathbf{V})] & \text{otherwise} \end{cases}$$

If $e = \mathbf{e}_{NPIF}$ and $q(\mathbf{S}_{PL}) = 1$ and $q(\mathbf{S}_{FU}) = 0$ and $q(\mathbf{P}) \geq \mathbf{Fp}$ and $q(\mathbf{P}_{old}) < \mathbf{Fp}$, then set $\hat{q} = q$.

If $e = \mathbf{e}_{NI}$ and $q(\mathbf{P}) \geq \mathbf{Cp}$ and $q(\mathbf{P}_{old}) < \mathbf{Cp}$, then $\hat{q} = q[\mathbf{S}_{NI} \leftarrow 1]$.

2: If $\hat{q}$ is defined and $(q, e, \hat{q}) \notin \delta$, then let $Q \leftarrow Q \cup \{\hat{q}\}$ and $\delta \leftarrow \delta \cup (q, e, \hat{q})$ and execute

GenTransMap(δ, Q)

the set of all valuations Val. This automaton reacts to events from the event set E by changing its state according to the following partial state-transition function

$$\mathbf{Trans} : \mathrm{Val} \times E \to \mathrm{Val}$$

where for all $S \in \mathrm{Val}$ and $e \in E$, the value $\hat{S} = \mathbf{Trans}(S, e)$ is defined as follows.

- If $e = \mathbf{e}_{PL}$ and either $S(\mathbf{T})$ belongs to $[\mathbf{T}_{pl,min}, \mathbf{T}_{pl,max}]$ or $S(\mathbf{T}_{old}) < \mathbf{T}_{pl,max} \leq S(\mathbf{T})$, then $\mathbf{Trans}(S, e)$ is defined and $\hat{S}$ coincides with S, except that $\hat{S}$ assigns the variable $\mathbf{S}_{PL}$ the value 1, i.e.

  $$\hat{S} = S[\mathbf{S}_{PL} \leftarrow 1]$$

  The condition on $\mathbf{T}$ simply expresses the requirement that a **Sheet too late** signal can take place only in the time interval $[\mathbf{T}_{pl,min}, \mathbf{T}_{pl,max}]$. The condition $S(\mathbf{T}_{old}) < \mathbf{T}_{pl,max} \leq S(\mathbf{T})$ allows for the detection of **Sheet too late** signal even if it takes place between sampling times.

- If $e = \mathbf{c}_{FU}$, that is a command to lift the fuse pinch is to be executed, then $\hat{S}$ is defined only if $S(\mathbf{S}_{FU}) = S(\mathbf{S}_{FU}) = 0$. That is, we can lift the fuse pinch if it is on the TTF tape, but we cannot lift it if it is in the process of being lifted or it has been already lifted. If $\hat{S}$ is defined, then it is the same as S except that it assigns the value 1 to the variable $\mathbf{S}_{FU}$ and it sets the value of $\mathbf{C}_{fu}$ and $\mathbf{C}_{fu,old}$ to 0, i.e.

  $$\hat{S} = S[\mathbf{S}_{FU} \leftarrow 1, \mathbf{C}_{fu} \leftarrow 0, \mathbf{C}_{fu,old} \leftarrow 0]$$

- If $e = \mathbf{e}_{FUc}$, that is, it is claimed that the procedure of lifting the fuse pinch has been completed, then $\hat{S}$ is defined if $S(\mathbf{S}_{FU}) = 1$ and $S(\mathbf{S}_{FU}) = 0$, and $\mathbf{T}_{fo}$ falls into the interval $(S(\mathbf{C}_{fu,old}), S(\mathbf{C}_{fu})]$. If all these conditions hold, then $\hat{S}$ is defined and it coincides with S, except that $\hat{S}$ sets the value of $\mathbf{S}_{FU}$ to 0 and the value of $\mathbf{S}_{FU}$ to 1, i.e.

  $$\hat{S} = S[\mathbf{S}_{FU} \leftarrow 1, \mathbf{S}_{FU} \leftarrow 0]$$

  Intuitively, the condition that $\mathbf{S}_{FU}$ is 1 stems from the fact that one cannot complete the procedure of lifting the fuse pinch if it has not been started yet (i.e. if $\mathbf{S}_{FU}$ is 0). Similarly, the requirement that $\mathbf{S}_{FU}$ is 0 is juts the formalization of the fact that the procedure of lifting the fuse pinch can be completed only once. Finally, the condition $S(\mathbf{C}_{fu,old}) < \mathbf{T}_{fo} \leq S(\mathbf{C}_{fu})$ simply says that the time necessary for completely lifting the pinch has elapsed. The reason that we cannot use the simpler condition $S(\mathbf{C}_{fu}) = \mathbf{T}_{fo}$ is that in our case we observe the progress of time not continuously, but only on discrete sampling times which are integer multiples of Δ. If $\mathbf{T}_{fo}$ is not an integer multiple of Δ, then the condition $S(\mathbf{C}_{fu}) = \mathbf{T}_{fo}$ might never be detected.

- If $e = \mathbf{c}_{FD}$ and $S(\mathbf{S}_{FU}) = 1$, then $\hat{S}$ is defined and it differs from S only in the fact that it assigns 1 to $\mathbf{S}_{FD}$ and 0 to $\mathbf{S}_{FU}$, i.e.

  $$\hat{S} = S[\mathbf{S}_{FD} \leftarrow 1, \mathbf{S}_{FU} \leftarrow 0]$$

  The intuition behind it is quite straightforward; one cannot put down the pinch if it has not been lifted.

- If $e = \mathbf{c}_A$ and the current time $\mathbf{T}$ is smaller than the maximal time T_{max}, then $\hat{S}$ is defined. The valuation $\hat{S}$ assigns to the discrete variables the same values as S and

it assigns the following values to each continuous variable.

$$\begin{aligned}
\hat{S}(\mathbf{T}) &= S(\mathbf{T}) + \Delta \\
\hat{S}(\mathbf{T}_{old}) &= S(\mathbf{T}) \\
\hat{S}(\mathbf{V}) &= \begin{cases} S(\mathbf{V}) + \mathbf{A}\Delta & \text{if } S(\mathbf{V}) + \mathbf{A}\Delta \leq \mathbf{V}_{max} \\ \mathbf{V}_{max} & \text{otherwise} \end{cases} \\
\hat{S}(\mathbf{C}_{fu}) &= S(\mathbf{C}_{fu}) + \Delta \text{ if } S(\mathbf{S}_{FU}) = 1 \\
\hat{S}(\mathbf{C}_{fu,old}) &= S(\mathbf{C}_{fu}) \text{ if } S(\mathbf{S}_{FU}) = 1 \\
\hat{S}(\mathbf{P}_{old}) &= S(\mathbf{P}) \\
\hat{S}(\mathbf{P}) &= S(\mathbf{P}) + \tau S(\mathbf{V}) + \tau^2 \mathbf{A} + \mathbf{V}_{max}(\Delta - \tau) \text{ where} \\
& \quad \tau = \min\{\Delta, (\mathbf{V}_{max} - S(\mathbf{V}))/\mathbf{A}\}
\end{aligned} \tag{4}$$

That equations above express the following simple action. We accelerate the TTF tape with a constant acceleration $\mathbf{A}$ and we are allowed to apply this acceleration as long as the speed of the TTF tape does not exceed $\mathbf{V}_{max}$. In addition, we are not allowed to apply the acceleration for longer than Δ time units. If we have reached the maximal speed $\mathbf{V}_{max}$ before the elapse of time Δ, then we just let the time remaining until Δ elapse and we do nothing. While the acceleration and/or waiting takes place, the value of the variable $\mathbf{P}$ changes according to the elementary physical law describing the change of the position under time-varying speed. Intuitively, the equations (4) are the result of discretization of the flow of the following differential equation

$$\begin{aligned}
\dot{\mathbf{V}} &= \begin{cases} \mathbf{A} & \text{if } \mathbf{V} \leq \mathbf{V}_{max} \\ 0 & \text{otherwise} \end{cases} \\
\dot{\mathbf{P}} &= \mathbf{V}
\end{aligned} \tag{5}$$

- If $e = \mathbf{c}_D$ and the current time $\mathbf{T}$ is smaller than the maximal time T_{max}, then $\hat{S}$ is defined. The valuation $\hat{S}$ assigns to the discrete variables the same values as S and it assigns the following values to each continuous variable.

$$\begin{aligned}
\hat{S}(\mathbf{T}) &= S(\mathbf{T}) + \Delta \\
\hat{S}(\mathbf{T}_{old}) &= S(\mathbf{T}) \\
\hat{S}(\mathbf{V}) &= \begin{cases} S(\mathbf{V}) - \mathbf{D}\Delta & \text{if } S(\mathbf{V}) - \mathbf{D}\Delta \geq \mathbf{V}_{min} \\ \mathbf{V}_{min} & \text{otherwise} \end{cases} \\
\hat{S}(\mathbf{C}_{fu}) &= S(\mathbf{C}_{fu}) + \Delta \text{ if } S(\mathbf{S}_{FU}) = 1 \\
\hat{S}(\mathbf{C}_{fu,old}) &= S(\mathbf{C}_{fu}) \text{ if } S(\mathbf{S}_{FU}) = 1 \\
\hat{S}(\mathbf{P}_{old}) &= S(\mathbf{P}) \\
\hat{S}(\mathbf{P}) &= S(\mathbf{P}) + \tau S(\mathbf{V}) - \tau^2 \mathbf{D} + \mathbf{V}_{min}(\Delta - \tau) \text{ where} \\
& \quad \tau = \min\{\Delta, (S(\mathbf{V}) - \mathbf{V}_{min})/\mathbf{D}\}
\end{aligned} \tag{6}$$

That equations above express the following simple action. We decelerate the TTF tape with a constant deceleration $\mathbf{D}$ and we are allowed to apply this deceleration as along as the speed of the TTF tape does not drop below $\mathbf{V}_{min}$. In addition, we are not allowed to apply the deceleration for longer than Δ time units. If we have reached the minimal speed $\mathbf{V}_{min}$ before the elapse of time Δ, then we just let the time remaining until Δ elapse and we do nothing. While the deceleration and/or waiting takes place, the value of the variable $\mathbf{P}$ changes according to the corresponding time-varying speed. Intuitively, the equations (4) are the result of

discretization of the flow of the following differential equation

$$\begin{aligned} \dot{\mathbf{V}} &= \begin{cases} -\mathbf{D} & \text{if } \mathbf{V} \geq \mathbf{V}_{min} \\ 0 & \text{otherwise} \end{cases} \\ \dot{\mathbf{P}} &= \mathbf{V} \end{aligned} \tag{7}$$

- If $e = \textbf{tick}$ and the current time $\mathbf{T}$ is smaller than the maximal time T_{max}, then $\hat{S}$ is defined. The valuation $\hat{S}$ assigns to the discrete variables the same values as S and it assigns the following values to each continuous variable.

$$\begin{aligned} \hat{S}(\mathbf{T}) &= S(\mathbf{T}) + \Delta \\ \hat{S}(\mathbf{T}_{old}) &= S(\mathbf{T}) \\ \hat{S}(\mathbf{V}) &= S(\mathbf{V}) \\ \hat{S}(\mathbf{C}_{fu}) &= S(\mathbf{C}_{fu}) + \Delta \text{ if } S(\mathbf{S}_{FU}) = 1 \\ \hat{S}(\mathbf{C}_{fu,old}) &= S(\mathbf{C}_{fu}) \text{ if } S(\mathbf{S}_{FU}) = 1 \\ \hat{S}(\mathbf{P}_{old}) &= S(\mathbf{P}) \\ \hat{S}(\mathbf{P}) &= S(\mathbf{P}) + \Delta S(\mathbf{V}) \end{aligned} \tag{8}$$

  The equations above express the following simple action. We let time Δ pass and we update the value of the variables $\mathbf{T}$ and $\mathbf{P}$ accordingly. The update rules simply express the fact that the position changes with a constant speed $\mathbf{V}$ during the time interval $[0, \Delta]$. Intuitively, the equations (8) are the result of sampling of the flow of the following differential equation

$$\begin{aligned} \dot{\mathbf{V}} &= 0 \\ \dot{\mathbf{P}} &= \mathbf{V} \end{aligned} \tag{9}$$

- If $e = \mathbf{e}_{NPIF}$, then $\hat{S}$ is defined only if $S(\mathbf{S}_{PL}) = 1$ and $S(\mathbf{S}_{FU}) = 0$, and $S(\mathbf{P}) \geq \mathbf{Fp}$ and $S(\mathbf{P}_{old}) < \mathbf{Fp}$. The valuation S coincides with the valuation S. The conditions above simply express the situation, under which the contamination of the fuse pinch occurs. The only less straightforward condition is $S(\mathbf{P}_{old}) < \mathbf{Fp} \leq S(\mathbf{P})$. This condition is equivalent to requiring that the fuse was passed in the interval $[\mathbf{T} - \Delta, \mathbf{T}]$, where $\mathbf{T}$ is the current time. The reason for not using the simpler condition $\mathbf{Fp} = S(\mathbf{P})$ is the following. We observe the variable $S(\mathbf{P})$ denoting the position of the image point at discrete time steps. Hence there is no guarantee that we will capture the moment when the $S(\mathbf{P})$ is exactly $\mathbf{Fp}$. However, if we see that the image point stood before the fuse pinch in the previous step and it is after the fuse pinch in the current step, then we can conclude that it must have been *exactly* at the fuse pinch at some time instance between the last and the current steps.

- If $e = \mathbf{e}_{NI}$, $\hat{S}$ is defined if $S(\mathbf{P}_{old}) < \mathbf{Cp}$, and $S(\mathbf{P}) \geq \mathbf{Cp}$. The valuation $\hat{S}$ is the same as the valuation S, except that it assigns 1 to the variable $\mathbf{S}_{NI}$. The conditions above just express the fact that $\mathbf{e}_{NI}$ is generated if the image point has passed the cleaner. The reason for using $S(\mathbf{P}_{old}) < \mathbf{Cp} \leq S(\mathbf{P})$ instead of $S(\mathbf{P}) = \mathbf{Cp}$ is similar to what was explained above. That is, we see the position only at discrete steps, hence in general we cannot detect when the image point reaches the cleaner. However, by checking the above condition we can determine whether the image point has reached the cleaner between now and the previous sampling.

With the discussion above in mind, it is easy to see that Algorithm 6.3.1 in combination with Algorithm 6.3.2 recursively computes the subset Q of V which is reachable from the initial state by the repeated application of the state transition map **Trans** defined above. The initial state q_0 is the state defined in Algorithm 6.3.1.

It can be easily seen that the algorithm in Algorithm 6.3.1 stops after finitely many steps and it returns a set Q of variable valuations and a set of triples δ of the form $(q, e, \hat{q})$ for some $\hat{q}, q \in Q$, $e \in E$. The returned set Q is then taken as the state-space of the automaton $\mathcal{T}$, and the set δ, interpreted as map $Q \times E \ni (q, e) \mapsto \hat{q} \in Q \iff (q, e, \hat{q}) \in \delta$ is taken as the state-transition map.

6.3.4 Initial state

The initial state q_0 of the automaton $\mathcal{T}$ is the state q_0 defined in the algorithm **GenStatTrans**, Algorithm 6.3.1.

6.3.5 Final states

The set of final states F are those variable valuations belonging to Q such that the value of the variable $\mathbf{S}_{NI}$ is one, i.e.

$$F = \{q \in Q \mid q(\mathbf{S}_{NI}) = 1\}$$

7 Formal model of the requirements

As it is customary in supervisory control theory, the requirements are modeled as a formal language. The language is defined over the set of events E where E is the set of discrete-events of the plant model $\mathcal{T}$, described in Subsection 6.2. The language allows for any sequence of events from E, as long as the following restrictions are met.

- Each sequence has to end in $\mathbf{e}_{NI}$. That is, the image point should end at the cleaner.
- No sequence may contain the event $\mathbf{e}_{NPIF}$. That is, no fuse pinch contamination is allowed to take place.
- The event $\mathbf{e}_{PL}$ occurs at most once and only as a first event or after some number **tick** events. The **Sheet too late** signal can occur only once. As long as it does not take place, no other action is allowed than letting the time progress.
- The event $\mathbf{c}_{FU}$ has to be followed by $\mathbf{e}_{FUc}$ and $\mathbf{c}_{FD}$. Moreover, after $\mathbf{c}_{FU}$ has occurred, the event $\mathbf{c}_D$ is not allowed to take place until event $\mathbf{e}_{FUc}$ and then $\mathbf{c}_{FD}$ took place. That is, the TTF tape is not allowed to be slow down as long as the fuse pinch is not on the TTF tape.
- If a sequence does not contain $\mathbf{e}_{PL}$, then it contains only $\mathbf{e}_{NI}$ and **tick**. That is, if no **Sheet too late** signal has arrived, then no action should take place except letting the time progress and reaching the cleaner.

Denote the language described above by K. An automaton accepting the language K is shown on Figure 5. On Figure 5, the dashed arrows are used to denote the transitions associated with uncontrollable events, the solid arrow are used to denote the transitions associated with the controllable events. The solid arrows without labels pointing outside the state are used to indicate that a certain state is final. Finally, the incoming arrow to $\mathbf{S}_0$ is used to indicate that $\mathbf{S}_0$ is the initial state.

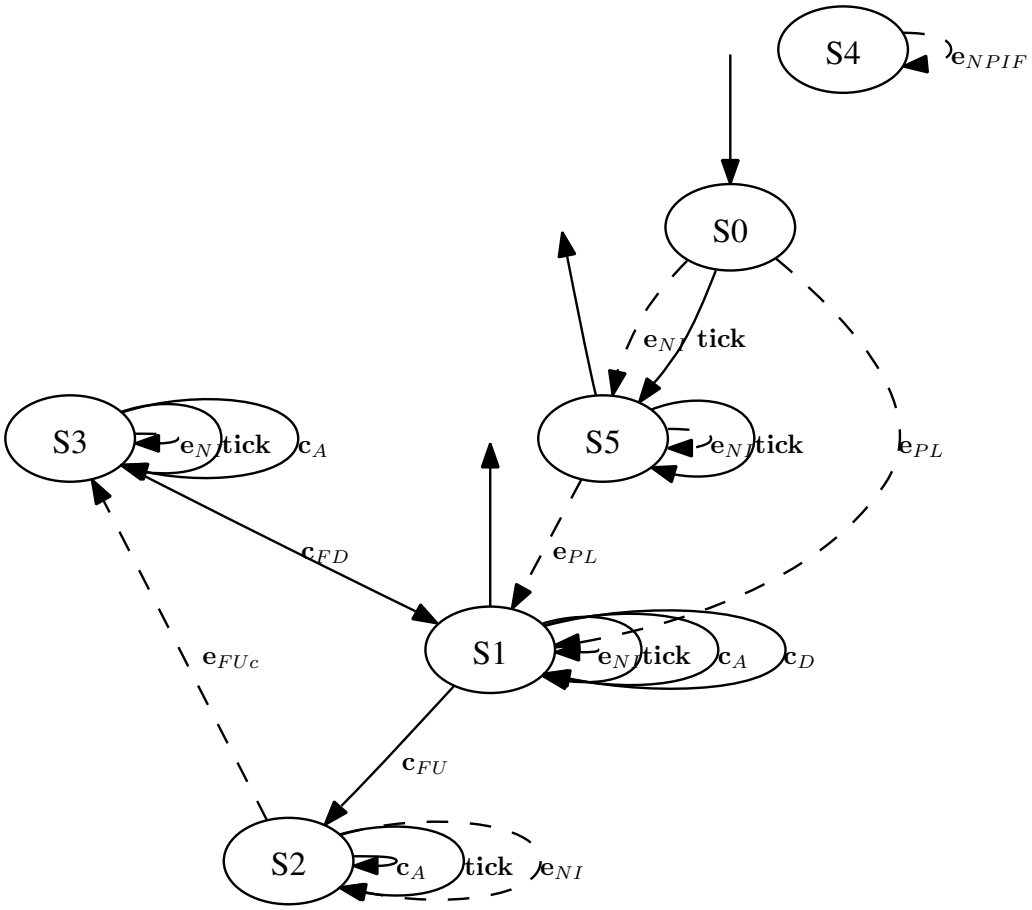

Figure 5: Automaton accepting the specification language K

In formal terms, the automaton $\mathcal{A}_{spec}$ accepting K shown on Figure 5 can be described as follows

$$\mathcal{A}_{spec} = (Q_{spec}, E, \delta_{spec}, F_{spec}, q_{0,spec})$$

where,

$Q_{spec} = \{\mathbf{S}0, \mathbf{S}1, \mathbf{S}2, \mathbf{S}3, \mathbf{S}4, \mathbf{S}5\}$ is the set of states

$E = E_{uc} \cup E_c = \{\mathbf{e}_{PL}, \mathbf{e}_{FUc}, \mathbf{c}_A, \mathbf{c}_D, \mathbf{c}_{FD}, \mathbf{c}_{FU}, \mathbf{e}_{NI}, \mathbf{e}_{NPIF}, \mathbf{tick}\}$ is the set of events. The set of events equals to that of for the automaton $\mathcal{T}$ modeling the plant. Similarly to $\mathcal{T}$, the controllable events are

$$E_c = \{\mathbf{c}_{FU}, \mathbf{c}_{FD}, \mathbf{c}_D, \mathbf{c}_A, \mathbf{tick}\}$$

and the uncontrollable events are

$$E_{uc} = \{\mathbf{e}_{PL}, \mathbf{e}_{NI}, \mathbf{e}_{NPIF}, \mathbf{e}_{FUc}\}$$

$\delta : Q_{spec} \times E \to Q_{spec}$ is the state-transition map, which is a partial map and it is defined as follows.

- For each $X \in \{\mathbf{e}_{NI}, \mathbf{tick}\}$, $\delta(\mathbf{S}0, X) = \mathbf{S}_5$, $\delta(\mathbf{S}0, \mathbf{e}_{PL}) = \mathbf{S}2$ and for all other events $Y \notin \{\mathbf{e}_{NI}, \mathbf{tick}, \mathbf{e}_{PL}\}$, $\delta(\mathbf{S}0, Y)$ is undefined.
- For each $X \in \{\mathbf{e}_{NI}, \mathbf{tick}\}$, $\delta(\mathbf{S}5, X) = \mathbf{S}5$, and $\delta(\mathbf{S}5, \mathbf{e}_{PL}) = \mathbf{S}_1$. For all other discrete events $Y \notin \{\mathbf{e}_{NI}, \mathbf{tick}, \mathbf{e}_{PL}\}$, $\delta(\mathbf{S}5, Y)$ is undefined.
- For each $X \in \{\mathbf{tick}, \mathbf{c}_A, \mathbf{c}_D, \mathbf{e}_{NI}\}$, $\delta(\mathbf{S}1, X) = \mathbf{S}1$ and $\delta(\mathbf{S}1, \mathbf{c}_{FU}) = \mathbf{S}2$. For $Y \notin \{\mathbf{tick}, \mathbf{c}_A, \mathbf{c}_D, \mathbf{e}_{NI}, \mathbf{c}_{FU}\}$, $\delta(\mathbf{S}1, Y)$ is undefined.
- For each $X \in \{\mathbf{tick}, \mathbf{c}_A, \mathbf{e}_{NI}\}$, $\delta(\mathbf{S}2, X) = \mathbf{S}2$ and $\delta(\mathbf{S}2, \mathbf{e}_{FUc}) = \mathbf{S}3$. For $Y \notin \{\mathbf{tick}, \mathbf{c}_A, \mathbf{e}_{NI}, \mathbf{e}_{FUc}\}$, $\delta(\mathbf{S}3, Y)$ is undefined.
- For each $X \in \{\mathbf{tick}, \mathbf{c}_A, \mathbf{e}_{NI}\}$, $\delta(\mathbf{S}3, X) = \mathbf{S}3$ and $\delta(\mathbf{S}1, \mathbf{c}_{FD}) = \mathbf{S}1$. For $Y \notin \{\mathbf{tick}, \mathbf{c}_A, \mathbf{e}_{NI}, \mathbf{c}_{FD}\}$, $\delta(\mathbf{S}3, Y)$ is undefined.
- For all $X \neq \mathbf{e}_{NPIF}$, $\delta(\mathbf{S}4, X)$ is undefined and $\delta(\mathbf{S}4, \mathbf{e}_{NPIF}) = \mathbf{S}4$.

$F_{spec} = \{\mathbf{S}1, \mathbf{S}5\}$ is the set of accepting states.

$q_{0,spec} = \mathbf{S}0$ is the initial state of the automaton $\mathcal{A}_{spec}$.

Notice that the state $\mathbf{S}4$ is in fact unreachable and can be removed from the automaton $\mathcal{A}_{spec}$. The only reason we included it into the model of the requirements, is that in order to get the supervisory control tools working, there had to be at least one transition in $\mathcal{A}_{spec}$ labeled by $\mathbf{e}_{NPIF}$. However, since the specification language explicitly forbids occurrence of $\mathbf{e}_{NPIF}$, the only solution is to add it as a state-transition involving states which are not reachable from the initial state.

8 Generation of the supervisor

The generation of the supervisor takes place according to the procedure outlined in Section 6 and illustrated by Figure 4. First, a finite state-automaton model $\mathcal{T}$ of the plant is generated. Second, using the requirement specification language K described in in Section 7 and the model $\mathcal{T}$ of the plant, the least restrictive non-blocking supervisor S is generated such that $L_m(\mathcal{T}/S) \subseteq K$.

From a practical point of view, the generation of the least restrictive supervisor S takes place as follows. Algorithm 6.3.1 for computing the finite-state automaton $\mathcal{T}$ was implemented in as python script **discretize_model_smart_new.py** . The script **discretize_model_smart_new.py** reads the values of the parameters listed in Subsection 6.1 and uses them to generate the corresponding finite-state automaton $\mathcal{T}$. The script writes the definition of the finite-state automaton model on the standard output, using the ASCII text input format of the TCT tool. A typical example of using **discretize_model_smart_new.py** under Linux would be the following

`python` **discretize_model_smart_new.py** `<parameter_file> > <automaton.ads>`

Here `<parameter_file>` is the name of the file containing the description of the model parameters, and `<automaton.ads>` is the name of the file where the automaton model of the toner system is going to be written, using the ASCII text input format of TCT. Examples of the parameters files and the outputs generated by the program will be presented in Appendix B.

The source of **discretize_model_smart_new.py** can be found in Appendix A. The automaton of Figure 5 recognizing the language K was encoded in the ASCII text input file format of the TCT tool as well. Using this and the file generated by **discretize_model_smart_new.py** , the least restrictive supervisor S was synthesized. The synthesis was repeated for a number of different parameter values. The list of models and supervisors obtained for various parameter values can be found in Appendix B.

9 Discussion and conclusions

In this section we will try to formulate some lessons learned from the use case discussed in the paper. In Subsection 9.1 we will discuss the pitfalls encountered when applying supervisory control to solve the presented use-case. In Subsection 9.2 we will describe possible future research directions.

9.1 Pitfalls and challenges in applying supervisor control to solving the use case

To begin with, it is not entirely clear whether the generated supervisor represents an adequate solution of the toner error-handling problem at hand. In a nutshell, the reason for this is the following

Extraction of control software from the supervisor The generated least restrictive supervisor is non-deterministic, i.e. it allows several choices of control inputs at any given point. However, the desired control software should generate at most one control action. Hence the problem arises how to extract a deterministic supervisor from the least restrictive one. In addition, a number of practical issues related to embedding of the extracted controller into the existing embedded software system should be addressed as well. We will elaborate on this point in Subsection 9.1.1.

Transition from the discrete-event domain to time-domain The supervisor and the presented model of the plant live in a discrete-event domain. The physical toner system and hence the actual control software lives in the time-domain. The transition between the time-domain and the discrete-event domain is not quite

clear. In fact, it is unclear how to justify the claim that the presented finite-state automaton $\mathcal{T}$ and the language K represent an adequate mathematical model of the physical toner system and the control objectives from Section 3. We will discuss this topic in more detail in Subsection 9.1.2

Large state-space of the model The size of the state-space of the toner model $\mathcal{T}$ is exponential in the chosen number of time steps. This might be a serious challenge. We will elaborate on this issue in Subsection 9.1.3

Below we elaborate on both issues in some more detail.

9.1.1 Extraction of control software

Notice that for the use-case treated in this paper, the controllable events are interpreted as control inputs, hence Remark 4 at the end of Section 4 needs to be taken into account.

To begin with, we need to establish a correspondence between the controllable events of the formal model and the control actions. This is relatively easy, in fact, it was already done in Subsection 6.2.1. Perhaps the only less straightforward step is the interpretation of the event **tick** as a control action. The event **tick** should be interpreted as follows; do not execute any control action but activate the supervisor again after time Δ has elapsed.

The second step is the extraction of a deterministic supervisor from the least restrictive one. One possibility for extracting such a deterministic supervisor is to use the method described in Remark 4, Subsection 4. In Appendix B the state diagram of a deterministic supervisor obtained using this method is shown. What the advantages and disadvantages are of the procedure outlined above remains to be seen.

Even after a deterministic implementable supervisor V has been obtained, implementing it as a control software poses a challenge. The reason for that lies in the problems around the physical interpretation of the supervisor V, see Subsection 9.1.2 for a more detailed discussion. Recall that in Section 4 we have sketched a possible implementation method for deterministic supervisor generating control inputs. Unfortunately, due to the presence of time, the sketched implementation need not be feasible. Proper ways of implementing the supervisor V remain a topic of future research.

9.1.2 Transition from discrete-event to real-time

To begin with, the following intuitive argument can be formulated for justifying why the obtained supervisor is a solution of the use case. If $L_m(\mathcal{T}/S) \subseteq K$, then it means that the event $\mathbf{e}_{NPIF}$ simply cannot occur in the closed-loop system. That is, if the control actions of the supervisor S are executed, then fuse pinch contamination cannot take place.

However, the argument above has several shortcomings. First, it is not quite clear why the automaton generated by Algorithm 6.3.1 would be a correct model of the toner system and in what sense. As it was pointed out before, the physical toner system lives in real time. Intuitively, the model generated by Algorithm 6.3.1 provides snapshots of the state of the physical toner system at sampling times. But it remains an open issue how to relate the measurements on the behavior of the physical toner system with the discrete-event behavior of the automaton $\mathcal{T}$.

Another difficult issue is the interpretation and hence implementation of the generated supervisor. Here we modeled the passage of time Δ by the controllable event **tick**. The

justification for this is that we wanted to enable the supervisor to say the following; "for the time being do nothing, but get back to me in Δ time units". However, by interpreting the time as controllable event, we also enable the supervisor to disable passage of time. If disabling **tick** is interpreted as "we must execute one of the control actions and we must not postpone it any longer", then this does not pose a problem. However, disabling **tick** can also be interpreted as stopping the progress of time, which is physically impossible. The above problem does not seem to arise in the least restrictive supervisor generated by the system, but it does arise in the deterministic supervisor which is extracted from a least restrictive one. An example of this phenomenon can be seen on the supervisor depicted on Figure 11, Appendix B.3. There, by disabling a **tick** event, one prevents $\mathbf{e}_{PL}$ occurring at later time instances. This is unrealistic, since $\mathbf{e}_{PL}$ models an uncontrollable external signal which may happen any time withing the indicated time interval. That is, the method outlined in Section 4 idoes not seem to be directly applicable to the implementation of the supervisor obtained for the toner system.

9.1.3 Complexity challenges

From the experiments it seems that the the number of time steps represents a problem only for the generation of the model $\mathcal{T}$. The TCT tool itself is able to generate the supervisor even for very big finite state automata in a matter of seconds. Since the program which generates the automaton $\mathcal{T}$ is written in the scripting language Python and it contains no optimization, we believe that the time needed for generating $\mathcal{T}$ can be reduced significantly by simply optimizing the code and possibly rewriting it in a lower level programming language such as C or C++. In addition, the generation of the model and the supervisor is expected to take place offline, during the development of new printers. In this case computational time is not a very serious concern anyway.

9.2 Future work

The use case raises a number of theoretical and applied problems which are worth investigating. Below we will list and discuss them one by one.

9.2.1 Further refinement of the model of the use-case

To begin with, further efforts are needed to check how realistic the models of the plant and of the requirements are. As it was already pointed out in Section 5, several simplifications and assumptions were made when building the model. The validity of these assumptions needs to be checked. However, in the light of the numerous open questions to be discussed in Subsection 9.1, one may question the wisdom of investing too much effort into perfecting the model along the lines presented in the paper. It might easily turn out that a different approach is more suitable for the control of printers in general, and for solving the current use-case in particular. Possible candidates for alternative solution methods include integer-linear programming and direct construction of hybrid controllers.

9.2.2 Comparison with the existing solution

As it was already pointed out in the introduction, the presented use-case has already been solved. It would be of great interest to compare the solution obtained by Océ with the solution obtained using supervisory control. There are several possible aspects worth comparing. Perhaps the most important are the following.

- Performance and complexity of the obtained controller
- Robustness with respect to the change of model parameters, reusability.
- Development time.

We expect that the internal solution might score high on the first aspect, but the solution obtained using supervisory control will perform better on the rest. The reason for this is the following. The controller is generated automatically based on the physical parameters of the system. Hence, if the parameters change, a new supervisor can be generated within minutes. On the other hand, Océ has decades of experience and a great deal of expertise in developing printer systems. Relying on this expertise Océ is likely to be able to come up with a very efficient solution.

9.2.3 Modeling in time domain versus discrete-event domain

As it was pointed out in Subsection 9.1.2 the transition from discrete-event domain to time domain represents a challenge. Even from the informal description of the toner system it is clear that it exhibits both discrete and continuous behavior. Hence, *hybrid systems* seem to be the most natural modeling formalism for the toner system. For more on hybrid systems see [11, 6]. In fact, the automaton $\mathcal{T}$ obtained by Algorithm 6.3.1 can be viewed as a discretization of a hybrid model of the toner system. However, supervisory control of hybrid systems is not that well developed. In fact, the methods of supervisory control for hybrid systems we found in the literature [9, 7, 8, 1] seem to be inadequate to address the use-case. The main challenge is that the solution of the use-case requires to take into account the timing issues. More precisely, the timing of control actions is critical for obtaining an adequate solution. [2] In contrast, the existing methods for supervisory control tend to abstract away from the timing.

We plan to work on using hybrid systems for modeling and solving the use case. In fact, this is already work in progress and the preliminary results are encouraging. In a nutshell, the approach is the following. We model the plant (the toner system) as a hybrid system, i.e. as a system containing both discrete and continuous behavior and living in real-time. We formulate the requirements in real-time as well. Then we convert the requirements to the discrete-event domain and we convert the hybrid system model to a discrete-event finite-state model. The latter is performed in two steps. First, we convert the hybrid system to a discrete-time system by sampling its evolution with sampling interval Δ. By further restricting attention to the finite-time behavior of the toner system we naturally get a discrete-event model which is essentially the same as the outcome of the Algorithm 6.3.1. We then solve a classical discrete-event supervisory control problem for the thus obtained model. Subsequently we translate the supervisor into a controller acting in real-time. The main idea behind the translation is that the resulting controller is activated on sampling times only and at each activation the controller computes the control actions to be performed. We aim at demonstrating that the thus obtained controller solves the original control problem formulated in real-time. Notice that the first few steps of the approach outlined above coincide with the procedure outlined in Section 6 and depicted on Figure 4.

The procedure above represents a theoretical challenge and it may take significant time to develop it in full detail. However, once the theoretical foundations have been laid and the software support has been developed, the procedure outlined above can be used

[2] Notice that the timing constraints cannot be easily formulated explicitly, but only as requirements on the evolution of continuous variables. In other words, timed automata does not seem to be a suitable formalism for modeling the use-case.

not only for solving the particular use-case, but for generating control software for other components of the printers as well.

Bibliography

[1] Rajeev Alur, Thomas Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proccedings of the IEEE*, 88(2):971–984, 2000.

[2] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publ, 1999.

[3] Samuel Eilenberg. *Automata, Languages and Machines.* Academic Press, New York, London, 1974.

[4] F. Gécseg and I Peák. *Algebraic theory of automata.* Akadémiai Kiadó, Budapest, 1972.

[5] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[6] Daniel Liberzon. *Switching in Systems and Control.* Birkhäuser, Boston, 2003.

[7] Thomas Moor, Jen M. Davoren, and Jörg Raisch. Modular supervisory control of a class of hybrid systems in a behavioural framework. In *Proc. European Control Conference ECC2001*, page 870âĂŞ875, Porto, Portugal, 2001.

[8] Thomas Moor and Jörg Raisch. Hierarchical hybrid control of a multiproduct batch plant. In *Proc. 16th IFAC World Congress*, Prague, Czech Republic, 2005.

[9] Thomas Moor, Jörg Raisch, and Siu O'Young. Discrete supervisory control of hybrid systems by l-complete approximations. *Journal of Discrete Event Dynamic Systems*, 12(1), 2002.

[10] P.J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25:206, 1987.

[11] Arjan van der Schaft and Hans Schumacher. *An Introduction to Hybrid Dybnamical Systems.* Springer-Verlag London, 2000.

[12] W.M. Wonham. Supervisory control of discrete-event systems. Lecture Notes, available at `http://se.wtb.tue.nl/sewiki/wonham/supervisory_control_news`.

[13] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.*, 25, 1987.

A Source code of the discretization algorithm

```
#! /usr/bin/env python2.4
import sys
import os
import copy
import ConfigParser
```

```
epsilon = 0.0001
def number_list (in_list, begin):
 out_list=dict([])
 counter = begin
 for element in in_list:
  out_list[str(element)]=counter
  counter = counter + 2

 return out_list

def number_events(contr_events,uncontr_events):
 numbered_events = dict([])
 numbered_uncontr_events = dict([])

 numbered_events = number_list(contr_events,1)
 numbered_uncontr_events = number_list(uncontr_events,0)

 for event in uncontr_events:
  numbered_events[str(event)] = numbered_uncontr_events[str(event)]

 return numbered_events




def generate_initial_state(predicate_list, parameters):
   dstate=dict([])

   for pred in predicate_list:
    dstate[pred]=False

   big_state=dict([('DISCRETE_STATE', dstate), \
            ('POSITION',0), \
     ('SPEED', parameters['speed_high']),\
     ('OLD_POSITION',0),\
     ('TIME',0), ('OLD_TIME',0)])

   return big_state



def own_copy(gstate, predicate_list):

 new_gstate=dict()

 new_gstate['DISCRETE_STATE']=dict()
 for pred in predicate_list:
  new_gstate['DISCRETE_STATE'][pred]=\
      gstate['DISCRETE_STATE'][pred]

 for (key,value) in gstate.iteritems():
  if not (key == 'DISCRETE_STATE'):
   new_gstate[key]=value

 return new_gstate
```

```
def generate_transition_map_for_state(gstate, event, parameters, predicate_list):

        new_gstate = own_copy(gstate, predicate_list)


 if gstate['DISCRETE_STATE']['IS_NI']:
  return (False,new_gstate)

 is_NI = ( parameters['cleaner_position'] <= \
                        new_gstate['POSITION'] ) and \
                       (parameters['cleaner_position'] >  \
                        new_gstate['OLD_POSITION'])
 if not (event  in ['NI', 'NPIF', 'PL', 'FD', 'FUc']) and is_NI:
  return (False,new_gstate)

 is_defined = True

 if event == 'DEC':
  is_defined = (new_gstate['SPEED'] - \
          parameters['speed_low'] > epsilon ) and \
  ( new_gstate['TIME'] < parameters['max_time'] - parameters['time_step'] )

  ramp_down = (new_gstate['SPEED'] - parameters['speed_low'])\
                                  /parameters['deceleration']
  if ramp_down > parameters['time_step']:
   ramp_down = parameters['time_step']


  new_gstate['SPEED']=new_gstate['SPEED'] - \
                                parameters['deceleration']*ramp_down
  new_gstate['OLD_POSITION'] = new_gstate['POSITION']
  new_gstate['POSITION'] = new_gstate['POSITION'] + \
                       new_gstate['SPEED'] * ramp_down + \
   parameters['speed_low'] * (parameters['time_step'] - ramp_down)
  new_gstate['OLD_TIME'] = new_gstate['TIME']
  new_gstate['TIME'] = new_gstate['TIME'] + parameters['time_step']

  if new_gstate['DISCRETE_STATE']['IS_FU']:
   is_defined = is_defined and \
   (new_gstate['DURATION'] < \
    parameters['fu_duration_max'])
   new_gstate['OLD_DURATION'] = new_gstate['DURATION']
   new_gstate['DURATION'] = \
   new_gstate['DURATION'] + parameters['time_step']


 elif event == 'ACC':
  is_defined = (parameters['speed_high'] - \
                                     new_gstate['SPEED'] > epsilon ) and \
  ( new_gstate['TIME'] < \
                         parameters['max_time'] - parameters['time_step'] )
  ramp_up = (parameters['speed_high'] - new_gstate['SPEED'])\
                                                  /parameters['acceleration']
```

```
 if ramp_up > parameters['time_step']:
  ramp_up = parameters['time_step']


 new_gstate['SPEED']=new_gstate['SPEED'] + \
                                  parameters['acceleration']*ramp_up
 new_gstate['OLD_POSITION'] = new_gstate['POSITION']
 new_gstate['POSITION'] = new_gstate['POSITION'] + \
                       new_gstate['SPEED'] * ramp_up + \
  parameters['speed_high'] * (parameters['time_step'] - ramp_up)


 new_gstate['OLD_TIME'] = new_gstate['TIME']
 new_gstate['TIME'] = new_gstate['TIME'] + parameters['time_step']

 if new_gstate['DISCRETE_STATE']['IS_FU']:
  is_defined = is_defined and \
  (new_gstate['DURATION'] < \
   parameters['fu_duration_max'])
  new_gstate['OLD_DURATION'] = new_gstate['DURATION']
  new_gstate['DURATION'] = \
  new_gstate['DURATION'] + parameters['time_step']

elif event == 'FU':
 is_defined = (not new_gstate['DISCRETE_STATE']['IS_FU']) and \
                 (not new_gstate['DISCRETE_STATE']['IS_FUc'])
 new_gstate['DISCRETE_STATE']['IS_FU']=True
 new_gstate['DURATION']=0
 new_gstate['OLD_DURATION']=0

elif event == 'FD':
 is_defined = new_gstate['DISCRETE_STATE']['IS_FUc']
 new_gstate['DISCRETE_STATE']['IS_FUc']=False

elif event == 'TICK':
 is_defined = ( new_gstate['TIME'] < \
                       parameters['max_time'] - parameters['time_step'])
 new_gstate['OLD_POSITION'] = new_gstate['POSITION']
 new_gstate['POSITION'] = new_gstate['POSITION'] + \
  new_gstate['SPEED'] * parameters['time_step']
 new_gstate['OLD_TIME'] = new_gstate['TIME']
 new_gstate['TIME'] = new_gstate['TIME'] + parameters['time_step']

 if new_gstate['DISCRETE_STATE']['IS_FU']:
  is_defined = is_defined and \
  (new_gstate['DURATION'] < \
   parameters['fu_duration_max'])
  new_gstate['OLD_DURATION'] = new_gstate['DURATION']
  new_gstate['DURATION'] = \
  new_gstate['DURATION'] + parameters['time_step']


elif event == 'FUc':
 if new_gstate['DISCRETE_STATE']['IS_FU']:
  is_defined = ( new_gstate['OLD_DURATION'] < \
    parameters['fu_duration_max']) and\
```

```
        (  parameters['fu_duration_max'] \
                                                <= new_gstate['DURATION'])
   new_gstate.pop('DURATION')
   new_gstate.pop('OLD_DURATION')
   new_gstate['DISCRETE_STATE']['IS_FU']  = False
   new_gstate['DISCRETE_STATE']['IS_FUc'] = True
  else:
   is_defined = False

 elif event == 'PL':
  is_defined = (not new_gstate['DISCRETE_STATE']['IS_PL'])\
                and ( (new_gstate['TIME'] <= parameters['max_pl_time'] and\
                 new_gstate['TIME'] >= parameters['min_pl_time']) or \
            ((new_gstate['OLD_TIME'] < parameters['max_pl_time']) and \
             #new_gstate['OLD_TIME'] > parameters['min_pl_time'] and \
    (parameters['max_pl_time'] <= new_gstate['TIME']) ))

  new_gstate['DISCRETE_STATE']['IS_PL'] = True

 elif event == 'NI':
  is_defined = ( parameters['cleaner_position'] <= \
      new_gstate['POSITION'] ) and \
   (parameters['cleaner_position'] >  \
    new_gstate['OLD_POSITION'])
  new_gstate['DISCRETE_STATE']['IS_NI'] = True

   elif event == 'NPIF':
  is_defined = (new_gstate['DISCRETE_STATE']['IS_PL'] and \
   (not new_gstate['DISCRETE_STATE']['IS_FUc'] )) and \
   (new_gstate['POSITION']  >=  parameters['fuse_position'])\
                       and (new_gstate['OLD_POSITION'] < parameters['fuse_position'])

 else:
  is_defined = False

 return (is_defined, new_gstate)


def generate_transition_map(gstate, event_list, parameters,\
                predicate_list, gstate_list=[], transition_map=[]):


 if not (own_str(gstate,predicate_list) in gstate_list):
  gstate_list.append(own_str(gstate, predicate_list))


 for event in event_list:
  (is_defined,new_gstate) = \
                generate_transition_map_for_state(\
                  gstate,event,parameters,predicate_list)
  if is_defined and \
                 (not ([own_str(gstate,predicate_list), \
                     str(event),own_str(new_gstate,predicate_list)] \
                                                        in transition_map)):
   transition_map.append([own_str(gstate, predicate_list),\
                       str(event),own_str(new_gstate,predicate_list)])
```

```
   if not (own_str(new_gstate, predicate_list) in  gstate_list):
    gstate_list.append(own_str(new_gstate, predicate_list))
    (transition_map,gstate_list)=generate_transition_map(\
                                 new_gstate, event_list,parameters,\
                                 predicate_list, gstate_list, transition_map)

 return (transition_map, gstate_list)

def is_state_marked(gstate):
# print "#State:"+ gstate+"\n"
 return ('ISNIiSTrue' in gstate)

def is_initial(gstate, predicate_list, parameters):
   initial=generate_initial_state(predicate_list, parameters)
   return gstate == own_str(initial,predicate_list)


def own_pred_str(pred):
 strings_pred=dict([('IS_PL', 'ISPL'),('IS_NI', 'ISNI'),\
 ('IS_FU', 'ISFU'), ('IS_FUc','ISFUc'),('IS_FD','ISFD'),\
        ('IS_NPIF','ISNPIF')])

 return strings_pred[pred]

def generate_reachable_set(gstate_list, transition_map, \
                                      initial_state, reach_set):

 if initial_state is reach_set:
  return

 reach_set.append(initial_state)

 for (old_state, event, new_state) in \
        transition_map:
  if old_state == initial_state:
        generate_reachable_set(gstate_list,\
                               transition_map, new_state, reach_set)

 return


def generate_reachable_state(gstate_list, transition_map, \
                                         predicate_list, parameters):
 is_initial_found = False
 for gstate in gstate_list:
  if is_initial(gstate, predicate_list, parameters):
   initial_state = gstate
   is_initial_found = True
   break

 if not is_initial_found:
  print "No initial state is found"
  return False

 #reach_state.append(initial_state)
```

```
 reach_state = []

 generate_reachable_set(gstate_list, transition_map, \
                                          initial_state, reach_state)

 return reach_state



def own_str(gstate, predicate_list):
 text = ""
 for pred in predicate_list:
  text=text+"and"+own_pred_str(pred)+\
                         "iS"+str(gstate['DISCRETE_STATE'][pred])

 text = text+"andPositionIs"+str(gstate['POSITION'])+\
                 "andOldPositionIs"+str(gstate['OLD_POSITION'])+"andSpeedIs"+\
               str(gstate['SPEED'])+\
       "andTimeIs" + str(gstate['TIME'])+\
               "andOldTimeIs"+str(gstate['OLD_TIME'])

 if gstate['DISCRETE_STATE']['IS_FU']:
  text=text+'andDurationIs'+str(gstate['DURATION'])+\
       'andOldDurationIs'+str(gstate['OLD_DURATION'])

 return text



def generate_asd_file(gstate_list, transition_map, \
            predicate_list, contr_events, uncontr_events, parameters):
#
  print "model"
 print "\n"
 state_counter = 0
 state_dict    = dict([])


 numbered_events = number_events(contr_events, uncontr_events)


 initial_found = False

        number_of_states=len(gstate_list)

 print "State size (State set will be (0,1....,size-1)):"

 print str(number_of_states)

 for gstate in gstate_list:

  if is_initial(gstate, predicate_list, parameters):
   state_dict[gstate] = state_counter
   state_counter = state_counter + 1
```

```
    initial_found = True


  if not initial_found:
   print "No initial state found"
   return

  for gstate in gstate_list:
   if not is_initial(gstate,predicate_list,parameters):
    state_dict[gstate] = state_counter
    state_counter = state_counter + 1


  print "Marker states:"
  for gstate in gstate_list:
          if is_state_marked(gstate):
    print str(state_dict[gstate])


  print "\n"
  print "Vocal states:\n"
  print "Transitions:"
  for transition in transition_map:
   old_state = transition[0]
   new_state = transition[2]
   event     = transition[1]

   print str(state_dict[old_state])+ \
   " " + str(numbered_events[event])+ " " + \
   str(state_dict[new_state])

  return

 def ReadParameters(file_name,external_parameters_list):
  parameters = dict([])

   fp = open(file_name)

   configf = ConfigParser.ConfigParser()

   configf.readfp(fp)



   data_list = configf.items('Fuse parameters')

  for element in data_list:
   key = element[0]
          value = element[1]
   parameters[key] = eval(value)

  parameters['real_speed_low']  = parameters['real_speed_min']

  if parameters.has_key('number_of_time_steps') and \
            (not parameters.has_key('max_time')):
   parameters['time_step'] = parameters['real_cleaner_position']\
```

```
            /(parameters['real_speed_low'] * \
                                (parameters['number_of_time_steps']-1))
 parameters['max_time'] = parameters['number_of_time_steps'] * \
                                   parameters['time_step']
elif parameters.has_key('number_of_time_steps') and \
           parameters.has_key('max_time'):
 parameters['time_step'] = parameters['max_time']\
                           / parameters['number_of_time_steps']
elif parameters.has_key('time_step') and \
     parameters.has_key('max_time') :
 parameters['number_of_time_steps'] = \
 int(parameters['max_time'] / parameters['time_step'])
else:
 print "Bad parameterization\n"
 sys._exit(-1)

parameters['acceleration'] = parameters['real_acceleration']
       parameters['deceleration'] = parameters['real_deceleration']


parameters['real_position_length'] = parameters['real_speed_low'] *\
                                              parameters['time_step']


parameters['speed_low' ] = parameters['real_speed_low']
parameters['speed_high'] = parameters['real_speed_max']

parameters['max_pl_time']        = parameters['real_time_pl_max']
       parameters['min_pl_time']              = parameters['real_time_pl_min']
parameters['fuse_position']            = parameters['real_fuse_position']
parameters['cleaner_position']         = parameters['real_cleaner_position']
parameters['fu_duration_max']          = parameters['real_time_fuse_open']


for (key,values) in parameters.iteritems():
 print "## Key: " + str(key)+ " value: " + \
       str(values) + "\n"

return parameters




# Global definitions

predicate_list         = ['IS_PL','IS_NI','IS_FU','IS_FUc']
controllable_events    = ['FU','FD', 'DEC', 'ACC', 'TICK']
uncontrollable_events  = ['PL','NI','NPIF', 'FUc']
events                 = controllable_events + uncontrollable_events

parameters=dict([])

external_parameters_list = [\
'real_fuse_position',\
'real_cleaner_position',\
```

```
'real_speed_max',\
'real_speed_min',
'real_speed_ratio',\
'number_of_time_steps',\
'real_time_fuse_open',\
'real_time_pl_max',\
'real_time_pl_min',\
'real_acceleration'
'real_deceleration'
]
value_list = [False,True]
conf_filename = 'test_fuse_param'


if len(sys.argv) < 2:
 print "Taking defualt configuration file " +  \
  str(conf_filename)+"\n"
else:
 conf_filename = sys.argv[1]
 print "#Reading configuration file " + \
  str(conf_filename) + "\n"


parameters = ReadParameters(conf_filename,\
                            external_parameters_list)

init_state=generate_initial_state(predicate_list,parameters)
(transition_map, gen_state_list) = generate_transition_map(init_state,\
                                       events, parameters, predicate_list)


generate_asd_file(gen_state_list, transition_map, predicate_list, \
                 controllable_events, uncontrollable_events, parameters)
```

B Examples of models generated with various parameters

In this section a number of supervisors presented generated for models of the toner system with different values of physical parameters described in Subsection 6.1. The various supervisors were generated as follows. First, a model of the toner system was generated using a particular set of parameters, subsequently the model was used to generate a supervisor. The particular set of parameters were fed in to the program generating the model by means of parameter files, as described in Section 8. The format of such a file is the following. It is just a text file containing lines of the form

<parameter_name> = <value>

where <paramater_name> is the name of the parameter, and <value> is the value assigned to it. In Table 1 we present the correspondence between the names used in the parameter files and the physical parameters described in Subsection 6.1 The file is expected to start with the string [Fuse parameters]. Notice that it is enough to specify either $\mathbf{N}_{max}$ or Δ and $\mathbf{T}_{max}$.

real_fuse_position	**Fp**
real_cleaner_position	**Cp**
real_speed_max	$\mathbf{V}_{max}$
real_speed_min	$\mathbf{V}_{min}$
real_deceleration	**D**
real_acceleration	**A**
real_time_fuse_open	$\mathbf{T}_{fo}$
real_time_pl_max	$\mathbf{T}_{pl,max}$
real_time_pl_min	$\mathbf{T}_{pl,min}$
number_of_time_steps	$\mathbf{N}_{max}$
time_step	Δ
max_time	$\mathbf{T}_{max}$

Table 1: Parameter names in the parameter files

The structure of the section is the following. For each set of parameter values we will present the parameter file specifying these values, the least restrictive supervisor generated for this model, and a deterministic supervisor extracted from the least restrictive one. The generated least restrictive supervisor and the extracted deterministic supervisor will be presented in the form of a state-transition diagram similar to the state-diagram of Figure 5. Similarly to the convention adopted in Figure 5, the solid arrows denote the state transitions corresponding to the controllable events, the dashed arrows denote the transitions corresponding to the uncontrollable events, the arrow pointing outwards indicate that a state is an accepting one, and the unique arrow with no source state pointing into a state indicates that this state is an initial one.

B.1

Parameter file

```
[Fuse parameters]
real_fuse_position    = 0.12348
real_cleaner_position = 0.17066
real_speed_max        = 0.488
real_speed_min        = 0.070
real_deceleration     = 4.7
real_acceleration     = 3
real_time_fuse_open   = 0.5
real_time_pl_max      = 0
real_time_pl_min      = 0
number_of_time_steps  = 28
#
```

The least restrictive supervisor is shown on Figure 6.

The deterministic supervisor extracted as discussed in Remark 4 is shown on Figure 7.

B.2

Parameter file

```
[Fuse parameters]
```

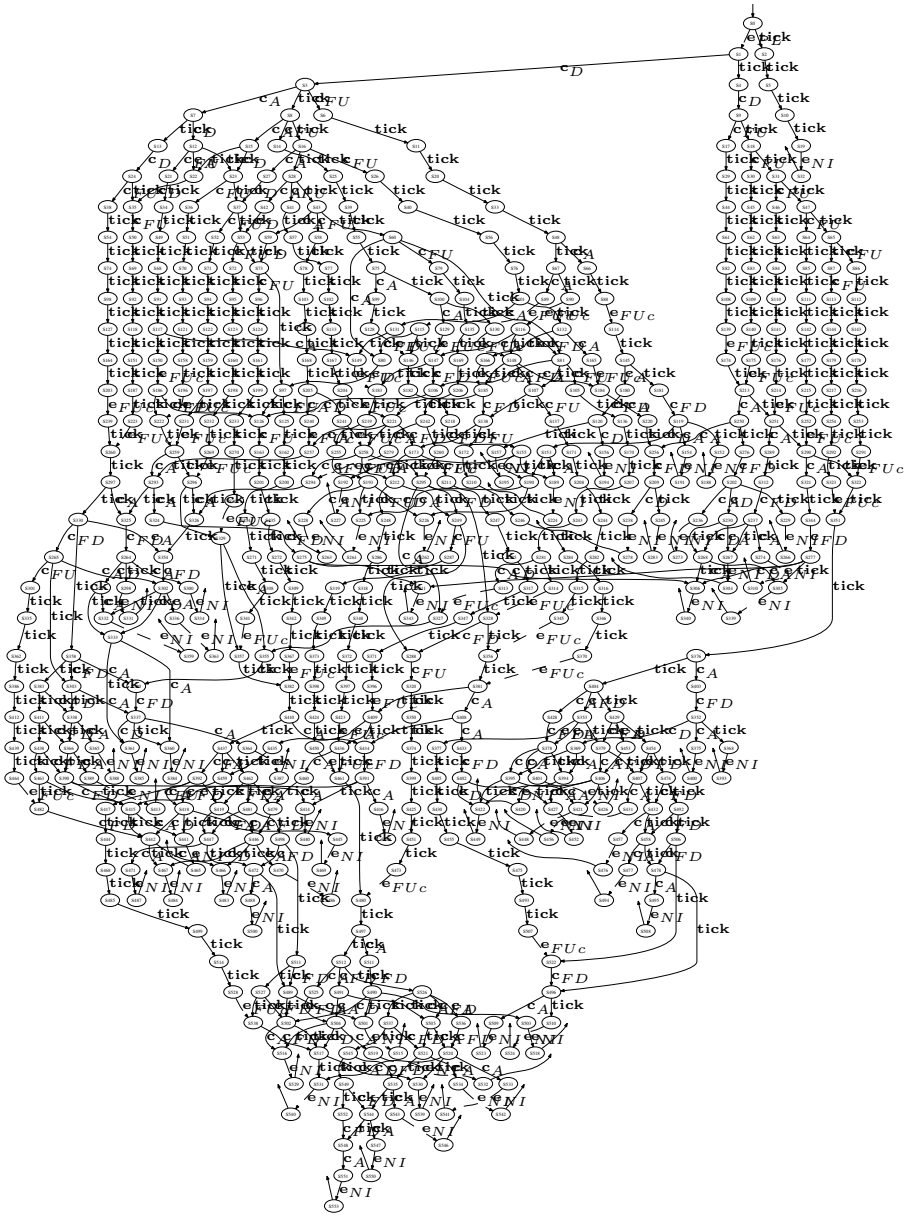

Figure 6: Least restrictive supervisor

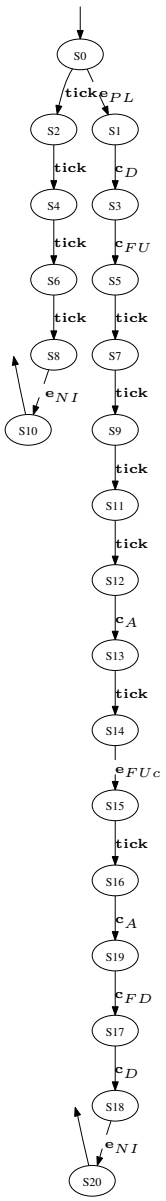

Figure 7: A deterministic supervisor extracted from the least restrictive one

```
real_fuse_position    = 0.12348
real_cleaner_position = 0.17066
real_speed_max        = 0.303
real_speed_min        = 0.070
real_deceleration     = 4.7
real_acceleration     = 3
real_time_fuse_open   = 0.94
real_time_pl_max      = 0
real_time_pl_min      = 0
number_of_time_steps  = 28
#
```

The least restrictive supervisor is shown on Figure 8. The deterministic supervisor extracted as discussed in Remark 4 is shown on Figure 9.

B.3

Parameter file

```
[Fuse parameters]
real_fuse_position    = 0.12348
real_cleaner_position = 0.17066
real_speed_max        = 0.303
real_speed_min        = 0.070
real_deceleration     = 4.7
real_acceleration     = 3
real_time_fuse_open   = 0.94
real_time_pl_max      = 0.1
real_time_pl_min      = 0
number_of_time_steps  = 28
```

The least restrictive supervisor is shown on Figure 10. The deterministic supervisor extracted as discussed in Remark 4 is shown on Figure 11.

B.4

Parameter file

```
[Fuse parameters]
real_fuse_position    = 0.12348
real_cleaner_position = 0.17066
real_speed_max        = 0.303
real_speed_min        = 0.070
real_deceleration     = 4.7
real_acceleration     = 3
real_time_fuse_open   = 0.5
real_time_pl_max      = 0.1
real_time_pl_min      = 0
number_of_time_steps  = 28
```

The least restrictive supervisor is shown on Figure 12. The deterministic supervisor extracted as discussed in Remark 4 is shown on Figure 13.

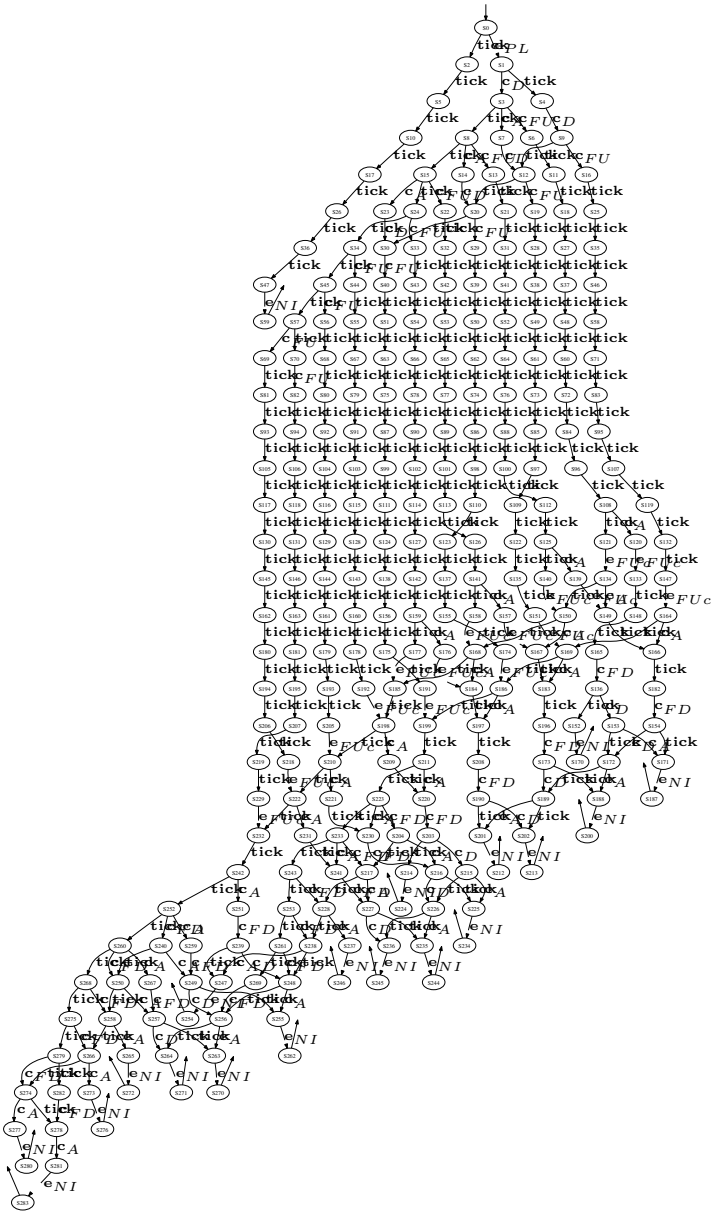

Figure 8: Least restrictive supervisor

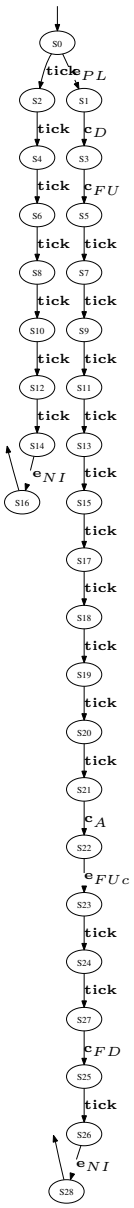

Figure 9: A deterministic supervisor extracted from the least restrictive one

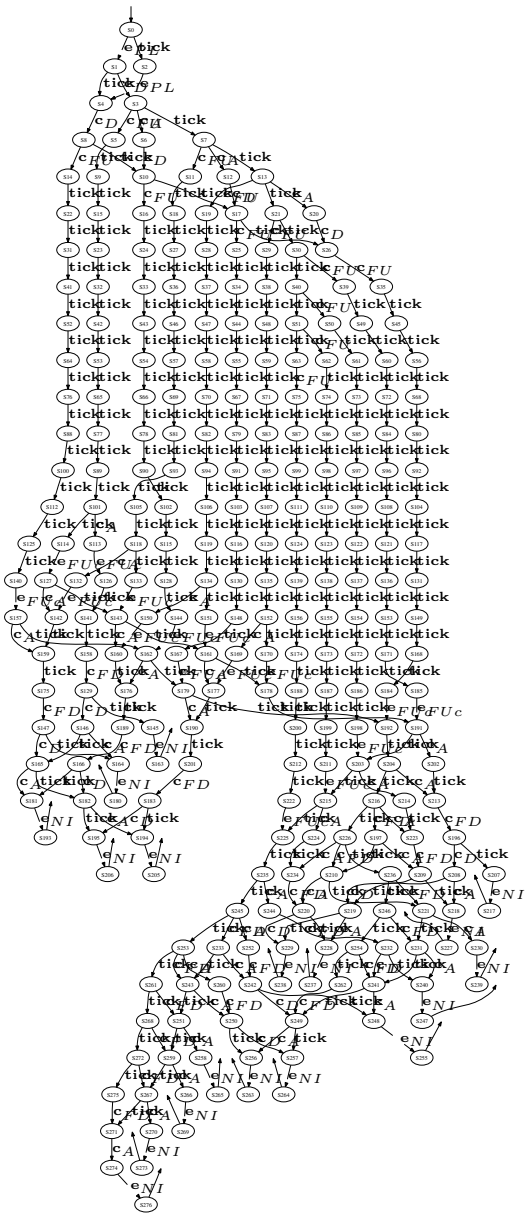

Figure 10: Least restrictive supervisor

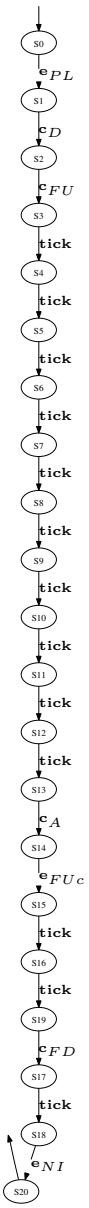

Figure 11: A deterministic supervisor extracted from the least restrictive one

Figure 12: Least restrictive supervisor

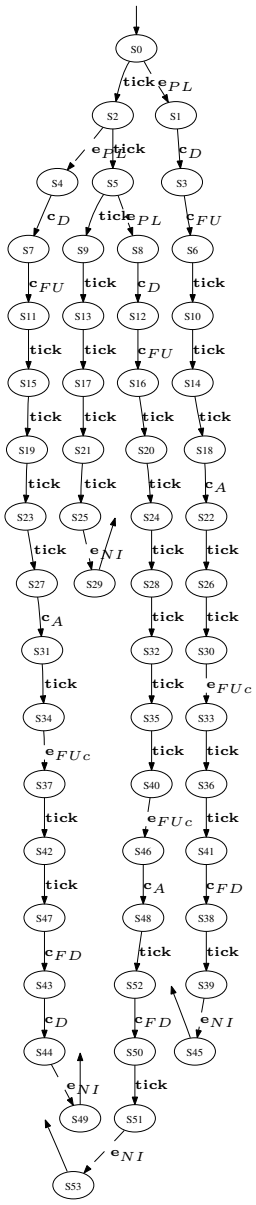

Figure 13: A deterministic supervisor extracted from the least restrictive one

B.5

Parameter file

```
[Fuse parameters]
real_fuse_position    = 0.12348
real_cleaner_position = 0.17066
real_speed_max        = 0.303
real_speed_min        = 0.070
real_deceleration     = 4.7
real_acceleration     = 3
real_time_fuse_open   = 0.5
real_time_pl_max      = 0.1
real_time_pl_min      = 0.05
number_of_time_steps  = 28
```

The least restrictive supervisor is shown on Figure 14. The deterministic supervisor extracted as discussed in Remark 4 is shown on Figure 15.

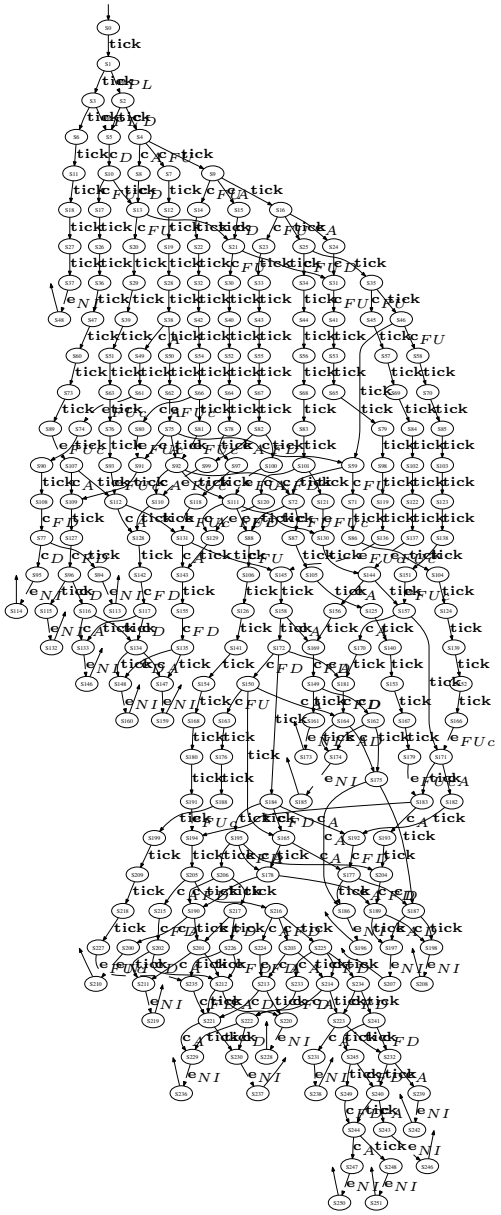

Figure 14: Least restrictive supervisor

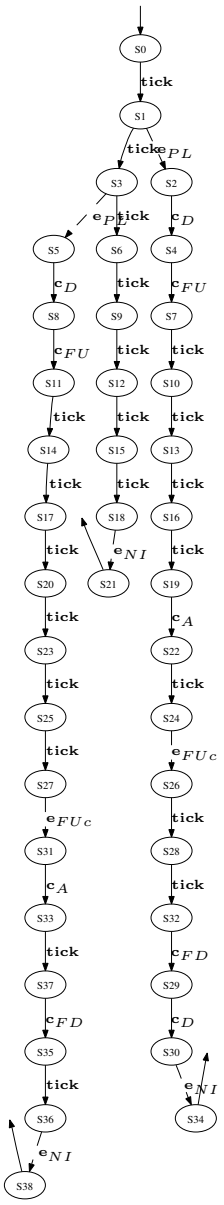

Figure 15: A deterministic supervisor extracted from the least restrictive one