Technical Report #2000-440

An Application of Discrete-Event Theory to Truck Dispatching

Department of Computing and Information Science

Queen's University, Ontario, Canada

Stéphane Blouin, Martin Guay and Karen Rudie

September 11, 2000

Acknowledgment

We would like to thank Dr. W.M.Wonham for providing the first author with facilities (office space and computer access) at the University of Toronto for the academic year 1999-2000 and for helpful comments on an earlier incarnation of this report. We are grateful to James Kresta and Mike Lipsett, at the Syncrude Research and Development Center in Edmonton (Alberta), for the information provided on the truck dispatching process.

Abstract

This report proposes a theoretical framework to automate the making of decisions for the discrete part of a large system. This research aims to provide a procedure that verifies possible decision scenarios and derives the optimal sequence of actions to take. For this purpose, two distinct theories are explored: discrete-event systems (DES) and vector discrete-event systems (VDES). The capabilities and limitations of these theories are examined with the help of an industrial example. The application consists of an oilsand extraction process whose discrete subsystem models the task of dispatching a fleet of trucks. A set of production and maintenance specifications are enforced while preserving the optimality in the execution of commands. In the DES framework, various supervisors (centralized, modular, hierarchical and timed) are synthesized. In VDES, only centralized supervisors are synthesized. Vector DES theory provides a more compact representation than DES. On the other hand, the DES framework remains a highly attractive approach due to its modelling capabilities and due to the availability of well-established tools.

Contents

List of Figures

List of Tables

1 Introduction

Complexity is an inherent limitation of current control techniques. Although it is possible to conceive of the control of large and complex systems, current tools are not amenable to provide simple and reliable ways of integrating and orchestrating decision-making mechanisms.

In manufacturing and industrial applications, there exists an important class of complex systems consisting of processes that exhibit interaction of continuous and discrete dynamics. In most cases, the continuous and discrete behaviours can be segregated so that they naturally form subsystems of the overall process. The continuous behaviour arises due to the physical aspects of the processes while the discrete component originates from control strategies implemented through programmable-logic controllers (PLC) or digital controllers.

Since complex systems can rapidly exceed human capability to capture detailed dynamics of the system's behaviour, the making of a decision is likely to be performed with the help of heuristics and thus may possibly be suboptimal. For situations where complexity increases or where the process configuration is continuously changing, the tasks of making adequate decisions and taking appropriate actions become even more difficult. Moreover, complex systems may have global objectives and performance requirements that require a mechanism to decompose global goals into subtasks and transmit the proper control actions to achieve them.

This report proposes a theoretical framework for making decisions based on the discrete part of the process only. The decision-making procedure for the overall (continuous and discrete) system is left for further project. The objective of the current research is to provide a systematic procedure for verifying possible decision scenarios and deriving the optimal sequence of actions to take. Two distinct theories are explored: discrete-event systems (DES) and vector discrete-event systems (VDES). The capabilities and limitations of these theories are examined with the help of a running industrial example.

The report is structured as follows. Section 2 presents the industrial application used throughout the report. It also gives some rules and assumptions for modelling. Section 3 provides the structure of the study as well as the basis of comparison for the DES and VDES approaches. Section 4 contains the modelling details and the supervisory controls for each of the cases studied. Observations on the implementation of DES and VDES theories are also provided. Finally, Section 5 concludes with the advantages and disadvantages of each approach as well as suggestions for future work. To complete the report, appendices A and C supply a brief summary of DES and VDES theories while appendices B and D contain computational details.

2 Problem Definition

This section presents the industrial application upon which the comparison of DES and VDES techniques will be based. The application of interest consists of the Syncrude oilsand extraction process as operated in Fort McMurray, Alberta. This process provides an industrial application that is rich in decision-making problems. This application also possesses interacting subsystems each of which has its own control challenges. This section proposes one approach to capture most of the rich behaviour inherent to the process without losing critical dynamics and yet challenging the DES and VDES theories to provide systematic procedures to derive a solution.

2.1 Industrial Context and Objectives

The oilsand separation process consists of three main operations: extraction, hydrotransport and separation as shown in Figure 1. The extraction part consists of shovels which are located at specific sites of the mine (Sites A, B, and C), each of which unloads its bucket into trucks. Trucks then transport the ore to a crusher (feeding a hydrotransport system), taking different routes without necessarily returning to the same shovel on the return trip. Distances separating the extraction locations from the crusher are of the order of two to four kilometers. The complete production has three shovels of two different bucket sizes (40 and 80 tons) dispersed over three locations and 30 trucks of two distinct volumes (240 and 320 tons).

As described above, the extraction behaviour is naturally driven by the occurrence of events (road conditions, payload capacity, etc.). Therefore, the extraction process will be better characterized by a set of discrete variables while hydrotransport and separation have continuous dynamics. Moreover, discrete actions taken by the extraction portion of the production affect the downstream continuous dynamics.

In the actual setup the dispatch of trucks is performed by a coordinator. The dispatch rules consist of a mixture of the dispatcher's experience and a set of heuristics. Clearly, the truck fleet size as well as the possibility of various scenarios can rapidly exceed the dispatcher's capability to capture fine details of the overall process. That is without considering the fact that the shovel locations naturally vary as mining operations proceed thus translating into changes of the route characteristics (length, quality, etc.).

In this context, it is natural to seek a systematic procedure for making decisions that will ensure effective use of the existing resources (e.g., trucks). This study focuses on the extraction process to propose decision tools by enhancing the dispatcher's vision of the various possibilities. These tools must also guarantee some degree of performance (optimality) based on the knowledge provided on the overall behaviour. Once the decision mechanism is formalized, control objectives related to the continuous parts can be considered.

2.2 Operating Rules and modelling

The following operation rules have been considered while modelling the extraction process:

- The average truck speed, under normal conditions, is 20 km/hr while it can go down to 15 km/hr on certain occasions. The quality of the roads (soft roads, hills,...) as well as the truck conditions (tire temperature, ...) can result in a reduction of the truck speed

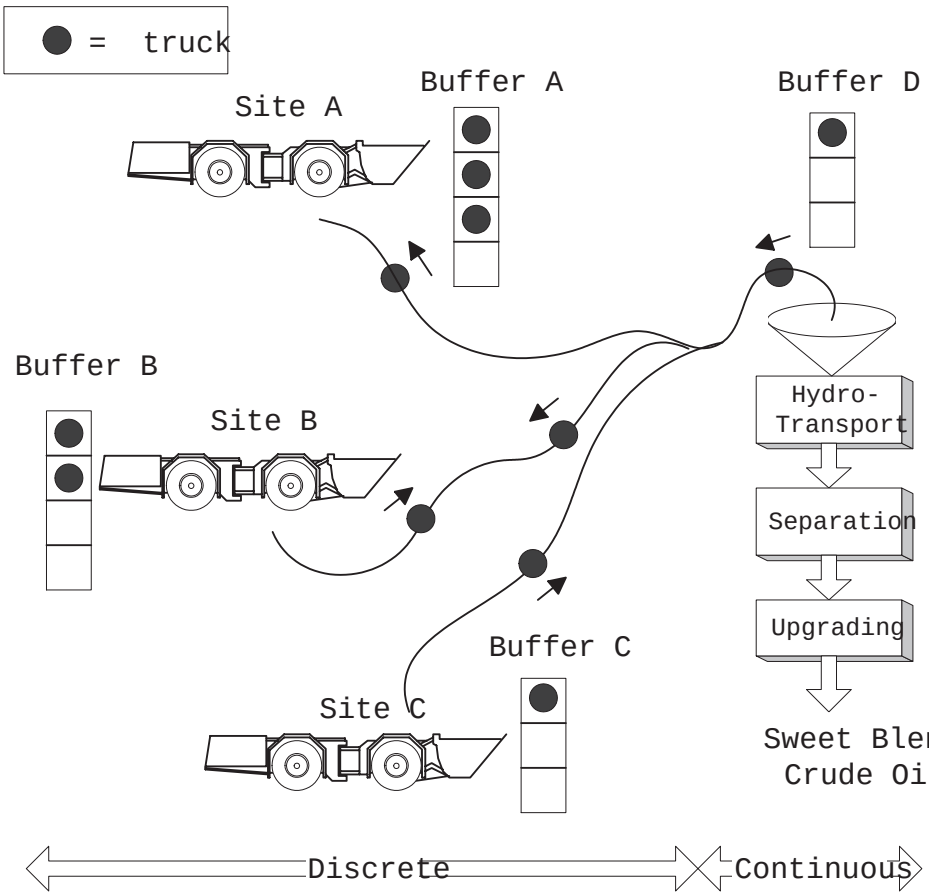

Figure 1: Oilsand Extraction Operation

or of its payload capacity. Therefore, a change of speed resulting from truck and/or road conditions can be captured by the time needed to execute an operation. We will borrow this point of view later in our discussion.

- The shovels do not break down.
- The production runs continuously except for coffee breaks. The latter are not handled here since they cause relatively short-lived (albeit significant) disruptions.
- As trucks may break down, the size of the truck fleet must compensate for breakdowns. Consequently, trucks in excess may have to wait in a queue before a shovel becomes available. Thus the shovels and the crusher should be equipped with queues.
- Additional information provided by Syncrude allowed the trucks to be modelled as finite state machines with the following states: idle, queued, loading, unloading, travelling, broken, and being repaired. For this project, Syncrude suggested that the idle and queued states be equivalent.
- Circulation rules, which govern the priority of passage between trucks at intersections, were also given but are not explicitly used at this time.

However, additional and fictitious scenarios are also introduced to enrich the model definition and explore the capabilities of the theory. Details of the scenarios will be provided in Section 3. As the applicability of any theory largely depends on its capability to translate effectively the production rules, the DES and VDES modelling steps will be explained extensively.

2.3 Some Simplifying Assumptions

To demonstrate the feasibility of the application of DES and VDES theories to truck dispatching, we choose to derive a simplified process that possesses all the key features of the original system. The main reason for this simplification is not conceptual but rather organizational. We want to preserve the main elements of the problem while ensuring an easy presentation of the otherwise large problem. Among the similarities, the determination of the most appropriate configuration and use of trucks considering the amount, location and capacity of shovels remain of primary interest.

The simplified approach is motivated by the following observation. In the oilsand extraction context, all trucks and shovels exhibit the same behaviour (identical DES and VDES representations). Therefore, by enlarging the fleet of trucks or shovels, one only augments the dimension of the problem without altering its complexity and therefore without changing the existence of a solution. Thus, what is achievable on a reduced scale (with fewer trucks and shovels) can be extended to the actual scale.

Finally, some aspects of the production are altered to facilitate this study by limiting the size of the overall system and controller. For instance, the shovels remain of 40 and 80 tons while the trucks' capacities are decreased to 80 and 160 tons. Distances are also reduced to minimize the size of the problem in a timed DES framework. For instance, a simplification was made on the time unit used by reducing hauling distances, or equivalently by increasing the hauling speed. Since all truck speeds are averages, time bounds can be used to reflect the speed span for the hauling time. If a speed decrease is required, the time bounds can be changed or another event can be defined with reduced time bounds.

3 Structure of the Study

It is known that the production configuration (mainly driven by the number of operational trucks and shovels) can change frequently. In this project, we perform sequential passes of increasing complexity to converge to a model sufficiently rich and complex in behavior. The strategy is to develop a stable structure and its respective controller before adding a new feature (Simon, [11]). This progressive approach shows how DES and VDES theories enable us to meet numerous specifications with a varying production layout. In the present case, the production layout is made more complex and more precise through five progressive scenarios listed in Table 1. These cases (labeled from 1 to 5) are differentiated by the following characteristics: the number of shovels (first column), the number of trucks (second column), the number of shovel locations (third column), the possibility of truck breakdowns (indicated by Yes or No in the fourth column) and the presence of queues at extraction and crusher sites (last column).

	Shovels	Trucks	Locations	Breakdowns	Queues
Case 1	1	3	1	No	0
Case 2	1	3	1	Yes	0
Case 3	2	3	1	Yes	0
Case 4	2	3	2	Yes	0
Case 5	2	3	2	Yes	1

Table 1: Progressive Production Layout

The approach taken is to cast the variable nature of the process within specifications while the invariant aspects of a piece of equipment are intrinsic to the plant description. Throughout sections 4.1 and 4.2, a set of seven specifications will be either implemented or, at least, modeled. In reference to the case number of Table 1, the specifications include the following:

Cases 1-5:	Bucket capacity for each truck,
Case 1:	Only one truck loaded at a time by the shovel,
Case 2:	Priority of repair of 160-ton trucks over 80-ton trucks,
Case 2:	Limit of one truck breakdown for four successfull loads,
Case 4:	After three 160-ton truck breakdowns, only 40-ton shovel can load the truck,
Case 5:	Queue for trucks at shovel sites,
Case 5:	Blend specification.

Table 2: Production Specifications

Among the untimed specifications of the process one finds the number of shovels available (which also affects the plant size), the size of the queues at the extraction and dumping areas, the load capacity of each truck, the blend of trucks required to satisfy a specific ore content, and the priorities of repair in case of trucks breakdowns. An example of a timed specification is the time needed for delivering ore at the crusher including the time required for loading, hauling and unloading.

The synthesis of a controller for the truck dispatching task is performed in two steps. First, Section 4.1 presents the modelling and synthesis aspects of DES theory. Then, Section 4.2 covers the same scenarios using VDES theory.

Among the existing techniques available in DES theory, we focus on centralized supervision, modular supervision, hierarchical supervision and timed DES. These concepts are summarized in Appendix A. Table 3 shows which of these supervisors is derived for each case.

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**
Centralised Superv.	X	X	X	X	X
Modular Superv.	X	X	X	X	X
Hierarchical Superv.		X		X	X
Timed Superv.	X	X			

Table 3: DES Supervisors for Different Cases

Full details for the DES supervisor synthesis are not provided here but only the main issues and observations are given. However, the main DES or TDES names are provided so the reader can refer to Appendix B of this report. The MAKEIT files (readable with any text editor) contain the history of all operations. Once again, for the sake of brevity, not all files are provided in this appendix. Even for the simplest cases the final result remains prohibitively large to be presented in a written form.

Since in Section 4.1 a DES supervisor is derived for each of the cases in Table 1 and specifications of Table 2, Section 4.2 mainly focuses on the modelling aspects of VDES to finally solve the last case (Case 5) whose full details can be found in Appendix D. The motivation for this procedure is that, contrary to the cumbersome solution of the full problem, it provides more insights than using the automated procedure in VTCT (a software package being developed at University of Toronto to perform VDES operations).

4 Control Synthesis for Truck Dispatching

In this section, the five cases described in Table 1 will be sequentially solved by using DES and VDES theory. This approach has the benefit of depicting the progressive increase in complexity (by inserting one new feature at a time) and its consequences (or computational cost). The objective here is to obtain stable structures up to the fifth case. Once this stage is reached, further features can be incorporated in the system to reflect the real process.

The models for DES, timed DES, and vector DES as well as their control synthesis techniques (modular and hierarchical) have the flavor of the work performed by Wonham and coworkers in [9], [2], [13],[6] and [7]. For more fundamentals on the topics, the reader is encouraged to refer to those references. An abbreviated introduction to the matter is also provided in appendices A and C. Other well-known timed DES and hierarchical DES approaches can be found in [1] and [3], respectively.

The text is organized as follows. Section 4.1 provides a thorough analysis of the process using DES theory. Then, Section 4.2 solves the same cases using VDES theory. Section 4.3 concludes by providing a discussion on various aspects from the DES and VDES approaches.

4.1 DES Supervisory Controls

In the next sections, the scenarios and specifications described in Section 3 will be sequentially modeled and solved using DES tools. The models for the plant and specification are also explained with the help of figures while their computational details can be found in Appendix B.

The software CTCT and TTCT have been used to synthesize supervisors for DES and TDES. The software can be found at the WWW site provided in [12].

4.1.1 Supervisory Controls for Case 1

The first scenario is composed of one shovel of 80 tons and three trucks (one of 80 tons and two of 160 tons) located at the same ore source. The distance between the extraction site and the crusher is identical for all trucks. Also, no breakdowns are allowed and no queues are adjoined to the shovel or to the crusher. The DES representing the plant is composed of $SHOVEL80$, $T1C80$, $T1C160$ and $T2C160$ (Figure 2) whose events are summarized in Table 4. The resulting plant DES $SYST1$ (Figure 3) is the shuffle product (product of DES depicting all possibilities of asynchronous events) of all finite-state machines in Figure 2. Note "slfp{ }" means self-loop at every state of the DES with events in brackets.

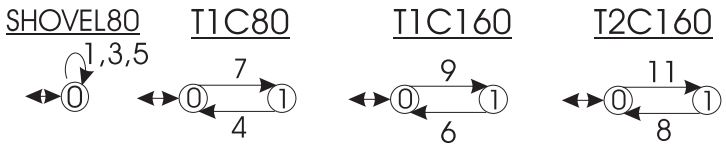

Figure 2: Case 1 - Shovel and Trucks DES

As there is only one shovel and no queue, many trucks can have access to the shovel and the loading of a specific truck must be differentiated from the loading of others. This justifies

Event	Meaning	Event	Meaning
1	loading of $T1C80$	7	full loading of $T1C80$
3	loading of $T1C160$	8	unloading of $T2C160$
4	unloading of $T1C80$	9	full loading of $T1C160$
5	loading of $T2C160$	11	full loading of $T2C160$
6	unloading of $T1C160$		

Table 4: Case 1 - Event Description for $SYST1$

the presence of controllable events 1, 3 and 5 for the loading of the 80-ton and the two 160-ton trucks. Since we represent the shovel DES as a singleton, the loading events must be controllable since the shovel can only load one truck at a time (supervisor must disable other loading events). The truck events, being loaded and unloaded, are controllable (eventually useful when full or partial loading will have to be chosen) and uncontrollable (depends on roads and truck conditions), respectively.

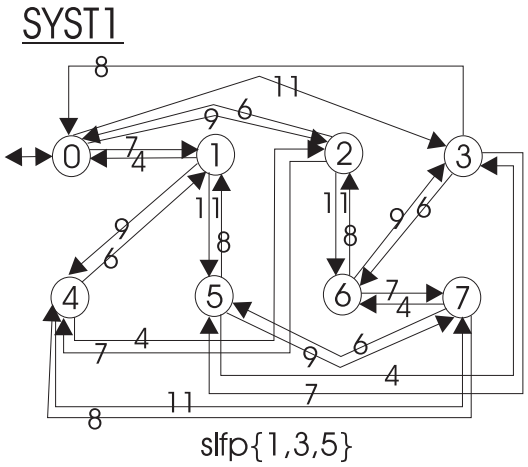

Figure 3: Case 1 - $SYST1$ DES

The first untimed specifications to be assigned to $SYST1$ refer to the bucket capacity of each truck. The capacities of trucks $T1C80$, $T1C160$ and $T2C160$ are specified by buffers of different sizes $B1$, $B2$ and $B3$ (Figure 4). The buffers translate the fact that only one bucket is required to fill $T1C80$ (buffer $B1$) while two are needed for each of $T1C160$ and $T2C160$ (buffers $B2$ and $B3$). Moreover, the buffers specify the requirement that only one truck is loaded at a time. For instance, in buffer $B2$ when event 3 occurs, other loading events 1 and 5 are not allowed until truck $T1C160$ is full, signified by event 9. All states of each buffer have self-loops of events of the other trucks to keep an independence of operation among the trucks.

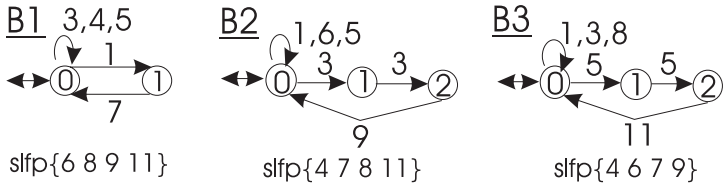

Figure 4: Case 1 - Buffer Specifications for $SYST1$

With this setup, a centralized supervisor $SUPER1$ is determined. As designed, the buffers $B1$, $B2$ and $B3$ enforce the truck capacity and the loading of one truck at a time. However, taken individually they do not prevent the underflow or overflow of the trucks. It is possible (in $B1$, $B2$ and $B3$, Figure 4) for loading events (namely 1, 3 and 5) to occur before unloading events (4, 6 and 8) take place at state 0. For instance, a faulty sequence in $B2$ would be 3-3-9-3 since a third loading occurs before an unloading, thus leading to a truck overflow for $T1C160$.

Therefore, modular supervisors must be designed to respect the underflow and overflow rules. As an example, consider $T1C80$ and its respective buffer $B1$. The underflow rule suggests that unloading must not occur when the truck is empty (avoid event 4 before 7). The overflow rule specifies that no other loading can happen before the truck is unloaded (avoid another event 1 before event 4) . These requirements are satisfied by adding an additional state to the buffer that brings the sequence loading-unloading in the right order (equivalent to including the truck model in the specification). Finally, we obtain the proper buffer supervisors $MB1$, $MB2$, and $MB3$ (Figure 5) whose intersection generates the modular supervisor $MSUPER14$. Moreover, the latter is optimal since it generates the same language as $SUPER1$ when put on-line with $SYST1$.

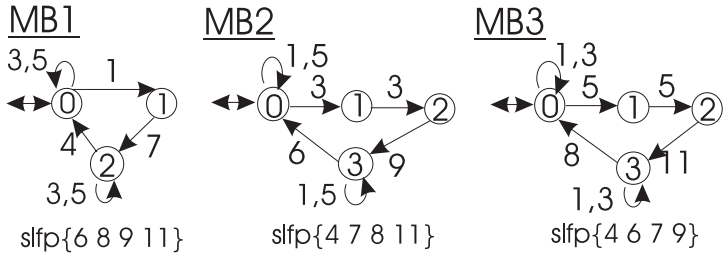

Figure 5: Case 1 - Modular Supervisors for $SYST1$

A timed DES that models this simple system is then developed to detect any possible problems. In fact, this first exercise raised some questions since the process description provided by Syncrude contained finite lower and upper time bounds for all events. Therefore, the upper time bounds of all controllable events were set to infinity as required by theory (Section A.3). However, a critical modelling question remains: "*Which one of the upper bound or controllability property should be kept when the effects of those properties on the overall process are not known?*" Obviously, this question refers to the need for a better understanding of the process and the answer requires a deeper analysis to assess the most realistic property for each event.

Arbitrary lower bounds of value 1 were given for shovel loading events 1, 3 and 5 while truck events had the time bounds: 4 [1 2], 7 [1 ∞], 6 [1 2], 9 [1 ∞], 8 [1 2], 11 [1 ∞] where elements in brackets have the following meanings [(lower time bound) (upper time bound)]. All controllable events are set to be forcible (an assumption that facilitates the first approach). With the previous specifications and given time bounds, no supervisor exists.

Similar to $SHOVEL80$, a partial expansion of $SYST1$ with the tick event and the above time bounds is provided in Figure 6. It is shown that if event 7 occurs at state 1 then, from state 5, event 4 becomes imminent and is impossible to stop. This is because event 4 reached its upper time bound (i.e., no more tick allowed) and there is a limit to the number of ticks the forcible events 1, 3, 5, 7, 9 and 11 can preempt. Similar conclusions can be drawn for events 6 and 8 such that there exists no controllable subsystem in $SYST1$. The selected time bounds are too restrictive for the system and the set of possible actions must be enlarged. This is performed

Event	Meaning	Event	Meaning
1	loading of $T1C80$	9	repair of $T1C80$
2	unloading of $T1C80$	10	unloading of $T2C160$
3	loading of $T1C160$	11	full loading of $T1C160$
4	breakdown of $T1C80$	12	breakdown of $T2C160$
5	loading of $T2C160$	13	repair of $T1C160$
6	unloading of $T1C160$	15	full load of $T2C160$
7	full loading of $T1C80$	17	repair of $T2C160$
8	breakdown of $T1C160$		

Table 5: Case 2 - Events Description for $SYST2$

by fixing the lower bounds of events 1, 3 and 5 to 0 (1 tick gap with upper time bounds of uncontrollable events 4, 6 and 8). This modification amounts to allowing the loading events 1, 3 and 5 to take place with no delay once they are given the possibility to occur. While the untimed supervisor has 28 states and 64 transitions, its timed equivalent has 375 and 899. No temporal specifications are added yet to the problem.

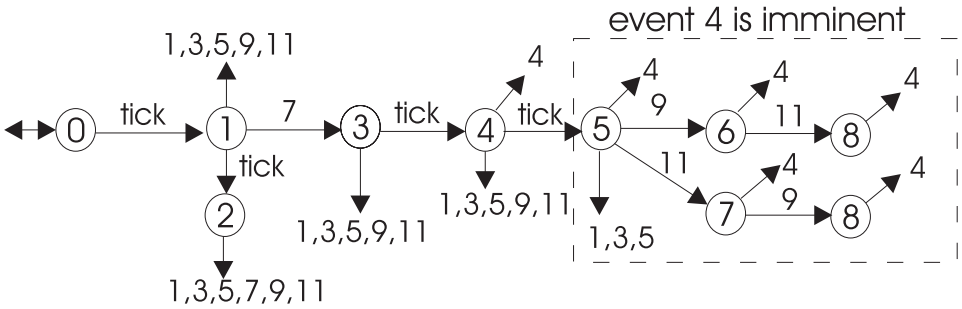

Figure 6: Case 1 - Partial Expansion of $SYST1$ with Timed Events

4.1.2 Supervisory Controls for Case 2

For the second scenario, truck breakdowns are introduced. Even if this is not a major change in the process description, this extension of truck models is used as a valuable refinement towards a more complex model. Names for trucks, shovels, buffers and modular buffer supervisor DES remain the same as in Section 4.1.1 but the global specification (resulting from $B1$, $B2$ and $B3$) is named $SPEC2$. Again the plant DES $SYST2$ is composed of trucks and shovel as shown in Figure 7. Due to the addition of breakdowns, the list of events is altered and the meaning of each event can be found in Table 5.

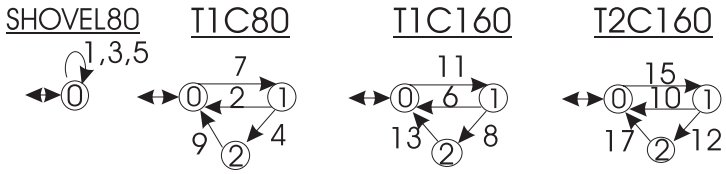

Figure 7: Case 2 - Shovel and Truck DES

With this new configuration a centralized supervisor $SUPER2$ is obtained. Once again, a

modular supervisor is built from the buffers DES. The changes required to synthesize the supervisors are similar to these of the previous case. In addition, care must be taken to account for the ordering of repairs compared to other events. To avoid overload occurring before an unload, only the repair events must lead to a loading (Figure 8). From proper supervisors $MB1$, $MB2$ and $MB3$ for the buffers, a modular and optimal supervisor $MSUPER2$ is determined.

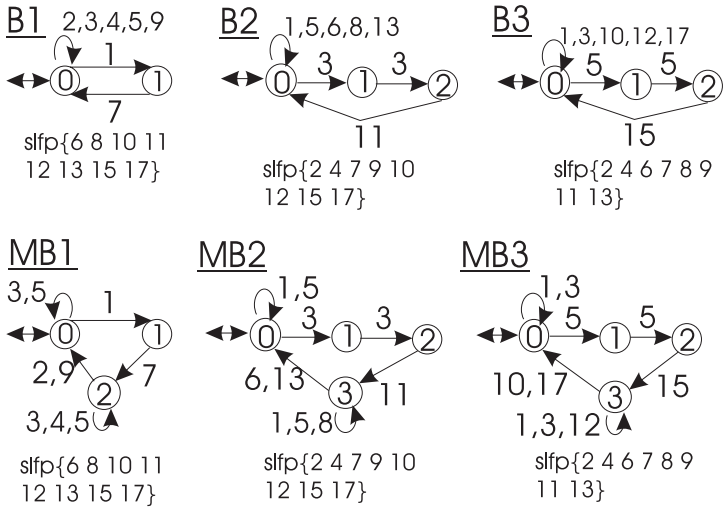

Figure 8: Case 2: Buffer Specification and Modular Supervisors for $SYST2$

With the possibility of breakdowns, a new specification is introduced. It consists of forcing the repair of any 160-ton trucks (if broken) before the repair of the 80-ton one (when broken). The specification $REPAIR$, built from two individual specifications $REPAIR1$ and $REPAIR2$ (Figure 9), can be used as a modular supervisor to guarantee the correct order of repairs. The combination of $SPEC2$ and $REPAIR$ provides the new specification $SPEC22$. From this we get the centralized supervisor $SUPER22$ and its modular equivalent $MSUPER22$ that is also optimal.

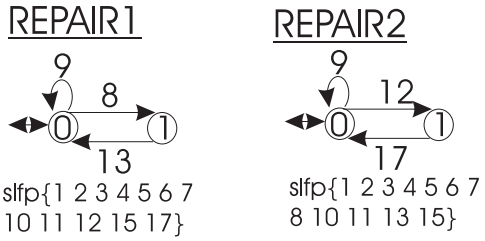

Figure 9: Case 2 - Second Specification

Now, we can also perform hierarchical supervision. Since we introduced breakdowns, it is possible for the manager to monitor the status of production by looking at how many breakdowns have occurred compared to the loads successfully brought to the crusher. For this purpose, a sample of the truck fleet is taken, here one 160-ton truck ($T1C160$), and the manager *vocalizes* the states representing the full and broken status (events 8 and 11).

The development of the hierarchical supervisor is performed as follows. The reader may refer to [12] for technical details. The states of $SYST2$ where events 8 and 11 lead are first vocalized

as states 10 and 11 (Figure 10). This means the following:

For event 8:	States 7 12 15 17 20 22 23 25 26	are vocalized as 10
For event 11:	States 2 5 8 10 13 16 18 21 24	are vocalized as 11

These vocalized states are a means of recording the occurrence of specific events (8 and 11) by tagging their target states with particular labels (10 and 11). The latter are not related to the low-level state labels and from now on, we strictly refer to states 10 and 11 as vocalized states. For convenience, the modified $SYST2$ is renamed GLO as it is our low level system. A portion of GLO is shown in Figure 10 where vocalized states are represented by a double circle. The DES GLO is then made output control consistent (named $OCGLO$) and then hierarchically consistent (named $HCGLO$) by appropriate operations [12].

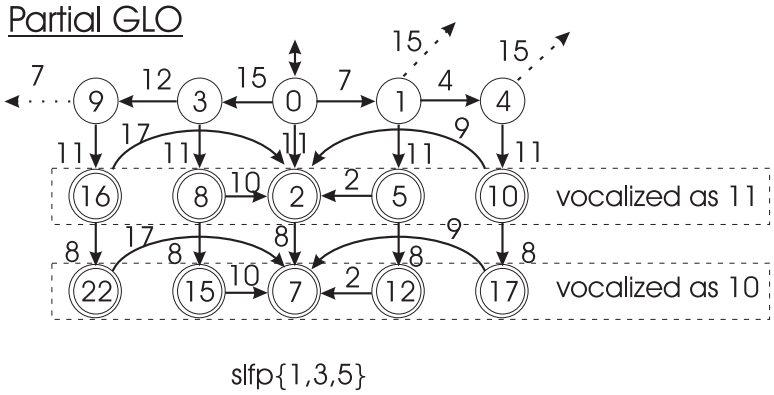

Figure 10: Case 2 - Partial GLO

By vocalizing some of the DES states we make abstraction of a portion of the low-level behaviour. In the sequel, we will refer to low-level sequences of events linking two vocalized states as *paths*. To guarantee a unique low-level execution following a high-level command, the high-level description must be enriched with the controllability nature of the paths. Namely, the controllable/uncontrollable paths of GLO will become the controllable/uncontrollable events of the high-level abstraction. Thus, a high-level event is controllable (resp., uncontrollable) based on the presence (resp., absence) of low-level controllable events within the path. For example, the path [0 11 2] linking state 0 to state 2 via event 11 is a controllable path while the path [8 10 2] is an uncontrollable one. The controllable (resp., uncontrollable) high-level event labels are identified by extending the low-level vocal state labels with an odd (resp., even) third digit. For instance, the low-level state 2 of Figure 10 (and vocalized as 11) becomes high-level events 111 and 110, due to the controllable and uncontrollable paths [0 11 2] and [8 10 2], respectively. As a result of this operation, the high-level alphabet is composed of events 100, 101, 110 and 111. The procedure described above leads to output control consistency (OCC).

The manager has a special interest in breakdowns (low-level event 8) and successful loadings (low-level event 11). By the definition of OCC, the controllable event 11 makes the path controllable, and since the uncontrollable event 8 always connects two vocalized states, it forms an uncontrollable path. So in the high-level abstraction of Figure 10, events 111 and 100 are the events of interest for the manager because they correspond to events 8 and 11 at the low level.

The manager is given limited observation and control over some of the low level events (here breakdown and full loading events). Suppose the manager wants to limit the number of break-

downs to one for each four successful loads of the truck. Using the simplified high-level alphabet only, the manager can draw his specification DES, named $SPECHI$, as in Figure 11. Once put on-line with the high-level map of the real production, we find that it cannot be done unless no production is allowed (a rather sad situation). For this specific case, the only capability the manager has is to stop the production whenever a certain number of breakdowns occurs (DES named $SPECHI2$, Figure 11). The manager can impose with success this new specification on the low-level system, the production.

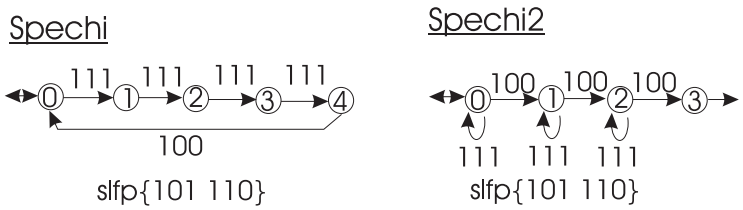

Figure 11: Case 2 - High Level Specifications

In the final step of the model refinement, time is introduced into the system using the following time bounds:

1 [0 ∞]	2 [2 3]	3 [0 ∞]	4 [1 2]	5 [0 ∞]
6 [2 3]	7 [1 ∞]	8 [1 2]	9 [2 ∞]	10 [2 3]
11 [1 ∞]	12 [1 2]	13 [2 ∞]	15 [1 ∞]	17 [2 ∞]

In this first approach, all controllable events are set to be forcible (which represents an ideal case that could be relaxed later). With the above time bounds, central and optimal modular supervisors are developed for the buffer specifications $B1$, $B2$ and $B3$ only (i.e., without any repair specification). The timed supervisors exist and they are similar to their untimed version except for the presence of the *tick* event. This confirms that any untimed specification should be worked out first in an untimed framework before switching to TDES because the size of the system increases dramatically when the occurrence of the *tick* event is introduced. For instance, the pair $(\#states, \#transitions)$ for the central supervisors when submitted to the buffer specification only is, respectively for the untimed and timed case, of value (72,243) and (1152,3073).

The temporal specification to be studied is to determine how fast one cycle (load/unload or load/breakdown/repair) can take. This is achieved by building a timer specification $TIMER$ [12] that is compared to the actual timed supervisory control $SUPER2$ (same name as untimed scenario but different supervisor). If among $TIMER$ there exists a supremal controllable sublanguage w.r.t. $SUPER2$ then the timer specification can be satisfied. Progressively, timers of 3, 4, 5 and 6 ticks were tried. For the first three timers there is no possible supervisor (see Appendix B) while for the fourth one TTCT failed. However, by inspection of any truck DES (since they all have the same time bounds) it can be seen that a minimum of seven ticks is required (for the longest uncontrollable path: 2 ticks before event 7 can occur (eligible after 1), 2 ticks before event 4 becomes imminent and 3 ticks before event 9 can take place (2 ticks to become eligible)).

The time bounds were initially assigned according to data provided for each event (load, unload, breakdown and repair) of Figure 7. For example, the breakdown events 4, 8 and 12 (if they occur) normally take place in the first period of hauling and this justified the time bounds [1 2] (compared to [2 3] for successful unload events 2, 6 and 10). However, giving

Event	Meaning	Event	Meaning
1	$T1C80$ loaded by $SHOVEL40$	10	unloading of $T2C160$
2	unloading of $T1C80$	11	$T2C160$ loaded by $SHOVEL80$
3	$T1C160$ loaded by $SHOVEL40$	12	breakdown of $T2C160$
4	breakdown of $T1C80$	13	full loading of $T1C80$
5	$T2C160$ loaded by $SHOVEL40$	17	repair of $T1C80$
6	unloading of $T1C160$	19	full loading of $T1C160$
7	$T1C80$ loaded by $SHOVEL80$	23	repair of $T1C160$
8	breakdown of $T1C160$	25	full loading of $T2C160$
9	$T1C160$ loaded by $SHOVEL80$	29	repair of $T2C160$

Table 6: Case 3 - Events Description for $SYST3$

time bounds by only looking at the event itself is misleading. We must also take into account other events leaving from the same node. With the actual time bounds the breakdown event always preempts the unload since it becomes imminent before the unload is eligible. To ensure fairness of occurrence of both events, we must permit at least some tick overlap and we should equate the two time upper bounds to include any late breakdown.

4.1.3 Supervisory Controls for Case 3

For the third case, another shovel of 40-ton capacity $SHOVEL40$ is added to the extraction site (Figure 12). Once again, events labels are redefined (Table 6) and the global system is named $SYST3$.

As the additional shovel is of different capacity than the other (we now have shovels of 40 and 80 tons) the trucks can be filled by a different number of buckets. The direct effect of this new feature is to bring another branch to all truck buffers $B1$, $B2$ and $B3$. Consider $B1$ the buffer associated with $T1C80$ (Figure 13), the buffer stipulates that the truck can be filled by one 80-ton bucket (event 7) or two 40-ton buckets (event 1 twice) both cases leading to a full truck (event 13). As all machinery is at one location, there is no need to distinguish between a load performed by any specific shovel. The modular buffer supervisors $MB1$, $MB2$ and $MB3$ are built with the same rules as in Case 2 and are themselves proper supervisors (Figure 13).

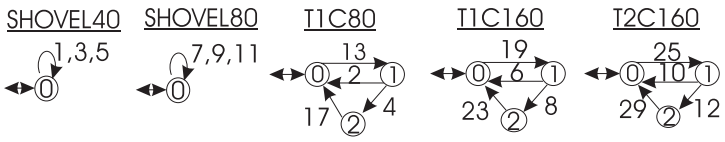

Figure 12: Case 3 - Shovel and Truck DES

4.1.4 Supervisory Controls for Case 4

This scenario brings a feature that disrupts both the truck and buffer DESs (shovels DESs remain identical). As in the real process, each shovel is located at a specific extraction site and trucks go to and from it to perform their loading and unloading. The two extraction sites are at different distances from the crusher and thus the travelling distance (important for the

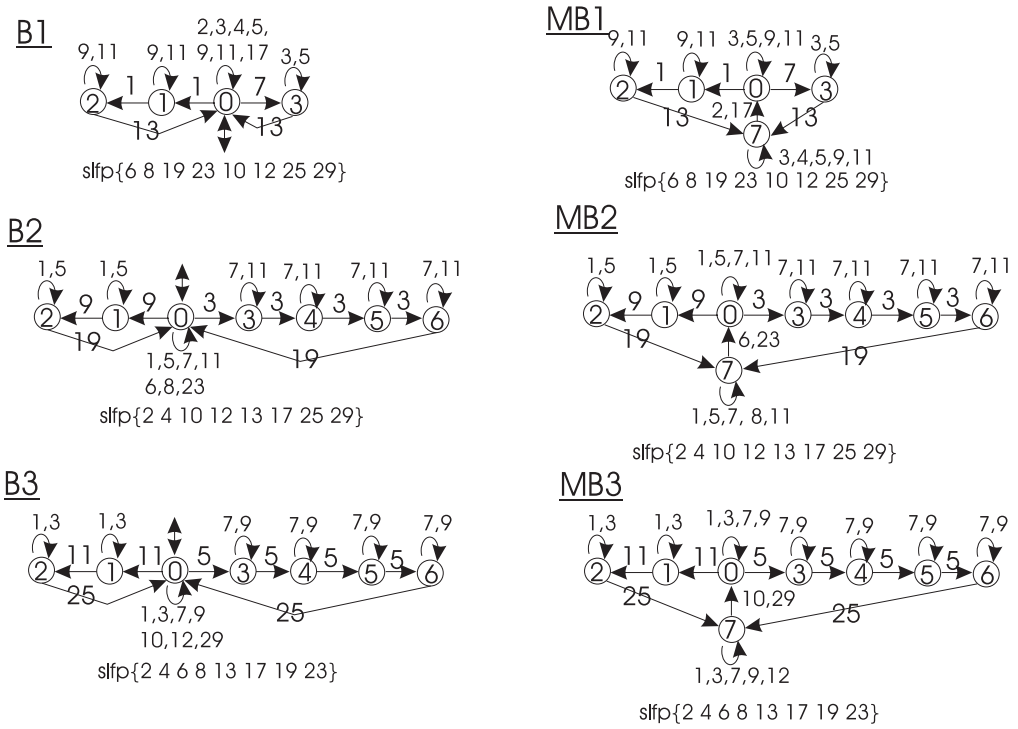

Figure 13: Case 3 - Buffers and Modular Supervisors

timed DES) will vary depending on where the truck is sent to or from. It is assumed that truck dispatching to different extraction sites is a controllable event and that it is performed from the crusher site which also corresponds to the location where the trucks start at the beginning of the day.

From now on, a successful load/unload cycle is described by the following sequence (where events for $T1C80$ of Figure 14 are provided in brackets as a reference): travel from crusher to extraction site (event 13 or 15), load at extraction site by a specific shovel (event 17 or 19), travel from extraction site to crusher (event 2 or 4), unload into crusher (event 21) and start over. A breakdown (event 6) may occur right after the loading and it is followed by a repair (event 23). By assigning a specific shovel to reach, the dispatcher directly enforces travelling distance requirements. In the TDES framework, we can relate a longer distance to a larger upper time bound. Thus, the choice of a shovel would imply the number of buckets needed to fill the truck, and the time required to travel back and forth between the shovel and the crusher. By distinguishing the loading at different locations, we can establish a relation between the dispatcher's choice and its implications on the production. For instance, the buffer $B1$ in Figure 14 has two different travelling distances to reach the shovels (namely events 13 or 15 for $T1C80$) and also different hauling distances to reach the crusher (events 17 or 19). This arrangement even allows us to set a higher average speed when the truck is empty and a lower one when it is full. Finally, breakdown events are not associated to a specific shovel because they originate from normal wear or road conditions. Table 7 provides the meaning for all events of Figure 14.

For the new system $SYST4$, a modular supervisor is built directly from the buffer DES $B1$, $B2$ and $B3$. In fact, one no longer needs to add an extra state to make the buffer DES a

Event	Meaning	Event	Meaning
1	$T1C80$ loaded by $SHOVEL40$	19	$SHOVEL80$ fully loaded $T1C80$
2	haul from $SHOVEL40$ to crusher	21	unloading of $T1C80$
3	$T1C160$ loaded by $SHOVEL40$	23	repair of $T1C80$
4	haul from $SHOVEL80$ to crusher	25	travel from crusher to $SHOVEL40$
5	$T2C160$ loaded by $SHOVEL40$	27	travel from crusher to $SHOVEL80$
6	breakdown of $T1C80$	29	$SHOVEL40$ fully loaded $T1C160$
7	$T1C80$ loaded by $SHOVEL80$	31	$SHOVEL80$ fully loaded $T1C160$
8	haul from $SHOVEL40$ to crusher	33	unloading of $T1C160$
9	$T1C160$ loaded by $SHOVEL80$	35	repair of $T1C160$
10	haul from $SHOVEL80$ to crusher	37	travel from crusher to $SHOVEL40$
11	$T2C160$ loaded by $SHOVEL80$	39	travel from crusher to $SHOVEL80$
12	breakdown of $T1C160$	41	$SHOVEL40$ fully loaded $T2C160$
13	travel from crusher to $SHOVEL40$	43	$SHOVEL80$ fully loaded $T2C160$
14	haul from $SHOVEL40$ to crusher	45	unloading of $T2C160$
15	travel from crusher to $SHOVEL80$	47	repair of $T2C160$
16	haul from $SHOVEL80$ to crusher		
17	$SHOVEL40$ fully loaded $T1C80$		
18	breakdown of $T2C160$		

Table 7: Case 4 - Events Description for $SYST4$

proper supervisor. This is because the shovel loading events (1, 3, 5, 7, 9 and 11) cannot take place before a truck reached a site (events 13, 15, 25, 27, 37 and 39). This strategy avoids any possible overflow. The untimed supervisor is of dimension (1302,4799) with buffer specifications only and of dimension (1302,4774) with the repair specification added.

Hierarchical supervisory control is also performed on the system. However it is done with two trucks only because with all trucks $PSYST4$ (projection of $SYST4$ without events 1, 3, 5, 7, 9, 11) still has over 1000 transitions to visually inspect and manually vocalize (a cumbersome task). The reader can refer to Case 2 for more details on the procedure. We vocalize $T1C160$ at states 2, 4 and 6 or equivalently states in $SYST4$ where events 12, 29 and 31 lead (respective vocal states are 10, 11 and 12).

From the definition of OCC (and for reasons similar to the ones given in Section 4.1.2), low-level events 12, 29 and 31 correspond to high-level events 100, 111 and 121, since they characterize in a unique fashion the paths to vocal states. The manager is now capable of blocking a truck from being loaded by a specific shovel. For the specification of Figure 15, the manager disables successfully the loading of truck $T1C160$ performed by $SHOVEL40$.

4.1.5 Supervisory Controls for Case 5

For the fifth and last scenario, a queue is added to the shovel $SHOVEL40$. Since the queue enters the problem as an additional specification (Figure 16) the truck, shovel and buffer DES are identical to the fourth case. The queue specification is developed as a modular supervisor. To do so, it must respect the system's behavior $SYST5$ (identical to $SYST4$) by allowing no more information than the system contains without being more restrictive than the centralized

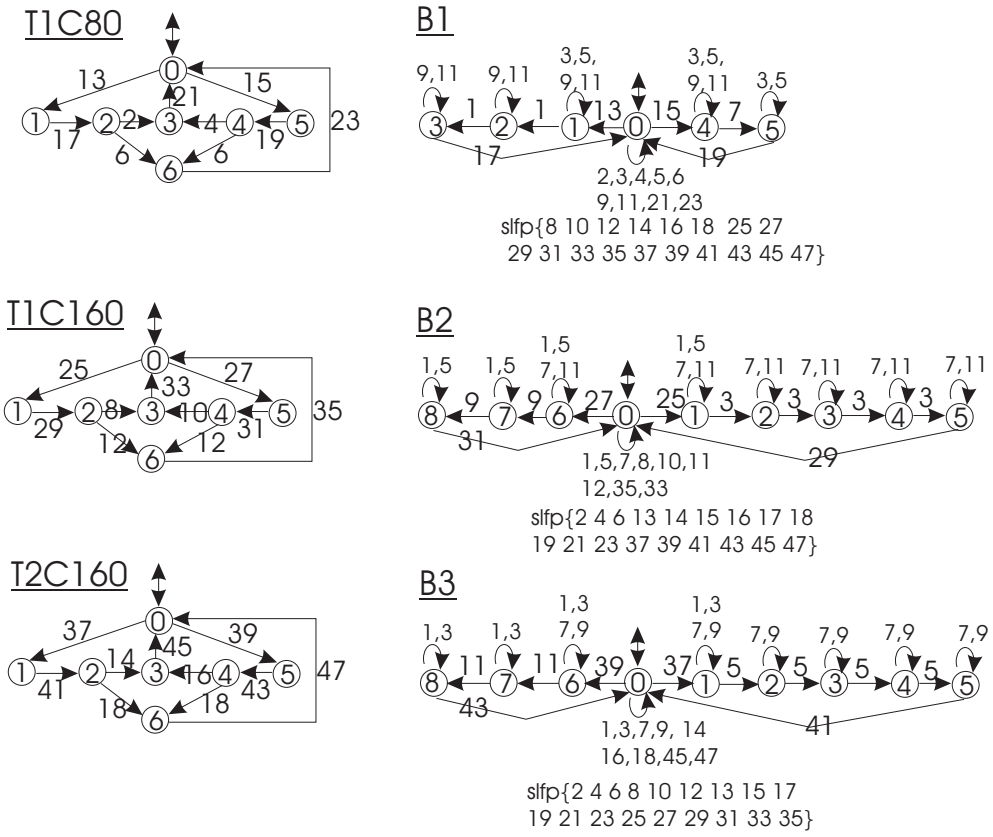

Figure 14: Case 4 - Truck and Buffer DES

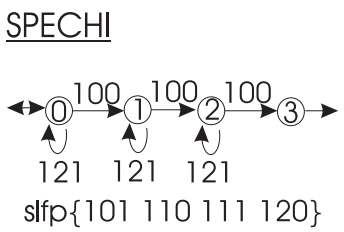

Figure 15: Case 4 - High Level Specification

supervisor. Again, only 2 trucks ($T1C80$ and $T1C160$) are considered even though the shovel events account for three trucks (i.e., contain loading events 5 and 11).

The specification $SPEC5$ includes the buffer specifications $B1$ and $B2$ as well as the queue specification Q. A modular supervisor $MSUPER5$ is found to be optimal. Its language, in the presence of the system $SYST5$, matches the language of the centralized supervisor $SUPER5$. One could think that with only two trucks present and a queue for two trucks at one site the supervisor size should not be different from the one found in case 4. In fact, $SUPER5$ is ten times larger than $SUPER4$. This increase in complexity is due to the fact that, in $SUPER4$, a truck could arrive at a site before another truck and still be loaded last. With the queue specification, we impose the rule "first come, first loaded". Thus the language is more specific and the DES larger.

Hierarchical supervisory control is now performed by vocalizing states following travel events

13 and 15. In $SYST5$ states 1, 6, 17, 18, 31 and 32 are vocalized with 13 while states 2, 9, 10, 21, 22, 35 and 36 are vocalized with 15. From the OCC definition (Section 4.1.2 and [12]), the high-level events of interest are 131 and 151. A new specification $SPECHI5$ is introduced to indicate that the dispatcher wants the truck $T1C80$ to be loaded twice by $SHOVEL40$ and then once by $SHOVEL80$ (Figure 17). This is because $SHOVEL40$ is farther than the other shovel and since the truck tires heat up when travelling too much while loaded, the short trip to $SHOVEL80$ permits its tires to cool down. This specification could also correspond to a requirement for a specific blend. For this case, a controllable high-level supervisor $SUPERHI5$ exists.

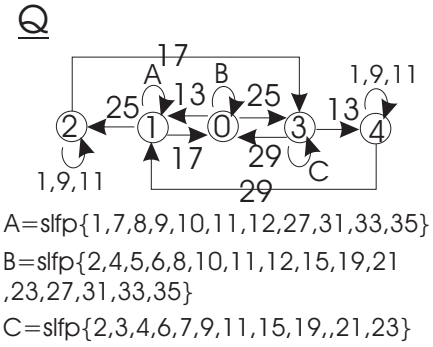

Figure 16: Case 5 - Queue DES

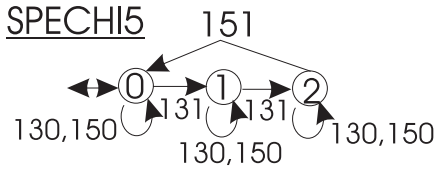

Figure 17: Case 5 - High Level Specification

4.2 VDES Supervisory Controls

The plant and specification DES developed in Section 4.1 are used as a starting point for VDES modelling. However, the details of how the specifications are adapted to a VDES approach are provided. As mentioned previously, only VDES *models* are provided for the first four cases while the fifth case is fully solved. A brief overview of VDES theory can be found in Appendix C while Appendix D contains all calculations related to the solution of the fifth case.

4.2.1 Modelling of Case 1

The first scenario is composed of one shovel of 80 tons and three trucks (one of 80 tons and two of 160 tons) located at the same ore source. The truck capacity specification can be represented by a VDES with an upper bound U representing the number of buckets needed for a specific truck. However, it is not immediately apparent how to derive a static predicate to enforce the requirement that when U loadings have taken place an unloading event must occur. For instance, consider the 80-ton shovel VDES modeled by three states x_1, x_2 and x_3 representing one full bucket, one bucket in a 160-ton truck and one bucket in a 80-ton truck, respectively (Figure 18). To respect the specification of loading only one truck at a time (say the loading of a 160-ton truck or event a_1), we should have a condition of the type: if $x_2 \geq 1$ then $[0\ 0\ 1][x_1\ x_2\ x_3]^T \leq 0$ until $\#x_2 = 2$. The previous condition requires that once the shovel initiated the loading of a 160-ton truck (if $x_2 \geq 1$), the loading of the 80-ton truck is disabled (then $[0\ 0\ 1][x_1\ x_2\ x_3]^T \leq 0$) until it completes the loading of the 160-ton truck with a second bucket (until $\#x_2 = 2$). This type of condition is a combination of static and dynamic predicates; it does not appear to be easily expressed either as a static predicate or entirely as a dynamic predicate. As such, the algorithms in [4] could not be immediately applied.

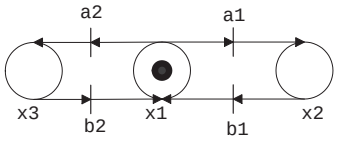

Figure 18: Simplest $SHOVEL80$ VDES

However, we can enforce the specification of having only one truck loaded at a time via the insertion of the truck capacity into the shovel VDES. For these reasons, the buffer and priority specifications are embedded into the shovel VDES (as shown in Figure 19). In the latter, the VDES representations of the shovel and the trucks are named $SHOVEL80$, $TC80$ and $TC160$, respectively. All events found in Figure 19 are defined in Table 8. In general, the meaning of the VDES states (Table 9) is an extension of the events (Table 8).

In VDES, a specification can be enforced by creating interdependences between various VDESs. An interdependence is generated via interconnections between the states of one VDES and the events of another VDES. This must be done in such a way that the number of tokens within each individual VDES remains invariant. In this sense, this is analogous to a mass balance. In Figure 32, an interdependence is assigned between the loading events (b_1 and b_2) and the shovel states (x_3 and x_4). For example, the connection $x_3 - b_1 - x_1$ is composed of an arrow leaving state x_3 to point towards event b_1 and another arrow exiting event b_1 to connect with state x_1. The first arrow enforces the requirement that event b_1 only fires when each

of state x_3 and x_5 possesses at least one token ($x_3 \geq 1$ and $x_5 \geq 1$) while the second arrow makes $SHOVEL80$ available to fire event a_1 or a_3 as soon as event b_1 has taken place. The connections $x_3 - b_1 - x_1$ thus ensure the transfer of a token from x_3 to x_1 for every firing of event b_1. Consequently, whenever event a_1 fires, event a_3 cannot take place until event b_1 occurs, which has the effect of disabling event b_2 in $TC80$ since a token is required in state x_4. Therefore, the Petri net representation of Figure 32 guarantees a certain sequence of events between VDESs $SHOVEL80$, $TC160$ and $TC80$.

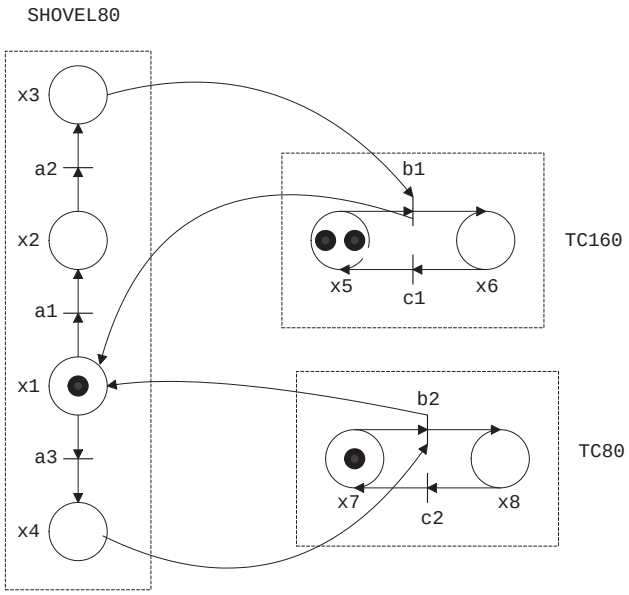

Figure 19: Petri net Representation of Case 1

Event	Meaning	Event	Meaning
a_1	first bucket in one $TC160$	c_1	unloading of one $TC160$
a_2	second bucket in one $TC160$	b_2	full loading of one $TC80$
a_3	first bucket in one $TC80$	c_2	unloading of one $TC80$
b_1	full loading of one $TC160$		

Table 8: Event Description for Case 1

Event	Meaning	Event	Meaning
x_1	idle status for $SHOVEL80$	x_5	idle (empty) status for trucks $TC160$
x_2	half-full loading of one truck $TC160$	x_6	full status for trucks $TC160$
x_3	full loading of one truck $TC160$	x_7	idle (empty) status for trucks $TC80$
x_4	full loading of one truck $TC80$	x_8	full status for trucks $TC80$

Table 9: State Description for Case 1

Similar to the DES case, loading events (b_1 or b_2) determine that a truck is ready to unload (event c_1 or c_2) and that the shovel is available to load another truck (event a_1 or a_3). In this sense, the sequence of events in the VDES of Figure 19 respects the behavior of the DES counterparts in Section 4.1.1. In fact, one notes that the truck VDES models are identical to the DES models of Figure 2.

A sequence of events like $a_1 - a_2 - b_1 - a_1 - a_2 - b_1 - a_1$ leads to an erroneous loading since event c_1 has never taken place and both 160-ton trucks have been filled already. To prevent such a (overflow) situation, dynamic predicates must be used. The dynamic predicates must record the number of loading cycles $\#a_i$ and compare it to the number of unloadings $\#c_i$. That is formulated by the following dynamic predicates

$$\begin{array}{ll} P_1 := & \#a_1 \leq \#c_1 + 2 \\ P_2 := & \#a_3 \leq \#c_2 + 1 \end{array} , \tag{1}$$

where the constants 1 and 2 represent the initial number of trucks in the idle status. An increase in the number of trucks or a change in fleet configuration (trucks idle or not) requires a change in these constants.

Since the Petri net representation does not include the controllable or uncontrollable nature of events, we recall it from Section 4.1. The uncontrollable events are c_1 and c_2 while the controllable ones are a_1, a_2, a_3, b_1 and b_2.

4.2.2 Modelling of Case 2

For the second scenario, truck breakdowns (events d_1 and d_2) and repairs (events e_1 and e_2) are introduced. The impact of additional events and states is to enlarge the VDES dimension and to render the loading dynamic predicates (P_1 and P_2) of Case 1 more complex. By extension, we get the resulting Petri net representation for Case 2 in Figure 20 with respective events defined in Table 10. One notices that the VDES structure for all trucks is again identical to the DES structure provided in Section 4.1.2. The latter contains all the justifications the reader needs to make connections between the VDES structure and the extraction process. In this case, the controllable events are a_1, a_2, a_3, b_1, b_2, e_1 and e_2. The uncontrollable events are c_1, d_1, c_2 and d_2.

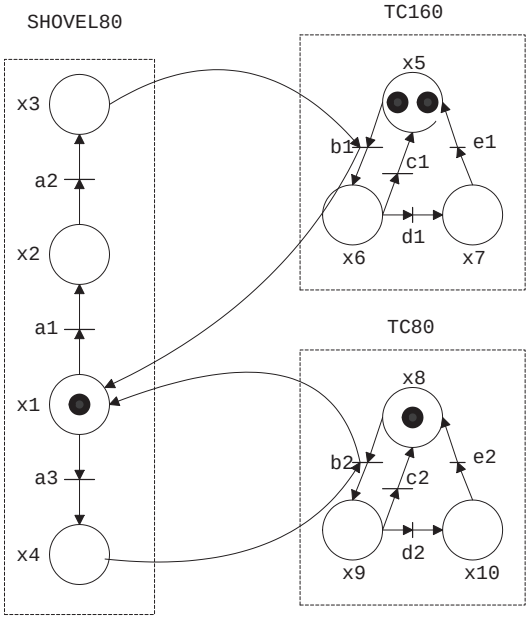

Figure 20: Petri net Representation of Case 2

Event	Meaning	Event	Meaning
a1	first bucket in one $TC160$ truck	e1	repair of one $TC160$ truck
a2	second bucket in one $TC160$ truck	b2	full loading of one $TC80$ truck
a3	first bucket in one $TC80$ truck	c2	unloading of one $TC80$ truck
b1	full loading of one $TC160$ truck	d2	breakdown of one $TC80$ truck
c1	unloading of one $TC160$ truck	e2	repair of one $TC80$ truck
d1	breakdown of one $TC160$ truck		

Table 10: Event Description for Case 2

The addition of events d_1, e_1, d_2 and e_2 alters the dynamic predicates P_1 and P_2 in the following manner

$$\begin{aligned} P_1 &:= \quad \#a_1 \leq (\#c_1 + \#e_1) + 2 \\ P_2 &:= \quad \#a_3 \leq (\#c_2 + \#e_2) + 1 \end{aligned} ,$$

where the sum of c_i and e_i represents the number of available trucks to be filled (coming back from unloading or repair).

For the specification of repair priority of 160-ton trucks over 80-ton trucks, it is easier to implement it as a VDES than as a predicate. In Figure 21, the state $x_{11} = 1$ guarantees that if events d_1 and d_2 occur, repair event e_2 cannot follow since $x_{11} = 0$ until event e_1 takes place thus giving priority of repair to $TC160$ over $TC80$. The presence of event f (assumed to take place at high speed) and state x_{12} is justified by the fact that a self-loop cannot be represented in the actual VDES framework.

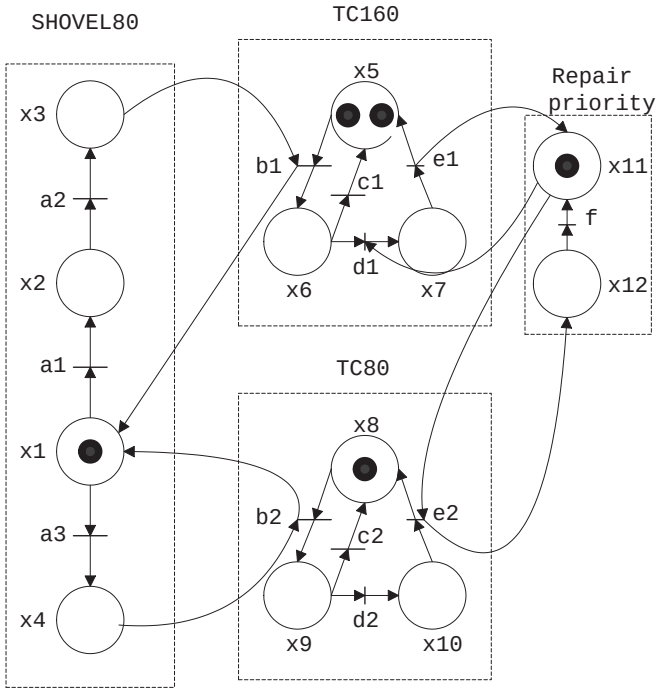

Figure 21: Specification of Case 2

As the truck dispatcher wants a maximum of one breakdown (d_1 or d_2) per four successful

unloads (c_1 or c_2), the following dynamic predicates ensure the specification is met.

$$\begin{aligned} P_3 &:= \quad -4\#d_1 + \#c_1 \geq 0 \\ P_4 &:= \quad -4\#d_2 + \#c_2 \geq 0 \end{aligned}$$

If more than one breakdown occurs for fewer than four unloadings, the above predicate is violated and the controllable loading event a_1 or a_3 will be disabled.

4.2.3 Modelling of Case 3

For this case, another shovel of capacity 40 tons, $SHOVEL40$, is added to the extraction site. As the additional shovel is of different capacity than the other (buckets of 40 and 80 tons now available), the trucks can be filled by a different number of buckets. The direct effect of this new feature is to give another loading option (two types of loading b_i or c_i) to all truck VDES representations as in Figure 22.

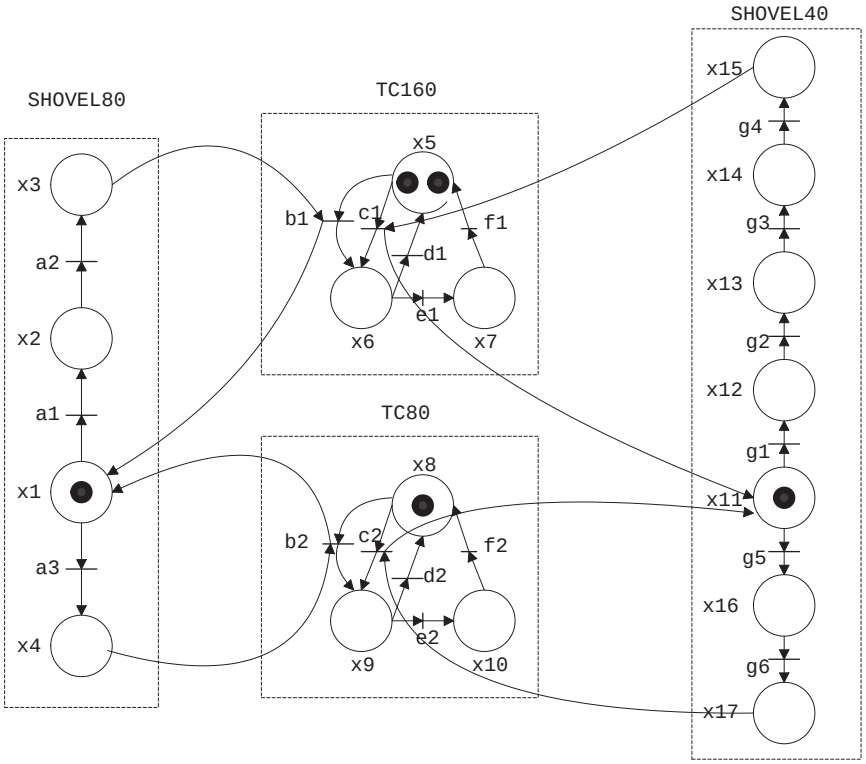

Figure 22: Petri net Representation of Case 3

One notices that in DES (Section 4.1.3) the addition of a shovel modified the buffer specification while in VDES it brings a modification to the truck VDES. In the present case, loading events from different shovels (events b_i and c_i) are distinguished while they were not in their DES equivalent. If only one truck loading event (say b_1) is linked to the loaded states of both shovels (x_3 and x_{15}), the occurrence of b_1 will require that $x_3 = x_{15} = 1$, allowing the loading of one truck by two shovels, an undesirable situation. As will be seen, the fourth case forces the distinction between loading performed by different shovels.

The predicates P_1 and P_2 preventing trucks overflow thus become

$$\begin{aligned} P_1 &:= \quad \#a_1 + \#g_1 \leq (\#d_1 + \#f_1) + 2 \\ P_2 &:= \quad \#a_3 + \#g_5 \leq (\#d_2 + \#f_2) + 1. \end{aligned}$$

The definition of events of Figure 22 is provided in Table 11. Finally, the controllable and uncontrollable events are a_1, a_2, a_3, $g_1, \ldots, g_6$, b_1, c_1, b_2, c_2, f_1, f_2 and d_1, e_1, d_2, e_2, respectively.

Event	Meaning	Event	Meaning
a1	first $SHOVEL80$ bucket in $TC160$	d2	unloading of $TC80$ truck
a2	second $SHOVEL80$ bucket in $TC160$	e2	breakdown of $TC80$ truck
a3	first $SHOVEL80$ bucket in $TC80$	f2	repair of $TC80$ truck
b1	full loading of $TC160$ by $SHOVEL80$	g1	first $SHOVEL40$ bucket in $TC160$
c1	full loading of $TC160$ by $SHOVEL40$	g2	second $SHOVEL40$ bucket in $TC160$
d1	unloading of $TC160$ truck	g3	third $SHOVEL40$ bucket in $TC160$
e1	breakdown of $TC160$ truck	g4	fourth $SHOVEL40$ bucket in $TC160$
f1	repair of $TC160$ truck	g5	first $SHOVEL40$ bucket in $TC80$
b2	full loading of $TC80$ by $SHOVEL80$	g6	second $SHOVEL40$ bucket in $TC80$
c2	full loading of $TC80$ by $SHOVEL40$		

Table 11: Event Description for Case 3

4.2.4 Modelling of Case 4

The extraction sites are now located at different distances from the crusher and thus the travelling distance varies depending on where the truck is sent to or came from. Consequently, hauling and loading events are associated with specific shovels. From now on, we assume that the truck dispatcher decides, via events c_1, c_2, h_1 and h_2, to which shovel the trucks are sent. In Figure 23 we observe that only the truck VDES are modified compared to Case 3. Here, hauling events c_1, c_2, h_1 and h_2 are needed to initiate the loading by shovels. The process events are defined in Table 12 where uncontrollable events are e_1, f_1, m_1, ,p_1, e_2, f_2, m_2 and p_2 while the others are controllable. Once again, the VDES models for the trucks match the ones developed for the DES approach in Section 4.1.4.

With this new configuration, the predicates P_1 and P_2 for the appropriate loading of trucks thus become

$$\begin{aligned} P_1 &:= \quad \#a_1 + \#b_1 \leq \#g_1 + \#n_1 + x_{12_0} \\ P_2 &:= \quad \#a_3 + \#b_5 \leq \#g_2 + \#n_2 + x_{19_0} \end{aligned},$$

where $x_{12_0} = 2$ and $x_{19_0} = 1$. This way, a more general case where the number of tokens in x_{12_0} and x_{19_0} are of different value could be considered without altering predicates P_1 and P_2. As mentioned before, a more detailed truck VDES provides an explicit relationship between the loading by a specific shovel (events d_i or k_i) and their respective hauling events (c_i or h_i). This ensures increased accuracy in the description of the process. Another specification to consider is that after three 160-ton truck breakdown events $\{f_1, p_1\}$, only the 40-ton shovel can load the 160-ton truck via event h_1. This specification is implemented as a VDES in Figure 24 in the following way. Since a self-loop cannot exist in the actual VDES framework, two extra states x_{26} and x_{27} as well as an event q (assumed to take place at a high speed and to be projected

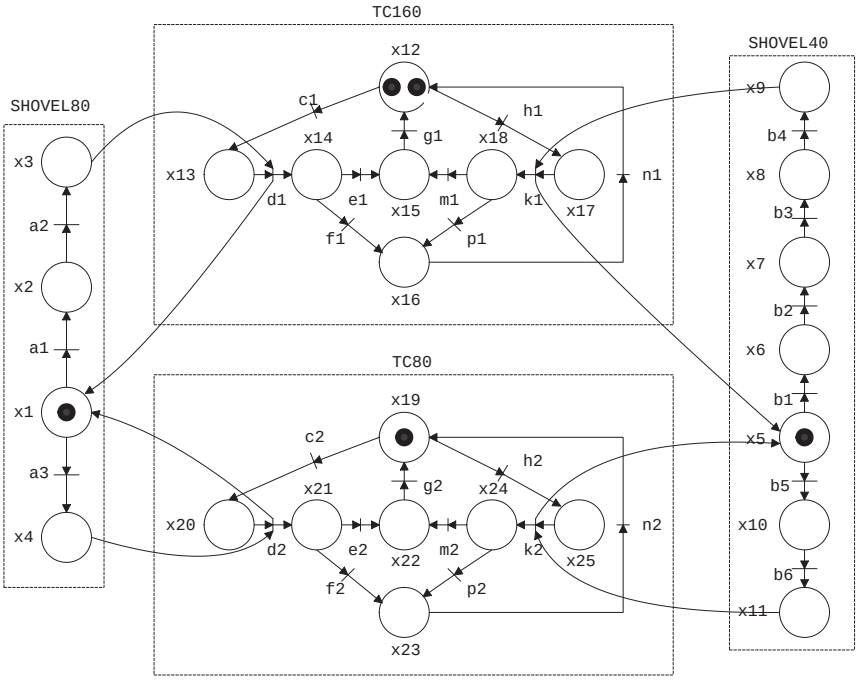

Figure 23: Petri net Representation of Case 4

Event	Meaning	Event	Meaning
a1	first $SHOVEL80$ bucket in $TC160$	h1	hauling of $TC160$ to $SHOVEL40$
a2	second $SHOVEL80$ bucket in $TC160$	k1	full loading of $TC160$ by $SHOVEL40$
a3	first $SHOVEL80$ bucket in $TC80$	m1	hauling of $TC160$ to crusher
b1	first $SHOVEL40$ bucket in $TC160$	n1	repair of $TC160$ truck
b2	second $SHOVEL40$ bucket in $TC160$	c2	hauling of $TC80$ to $SHOVEL80$
b3	third $SHOVEL40$ bucket in $TC160$	d2	full loading of $TC80$ by $SHOVEL80$
b4	fourth $SHOVEL40$ bucket in $TC160$	e2	hauling of $TC80$ to crusher
b5	first $SHOVEL40$ bucket in $TC80$	f2,p2	breakdown of $TC80$ truck
b6	second $SHOVEL40$ bucket in $TC80$	g2	unloading of $TC80$ truck
c1	hauling of $TC160$ to $SHOVEL80$	h2	hauling of $TC80$ to $SHOVEL40$
d1	full loading of $TC160$ by $SHOVEL80$	k2	full loading of $TC80$ by $SHOVEL40$
e1	hauling of $TC160$ to crusher	m2	hauling of $TC80$ to crusher
f1,p1	breakdown of $TC160$ truck	n2	repair of $TC80$ truck
g1	unloading of $TC160$ truck		

Table 12: Event Description for Case 4

out later on) are added. For every event f_1 or p_1 state x_{27} loses one token. As long as $x_{27} \geq 1$, event c_1 can take place. Consequently, after three breakdown events (or $\#f_1 + \#p_1 = 3$) event c_1 is disabled (since $x_{27} = 0$) while loading event h_1 remains possible. Thus only event h_1 can perform the loading of $TC160$ via $SHOVEL80$.

One notices that even if $TC160$ includes all trucks of 160 tons, the dispatcher can also enforce some preferred sequences of truck loadings (e.g., all $TC160$ trucks are loaded two times by $SHOVEL40$ and then three times by $SHOVEL80$).

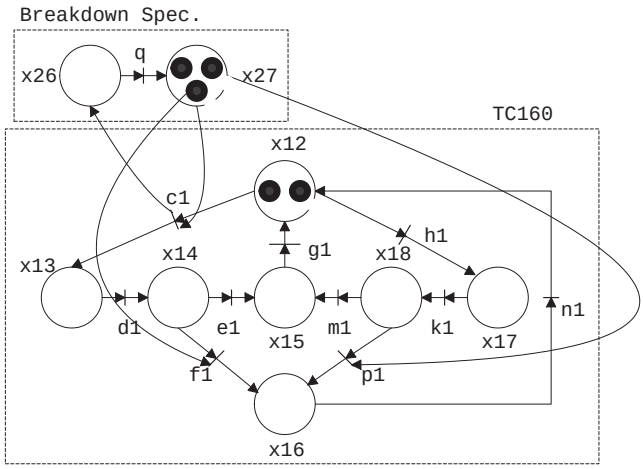

Figure 24: Specification of Case 4

4.2.5 Modelling and Supervisory Control of Case 5

For the fifth and last case, a blend specification is considered for 80-ton trucks. It consists of forcing two loadings by $SHOVEL40$ (event k_2) for each loading performed by $SHOVEL80$ (event d_2). In Figure 25, as $x_{29} = 2$ and event h_2 only displaces one token at a time (by default), event h_2 can take place twice, thus augmenting state x_{28} to two. As event c_2 decreases state x_{28} and increases state x_{29} by two units (weight of 2 on arrows entering and exiting event c_2), it cannot occur when $x_{28} = 1$. Therefore, there will be two events h_2 for each event c_2.

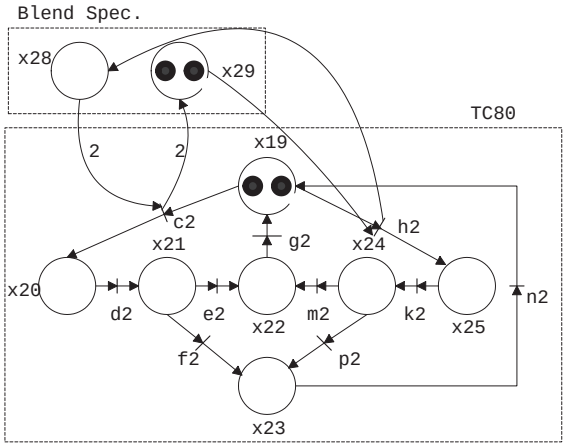

Figure 25: Specification of Case 5

The queue specification enforces the rule "first come, first loaded" at a shovel (or extraction) site and thus ensures fairness of events. In a VDES framework, all trucks of the same capacity (all 80-ton trucks or all 160-ton trucks) are modeled by a unique VDES. For the purpose of queueing, the truck capacity is the most crucial information and knowing which specific truck (for instance, the blue or the red one) is loaded becomes irrelevant. Therefore, the queue specification should instead record the occurrence of events originating from trucks of different capacities. Namely, the specification requires that tokens of the VDES representing the 80-ton

and 160-ton trucks be distinguishable. The feature of differentiating tokens is impossible in the actual VDES framework, and therefore the queue specification cannot be achieved. Instead, Colored Petri nets may be used even though there exists no relation to convert the "first come, first used" rule into a VDES. One difficulty that may arise is that for this rule to be active in the actual VDES framework it should be made universal (i.e., to differentiate the trucks at all places).

Now, the system developed in Cases 4 and 5 with the specifications listed below will be used to synthesize an optimal supervisory control.

1 Bucket capacity for each truck,
2 Only one truck loaded at a time by the shovel,
3 Priority of repair of 160-ton trucks over 80-ton trucks,
4 After three 160-ton truck breakdowns, only 40-ton shovel can unload the truck,
5 Limit of one truck breakdown for four successful load,
6 Blend specification

The Petri net representation resulting from the modelling of previous cases is shown in Figure 26. Since dynamic predicates (P_1, P_2) are analogous to (P_3, P_4), only the implementation of dynamic predicates P_2 and P_3 is shown for clarity purposes. Appendix D provides the calculations to derive the supervisory rules to implement the dynamic predicates P_1, P_2, P_3 and P_4

$$\begin{aligned} P_1 &:= \#a_1 + \#b_1 \leq \#g_1 + \#n_1 + x_{12_0} \\ P_2 &:= \#a_3 + \#b_5 \leq \#g_2 + \#n_2 + x_{19_0} \\ P_3 &:= -4(\#f_1 + \#p_1) + \#g_1 \geq 0 \\ P_4 &:= -4(\#f_2 + \#p_2) + \#g_2 \geq 0. \end{aligned}$$

The end result, as given in Appendix D, is that all specifications (with the exception of the queue specification) and predicates can be successfully implemented in a VDES framework.

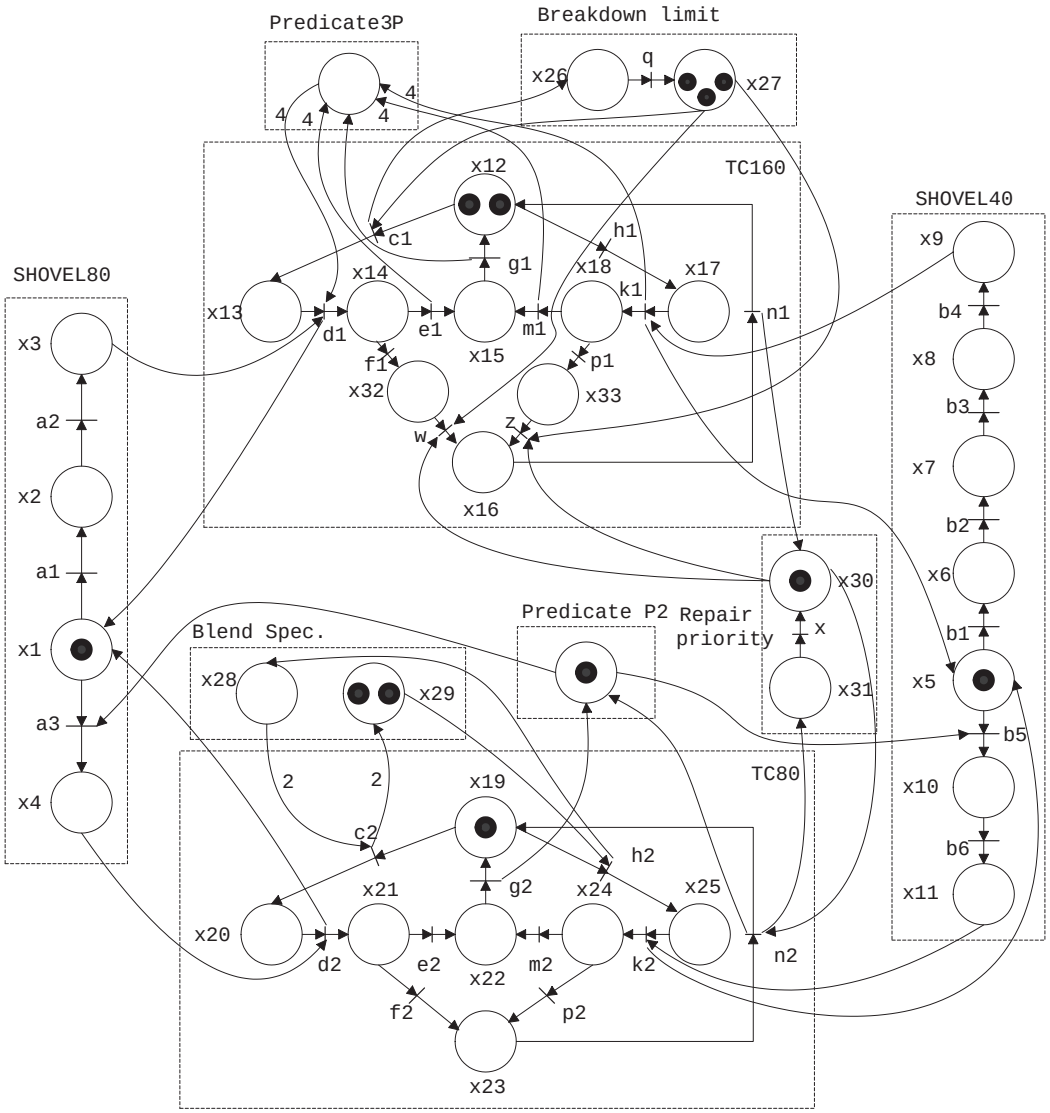

Figure 26: Production and VDES Specifications

4.3 Discussion

This section provides a discussion on various aspects of Section 4.1 and Section 4.2. The text is organized as follows. Section 4.3.1 analyzes the objectives that could be achieved using DES and VDES theories in light of the original process. Then, Section 4.3.2 and Section 4.3.3 give technical comments on the DES and VDES approaches, respectively.

4.3.1 Scope

The following paragraphs refer to additional specifications than the ones found in Table 2. This section aims at providing an analysis of obstructions or difficulties fulfilling these specifications in the DES and VDES approaches.

Consider the specification of having two trucks unloading at the same time. A possible TDES strategy would be to use a small enough time unit such that two unloadings are a tick apart (given two trucks are present at the crusher). This is realistic since real simultaneous events are rare if the time unit is chosen appropriately (small enough compared to the time taken for unloading). In a VDES approach, simultaneous events are theoretically allowed but the main theorems are not designed to consider such a scenario.

Another specification of interest is that a truck may be partially broken down but still operational with a reduced capacity (payload or speed). In a DES framework, such a specification can be embedded into the buffers by the insertion of an additional transition leading to the end of a cycle before the truck is full. For instance, for the DES $B1$ of Case 4 (Figure 14) one can add the controllable event 49 that stops the loading of a second bucket (Figure 27).

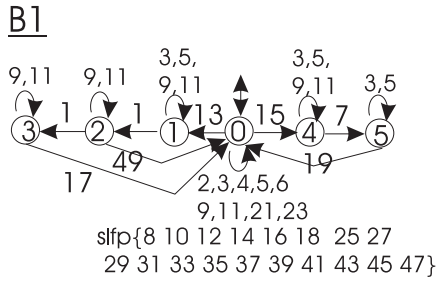

Figure 27: Reduced Load Buffer for $T1C80$

4.3.2 Comments on the DES Approach

The main drawback of the DES approach is that the increasing size of the overall system may render the implementation of centralized supervisors prohibitive. When possible, the strategy of this research has been to develop modular supervisors to reduce the effective size (and complexity) of the overall system. As specifications were designed to act as optimal modular supervisors, the proposed approach has been to increment progressively the number of trucks. However, the optimal nature of the modular supervisor must be verified after each new addition of components (stable intermediate structure).

As seen in Section 4.1, the addition of timed events gives a representation that is closer to reality. However, the assignment of time bounds requires deep understanding of the process and it must be performed carefully with respect to the overall process, as seen in Section 4.1.1 and Section 4.1.2. A more critical issue remains the proper selection of time units such that the dominant dynamics are captured without penalizing the computational cost of having a detailed representation of the process. Also, any untimed specification should be worked out first in an untimed framework before converting it to TDES. Finally, the major drawback of using timed DES is the explosion in the number of discrete states.

In hierarchical control, there is a trade-off between the level of refinements the low-level requires so that the high-level supervisor can take adequate actions.

4.3.3 Comments on the VDES Approach

There are stringent conditions under which a VDES implementation (or VDESI) can be derived. The main conditions are as follows: i) the uncontrollable subsystem G_u must be loop-free, ii) the uncontrollable events involved in predicates cannot be preceded by more than one state. For our special application (see Appendix D), the first condition is naturally satisfied by the large number of controllable events while for the second condition, an easy fix has been found. Since such a fix may not always be possible, there is no guarantee that a VDES implementation of an optimal supervisor can be derived.

Another limitation is the modelling of queues that does not seem to be easily achievable within the actual VDES framework. If machines within a VDES need to be differentiated, a formalism similar to the one in Colored Petri nets should be established.

Since self-loops cannot be represented in the actual VDES framework, the final language of a VDES is obtained by projecting out the dummy events that have been added to circumvent the need for self-loops (for example, events x and y in Figure 26).

5 Conclusions

This last section summarizes the capabilities of DES and VDES theories to solve a decision problem when applied to an oilsand extraction process. In Section 5.1, we perform a comparison of the advantages and inconveniences of the DES and VDES approaches. Section 5.2 concludes by providing possible direction for further research work that would benefit both the oilsand extraction application and DES theory.

5.1 DES versus VDES

Unlike other theories, VDES and DES theories answer the question *"With a given setup, how can the equipment be operated in an optimal way to satisfy the specifications?"* and not *"What is the optimal setup to fulfill the specifications?"* This is particularly well-suited for a truck dispatching task where daily conditions (not under control) affect the production setup (i.e., the number of trucks, ore location, ore content, and number of shovels available for extraction).

All specifications provided by Syncrude were successfully tackled or at least a scheme of solution was provided (Section 4.3.1). Moreover, additional and fictitious specifications were considered to show the DES/VDES theory and the capabilities of the associated tools. With the DES approach, the specifications have been solved sequentially and often in an isolated manner. This approach was intentional since even with a reduced level of complexity, the solution to a DES problem often exceeds the capability of standard representation techniques (in this case, a direct translation into state charts[1] would be beneficial). Unlike DES, VDES possesses a compact representation when many machines exhibit the same behaviour. This explains why in Section 4.2.5 all specifications could be considered at the same time without omitting information about the overall behaviour.

In some cases, VDES theory offers fewer options than DES to implement a specification. In DES, a specification can always be embedded in the plant DES or specified as a separate DES with no increase in complexity. As in Section 4.2.1, it appears that sometimes VDES theory must inevitably embed certain specifications to preserve some dynamics of interest. Fortunately, after performing the appropriate changes, the overall VDES behaviour corresponds exactly to the DES behavior. Even though VDES is dual to DES[2], it appears that not all untimed DES features can be captured when expressed in a VDES framework. In this respect, the limitations of VDES theory in implementing the queue specification remain considerable in the actual industrial context.

Obviously, modular supervisory control remains an interesting manner to implement DES supervisors at a reduced cost. The DES theory also offers hierarchical supervisory control that can provide a truck dispatcher with a simplified picture of the whole production, thus facilitating the decision-making task. On the other hand, vector DES offers an equivalence to modular supervisory control by allowing the conjunction of many predicates. To our knowledge, there is no hierarchical technique in VDES but since the representation is more compact, a manager has the ability to focus on a small portion of the process while preserving a complete picture of the overall behaviour.

[1] A state chart is a graphical formalism used to describe complex system languages (object of infinite length) by providing an equivalent economical (or finite dimension) representation in a clear and realistic format [5].

[2] In VDES a behaviour is characterized by the evolution of its states rather than by the sequence of events.

Since Syncrude's production is mostly composed of identical machinery, once two or three pieces of equipment are successfully introduced into the process the procedure developed in a DES framework can be easily extended to a larger fleet. As mentioned earlier, the main benefit of using VDES is the compactness of its representation. As seen in Section 4.2, with a VDES approach the complete fleet of 30 trucks can be more easily represented than in DES. For example, in Figure 26, the insertion of 30 trucks (tokens) amounts to setting x_{12_0} and x_{19_0} such that $x_{12_0} + x_{19_0} = 30$ (if trucks are only allowed to start from the Idle position). Therefore, the whole truck fleet is modeled without enlarging the dimension of the system. Moreover, the supervisor built for a smaller fleet remains adequate.

5.2 Future Work

One of the crucial tasks remains the translation of (qualitative and/or quantitative) objectives into suitable DES or VDES equivalences. Even though this was accomplished easily in the present case, there exists no systematic procedure to convert an objective into a language. Any breakthrough in this direction would significantly benefit the use of DES and VDES as a solution scheme for decision problems. While performing a conversion of a system's behaviour to a language, important considerations are the determination (or "identification") for each event of whether it is controllable or uncontrollable and the value of its time bounds. As seen in Section 4.1.1, there must be consistency between the untimed and the timed DES representation of a process.

Throughout this work, a progressive and incremental approach has been shown to be fruitful in the development of DES/VDES supervisory controls. We believe that this is an appropriate strategy for any future work that will be performed with an application such as the oilsand extraction.

In reference to Table 2, some further extensions could be brought to the fifth case. The next logical steps to take for the continuation of the work, so that the overall model is closer to the original process, could be summarized by four major additions (Table 13). This amounts to the addition of more trucks as well as queues that are brought in wherever a truck can go (i.e., at the shovel location and the crusher). Finally, from case 9 the enlargement to thirty trucks and three locations will only be a matter of time since the main issues will have already been addressed.

	Shovels	Trucks	Locations	Breakdowns	Queues
Case 6	2	3	2	Yes	2
Case 7	2	3	2	Yes	3
Case 8	2	4	2	Yes	3
Case 9	2	5	2	Yes	3

Table 13: Possible Extensions to Mimic the Overall Production

Since VDES is simpler in its representation and calculation, we first believed that - prior to using DES - a VDES approach could be used as a first and rapid solution for synthesizing an optimal supervisory control. However, if the VDES approach does not yield an optimal supervisor, it does not automatically follow that DES is incapable of solving the problem in an optimal fashion (the queue specification is an example). Also, if VDES has a solution it may be blocking (from the theory) and therefore a nonblocking optimal DES supervisor may not

exist. The last issue should be investigated more deeply with a simple system. From there, two possible research directions are possible. One could refine the VDES theory and provide stronger conditions that guarantee a nonblocking VDES supervisor, or one could determine additional conditions that ensure the existence of a nonblocking optimal DES supervisor when VDES theory provides a solution. If there exist no such conditions, VDES will remain a less attractive tool than DES from an application point of view. Also, VDES theory should be enhanced so that queue specifications can be handled.

This study has revealed that the DES theory is a more suitable framework than VDES for the decision problem of the discrete part of the oilsand extraction process. The next step will be to incorporate the continuous dynamics and see how DES theory can be used for the overall decision problem.

A DES Theoretical Background

The following sections present an informal introduction to DES theory and its objectives. It is by no means an exhaustive picture of the capability of the whole theory but it contains the necessary information to facilitate the reading and the understanding of the present study. For greater details, the reader is referred to [8] and [9].

A.1 Discrete-event systems

Discrete-event systems are systems driven by the occurrence of events bringing the system from one state to another. Events are assumed to be instantaneous (i.e., without time reference) and can be of two types: *controllable* or *uncontrollable*. Uncontrollable events are events that cannot be stopped while controllable events can be disabled by some "control" mechanism. In a DES, the set of all possible events forms the *alphabet*. The behavior of a DES is characterized by the sequences of events, or *language*, that it is capable of generating. Similarly, the control rules (desired sequences of events), called here *specifications*, can be translated into a finite-state machine representation. For this reason, we speak of a DES and its respective language interchangeably.

For the control of DES, there will be two languages of interest: one for the *plant* representing the system to control, and one for the specifications describing some desired behavior. For a system composed of subsystems (i.e., here the shovels and trucks), the plant DES is defined as the Cartesian product of its subsystems DES. A *supervisor* (terminology for a DES controller) for a plant DES is itself a DES that enables and disables the events such that the plant meets the specifications. Thus some events are shared between the plant and the supervisor. The techniques discussed in the following section can be used to synthesize different types of supervisors based on supervision needs (centralized, decentralized (modular) or hierarchical (multilayers)). The DES theory guarantees that all synthesized supervisors are optimal in the sense that the maximum number of events is always enabled (i.e., it provides the least restrictive supervisor).

Discrete-event systems are represented by transition diagrams (directed graphs) of finite dimension where events are described by arrows and states by circles. In all the DESs shown in this report, states are labelled by numbers, starting with the initial state identified by zero. As only events can be shared, common states labels between two or more DES bear no particular meaning. In a transition graph, controllable and uncontrollable events are distinguished by odd and even values, respectively. An exiting arrow attached to a state means that the state is of particular interest (called a *marked state*).

An example of a DES is provided in Figure 28 showing the DES for a shovel (named $SHOVEL80$) and a truck of 80 tons of capacity (named $T1C80$). The shovel DES has only one state (no breakdown) and continuously loads trucks (loading state 0). Possible events are 1, 3 and 5 (all controllable) and they represent the loading of trucks: 1 stands for the loading of the 80-ton truck $T1C80$, 3 for one 160-ton truck $T1C160$ and 5 for the other 160-ton truck $T2C160$. The loadings are assumed to be controllable since any loading can always be stopped from occurring. The truck DES has two states representing its empty or full mode (respectively, state 0 and state 1). Events 7 and 4 depict a full loading and an unloading. The loading event is defined to be controllable as it is always possible to find a mechanism to stop the loading of

a truck. The unloading event is assumed to be uncontrollable since an unloading is successful only in the absence of breakdowns (uncontrollable by nature). A DES can be made as precise as required and the approach will demonstrate the effect of more precise system descriptions and specifications.

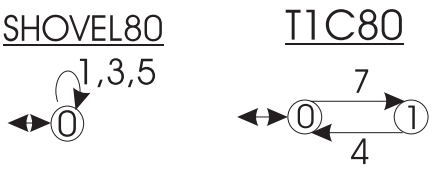

Figure 28: DES Representation of 80 Ton Shovel and Truck

A.2 Supervisors for DES

Different types of supervisors can be synthesized for DESs. The most common and easily computed one is the centralized supervisor, which consists of a global supervisor that handles all specifications. The main disadvantage is that its size is often so large that one loses sight of the original features. This often renders the implementation of a centralized supervisor prohibitive.

An alternative to the centralized supervisor is the decentralized (or modular) supervisor. It performs the same task as its centralized counterpart but has as many supervisory modules as local specifications. Thus it is easier to implement and the supervisory objective is more apparent from the modular structure. This type of supervisor is preferred in the present context since the real production (30 trucks and 3 shovels) leads to a rather large system that would benefit from modular supervisors. Figure 29 shows that a centralized supervisor merges all local specifications (Sp.1 to Sp. N) into a global one to develop its supervisory language. Each modular supervisor (MS 1 to MS N) satisfies a local specification and an event is enabled when it is enabled by all modular supervisors.

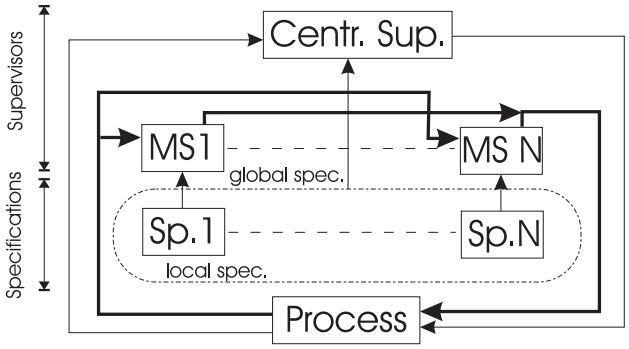

Figure 29: Centralised and Decentralised Supervisors

The third type of supervisor of interest is the hierarchical supervisor. It consists of a higher level supervisor designed for management purposes. It only observes some of the process events that are of specific interest. These events are said to be *vocalized* or viewed by the high-level supervisor. As a result, the hierarchical supervisor has a reduced version of the process but can still impose its specifications at the higher level, while its corresponding low-level supervisor

translates and implements these specifications. This type of supervisor is a key element of DES theory. It serves as a decision-making process for the system that looks at relevant information only. Centralized and decentralized supervisors can be synthesized in the high-level language.

A.3 Timed DES

Timed DES introduces the notion of time by associating time bounds (lower and upper) to all events of a DES. The lower bound represents the minimum time required to elapse before the event takes place. The upper time bound is the maximum time before an event must occur. The time measure is the *tick* of a global clock under which all events are synchronized. That way, when a tick occurs, events that are *eligible* (events whose occurrence is possible) see their time counter augmented by one unit. By definition, controllable events have an infinite upper time bound since disabling an event is analogous to giving the event an eternity to occur. Moreover, an event can be set to beat the clock by preempting a tick from occurring and preempting other events. Such events are said to be *forcible*.

Consider the truck DES $T1C80$ (Figure 28) and define the time bounds for event 4 as [1 2] and for event 7 as [1 ∞] where the values in brackets represent, respectively, the lower and upper time bounds. As seen in Figure 30, event 7 can only occur after a tick elapsed and before an infinite number of ticks. Similarly, event 4 cannot take place before at least one tick elapses and it must occur before the third one. One easily sees that the introduction of time has a significant impact on the size of the state representation of a DES (from 2 to 9 states) [10].

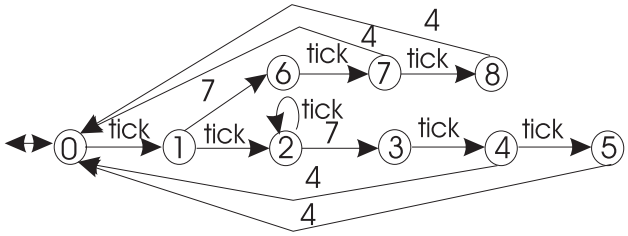

Figure 30: TDES for $SHOVEL80$

The size of the resulting TDES is related to the choice of time unit size (seconds, fraction of minutes, minutes, etc.). The finer the time unit the larger the TDES becomes, allowing more possibilities for supervisory control. However, a smaller time unit increases the computational cost of determining an appropriate supervisor, which may become a limiting factor. Conversely, if the time unit is too large the resulting model may fail to accurately represent the original process. This can lead to a reduction of supervisory capabilities. These issues are illustrated in [10]. Moreover, the reader can refer to Brandin and Wonham [2] for additional details on TDES.

B Printouts for DES Supervisory Controls

B.1 Printouts for Case 1

********** UNTIMED OPERATIONS **********

SYST1 # states: 8 state set: 0 ... 7 initial state: 0

marker states: 0

vocal states: none

transitions: 48

transitions:

[0, 1, 0] [0, 3, 0] [0, 5, 0] [0, 7, 1] [0, 9, 2] [0, 11, 3] [1, 1, 1] [1, 3, 1] [1, 4, 0] [1, 5, 1] [1, 9, 4] [1, 11, 5] [2, 1, 2] [2, 3, 2] [2, 5, 2] [2, 6, 0] [2, 7, 4] [2, 11, 6] [3, 1, 3] [3, 3, 3] [3, 5, 3] [3, 7, 5] [3, 8, 0] [3, 9, 6] [4, 1, 4] [4, 3, 4] [4, 4, 2] [4, 5, 4] [4, 6, 1] [4, 11, 7] [5, 1, 5] [5, 3, 5] [5, 4, 3] [5, 5, 5] [5, 8, 1] [5, 9, 7] [6, 1, 6] [6, 3, 6] [6, 5, 6] [6, 6, 3] [6, 7, 7] [6, 8, 2] [7, 1, 7] [7, 3, 7] [7, 4, 6] [7, 5, 7] [7, 6, 5] [7, 8, 4]

****

SPEC1 # states: 6 state set: 0 ... 5 initial state: 0

marker states: 0

vocal states: none

transitions: 21

transitions:

[0, 1, 1] [0, 3, 2] [0, 4, 0] [0, 5, 3] [0, 6, 0] [0, 8, 0] [1, 6, 1] [1, 7, 0] [1, 8, 1] [2, 3, 4] [2, 4, 2] [2, 8, 2] [3, 4, 3] [3, 5, 5] [3, 6, 3] [4, 4, 4] [4, 8, 4] [4, 9, 0] [5, 4, 5] [5, 6, 5] [5, 11, 0]

****

SUPER1 # states: 28 state set: 0 ... 27 initial state: 0

marker states: 0

vocal states: none

transitions: 64

transitions:

[0, 1, 1] [0, 3, 2] [0, 5, 3] [1, 7, 4] [2, 3, 5] [3, 5, 6] [4, 3, 7] [4, 4, 0] [4, 5, 8] [5, 9, 9] [6, 11, 10] [7, 3, 11] [7, 4, 2] [8, 4, 3] [8, 5, 12] [9, 1, 13] [9, 5, 14] [9, 6, 0] [10, 1, 15] [10, 3, 16] [10, 8, 0] [11, 4, 5] [11, 9, 17] [12, 4, 6] [12, 11, 18] [13, 6, 1] [13, 7, 17] [14, 5, 19] [14, 6, 3] [15, 7, 18] [15, 8, 1] [16, 3, 20] [16, 8, 2] [17, 4, 9] [17, 5, 21] [17, 6, 4] [18, 3, 22] [18, 4, 10] [18, 8, 4] [19, 6, 6] [19, 11, 23] [20, 8, 5] [20, 9, 23] [21, 4, 14] [21, 5, 24] [21, 6, 8] [22, 3, 25] [22, 4, 16] [22, 8, 7] [23, 1, 26] [23, 6, 10] [23, 8, 9] [24, 4, 19] [24, 6, 12] [24, 11, 27] [25, 4, 20] [25, 8, 11] [25, 9, 27] [26, 6, 15] [26, 7, 27] [26, 8, 13] [27, 4, 23] [27, 6, 18] [27, 8, 17]

****

MSUPER14 # states: 28 state set: 0 ... 27 initial state: 0

marker states: 0

vocal states: none

transitions: 64

transitions:

[0, 1, 1] [0, 3, 2] [0, 5, 3] [1, 7, 4] [2, 3, 5] [3, 5, 6] [4, 3, 7] [4, 4, 0] [4, 5, 8] [5, 9, 9] [6, 11, 10] [7, 3, 11] [7, 4, 2] [8, 4, 3] [8, 5, 12] [9, 1, 13] [9, 5, 14] [9, 6, 0] [10, 1, 15] [10, 3, 16] [10, 8, 0] [11, 4, 5] [11, 9, 17] [12, 4, 6] [12, 11, 18] [13, 6, 1] [13, 7, 17] [14, 5, 19] [14, 6, 3] [15, 7, 18] [15, 8, 1] [16, 3, 20] [16, 8, 2] [17, 4, 9] [17, 5, 21] [17, 6, 4] [18, 3, 22] [18, 4, 10] [18, 8, 4] [19, 6, 6] [19, 11, 23] [20, 8, 5] [20, 9, 23] [21, 4, 14] [21, 5, 24] [21, 6, 8] [22, 3, 25] [22, 4, 16] [22, 8, 7] [23, 1, 26] [23, 6, 10] [23, 8, 9] [24, 4, 19] [24, 6, 12] [24, 11, 27] [25, 4, 20] [25, 8, 11] [25, 9, 27] [26, 6, 15] [26, 7, 27] [26, 8, 13] [27, 4, 23] [27, 6, 18] [27, 8, 17]

****

SHOVEL80 = Create(SHOVEL80,[mark 0],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)

T1C80 = Create(T1C80,[mark 0],[tran [0,7,1],[1,4,0]]) (2,2)

T1C160 = Create(T1C160,[mark 0],[tran [0,9,1],[1,6,0]]) (2,2)

T2C160 = Create(T2C160,[mark 0],[tran [0,11,1],[1,8,0]]) (2,2)

B1 = Create(B1,[mark 0],[tran [0,1,1],[1,7,0]]) (2,2)

B1 = Selfloop(B1,[6,8,9,11]) (2,10) Computing time = 00:00:00.00

B1 = Edit(B1,[trans +[0,3,0],+[0,4,0],+[0,5,0]]) (2,13)

B2 = Create(B2,[mark 0],[tran [0,1,0],[0,3,1],[0,5,0],[0,6,0],[1,3,2],[2,9 ,0]]) (3,6)

B2 = Selfloop(B2,[4,7,8,11]) (3,18) Computing time = 00:00:00.00

B3 = Create(B3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,1],[0,8,0],[1,5,2],[2,1 1,0]]) (3,6)

B3 = Selfloop(B3,[4,6,7,9]) (3,18) Computing time = 00:00:00.00

MEETB1B2 = Meet(B1,B2) (4,19) Computing time = 00:00:00.00

SPEC1 = Meet(MEETB1B2,B3) (6,21) Computing time = 00:00:00.00

TC160 = Sync(T1C160,T2C160) (4,8) Computing time = 00:00:00.00

TRUCKS = Sync(TC160,T1C80) (8,24) Computing time = 00:00:00.06

SYST1 = Sync(TRUCKS,SHOVEL80) (8,48) Computing time = 00:00:00.00

SUPER1 = Supcon(SYST1,SPEC1) (28,64) Computing time = 00:00:00.00

SUPER1 = Condat(SYST1,SUPER1) Controllable. Computing time = 00:00:00.00

false = Nonconflict(B1,SYST1) Computing time = 00:00:00.00

DATB1 = Condat(SYST1,B1) Uncontrollable. Computing time = 00:00:00.00

MB14 = Create(MB14,[mark 0],[tran [0,1,1],[0,3,0],[0,5,0],[1,7,2],[2,3,2], [2,4,0],[2,5,2]]) (3,7)

MB14 = Selfloop(MB14,[6,8,9,11]) (3,19) Computing time = 00:00:00.06

true = Nonconflict(MB14,SYST1) Computing time = 00:00:00.00

DATMB14 = Condat(SYST1,MB14) Controllable. Computing time = 00:00:00.00

MB24 = Create(MB24,[mark 0],[tran [0,1,0],[0,3,1],[0,5,0],[1,3,2],[2,9,3], [3,1,3],[3,5,3],[3,6,0]]) (4,8)

MB24 = Selfloop(MB24,[4,7,8,11]) (4,24) Computing time = 00:00:00.00

true = Nonconflict(MB24,SYST1) Computing time = 00:00:00.00

DATMB24 = Condat(SYST1,MB24) Controllable. Computing time = 00:00:00.00

MB34 = Create(MB34,[mark 0],[tran [0,1,0],[0,3,0],[0,5,1],[1,5,2],[2,11,3] ,[3,1,3],[3,3,3],[3,8,0]]) (4,8)

MB34 = Selfloop(MB34,[4,6,7,9]) (4,24) Computing time = 00:00:00.00

true = Nonconflict(MB34,SYST1) Computing time = 00:00:00.00

DATMB34 = Condat(SYST1,MB34) Controllable. Computing time = 00:00:00.00

true = Nonconflict(MB14,MB24) Computing time = 00:00:00.00

true = Nonconflict(MB14,MB34) Computing time = 00:00:00.00

true = Nonconflict(MB24,MB34) Computing time = 00:00:00.00

MB14MB24 = Meet(MB14,MB24) (10,41) Computing time = 00:00:00.00

MSUPER14 = Meet(MB14MB24,MB34) (28,64) Computing time = 00:00:00.00

true = Nonconflict(MSUPER14,SYST1) Computing time = 00:00:00.00

TMSUP14 = Trim(MSUPER14) (28,64) Computing time = 00:00:00.00

true = Isomorph(MSUPER14,TMSUP14;identity) Computing time = 00:00:00.00

TEST14 = Meet(MSUPER14,SYST1) (28,64) Computing time = 00:00:00.00

true = Isomorph(SUPER1,TEST14;identity) Computing time = 00:00:00.06

********** TIMED OPERATIONS **********

shovel80 = ACreate(shovel80,[mark 0],[timebounds [1,0,1000],[3,0,1000],[5, 0,1000]],[forcible 1,],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)

t1c80 = ACreate(t1c80,[mark 0],[timebounds [4,1,2],[7,1,1000]],[forcible 7],[tran [0,7,1],[1,4,0]]) (2,2)

t1c160 = ACreate(t1c160,[mark 0],[timebounds [6,1,2],[9,1,1000]],[forcible 9],[tran [0,9,1],[1,6,0]]) (2,2)

t2c160 = ACreate(t2c160,[mark 0],[timebounds [8,1,2],[11,1,1000]],[forcibl e 11],[tran [0,11,1],[1,8,0]]) (2,2)

tc160 = Comp(t1c160,t2c160) (4,8) Computing time = 0 sec

trucks = Comp(tc160,t1c80) (8,24) Computing time = 0 sec

system1 = Comp(trucks,shovel80) (8,48) Computing time = 0 sec

system1 = TimedGraph(system1) (125,664) Computing time = 0 sec

b1 = Create(b1,[mark 0],[tran [0,0,0],[0,1,1],[0,3,0],[0,4,0],[0,5,0],[1,0 ,1],[1,7,0]],[forcible 1,3,5,7]) (2,7)

b1 = Selfloop(b1,[6,8,9,11],[new forcible 9,11]) (2,15) Computing time = 1 sec

b2 = Create(b2,[mark 0],[tran [0,0,0],[0,1,0],[0,3,1],[0,5,0],[0,6,0],[1,0 ,1],[1,3,2],[2,0,2],[2,9,0]],[forcible 1,3,5,9]) (3,9)

b2 = Selfloop(b2,[4,7,8,11],[new forcible 7,11]) (3,21) Computing time = 0 sec

b3 = Create(b3,[mark 0],[tran [0,0,0],[0,1,0],[0,3,0],[0,5,1],[0,8,0],[1,0 ,1],[1,5,2],[2,0,2],[2,11,0]],[forcible 1,3,5,11]) (3,9)

b3 = Selfloop(b3,[4,6,7,9],[new forcible 7,9]) (3,21) Computing time = 0 sec

b1b2 = Meet(b1,b2) (4,23) Computing time = 0 sec

spec1 = Meet(b1b2,b3) (6,27) Computing time = 0 sec

super1 = Supcon(system1,spec1) (375,899) Computing time = 0 sec

super1 = Condat(system1,super1) Computing time = 0 sec

B.2 Printouts for Case 2

********** UNTIMED OPERATIONS **********

PSYST2 # states: 27 state set: 0 ... 26 initial state: 0

Projection of SYST without the selfoop at each state with events 1,3 and 5

marker states: 0

vocal states: none

transitions: 108

transitions:

[0, 7, 1] [0, 11, 2] [0, 15, 3] [1, 2, 0] [1, 4, 4] [1, 11, 5] [1, 15, 6] [2, 6, 0] [2, 7, 5] [2, 8, 7] [2, 15, 8] [3, 7, 6] [3, 10, 0] [3, 11, 8] [3, 12, 9] [4, 9, 0] [4, 11, 10] [4, 15, 11] [5, 2, 2] [5, 4, 10] [5, 6, 1] [5, 8, 12] [5, 15, 13] [6, 2, 3] [6, 4, 11] [6, 10, 1] [6, 11, 13] [6, 12, 14] [7, 7, 12] [7, 13, 0] [7, 15, 15] [8, 6, 3] [8, 7, 13] [8, 8, 15] [8, 10, 2] [8, 12, 16] [9, 7, 14] [9, 11, 16] [9, 17, 0] [10, 6, 4] [10, 8, 17] [10, 9, 2] [10, 15, 18] [11, 9, 3] [11, 10, 4] [11, 11, 18] [11, 12, 19] [12, 2, 7] [12, 4, 17] [12, 13, 1] [12, 15, 20] [13, 2, 8] [13, 4, 18] [13, 6, 6] [13, 8, 20] [13, 10, 5] [13, 12, 21] [14, 2, 9] [14, 4, 19] [14, 11, 21] [14, 17, 1] [15, 7, 20] [15, 10, 7] [15, 12, 22] [15, 13, 3] [16, 6, 9] [16, 7, 21] [16, 8, 22] [16, 17, 2] [17, 9, 7] [17, 13, 4] [17, 15, 23] [18, 6, 11] [18, 8, 23] [18, 9, 8] [18, 10, 10] [18, 12, 24] [19, 9, 9] [19, 11, 24] [19, 17, 4] [20, 2, 15] [20, 4, 23] [20, 10, 12] [20, 12, 25] [20, 13, 6] [21, 2, 16] [21, 4, 24] [21, 6, 14] [21, 8, 25] [21, 17, 5] [22, 7, 25] [22, 13, 9] [22, 17, 7] [23, 9, 15] [23, 10, 17] [23, 12, 26] [23, 13, 11] [24, 6, 19] [24, 8, 26] [24, 9, 16] [24, 17, 10] [25, 2, 22] [25, 4, 26] [25, 13, 14] [25, 17, 12] [26, 9, 22] [26, 13, 19] [26, 17, 17]

****

GHI # states: 7 state set: 0 ... 6 initial state: 0

marker states: 0 1 2 3 4 5 6

vocal states: none

transitions: 17

transitions:

[0,111, 1] [1,100, 2] [1,110, 3] [1,111, 1] [2,100, 4] [2,101, 2] [2,111, 1] [3,100, 4] [3,110, 5] [3,111, 1] [4,100, 6] [4,101, 2] [4,111, 1] [5,100, 6] [5,111, 1] [6,101, 4] [6,111, 1]

****

SHOVEL80 = Create(SHOVEL80,[mark 0],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)

T1C80 = Create(T1C80,[mark 0],[tran [0,7,1],[1,2,0],[1,4,2],[2,9,0]]) (3, 4)

T1C160 = Create(T1C160,[mark 0],[tran [0,11,1],[1,6,0],[1,8,2],[2,13,0]]) (3,4)

T2C160 = Create(T2C160,[mark 0],[tran [0,15,1],[1,10,0],[1,12,2],[2,17,0]]) (3,4)

B1 = Create(B1,[mark 0],[tran [0,1,1],[0,2,0],[0,3,0],[0,4,0],[0,5,0],[0,9 ,0],[1,7,0]]) (2,7)

B1 = Selfloop(B1,[6,8,10,11,12,13,15,17]) (2,23) Computing time = 00:00:00.00

B2 = Create(B2,[mark 0],[tran [0,1,0],[0,3,1],[0,5,0],[0,6,0],[0,8,0],[0,1 3,0],[1,3,2],[2,11,0]]) (3,8)

B2 = Selfloop(B2,[2,4,7,9,10,12,15,17]) (3,32) Computing time = 00:00:00.00

B3 = Create(B3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,1],[0,10,0],[0,12,0],[0 ,17,0],[1,5,2],[2,15,0]]) (3,8)

B3 = Selfloop(B3,[2,4,6,7,8,9,11,13]) (3,32) Computing time = 00:00:00.00

TC160 = Sync(T1C160,T2C160) (9,24) Computing time = 00:00:00.00

TRUCKS = Sync(TC160,T1C80) (27,108) Computing time = 00:00:00.00

SYST2 = Sync(TRUCKS,SHOVEL80) (27,189) Computing time = 00:00:00.05

MEETB1B2 = Meet(B1,B2) (4,37) Computing time = 00:00:00.00

SPEC2 = Meet(MEETB1B2,B3) (6,47) Computing time = 00:00:00.00

SUPER2 = Supcon(SYST2,SPEC2) (72,243) Computing time = 00:00:00.00

SUPER2 = Condat(SYST2,SUPER2) Controllable. Computing time = 00:00:00.00

TSYST2 = Trim(SYST2) (27,189) Computing time = 00:00:00.00

true = Isomorph(SYST2,TSYST2;identity) Computing time = 00:00:00.00

MB1 = Create(MB1,[mark 0],[tran [0,1,1],[0,3,0],[0,5,0],[1,7,2],[2,2,0],[2 ,3,2],[2,4,2],[2,5,2],[2,9,0]]) (3,9)

MB1 = Selfloop(MB1,[6,8,10,11,12,13,15,17]) (3,33) Computing time = 00:00:00.00

true = Nonconflict(MB1,SYST2) Computing time = 00:00:00.00

MB3 = Create(MB3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,1],[1,5,2],[2,15,3],[3,1,3],[3,3,3],[3,10,0]

,[3,12,3],[3,17,0]]) (4,10)

MB3 = Selfloop(MB3,[2,4,6,7,8,9,11,13]) (4,42) Computing time = 00:00:00.00

MB2 = Create(MB2,[mark 0],[tran [0,1,0],[0,3,1],[0,5,0],[1,3,2],[2,11,3],[3,1,3],[3,5,3],[3,6,0],

[3,8,3],[3,13,0]]) (4,10)

MB2 = Selfloop(MB2,[2,4,7,9,10,12,15,17]) (4,42) Computing time = 00:00:00.06

true = Nonconflict(MB1,SYST2) Computing time = 00:00:00.00

true = Nonconflict(MB2,SYST2) Computing time = 00:00:00.05

true = Nonconflict(MB3,SYST2) Computing time = 00:00:00.00

MB1MB2 = Meet(MB1,MB2) (10,75) Computing time = 00:00:00.00

MSUPER2 = Meet(MB1MB2,MB3) (28,128) Computing time = 00:00:00.00

true = Nonconflict(MSUPER2,SYST2) Computing time = 00:00:00.00

TMSUPER2 = Trim(MSUPER2) (28,128) Computing time = 00:00:00.00

true = Isomorph(MSUPER2,TMSUPER2;identity) Computing time = 00:00:00.00

TEST2 = Meet(MSUPER2,SYST2) (72,243) Computing time = 00:00:00.00

true = Isomorph(SUPER2,TEST2;identity) Computing time = 00:00:00.05

REP1 = Create(REP1,[mark 0],[tran [0,8,1],[0,9,0],[1,13,0]]) (2,3)

REP1 = Selfloop(REP1,[1,2,3,4,5,6,7,10,11,12,15,17]) (2,27) Computing time = 00:00:00.00

REP2 = Create(REP2,[mark 0],[tran [0,9,0],[0,12,1],[1,17,0]]) (2,3)

REP2 = Selfloop(REP2,[1,2,3,4,5,6,7,8,10,11,13,15]) (2,27) Computing time = 00:00:00.00

REPAIR = Meet(REP1,REP2) (4,49) Computing time = 00:00:00.00

true = Nonconflict(REPAIR,SYST2) Computing time = 00:00:00.00

DATREP = Condat(SYST2,REPAIR) Controllable. Computing time = 00:00:00.00

MSUPER22 = Meet(MSUPER2,REPAIR) (112,396) Computing time = 00:00:00.00

true = Nonconflict(MSUPER22,SYST2) Computing time = 00:00:00.00

TMSUP22 = Trim(MSUPER22) (112,396) Computing time = 00:00:00.00

true = Isomorph(MSUPER22,TMSUP22;identity) Computing time = 00:00:00.06

SPEC22 = Meet(SPEC2,REPAIR) (24,141) Computing time = 00:00:00.00

SUPER22 = Supcon(SYST2,SPEC22) (72,234) Computing time = 00:00:00.00

SUPER22 = Condat(SYST2,SUPER22) Controllable. Computing time = 00:00:00.00

TEST22 = Meet(SYST2,MSUPER22) (72,234) Computing time = 00:00:00.00

true = Isomorph(SUPER22,TEST22;identity) Computing time = 00:00:00.00

PSYST2 = Project(SYST2,Null[1,3,5]) (27,108) Computing time = 00:00:00.06

GLO = Edit(SYST2V) (27,189)

OCGLO = Outconsis(GLO) (44,309) Computing time = 00:00:00.06

HCGLO = Hiconsis(OCGLO) (44,309) Computing time = 00:00:00.11

GHI = Higen(HCGLO) (7,17) Computing time = 00:00:00.11

SPECHI = Create(SPECHI,[mark 0],[tran [0,111,1],[1,111,2],[2,111,3],[3,111 ,4],[4,100,0]]) (5,5)

SPECHI = Selfloop(SPECHI,[101,110]) (5,15) Computing time = 00:00:00.00

TSPECHI = Trim(SPECHI) (5,15) Computing time = 00:00:00.00

true = Isomorph(TSPECHI,SPECHI;identity) Computing time = 00:00:00.05

MEETHI = Meet(SPECHI,GHI) (16,27) Computing time = 00:00:00.05

TMEETHI = Trim(MEETHI) (16,27) Computing time = 00:00:00.06

SUPERHI = Supcon(GHI,SPECHI) (1,0) Computing time = 00:00:00.06

SUPERHI = Condat(GHI,SUPERHI) Controllable. Computing time = 00:00:00.00

SPECHI2 = Create(SPECHI2,[mark 0],[tran [0,100,1],[0,111,0],[1,100,2],[1,1 11,1],[2,100,3],[2,111,2]]) (4,6)

SPECHI2 = Selfloop(SPECHI2,[101,110]) (4,14) Computing time = 00:00:00.00

SUPERHI2 = Supcon(GHI,SPECHI2) (1,0) Computing time = 00:00:00.00

SUPERHI2 = Condat(GHI,SUPERHI2) Controllable. Computing time = 00:00:00.00

SPECHI2 = Edit(SPECHI2,[mark +[3]]) (4,14)

SUPERHI2 = Supcon(GHI,SPECHI2) (10,16) Computing time = 00:00:00.00

SUPERHI2 = Condat(GHI,SUPERHI2) Controllable. Computing time = 00:00:00.00

********** TIMED OPERATIONS **********

shovel80 = ACreate(shovel80,[mark 0],[timebounds [1,0,1000],[3,0,1000],[5, 0,1000]],[forcible 1,],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)

t1c80 = ACreate(t1c80,[mark 0],[timebounds [2,2,3],[4,1,2],[7,1,1000],[9,2 ,1000]],[forcible 7,],[tran [0,7,1],[1,2,0],[1,4,2],[2,9,0]]) (3, 4)

t1c160 = ACreate(t1c160,[mark 0],[timebounds [6,2,3],[8,1,2],[11,1,1000],[13,2,1000]],[forcible 11,],[tran [0,11,1],[1,6,0],[1,8,2],[2,13,0]]) (3,4)

t2c160 = ACreate(t2c160,[mark 0],[timebounds [10,2,3],[12,1,2],[15,1,1000] ,[17,2,1000]],[forcible 15,],[tran [0,15,1],[1,10,0],[1,12,2],[2, 17,0]]) (3,4)

tc160 = Comp(t1c160,t2c160) (9,24) Computing time = 0 sec

trucks = Comp(tc160,t1c80) (27,108) Computing time = 0 sec

system2 = Comp(trucks,shovel80) (27,189) Computing time = 0 sec

system2 = TimedGraph(system2) (512,2839) Computing time = 0 sec

b1 = Create(b1,[mark 0],[tran [0,0,0],[0,1,1],[0,2,0],[0,3,0],[0,4,0],[0,5 ,0],[0,9,0],[1,0,1],[1,7,0]],[forcible 1,3,5,7,9]) (2,9)

b1 = Selfloop(b1,[6,8,10,11,12,13,15,17],[new forcible 11,13,15,17]) (2,2 5) Computing time = 0 sec

b2 = Create(b2,[mark 0],[tran [0,0,0],[0,1,0],[0,3,1],[0,5,0],[0,6,0],[0,8 ,0],[0,13,0],[1,0,1],[1,3,2],

[2,0,2],[2,11,0]],[forcible 1,3,5,11,13]) (3,11)

b2 = Selfloop(b2,[2,4,7,9,10,12,15,17],[new forcible 7,9,15,17]) (3,35) Computing time = 0 sec

b3 = Create(b3,[mark 0],[tran [0,0,0],[0,1,0],[0,3,0],[0,5,1],[0,10,0],[0, 12,0],[0,17,0],[1,0,1],[1,5,2],
[2,0,2],[2,15,0]],[forcible 1,3,5,15,17]) (3,11)

b3 = Selfloop(b3,[2,4,6,7,8,9,11,13],[new forcible 7,9,11,13]) (3,35) Computing time = 0 sec

b1b2 = Meet(b1,b2) (4,41) Computing time = 0 sec

spec2 = Meet(b1b2,b3) (6,53) Computing time = 0 sec

super2 = Supcon(system2,spec2) (1152,3073) Computing time = 17 sec

timer = Create(timer,[mark 0,1,2,3,4,5,6],[tran [0,0,1],[1,0,2],[2,0,3],[3 ,0,4],[4,0,5],[5,0,6]]) (7,6)

timer = Selfloop(timer,[1,2,3,4,5,6,7,8,9,10,11,12,13,15,17],[new forcible 1,3,5,7,9,11,13,15,17])
(7,111) Computing time = 0 sec

timer5 = Edit(timer,[mark -[6]],[states -[6]],[trans +[5,0,6],-[5,0,6]]) (7,95)

timer5 = Minstate(timer5) (6,95) Computing time = 0 sec

mb1 = Create(mb1,[mark 0],[tran [0,1,1],[0,3,0],[0,5,0],[1,7,2],[2,2,0],[2 ,3,2],[2,4,2],[2,5,2],
[2,9,0]],[forcible 1,3,5,7,9]) (3,9)

mb1 = Selfloop(mb1,[0,6,8,10,11,12,13,15,17],[new forcible 11,13,15,17]) (3,36) Computing time
= 0 sec

datmb1 = Condat(system2,mb1) Computing time = 0 sec

mb2 = Create(mb2,[mark 0],[tran [0,1,0],[0,3,1],[0,5,0],[1,3,2],[2,11,3],[3,1,3],[3,5,3],[3,6,0],[3,8,3],
[3,13,0]],[forcible 1,3,5,11,13]) (4,10)

mb2 = Selfloop(mb2,[0,2,4,7,9,10,12,15,17],[new forcible 7,9,15,17]) (4,4 6) Computing time =
0 sec

datmb2 = Condat(system2,mb2) Computing time = 0 sec

mb3 = Create(mb3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,1],[1,5,2],[2,15,3],[3,1,3],[3,3,3],[3,10,0],[3,12,3],
[3,17,0]],[forcible 1,3,5,15,17]) (4,10)

mb3 = Selfloop(mb3,[0,2,4,6,7,8,9,11,13],[new forcible 7,9,11,13]) (4,46) Computing time = 0
sec

datmb3 = Condat(system2,mb3) Computing time = 0 sec

mb1mb2 = Meet(mb1,mb2) (10,85) Computing time = 0 sec

msuper2 = Meet(mb1mb2,mb3) (28,156) Computing time = 0 sec

true = Nonconflict(msuper2,system2) Computing time = 0 sec

tmsuper2 = Trim(msuper2) (28,156) Computing time = 0 sec

true = Isomorph(msuper2,tmsuper2;identity) Computing time = 0 sec

test2 = Meet(system2,msuper2) (1152,3073) Computing time = 0 sec

true = Isomorph(super2,test2;identity) Computing time = 0 sec

timer4 = Edit(timer5,[mark -[5]],[states -[5]],[trans -[4,0,5]]) (6,79)

timer4 = Minstate(timer4) (5,79) Computing time = 0 sec

timer3 = Edit(timer4,[mark -[4]],[states -[4]],[trans -[3,0,4]]) (5,63)

timer3 = Minstate(timer3) (4,63) Computing time = 0 sec

stimer4 = Supcon(super2,timer4) (0,0) Computing time = 7 sec

stimer4 = Condat(super2,stimer4) Computing time = 0 sec

stimer5 = Supcon(super2,timer5) (0,0) Computing time = 73 sec

timer6 = Edit(timer) (7,111)

reps80 = Create(reps80,[mark 0],[tran [0,0,0],[0,1,0],[0,2,0],[0,4,1],[0,7 ,0],[1,0,2],[1,9,0],[2,0,3],[2,9,0],
[3,0,4],[3,9,0],[4,9,0]],[forcible 1,7,9]) (5,12)

reps80 = Selfloop(reps80,[3,5,6,8,10,11,12,13,15,17],[new forcible 3,5,11, 13,15,17]) (5,62) Computing time = 0 sec

sreps80 = Supcon(system2,reps80) (576,2919) Computing time = 0 sec

sreps80 = Condat(system2,sreps80) Computing time = 1 sec

spec22 = Meet(spec2,reps80) (26,179) Computing time = 0 sec

super22 = Supcon(system2,spec22) (1280,3377) Computing time = 17 sec

spec22 = Condat(system2,spec22) Computing time = 75 sec

reps802 = Create(reps802,[mark 0],[tran [0,0,0],[0,1,0],[0,2,0],[0,4,1],[0 ,7,0],[1,0,2],[1,9,0],[2,0,3],
[2,9,0],[3,0,4],[3,9,0],[4,0,5],[4,9,0]],[forcible 1,7,9]) (6,13)

reps802 = Selfloop(reps802,[3,5,6,8,10,11,12,13,15,17],[new forcible 3,5,1 1,13,15,17]) (6,73) Computing time = 1 sec

spec23 = Meet(spec2,reps802) (31,208) Computing time = 0 sec

super23 = Supcon(system2,spec23) (1280,3377) Computing time = 122 sec

super23 = Condat(system2,super23) Computing time = 178 sec

B.3 Printouts for Case 3

********** UNTIMED OPERATIONS **********

SHOVEL40 = Create(SHOVEL40,[mark 0],[tran [0,1,0],[0,3,0]]) (1,2)

SHOVEL80 = Create(SHOVEL80,[mark 0],[tran [0,7,0],[0,9,0]]) (1,2)

T1C80 = Create(T1C80,[mark 0],[tran [0,13,1],[1,2,0],[1,4,2],[2,17,0]]) (3,4)

T1C160 = Create(T1C160,[mark 0],[tran [0,19,1],[1,6,0],[1,8,2],[2,23,0]]) (3,4)

B1 = Create(B1,[mark 0],[tran [0,1,1],[0,2,0],[0,3,0],[0,4,0],[0,7,3],[0,9 ,0],[0,17,0],
[1,1,2],[1,9,1],[2,9,2],[2,13,0],[3,3,3],[3,13,0]]) (4, 13)

B1 = Selfloop(B1,[6,8,19,23]) (4,29) Computing time = 00:00:00.00

B2 = Create(B2,[mark 0],[tran [0,1,0],[0,3,3],[0,6,0],[0,7,0],[0,8,0],[0,9 ,1],[0,23,0],
[1,1,1],[1,9,2],[2,1,2],[2,19,0],[3,3,4],[3,7,3],[4,3,5] ,[4,7,4],[5,3,6],[5,7,5],[6,7,6],[6,19,0]]) (7,19)

B2 = Selfloop(B2,[2,4,13,17]) (7,47) Computing time = 00:00:00.00

MB1 = Create(MB1,[mark 0],[tran [0,1,1],[0,3,0],[0,7,3],[0,9,0],[1,1,2],[1 ,9,1],[2,9,2],[2,13,4],
[3,3,3],[3,13,4],[4,2,0],[4,3,4],[4,4,4],[4,9 ,4],[4,17,0]]) (5,15)

MB1 = Selfloop(MB1,[6,8,19,23]) (5,35) Computing time = 00:00:00.00

MB2 = Create(MB2,[mark 0],[tran [0,1,0],[0,3,3],[0,7,0],[0,9,1],[1,1,1],[1 ,9,2],[2,1,2],
[2,19,7],[3,3,4],[3,7,3],[4,3,5],[4,7,4],[5,3,6],[5,7, 5],[6,7,6],[6,19,7],[7,1,7],[7,6,0],[7,7,7],[7,8,7],[7,23,0]])
(8,2 1)

MB2 = Selfloop(MB2,[2,4,13,17]) (8,53) Computing time = 00:00:00.00

TRUCKS = Sync(T1C80,T1C160) (9,24) Computing time = 00:00:00.00

SHOVELS = Sync(SHOVEL40,SHOVEL80) (1,4) Computing time = 00:00:00.00

SYST3 = Sync(SHOVELS,TRUCKS) (9,60) Computing time = 00:00:00.06

MEETB1B2 = Meet(B1,B2) (18,71) Computing time = 00:00:00.00

SPEC3 = Meet(B1,B2) (18,71) Computing time = 00:00:00.00

SUPER3 = Supcon(SYST3,SPEC3) (44,109) Computing time = 00:00:00.00

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.00

MSUPER3 = Meet(MB1,MB2) (30,90) Computing time = 00:00:00.00

TEST3 = Meet(SYST3,MSUPER3) (44,109) Computing time = 00:00:00.00

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.00

SHOVEL80 = Edit(SHOVEL80,[trans +[0,11,0]]) (1,3)

B1 = Edit(B1,[trans +[0,11,0],+[1,11,1],+[2,11,2]]) (4,32)

MB1 = Edit(MB1,[trans +[0,11,0],+[1,11,1],+[2,11,2],+[4,11,4]]) (5,39)

B2 = Edit(B2,[trans +[0,11,0],+[3,11,3],+[4,11,4],+[5,11,5],+[6,11,6]]) (7,52)

MB2 = Edit(MB2,[trans +[0,11,0],+[3,11,3],+[4,11,4],+[5,11,5],+[6,11,6],+[7,11,7]]) (8,59)

SHOVELS = Sync(SHOVEL40,SHOVEL80) (1,5) Computing time = 00:00:00.00

SYST3 = Sync(SHOVELS,TRUCKS) (9,69) Computing time = 00:00:00.00

SPEC3 = Meet(B1,B2) (18,78) Computing time = 00:00:00.00

SUPER3 = Supcon(SYST3,SPEC3) (44,136) Computing time = 00:00:00.00

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.00

MSUPER3 = Meet(MB1,MB2) (30,106) Computing time = 00:00:00.00

TEST3 = Meet(MSUPER3,SYST3) (44,136) Computing time = 00:00:00.00

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.06

SHOVEL40 = Edit(SHOVEL40,[trans +[0,5,0]]) (1,3)

SHOVELS = Sync(SHOVEL40,SHOVEL80) (1,6) Computing time = 00:00:00.00

SYST3 = Sync(SHOVELS,TRUCKS) (9,78) Computing time = 00:00:00.00

B1 = Edit(B1,[trans +[0,5,0],+[3,5,3]]) (4,34)

B2 = Edit(B2,[trans +[0,5,0],+[1,5,1],+[2,5,2]]) (7,55)

MB1 = Edit(MB1,[trans +[0,5,0],+[3,5,3],+[4,5,4]]) (5,42)

MB2 = Edit(MB2,[trans +[1,5,1],+[2,5,2],+[7,5,7]]) (8,62)

MB2 = Edit(MB2,[trans +[0,5,0]]) (8,63)

SPEC3 = Meet(B1,B2) (18,82) Computing time = 00:00:00.00

MSUPER3 = Meet(MB1,MB2) (30,116) Computing time = 00:00:00.05

SUPER3 = Supcon(SYST3,SPEC3) (44,154) Computing time = 00:00:00.00

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.00

TEST3 = Meet(MSUPER3,SYST3) (44,154) Computing time = 00:00:00.00

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.06

T2C160 = Create(T2C160,[mark 0],[tran [0,25,1],[1,10,0],[1,12,2],[2,29,0]]) (3,4)

TRUCKS = Sync(TRUCKS,T2C160) (27,108) Computing time = 00:00:00.00

SYST3 = Sync(TRUCKS,SHOVELS) (27,270) Computing time = 00:00:00.00

B1 = Selfloop(B1,[10,12,25,29]) (4,50) Computing time = 00:00:00.06

B2 = Selfloop(B2,[10,12,25,29]) (7,83) Computing time = 00:00:00.00

MB1 = Selfloop(MB1,[10,12,25,29]) (5,62) Computing time = 00:00:00.00

MB2 = Selfloop(MB2,[10,12,25,29]) (8,95) Computing time = 00:00:00.00

SPEC3 = Meet(B1,B2) (18,154) Computing time = 00:00:00.00

SUPER3 = Supcon(SYST3,SPEC3) (132,638) Computing time = 00:00:00.00

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.05

MSUPER3 = Meet(MB1,MB2) (30,236) Computing time = 00:00:00.00

TEST3 = Meet(SYST3,MSUPER3) (132,638) Computing time = 00:00:00.00

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.00

B3 = Create(B3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,0],[0,7,0],[0,9,0],[0,1 0,0],[0,11,1],[0,12,0],
[0,29,0],[1,1,1],[1,3,1],[1,11,2],[2,1,2],[2,3 ,2],[2,25,0]]) (3,15)

B3 = Selfloop(B3,[2,4,6,8,13,17,19,23]) (3,39) Computing time = 00:00:00.00

MB3 = Create(MB3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,0],[0,7,0],[0,9,0],[0 ,11,1],[1,1,1],
[1,3,1],[1,11,2],[2,1,2],[2,3,2],[2,25,3],[3,1,3],[3, 3,3],[3,5,3],[3,7,3],[3,9,3],[3,10,0],[3,12,3],[3,29,0]])
(4,20)

MB3 = Selfloop(MB3,[2,4,6,8,13,17,19,23]) (4,52) Computing time = 00:00:00.00

SPEC3 = Meet(SPEC3,B3) (32,214) Computing time = 00:00:00.00

SUPER3 = Supcon(SYST3,SPEC3) (186,714) Computing time = 00:00:00.05

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.00

MSUPER3 = Meet(MSUPER3,MB3) (92,430) Computing time = 00:00:00.00

TEST3 = Meet(MSUPER3,SYST3) (186,714) Computing time = 00:00:00.00

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.05

B3 = Create(B3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,3],[0,7,0],[0,9,0],[0,1 0,0],[0,11,1],[0,12,0],
[0,29,0],[1,1,1],[1,3,1],[1,11,2],[2,1,2],[2,3 ,2],[2,25,0],[3,5,4],[3,7,3],[3,9,3],[4,5,5],[4,7,4],[4,9,4],
[5,5,6],[5,7,5],[5,9,5],[6,7,6],[6,9,6],[6,25,0]]) (7,27)

B3 = Selfloop(B3,[2,4,6,8,13,17,19,23]) (7,83) Computing time = 00:00:00.00

MB3 = Create(MB3,[mark 0],[tran [0,1,0],[0,3,0],[0,5,3],[0,7,0],[0,9,0],[0 ,11,1],[1,1,1],
[1,3,1],[1,11,2],[2,1,2],[2,3,2],[2,25,7],[3,5,4],[3, 7,3],[3,9,3],[4,5,5],[4,7,4],[4,9,4],[5,5,6],[5,7,5],[5,9,5],[6,7,6]
,[6,9,6],[6,25,7],[7,1,7],[7,3,7],[7,7,7],[7,9,7],[7,10,0],[7,12,7], [7,29,0]]) (8,31)

MB3 = Selfloop(MB3,[2,4,6,8,13,17,19,23]) (8,95) Computing time = 00:00:00.00

MEETB1B2 = Meet(B1,B2) (18,154) Computing time = 00:00:00.00

SPEC3 = Meet(MEETB1B2,B3) (48,310) Computing time = 00:00:00.00

SUPER3 = Supcon(SYST3,SPEC3) (258,918) Computing time = 00:00:00.00

SUPER3 = Condat(SYST3,SUPER3) Controllable. Computing time = 00:00:00.06

MB1MB2 = Meet(MB1,MB2) (30,236) Computing time = 00:00:00.00

MSUPER3 = Meet(MB1MB2,MB3) (132,584) Computing time = 00:00:00.00

TEST3 = Meet(MSUPER3,SYST3) (258,918) Computing time = 00:00:00.06

true = Isomorph(SUPER3,TEST3;identity) Computing time = 00:00:00.05

********** TIMED OPERATIONS **********

shovel40 = ACreate(shovel40,[mark 0],[timebounds [1,1,1000],[3,1,1000],[5, 1,1000]],[forcible 1,],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)

shovel80 = ACreate(shovel80,[mark 0],[timebounds [7,1,1000],[9,1,1000],[11 ,1,1000]],[forcible 7,],[tran [0,7,0],[0,9,0],[0,11,0]]) (1,3)

t1c80 = ACreate(t1c80,[mark 0],[timebounds [2,3,4],[4,2,3],[6,1,2],[13,3,1 000],[15,2,1000],
[17,3,1000],[19,2,1000],[21,1,1000],[23,2,1000]],[forcible 13,15,17,19,],[tran [0,13,1],
[0,15,5],[1,17,2],[2,2,3],[2,6,6],[3,21,0],[4,4,3],[4,6,6],[5,19,4],[6,23,0]]) (7,10)

t1c160 = ACreate(t1c160,[mark 0],[timebounds [8,3,4],[10,2,3],[12,1,2],[25,3,1000],
[27,2,1000],[29,5,1000],[31,3,1000],[33,1,1000],[35,2,1000]],[forcible 25,27,29,31,],
[tran [0,25,1],[0,27,5],[1,29,2], [2,8,3],[2,12,6],[3,33,0],[4,10,3],[4,12,6],[5,31,4],[6,35,0]]) (7,10)

t2c160 = ACreate(t2c160,[mark 0],[timebounds [14,3,4],[16,2,3],[18,1,2],[3 7,3,1000],[39,2,1000],
[41,5,1000],[43,3,1000],[45,1,1000],[47,2,1000]],[forcible 37,39,41,43,],
[tran [0,37,1],[0,39,5],[1,41,2],[2,14,3],[2,18,6],[3,45,0],[4,16,3],
[4,18,6],[5,43,4],[6,47,0]]) (7,10)
tc160 = Comp(t1c160,t2c160) (49,140) Computing time = 0 sec
trucks = Comp(tc160,t1c80) (343,1470) Computing time = 0 sec
shovels = Comp(shovel40,shovel80) (1,6) Computing time = 0 sec
system = Comp(shovels,trucks) (343,3528) Computing time = 0 sec
trucks = TimedGraph(trucks) (13750,31280) Computing time = 19 sec
shovels = TimedGraph(shovels) (64,256) Computing time = 0 sec
t1c80 = AEdit(t1c80,[changed timebounds [17 [0,1000]],[19 [0,1000]]]) (7,10)
t1c160 = AEdit(t1c160,[changed timebounds [29 [0,1000]],[31 [0,100 0]]]) (7,10)
t2c160 = AEdit(t2c160,[changed timebounds [41 [0,1000]],[43 [0,100 0]]]) (7,10)
tc160 = Comp(t1c160,t2c160) (49,140) Computing time = 0 sec
trucks = Comp(tc160,t1c80) (343,1470) Computing time = 0 sec
trucks = TimedGraph(trucks) (4913,13779) Computing time = 1 sec

B.4 Printouts for Case 4

****

SHOVEL40 = Create(SHOVEL40,[mark 0],[tran [0,1,0],[0,3,0],[0,5,0]]) (1,3)
SHOVEL80 = Create(SHOVEL80,[mark 0],[tran [0,7,0],[0,9,0],[0,11,0]]) (1,3)
T1C80 = Create(T1C80,[mark 0],[tran [0,13,1],[0,15,5],[1,17,2],[2,2,3],[2, 6,6],
[3,21,0],[4,4,3],[4,6,6],[5,19,4],[6,23,0]]) (7,10)
B1 = Create(B1,[mark 0],[tran [0,2,0],[0,3,0],[0,4,0],[0,5,0],[0,6,0],[0,9,0],[0,11,0],
[0,13,1],[0,15,4],[0,21,0],[0,23,0],[1,1,2],[1,3,1],[1,5,1],[1,9,1],[1,11,1],[2,1,3],
[2,9,2],[2,11,2],[3,9,3],[3,11,3],[3,17,0],[4,3,4],[4,5,4],[4,7,5],[4,9,4],
[4,11,4],[5,3,5],[5,5,5],[5,19,0]]) (6,30)
SHOVELS = Sync(SHOVEL40,SHOVEL80) (1,6) Computing time = 00:00:00.05
SYST4 = Sync(SHOVELS,T1C80) (7,52) Computing time = 00:00:00.06
SUPER4 = Supcon(SYST4,B1) (10,47) Computing time = 00:00:00.00
SUPER4 = Condat(SYST4,SUPER4) Controllable. Computing time = 00:00:00.00
true = Nonconflict(B1,SYST4) Computing time = 00:00:00.00
TB1 = Trim(B1) (6,30) Computing time = 00:00:00.00

true = Isomorph(TB1,B1;identity) Computing time = 00:00:00.00

TEST4 = Meet(SYST4,B1) (10,47) Computing time = 00:00:00.05

true = Isomorph(SUPER4,TEST4;identity) Computing time = 00:00:00.00

MB1 = Edit(B1) (6,30)

true = Nonconflict(MB1,SYST4) Computing time = 00:00:00.00

TMB1 = Trim(MB1) (6,30) Computing time = 00:00:00.00

true = Isomorph(MB1,TMB1;identity) Computing time = 00:00:00.00

TEST4 = Meet(MB1,SYST4) (10,47) Computing time = 00:00:00.00

true = Isomorph(SUPER4,TEST4;identity) Computing time = 00:00:00.05

DATMB1 = Condat(SYST4,MB1) Controllable. Computing time = 00:00:00.00

T1C160 = Create(T1C160,[mark 0],[tran [0,25,1],[0,27,5],[1,29,2],[2,8,3],[2,12,6],[3,33,0],
[4,10,3],[4,12,6],[5,31,4],[6,35,0]]) (7,10)

B2 = Create(B2,[mark 0],[tran [0,1,0],[0,5,0],[0,7,0],[0,8,0],[0,10,0],[0,11,0],[0,12,0],[0,25,1],
[0,27,6],[0,33,0],[0,35,0],[1,1,1],[1,3,2],[1,5,1],[1,7,1],[1,11,1],[2,3,3],[2,7,2],
[2,11,2],[3,3,4],[3,7,3],[3,11,3],[4,3,5],[4,7,4],[4,11,4],[5,7,5],[5,11,5],[5,29,0],
[6,1,6],[6,5,6],[6,7,6],[6,9,7],[6,11,6],[7,1,7],[7,5,7], [7,9,8],[8,1,8],
[8,5,8],[8,31,0]]) (9,39)

B2 = Selfloop(B2,[2,4,6,13,15,17,19,21,23]) (9,120) Computing time = 00:00:00.00

B1 = Selfloop(B1,[8,10,12,25,27,29,31,33,35]) (6,84) Computing time = 00:00:00.00

SPEC4 = Meet(B1,B2) (44,214) Computing time = 00:00:00.05

TRUCKS = Sync(T1C80,T1C160) (49,140) Computing time = 00:00:00.00

SYST4 = Sync(TRUCKS,SHOVELS) (49,434) Computing time = 00:00:00.00

SUPER4 = Supcon(SYST4,SPEC4) (120,461) Computing time = 00:00:00.00

SUPER4 = Condat(SYST4,SUPER4) Controllable. Computing time = 00:00:00.00

MB1 = Edit(B1) (6,84)

MB2 = Edit(B2) (9,120)

DATMB1 = Condat(SYST4,MB1) Controllable. Computing time = 00:00:00.00

DATMB2 = Condat(SYST4,MB2) Controllable. Computing time = 00:00:00.06

MSUPER4 = Meet(MB1,MB2) (44,214) Computing time = 00:00:00.05

true = Nonconflict(SYST4,MSUPER4) Computing time = 00:00:00.00

TMSUPER4 = Trim(MSUPER4) (44,214) Computing time = 00:00:00.00

true = Isomorph(TMSUPER4,MSUPER4;identity) Computing time = 00:00:00.00

TEST4 = Meet(MSUPER4,SYST4) (120,461) Computing time = 00:00:00.00

true = Isomorph(SUPER4,TEST4;identity) Computing time = 00:00:00.00

********** Hierarchical supervisory control **********

SYST4V = Edit(SYST4,[voc [11,11],[12,12],[17,11],[18,12],[21,11],[22,12],[24,10],[29,11],

[30,12],[32,10],[33,11],[34,12],[36,10],[37,11],[3 8,12],[39,11],[40,12],[42,10],[44,10],[46,10],[48,10]]) (49,434)

GLO = Edit(SYST4V) (49,434)

OCGLO = Outconsis(GLO) (60,529) Computing time = 00:00:00.05

HCGLO = Hiconsis(OCGLO) (60,529) Computing time = 00:00:00.77

true = Isomorph(OCGLO,HCGLO;identity) Computing time = 00:00:00.00

GHI = Higen(HCGLO) (8,26) Computing time = 00:00:00.05

SPECHI = Create(SPECHI,[mark 0,3],[tran [0,100,1],[0,121,0],[1,100,2],[1,1 21,1],[2,100,3],[2,121,2]]) (4,6)

SPECHI = Selfloop(SPECHI,[101,110,111,120]) (4,22) Computing time = 00:00:00.00

SUPERHI = Supcon(GHI,SPECHI) (18,44) Computing time = 00:00:00.00

SUPERHI = Condat(GHI,SUPERHI) Controllable. Computing time = 00:00:00.00

****

T2C160 = Create(T2C160,[mark 0],[tran [0,37,1],[0,39,5],[1,41,2],[2,14,3], [2,18,6],[3,45,0],

[4,16,3],[4,18,6],[5,43,4],[6,47,0]]) (7,10)

B3 = Create(B3,[mark 0],[tran [0,1,0],[0,3,0],[0,7,0],[0,9,0],[0,14,0],[0, 16,0],[0,18,0],[0,37,1],

[0,39,6],[0,45,0],[0,47,0],[1,1,1],[1,3,1],[1 ,5,2],[1,7,1],[1,9,1],[2,5,3],[2,7,2],[2,9,2],[3,5,4],[3,7,3],[3,9,3],

[4,5,5],[4,7,4],[4,9,4],[5,7,5],[5,9,5],[5,41,0],[6,1,6],[6,3,6],[6, 7,6],[6,9,6],[6,11,7],[7,1,7],

[7,3,7],[7,11,8],[8,1,8],[8,3,8],[8,43, 0]]) (9,39)

B3 = Selfloop(B3,[2,4,6,8,10,12,13,15,17,19,21,23,25,27,29,31,33,35]) (9, 201) Computing time = 00:00:00.05

B1 = Selfloop(B1,[14,16,18,37,39,41,43,45,47]) (6,138) Computing time = 00:00:00.00

B2 = Selfloop(B2,[14,16,18,37,39,41,43,45,47]) (9,201) Computing time = 00:00:00.00

TC160 = Sync(T1C160,T2C160) (49,140) Computing time = 00:00:00.00

TRUCKS = Sync(TC160,T1C80) (343,1470) Computing time = 00:00:00.05

SYST4 = Sync(SHOVELS,TRUCKS) (343,3528) Computing time = 00:00:00.05

MEETB1B2 = Meet(B1,B2) (44,610) Computing time = 00:00:00.00

SPEC4 = Meet(MEETB1B2,B3) (258,1514) Computing time = 00:00:00.06

SUPER4 = Supcon(SYST4,SPEC4) (1302,4799) Computing time = 00:00:00.11

SUPER4 = Condat(SYST4,SUPER4) Controllable. Computing time = 00:00:00.17

DATMB1 = Condat(SYST4,MB1) Controllable. Computing time = 00:00:00.00

DATMB2 = Condat(SYST4,MB2) Controllable. Computing time = 00:00:00.00

MB1 = Edit(B1) (6,138)

MB2 = Edit(B2) (9,201)

MB3 = Edit(B3) (9,201)

MB1MB2 = Meet(MB1,MB2) (44,610) Computing time = 00:00:00.06

MSUPER4 = Meet(MB1MB2,MB3) (258,1514) Computing time = 00:00:00.05

DATMB1 = Condat(SYST4,MB1) Controllable. Computing time = 00:00:00.06

DATMB2 = Condat(SYST4,MB2) Controllable. Computing time = 00:00:00.05

DATMB3 = Condat(SYST4,MB3) Controllable. Computing time = 00:00:00.00

true = Nonconflict(SYST4,MSUPER4) Computing time = 00:00:00.00

TMSUPER4 = Trim(MSUPER4) (258,1514) Computing time = 00:00:00.00

true = Isomorph(TMSUPER4,MSUPER4;identity) Computing time = 00:00:00.11

TEST4 = Meet(MSUPER4,SYST4) (1302,4799) Computing time = 00:00:00.05

true = Isomorph(SUPER4,TEST4;identity) Computing time = 00:00:00.28

********** Second specification **********

REP1 = Create(REP1,[mark 0],[tran [0,12,1],[0,23,0],[1,35,0]]) (2,3)

REP1 = Selfloop(REP1,[2,4,6,8,10,13,14,15,16,17,18,19,21,25,27,29,31,33,37 ,39,41,43,45,47]) (2,51) Computing time = 00:00:00.00

REP1 = Selfloop(REP1,[1,3,5,7,9,11]) (2,63) Computing time = 00:00:00.05

REP2 = Create(REP2,[mark 0],[tran [0,18,1],[0,23,0],[1,47,0]]) (2,3)

REP2 = Selfloop(REP2,[2,4,6,8,10,12,13,14,15,16,17,19,21,25,27,29,31,33,35 ,37,39,41,43,45]) (2,51) Computing time = 00:00:00.00

REP2 = Selfloop(REP2,[1,3,5,7,9,11]) (2,63) Computing time = 00:00:00.00

DATREP1 = Condat(SYST4,REP1) Controllable. Computing time = 00:00:00.00

DATREP2 = Condat(SYST4,REP2) Controllable. Computing time = 00:00:00.00

REPAIR = Meet(REP1,REP2) (4,121) Computing time = 00:00:00.05

SPEC42 = Meet(SPEC4,REPAIR) (1032,5521) Computing time = 00:00:00.05

SUPER42 = Supcon(SYST4,SPEC42) (1302,4774) Computing time = 00:00:00.06

SUPER42 = Condat(SYST4,SUPER42) Controllable. Computing time = 00:00:00.05

true = Nonconflict(REPAIR,SYST4) Computing time = 00:00:00.00

DATREP = Condat(SYST4,REPAIR) Controllable. Computing time = 00:00:00.00

MSUPER42 = Meet(MSUPER4,REPAIR) (1032,5521) Computing time = 00:00:00.00

true = Nonconflict(MSUPER42,SYST4) Computing time = 00:00:00.06

TMSUP42 = Trim(MSUPER42) (1032,5521) Computing time = 00:00:00.00

true = Isomorph(MSUPER42,TMSUP42;identity) Computing time = 00:00:00.22

TEST42 = Meet(MSUPER42,SYST4) (1302,4774) Computing time = 00:00:00.05

true = Isomorph(SUPER42,TEST42;identity) Computing time = 00:00:00.28

B.5 Printouts for Case 5

****

Q = Create(Q,[mark 0],[tran [0,2,0],[0,4,0],[0,5,0],[0,6,0],[0,7,0],[0,8,0],[0,9,0],
[0,10,0],[0,11,0],[0,12,0],[0,13,1],[0,15,0],[0,19,0],[0,21,0],[0,23,0],[0,25,3],
[0,27,0],[0,31,0],[0,33,0],[0,35,0],[1,1,1],[1,7,1],[1,8,1],[1,9,1],[1,10,1],
[1,11,1],[1,12,1],[1,17,0],[1,25,2],[1,27,1],[1,31,1],[1,33,1],[1,35,1],[2,1,2],
[2,9,2],[2,11,2],[2,17,3],[3,2,3],[3,3,3],[3,4,3],[3,6,3],[3,7,3],[3,9,3],[3,11,3],
[3,13,4],[3,15,3],[3,19,3],[3,21,3],[3,23,3],[3,29,0],[4,3,4],[4,7,4],[4,11,4],[4,29,1]]) (5,54)

SPEC5 = Meet(B1,B2) (44,214) Computing time = 00:00:00.00

SYST5 = Sync(SHOVELS,TRUCKS) (49,434) Computing time = 00:00:00.00

ALLSP5 = Allevents(SPEC5) (1,24) Computing time = 00:00:00.00

ALLQ = Allevents(Q) (1,24) Computing time = 00:00:00.06

true = Isomorph(ALLSP5,ALLQ;identity) Computing time = 00:00:00.00

true = Nonconflict(SYST5,Q) Computing time = 00:00:00.00

DATQ = Condat(SYST5,Q) Controllable. Computing time = 00:00:00.00

MB1 = Edit(B1) (6,84)

MB2 = Edit(B2) (9,120)

MB1MB2 = Meet(MB1,MB2) (44,214) Computing time = 00:00:00.00

TQ = Trim(Q) (5,54) Computing time = 00:00:00.00

true = Isomorph(TQ,Q;identity) Computing time = 00:00:00.00

MQ = Edit(Q) (5,54)

MSUPER5 = Meet(MQ,MB1MB2) (45,207) Computing time = 00:00:00.06

true = Nonconflict(MSUPER5,SYST5) Computing time = 00:00:00.00

TEST5 = Meet(MSUPER5,SYST5) (121,446) Computing time = 00:00:00.00

SPEC5 = Meet(SPEC5,Q) (45,207) Computing time = 00:00:00.06

SUPER5 = Supcon(SYST5,SPEC5) (121,446) Computing time = 00:00:00.00

SUPER5 = Condat(SYST5,SUPER5) Controllable. Computing time = 00:00:00.06

true = Isomorph(SUPER5,TEST5;identity) Computing time = 00:00:00.05

PSYST5 = Project(SYST5,Null[1,3,5,7,9,11]) (49,140) Computing time = 00:00:00.05

SYST5V = Edit(SYST5,[voc [1,13],[2,15],[6,13],[9,15],[10,15],[17,13],[18,13],
[21,15],[22,15],[31,18],[32,18],[35,15],[36,15]]) (49,434)

SYST5V = Edit(SYST5V,[voc [31,13],[32,13]]) (49,434)

GLO = Edit(SYST5V) (49,434)

OCGLO = Outconsis(GLO) (53,466) Computing time = 00:00:00.00

HCGLO = Hiconsis(OCGLO) (53,466) Computing time = 00:00:00.33

true = Isomorph(HCGLO,OCGLO;identity) Computing time = 00:00:00.00

GHI = Higen(HCGLO) (3,8) Computing time = 00:00:00.00

SPECHI5 = Create(SPECHI5,[mark 0],[tran [0,130,0],[0,131,1],[0,150,0],[1,130,1],
[1,131,2],[1,150,1],[2,130,2],[2,150,2],[2,151,0]]) (3,9)

SUPERHI5 = Supcon(GHI,SPECHI5) (6,9) Computing time = 00:00:00.00

SUPERHI5 = Condat(GHI,SUPERHI5) Controllable. Computing time = 00:00:00.00

C VDES Theoretical Background

The following paragraphs contain an informal introduction to VDES theory that is meant to facilitate the reading and understanding of the present study. More details can be found in [6] and [7].

C.1 Vector Discrete-Event Systems

Vector discrete-event systems are DESs in which the state transition structure is given a more fundamental role. In fact, in a DES a system's behaviour is characterized by the occurrence of events while a VDES behaviour will be defined through its discrete states. We usually borrow the Petri net formalism as a representation means for VDES. In this formalism, each system's state plays the role of a buffer containing some tokens. In fact, the value of a state provides the number of tokens associated with its buffer. For instance, in Figure 31 the presence of one token in state x_1 and the absence of tokens in states x_2 and x_3 translate into $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$. As in DES, each state of a VDES represents a particular status of a machine (idle, breakdown, etc.). Thus, at any moment the number and location of tokens in the states of a VDES characterize the number of machines of a certain condition. The evolution of a VDES depends upon the occurrence[3] of events (as a_1, a_2, b_1 and b_2 in Figure 31) whose effect is to displace tokens from one location to another.

To represent the evolution of a VDES graphically, we adopt the following rules. For a given event, a state with an event arrow pointing out sees its value decreased while a state with an incoming arrow experiences an increase of its state value. Consequently, a discrete state sees its value augmented (resp., decreased) when one of its entering (resp., exiting) events fires. Normally, the firing of an event decreases by one the number of tokens in all preceding states. Otherwise, a weight (an integer) will be assigned to the event arrows indicating how many tokens are displaced. Moreover, an event can only fire when its attached upstream states possess a sufficient number of tokens. For instance, only events a_1 and a_2 (whose firing require one token) can occur in Figure 31 because $x_1 = 1$. If a_1 fires, the state values will become $x_1 = 0$, $x_2 = 1$ and $x_3 = 0$ as event a_1 leaves x_1 and points into x_2 while x_3 remains unaltered. Therefore, when event b_1 takes place, the original configuration of our example (Figure 31) is recovered.

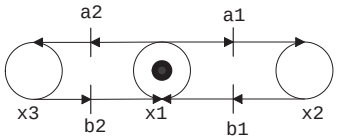

Figure 31: Example of VDES

Formally, a VDES $G = (X, \Sigma, \xi, X_o, X_m)$ is a DES equipped with a vector structure and in which we make use of vector addition over the integers representing the discrete-state values. Thus, a VDES is mainly characterized by an alphabet Σ, a transition function $\xi : X \times \Sigma \to X$, a state vector X and its initial value X_o. Given the state vector X of a VDES and an event σ, the transition function $\xi(X, \sigma)$ computes the next value of X resulting from the firing of σ

[3] As in Petri net theory, we will refer to the "firing" of an event for its occurrence.

in the following manner:

$$\xi(X, \sigma) = X + EV(\sigma).$$

The displacement matrix E converts the firing of event σ into its respective incremental and decremental effects on state vector X. Therefore with $\Sigma = \{\sigma_1\ \sigma_2\ \ldots\ \sigma_m\}$, the matrix E can be written as $E = [e_{\sigma_1}\ e_{\sigma_1}\ \ldots\ e_{\sigma_m}]$ where the column vector e_{σ_i} is the displacement vector associated with event σ_i. In this sense, the matrix E is analogous to a transition matrix since it completely characterizes a VDES. The occurrence vector $V(s)$ extracts from a given sequence of events (or string) s the number of occurrences of event $\sigma_i \in \Sigma$. While entries in E can take negative values, we require that for all $x_i \in X$ we have $x_i \geq 0$. For instance, the firing of event a_1 can be written as $V = [\#a_1\ \#a_2\ \#b_1\ \#b_2]^T = [1\ 0\ 0\ 0]^T$. Also, the effect of the firing of a_1 over $X = [x_1\ x_2\ x_3]^T$ is recorded in the first column of E as $e_{a_1} = [-1\ 1\ 0]^T$ since x_1 is decreased by 1 while x_2 is increased by 1 and x_3 is unaffected by event a_1. With the configuration of Figure 31, we have $X_o = [x_{1o}\ x_{2o}\ x_{3o}] = [1\ 0\ 0]^T$ and the transition following the firing of a_1 is written as

$$\xi(X_o, a1) = X_o + EV(a_1) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{array}{c} \begin{matrix} e_{a_1} & e_{a_2} & e_{b_1} & e_{b_2} \end{matrix} \\ \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \end{array} \begin{bmatrix} \#a_1 = 1 \\ \#a_2 = 0 \\ \#b_1 = 0 \\ \#b_2 = 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},$$

where columns 2 through 4 of matrix E take into account the decremental/incremental effects of events a_2, b_1 and b_2, respectively.

Unlike DES, a VDES representation of similar machines can be grouped into one VDES where the number of machines is indicated by the number of tokens within the machine (i.e., the sum of its states). For instance, the two 160-ton trucks of Figure 32 are modeled by $TC160$ with two tokens in the idle position, x_5, representing the initial status of 160-ton trucks. Despite the token exchange between trucks $TC80$, $TC160$, and shovel $SHOVEL80$, the total number of tokens in each VDES (i.e., the sum of all machine states) remains invariant (i.e., $x_1 + x_2 + x_3 + x_4 = 1$, $x_5 + x_6 = 2$ and $x_7 + x_8 = 1$).

In the actual VDES framework, a self-loop of an event cannot be represented since there is no means to differentiate a self-loop (namely an entry of value $1 - 1 = 0$) from no event (value 0) in matrix E. The usual way to circumvent this limitation is to define an additional dummy state and event so that the self-loop is represented by the sequential firing of two events. Furthermore, the extra event is assumed to take place rapidly and it is further projected out to extract the appropriate behaviour.

C.2 Control of Vector Discrete-Event Systems

Constraints on a VDES can be imposed on the value taken by the states or by the number of times a state is visited. The first type of constraint on the states X of a VDES can be implemented via a *static predicate* P_{static}, while the second type is enforced by a *dynamic predicate* P_{dyn}. Given a state set X, a predicate $P : X \to \{0, 1\}$ is a function placing conditions (0 and 1) on elements of X. Thus, with a predicate P can be associated a subset $X_p \subseteq X$ respecting P. We should think of a predicate P as a specification for the controlled behaviour of a VDES named G. To complete the analogy with specifications, the values 0 and 1 generated

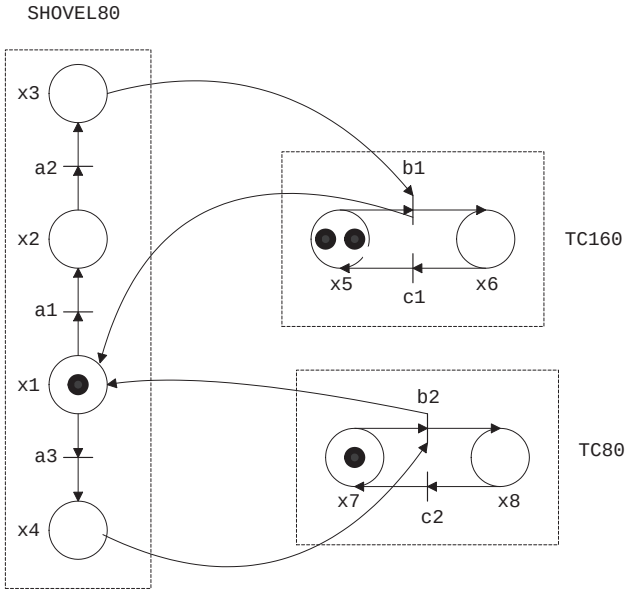

Figure 32: Interacting VDESs

by a predicate and assigned to the states of a VDES mean that a state is forbidden and allowed, respectively. The implementation of a specification is performed through its conversion into a static or a dynamic predicate. The following text provides information on the modelling of specifications.

A static predicate assigns constraints on the state values. By convention, we write a static predicate as

$$x \models P_{static} \quad \text{iff} \quad x = aX \leq b,$$

where $x \models P_{static}$ means "a state $x \in X_p$ satisfies the predicate P_{static}". For instance, the predicate P imposing $x_2 \leq 0$ (by convention of representation even though $x_2 < 0$ is not possible) on the system of Figure 31 could be rewritten with $a = [0\ 1\ 0]$, $X = [x_1\ x_2\ x_3]^T$ and $b = 0$.

A predicate P_{static} is enforced on a VDES by suitably enabling/disabling events $\sigma \in \Sigma$ at some $x \in X$. For this purpose, a state feedback control (SFBC) is adjoined to the VDES. A SFBC takes the value X as inputs and in return generates a collection of predicates $\{f_\sigma\}$ disabling or enabling σ at specific $x \in X$. In the previous example ($x_2 \leq 0$ imposed on VDES of Figure 31), a SFBC will disable event a_1 at state x_1 when $x_2 = 0$. More formally, one can write

$$\begin{array}{lll} f_{a_1}(X) = 1 & \text{if} \quad \xi(X, a_1)! \quad \text{and} & x_2 < 0 \\ f_{a_1}(X) = 0 & \text{otherwise} & \end{array} .$$

This way, whenever a transition with event a_1 exists (namely $\xi(X, a_1)!$) event a_1 cannot be enabled ($f_{a_1}(X) = 1$) since the condition $x_2 < 0$ is never satisfied. Consequently, event a_1 is always disabled ($f_{a_1}(X) = 0$). We usually require a SFBC to be balanced, so that it enables the largest possible set of reachable events satisfying the predicate. This property guarantees some optimal behaviour that is preserved under conjunction of SFBCs.

In some cases, the specification not only depends on the present status of the system but also requires knowledge of the past behaviour. Consequently, the past behaviour is captured by a

dynamic predicate P_{dyn}, which by convention takes the form

$$v \models P_{dyn} \quad \text{iff} \quad d - cv \geq 0,$$

where v, d and c represent the occurrence vector of events contained in P_{dyn}, an integer value, and the coefficient vector of v, respectively. For instance, to enforce $P_{dyn} := \#c_1 \geq \#c_2$ on the VDES of Figure 32, thus specifying that the number of events c_2 in $TC80$ can occur at most "$\#c_1$" times, one would define $v = [v_{c_1}\ v_{c_2}]^T$, $d = 0$ and $c = [-1\ 1]$ where v_{c_1} represents $\#c_1$ and v_{c_2} represents $\#c_2$.

Roughly speaking, a dynamic predicate sets a counter on how often a set of states is visited. Similarly to SFBC, we define a dynamic SFBC (or DSFBC) as the controller that implements the dynamic predicate P_{dyn} over a VDES G. The implementation of a dynamic predicate requires a transformation into an equivalent static predicate as follows. One defines an additional state set Y (namely a collection of event counters) extending the VDES state set X. Then, one converts the dynamic predicate into a static one by ensuring that $\forall y \in Y,\ y \geq 0$. For instance, in the example $P_{dyn} := \#c_1 \geq \#c_2$, one could define a new state set $Y = \{y_1, y_2\}$ where $y_1 = \#c_1$ and $y_2 = \#c_2$ (both guaranteed to be positive). Therefore, with $P_{static} := y_2 - y_1 \leq 0$, P_{dyn} is equivalent to the static predicate P_{static} if one implements P_{static} over the enlarged state set, denoted by $X \oplus Y$.

The conversion of a dynamic predicate P_{dyn} to a static one P_{static} is performed by defining a memory VDES H for G. Formally, a memory $H = (Y, \alpha, \eta, Y_o, Y)$ is a DES where $L(H) \supseteq L(G)$ and whose marker set is the entire state set. The resulting enlarged system (in the sense of adjoining state sets X and Y. i.e., $X \oplus Y$), denoted by $G \oplus H$, as well as the static predicate P_{static} effectively implement P_{dyn} if $L(G \oplus H, P_{static}) = L(G, P_{dyn})$ holds. Therefore, a DSFBC implements the dynamic predicate P_{dyn} over G via $G \oplus H$ and P_{static}. The major advantage of converting dynamic predicates to static ones is that it permits the conjunction of both types of predicates into a static predicate framework, thus enabling combinations of various specifications.

C.3 Supervisors of Vector Discrete-Event Systems

The key property to derive an optimal SFBC relates to the uncontrollable subsystem of G, denoted G_u. A subsystem of G, which is also a VDES, is obtained by picking out a subset of the component of the state vector X and a subset of the event set Σ. For instance, the subsystem G_u contains the complete state set X and the subset of uncontrollable events $\Sigma_u \subseteq \Sigma$. Similarly, we further consider $E_u \subseteq E$ consisting of the displacement submatrix (with columns properly reorganized) due to uncontrollable events. If G_u is loop-free, then an optimal SFBC can possibly be determined. In fact, the existence of a SFBC is related to the solution of a maximization problem

$$\text{maximize } aE_u v^* \text{ subject to } v^* \geq 0,\ X + E_u v^* \geq 0,$$

where a and E_u are as defined above while v^*, the solution of the maximization problem, is an occurrence vector. Therefore, a solution to the above maximization problem, if it exists, is optimal in the sense that a maximal number of event occurrences is enabled. The constraints $v^* \geq 0$ and $X + E_u v^* \geq 0$ insure that the occurrences of events in v^* (positive integers) are such that the state values can still be represented by natural numbers. A SFBC is said to

be optimal if with the VDES initial values X_0 and the solution of the maximization problem v^*, the condition $a(X_0 + E_u v^*(X_0)) \leq b$ is satisfied. Namely, this insures that under the flow of events in v^* and with the VDES in its initial configuration, the static predicate remains valid. Finally, in the presence of event $\sigma \in \Sigma_c$ (the subset of controllable events) the predicates generated by the SFBC, denoted by f_σ, are obtained from the following equation

$$\begin{array}{ll} f_\sigma(X \oplus Y) = 1 & \text{if} \quad \xi \oplus \eta(X \oplus Y, \sigma)! \quad \text{and} \quad a(X \oplus Y)_{new} + aE_u v^*((X \oplus Y)_{new}) \leq b \\ f_\sigma(X \oplus Y) = 0 & \text{otherwise} \end{array} ,$$

where $(X \oplus Y)_{new} = (X \oplus Y) + (e_\sigma \oplus h_\sigma)$ with e_σ and h_σ the displacement vectors for states X and Y, respectively. Therefore, the above expression provides the enablement (resp., disablement) rules of event σ represented by value 1 (resp., 0) depending on the value of the state $X \oplus Y$. One notices that event σ is enabled (i.e., $f_\sigma(X \oplus Y) = 1$) only if event σ remains a transition of the enlarged system $\xi \oplus \eta(X \oplus Y, \sigma)!$ while not violating the static predicate over the enlarged state set $X \oplus Y$.

After the computation of a SFBC, the final step is to derive a VDES implementation (or VDESI) of that SFBC. Namely, we convert the SFBC into a VDES with suitable interdependence relations with the controlled VDESs. This is performed in three steps. First, we determine the subsystem $G^\nabla \subseteq G_u$ uniquely composed of uncontrollable events decreasing the specification coordinates Y [12]. Then, we investigate its loop-freeness. Secondly, we verify that G^∇ decrements at most one state in G. Finally, we rewrite the specification so that it is composed of controllable events. Therefore, we can disable the controllable event when the specification reaches its limit. An algorithm performing this last step is provided in [4]. If the above procedure is successful a VDESI exists. Occasionally, a VDESI can be derived directly from the VDES to control and the dynamic predicate.

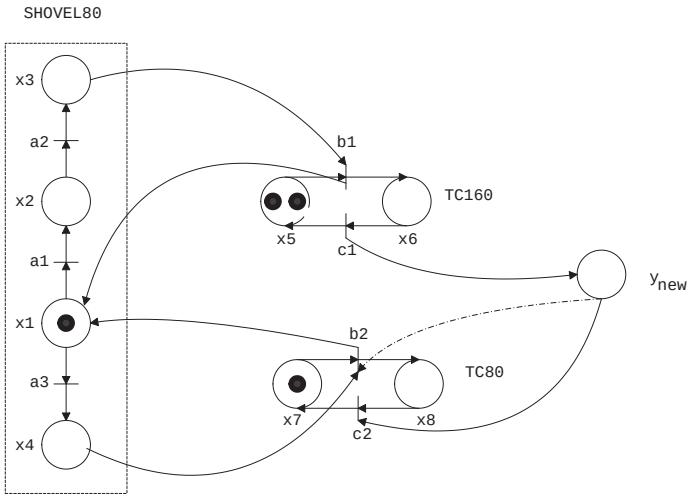

Figure 33: VDESI on Interacting VDESs

A simplified schematic of the computation of a VDESI is now provided. Consider the predicate $P_{dyn} := \#c_1 \geq \#c_2$ modeled by a single state VDES $y_{new} := \#c_1 - \#c_2 \geq 0$ in Figure 33. Events b_i and c_i, $i = 1, 2$ are assumed to be controllable and uncontrollable, respectively. To design a VDESI, we follow the three-step procedure described in the previous paragraph. By inspection of Figure 33, one notices that G^∇ is composed of event c_2 since this is the unique uncontrollable event decrementing y_{new}. Thus G^∇ is automatically loop-free. Moreover, event

c_2 only decreases state x_8 of G (namely of VDES $TC80$). To make the VDESI controllable, one can replace $\#c_2$ (solid line) with $\#b_2$ (dotted line) in y_{new}. Despite the modification of the predicate, its true behaviour is preserved because event c_2 is preceded by event b_2 whose firing leads to the unstoppable firing of event c_2. Moreover, the modified predicate provides a means to disable event b_2 when necessary.

C.4 Procedures to Compute SFBC, DSFBC and VDESI

The following procedures summarize the various steps given in Section C.1, Section C.2 and Section C.3.

$P_{static} \rightarrow$ SFBC: To extract a SFBC from a static predicate,

1. Identify the state vector X and its initial value X_o,
2. From the alphabet Σ build the displacement matrix E,
3. Identify the uncontrollable events $\Sigma_u = \{\sigma_1 \ \ldots \ \sigma_p\}$ and extract the uncontrollable displacement submatrix E_u,
4. From inspection of the VDES or E_u verify that G_u is loop-free,
5. Write down the specification as a standard static predicate

$$x \models P_{static} \ \text{ iff } \ aX \leq b,$$

   from which a and b are obtained.

6. Form vector $v^* = [v_1 \ \ldots \ v_p]$ representing the number of occurrences of $\sigma_1, \ \ldots, \ \sigma_p$,
7. Determine v^* solving the maximization problem

$$\text{maximize } aE_uv^* \text{ subject to } v^* \geq 0 \text{ and } \ X + E_uv^* \geq 0.$$

   This amounts to the following:

   - write all constraints $v^* \geq 0$ and $X + E_uv^* \geq 0$ explicitly,
   - extract the set of effective constraints,
   - choose v^* maximizing aE_uv^* while respecting the reduced set of constraints.

8. Check if

$$aX_0 + aE_uv^*(X_0) \leq b$$

   holds. If so, there exists an optimal SFBC f_σ for predicate P_{static}. If not, no SFBC exists that enforces predicate P_{static}.

9. Define the optimal SFBC f_σ so that for $\sigma \in \Sigma_c$ it enables σ (i.e., $f_\sigma(X) = 1$) if

$$\xi(X, \sigma)! \quad \text{and} \quad aX_{new} + aE_uv^*(X_{new}) \leq b$$

   holds. Also, $X_{new} = X + e_\sigma$ must satisfy $X_{new} \geq 0$.

$P_{dyn} \rightarrow P_{static}$: To convert a dynamic predicate into a static one,

1. Identify the state vector X and its initial value X_o,
2. Define the alphabet $\Sigma = \{\alpha,\ \beta,\ \ldots\}$ and the occurrence vector $v = [v_\alpha\ v_\beta\ \ldots] = [\#\alpha\ \#\beta\ \ldots]$,
3. Write down the specification as a standard dynamic predicate

$$v \models P_{dyn} \ \text{ iff } \ d - cv \geq 0,$$

from which c and d are obtained.

4. Associate to P_{dyn} a VDES memory,

$$H = \{Y, \Sigma, \eta, Y_o, Y\}$$

where $Y \geq 0$ and Y_o represent the state set and its initial value. The transition function is defined as $\eta(Y, \sigma) = Y + h_\sigma$ with the displacement vector h_σ for state set Y.

5. Enlarge the state vector X by Y such that for the enlarged VDES, denoted by $G \oplus H$, we have state vectors of the form $X \oplus Y = [x_1\ \ldots\ y_1\ \ldots]$ and $(X \oplus Y)_o = [x_{1o}\ \ldots\ y_{1o}\ \ldots]$.
6. Redefine the dynamic predicate P_{dyn} as a static one P_{static} by using the states $X \oplus Y$. Thus we have

$$d - cv \geq 0 \ \equiv \ b - a(X \oplus Y) \geq 0 \ \equiv \ a(X \oplus Y) \leq b,$$

from which a and b are extracted. Also, Y_0 is obtained by setting all event occurrences equal to zero in the state set Y.

7. Verify that in all cases $X \oplus Y \geq 0$. If not, the state set Y must be redefined and step 4 must be reinitiated.

$P_{dyn} \rightarrow$ DSFBC: To convert a dynamic predicate into DSFBC,

1. Initiate procedure $P_{dyn} \rightarrow P_{static}$,
2. Continue with procedure $P_{static} \rightarrow$ SFBC by replacing X, ξ and e_σ with the equivalent for the enlarged system i.e., $X \oplus Y$, $\xi \oplus \eta$ and $e_\sigma \oplus h_\sigma$.

(D)SFBC $\rightarrow$ VDESI: To convert a SFBC or DSFBC into a VDESI,

1. From procedure $P_{dyn} \rightarrow$ DSFBC or procedure $P_{static} \rightarrow$ SFBC extract $G^\nabla \subseteq G_u$,
2. Verify that G^∇ is loop-free,
3. Verify that each uncontrollable event in G^∇ decrements at most one state of G,
4. Rewrite the specification so that it is composed of controllable events. This can be done with the algorithm provided in [12] and [4].

D VDES solution for Case 5

This section provides the calculations for Case 5 of Section 4.2.5. We refer to G as the VDES of Figure 26 and give it the following definition $G = (X, \Sigma, \xi, X_o, X_m)$ where X and ξ represent the state set and the transition function, respectively.

The first phase is to convert the dynamic predicates developed previously into optimal dynamic state feedback controls ($P_{dyn} \rightarrow$ DSFBC). This can be done because the uncontrollable subsystem G_u is loop-free (shown later on). Let us consider the following predicates:

$$\begin{aligned} P_1 &:= \#a_1 + \#b_1 \leq \#g_1 + \#n_1 + x_{12_0} \\ P_2 &:= \#a_3 + \#b_5 \leq \#g_2 + \#n_2 + x_{19_0} \\ P_3 &:= -4(\#f_1 + \#p_1) + \#g_1 \geq 0 \\ P_4 &:= -4(\#f_2 + \#p_2) + \#g_2 \geq 0. \end{aligned}$$

We solve for dynamic predicates P_1 and P_3 only since their equivalent versions in $TC80$ (P_2 and P_4) can be derived similarly. Consider the vector of event occurrences defined as $v = [v_{a1}\ v_{b1}\ v_{g1}\ v_{n1}\ \ldots]^T$. The predicate P_1 can be put in the following form:

$$\begin{aligned} P_1 \ :=\ & d - cv \geq 0 \\ & x_{12_0} - [1\ 1\ -1\ -1\ \ldots] \begin{bmatrix} v_{a1} \\ v_{b1} \\ v_{g1} \\ v_{n1} \\ \vdots \end{bmatrix} \geq 0 \\ & [1\ -1] \left(\begin{bmatrix} x_{12_0} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 1 & \ldots \\ 1 & 1 & 0 & 0 & \ldots \end{bmatrix} \begin{bmatrix} v_{a1} \\ v_{b1} \\ v_{g1} \\ v_{n1} \\ \vdots \end{bmatrix} \right) \geq [1\ -1] \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \end{aligned}$$

such that $x \models P_1$ iff $cv \leq d$. Then we associate a memory $H = (Y, \Sigma, \eta, Y_o, Y)$ where $Y = \{y_1, y_2\}$ and $Y_0 = d = [x_{12_0}\ 0]^T$. Therefore we define $G \oplus H$, an enlarged VDES with additional states,

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_{12_0} + \#g_1 + \#n_1 \\ \#a_1 + \#b_1 \end{bmatrix},$$

with the following non-trivial initial values,

$$x_{1_0} = 1, \quad x_{5_0} = 1, \quad x_{12_0} = 2, \quad x_{19_0} = 1, \quad x_{27_0} = 3, \quad x_{29_0} = 2, \quad x_{30_0} = 1, \quad y_{1_0} = x_{12_0} = 2.$$

The above calculations consist of the first steps to convert the dynamic predicate into an enlarged VDES with a static predicate ($P_{dyn} \rightarrow P_{static}$). It remains to determine the existence of an optimal DSFBC for the static predicate. From the previous steps we have that $X \oplus Y = [x_1\ \ldots\ x_{31}\ y_1\ y_2]^T$ and $\Sigma_u = \{e_1, f_1, m_1, p_1, e_2, f_2, m_2, p_2\}$ such that the occurrence matrix for

uncontrollable events is

$$E_u = \begin{bmatrix} & e_1 & f_1 & m_1 & p_1 & e_2 & f_2 & m_2 & p_2 \\ x_{14} & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_{15} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_{16} & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ x_{18} & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ x_{21} & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ x_{22} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ x_{23} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ x_{24} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ x_{27} & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ x_{30} & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ y_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

where one notes that there exists no loop made of uncontrollable events only. Therefore, G_u is loop-free. Then P_1 is rewritten as a static predicate function of the enlarged state set $X \oplus Y$ such as

$$x \models P_1 \text{ iff } [0 \ \ldots \ 0 \ -1 \ 1] \begin{bmatrix} x_1 \\ \vdots \\ x_{31} \\ y_1 \\ y_2 \end{bmatrix} \leq 0,$$

with $a = [0 \ \ldots \ 0 \ -1 \ 1]$ and $b = 0$. The next step is to maximize $aE_u v^*$ where $v^* = [v_{e_1} \ v_{f_1} \ v_{m_1} \ v_{p_1} \ v_{e_2} \ v_{f_2} \ v_{m_2} \ v_{p_2}]$. However, since the predicate P_1 does not contain any uncontrollable events (i.e., $aE_u v^* = 0$) the static predicate P_1 is already optimal. Before a VDES implementation is derived for P_1, we treat the predicate P_3 by repeating the steps taken for P_1 (i.e., $P_{dyn} \rightarrow$ DSFBC)).

For predicate $P_3 := -4(\#f_1 + \#p_1) + \#g_1 \geq 0$, we have $v = [v_{f1} \ v_{g1} \ v_{p1} \ \ldots]^T$ and

$$P_3 \ := \ [1 \ -1] \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & \ldots \\ 4 & 0 & 4 & \ldots \end{bmatrix} \begin{bmatrix} v_{f1} \\ v_{g1} \\ v_{p1} \\ \vdots \end{bmatrix} \right) \geq [1 \ -1] \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

such that we get two additional states

$$\begin{bmatrix} y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \#g_1 \\ 4(\#f_1 + \#p_1) \end{bmatrix},$$

The state set for predicates P_1 and P_3 thus becomes $X \oplus Y = [x_1 \ \ldots \ x_{31} \ y_1 \ y_2 \ y_3 \ y_4]^T$ while Σ_u remain unchanged and E_u is augmented by two lines. For the initial values, we simply adjoin $y_{30} = y_{40} = 0$ to initial values derived for P_1 to obtain $(X \oplus Y)_0$. Therefore, the static

equivalent to P_3 is

$$x \models P_3 \text{ iff } [0 \ \ldots \ 0 \ -1 \ 1] \begin{bmatrix} x_1 \\ \vdots \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \leq 0$$

where $a = [0 \ \ldots \ 0 \ -1 \ 1]$ and $b = 0$. Unlike P_1, predicate P_3 contains uncontrollable events f_1 and p_1. Therefore, the maximization problem of $aE_u v^*$ or

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 1] \begin{bmatrix} & e_1 & f_1 & m_1 & p_1 & e_2 & f_2 & m_2 & p_2 \\ x_{14} & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_{15} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_{16} & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ x_{18} & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ x_{21} & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ x_{22} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ x_{23} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ x_{24} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ x_{27} & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ x_{30} & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ y_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ y_4 & 0 & 4 & 0 & 4 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_{e_1} \\ v_{f_1} \\ v_{m_1} \\ v_{p_1} \\ v_{e_2} \\ v_{f_2} \\ v_{m_2} \\ v_{p_2} \end{bmatrix} =$$

$$4v_{f_1} + 4v_{p_1}$$

is formulated as *Maximize* $aE_u v^* = 4v_{f_1} + 4v_{p_1}$ *subject to* $v^* \geq 0$ *and* $X + E_u v^* \geq 0$. The constraint $v^* \geq 0$ is true since we represent the event occurrences by positive integers while the development of $X + E_u v^* \geq 0$ gives

$$\begin{array}{llllllllll} x_{14} & \geq & v_{e_1} + v_{f_1}, & x_{15} & \geq & -v_{e_1} - v_{m_1}, & x_{16} & \geq & -v_{f_1} - v_{p_1}, \\ x_{18} & \geq & v_{m_1} + v_{p_1}, & x_{21} & \geq & v_{e_2} + v_{f_2}, & x_{22} & \geq & -v_{e_2} - v_{m_2}, \\ x_{23} & \geq & -v_{f_2} - v_{p_2}, & x_{24} & \geq & v_{m_2} + v_{p_2}, & x_{27} & \geq & v_{f_1} + v_{p_1}, \\ x_{30} & \geq & v_{f_1} + v_{p_1}, & y_4 & \geq & -4v_{f_1} - 4v_{p_1} & & & \end{array}$$

In the above set of constraints, the events belonging to $TC80$ and $TC160$ are unrelated. Therefore, the constraints concerned with events contained in the function to maximize $aE_u v^* = 4v_{f_1} + 4v_{p_1}$ (e.i., f_1 and p_1) reduce to

$$x_{14} \geq v_{e_1} + v_{f_1}, \quad x_{18} \geq v_{m_1} + v_{p_1}, \quad x_{30} \geq v_{f_1} + v_{p_1}, \quad x_{27} \geq v_{f_1} + v_{p_1}$$

since constraints $x_{16} \geq -v_{f_1} - v_{p_1}$ and $y_4 \geq -4v_{f_1} - 4v_{p_1}$ are satisfied by the fact that $x_j \geq 0$, $y_j \geq 0$ and $v_j \geq 0$ for all j by definition. State x_{30} takes values in $\{0, 1\}$ while states x_{14} and x_{18} can take values in $\{0, 1, 2\}$.

Therefore, the maximum value for $4(v_{f_1} + v_{p_1})$ is when $v_{e_1} = v_{m_1} = 0$ such that $v_{f_1} = x_{14}$ and $v_{p_1} = x_{18}$. Consequently, we define the occurrence vector v^* which solves the maximization

problem as

$$v^* = \begin{bmatrix} v_{e_1} \\ v_{f_1} \\ v_{m_1} \\ v_{p_1} \\ v_{e_2} \\ v_{f_2} \\ v_{m_2} \\ v_{p_2} \end{bmatrix} = \begin{bmatrix} 0 \\ x_{14} \\ 0 \\ x_{18} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}.$$

An optimal $SFBC$ for P_3, named f_σ^3, exists since $a(X \oplus Y)_0 + aE_u v^*((X \oplus Y)_0) = -y_{3_0} + y_{4_0} + 4(x_{14_0} + x_{18_0}) \leq b = 0$ when computed with a, b, $(X \oplus Y)_0$, E_u and v^* provided above. The optimal $SFBC$ f_σ^3 for $\sigma \in \Sigma_c$ is such that

$$\begin{array}{lllll} f_\sigma^3(X \oplus Y) = 1 & \text{if} & \xi \oplus \eta(X \oplus Y, \sigma)! & \text{and} & a(X \oplus Y)_{new} + aE_u v^*((X \oplus Y)_{new}) \leq b \\ & \text{if} & \xi \oplus \eta(X \oplus Y, \sigma)! & \text{and} & -y_{3,new} + y_{4,new} + 4(x_{14,new} + x_{18,new}) \leq 0 \ , \\ f_\sigma^3(X \oplus Y) = 0 & \text{otherwise} & & & \end{array}$$

Moreover, the above $SFBC$ has a balanced equivalent so that the set of predicates P_1, P_2, P_3 and P_4 can be further considered as a unique dynamic predicate by the conjunction $P_{dyn} = \bigwedge P_i$, $i = 1, 2, 3, 4$.

The last step is to derive a VDES implementation (or VDESI) of f_σ^3 (i.e., (D)SFBC $\rightarrow$ VDESI). We convert predicate P_3 into a one-dimensional VDES (represented by only one state $y_{new} = -4(\#f_1 + \#p_1) + \#g_1$ whose displacement vector is $h_\sigma = [\ldots\ h_{f_1}\ h_{g_1}\ \ldots\ h_{p_1}\ \ldots] = [\ldots\ -4\ 1\ \ldots\ -4\ \ldots]$). The connections attached to state y_{new} are represented by solid arrows in Figure 34). One notices that a weight of 4 tokens characterizes the arrows exiting state y_{new} thus meaning that 4 tokens are removed each time event f_1 or p_1 fires.

There are only two uncontrollable events $\{f_1, p_1\}$ with decremental effects on $Y = \{y_{new}\}$. Thus events f_1 and p_1 form G^∇, which is loop-free (for reasons similar to the ones given above where P_3 is represented by states $\{y_1, y_2\}$). However, the states X of G that are decreased by the firing of events in G^∇ (i.e., $\{f_1, p_1\}$) are $\{x_{14}, x_{18}, x_{30}\}$. In particular, since two states $\{x_{14}, x_{30}\}$ (resp., $\{x_{18}, x_{30}\}$) are upstream of event f_1 (resp., p_1), a VDESI does not exist. (In reference to [12], this will translate into $\Sigma_u^- = \Sigma^\nabla = \{f_1, p_1\}$, $I^\nabla = \{x_{14}, x_{18}\}$ where $\{f_1, p_1\}$ is a set of leaf events such that for $\sigma = f_1 \in \Sigma^\nabla$ we have $|\sigma^\uparrow| > 1$.) Fortunately, a designer can circumvent the problem with a simple fix. It consists of adding two additional states and events (t_1 and v_1) to all truck VDES as shown in Figure 34. The additional events can be either controllable or uncontrollable. The modification of the truck VDES decouples the repair priority from the predicate P_3 by providing only one source state to events f_1 and p_1. With the new system, all conditions for a VDESI that is equivalent to f_σ^3 hold.

The algorithm in [4] provides a controllable equivalence to the dynamic predicate P_3 by computing new initial values $y_{new,0}$ and new displacement vectors $h_{new,\sigma}$ where σ is related to uncontrollable events in P_3. As a result, we have for event f_1 that

$$y_{new,0} = 0, \quad h_{new,d_1} = -4, \quad h_{new,e_1} = 4, \quad h_{new,f_1} = 0.$$

The procedure determines the equivalent $\#f_1 = \#d_1 - \#e_1$ that is used to rewrite predicate P_3 and alter the VDESI. Consequently, the solid arrows of Figure 34 are replaced by dotted arrows

leaving e_1 and pointing to d_1 thus enabling us to stop the progression of $\#f_1$ by disabling d_1. When considering the modifications originating from uncontrollable events $\{f_1, p_1\}$, the VDESI derived from dynamic predicate P_3 is given by

$$y_{new,0} = 0, \quad [\ h_{d_1} \quad h_{e_1} \quad h_{f_1} \quad h_{k_1} \quad h_{m_1} \quad h_{p_1}\ ]_{new} = [\ -4 \quad 4 \quad 0 \quad -4 \quad 4 \quad 0\ ],$$

whose final result is shown in Figure 34. Finally, for clarity purposes only predicates P_2 (analogous to P_1) and P_3 (Figure 34) are implemented on the complete system of Case 5 to result into Figure 26.

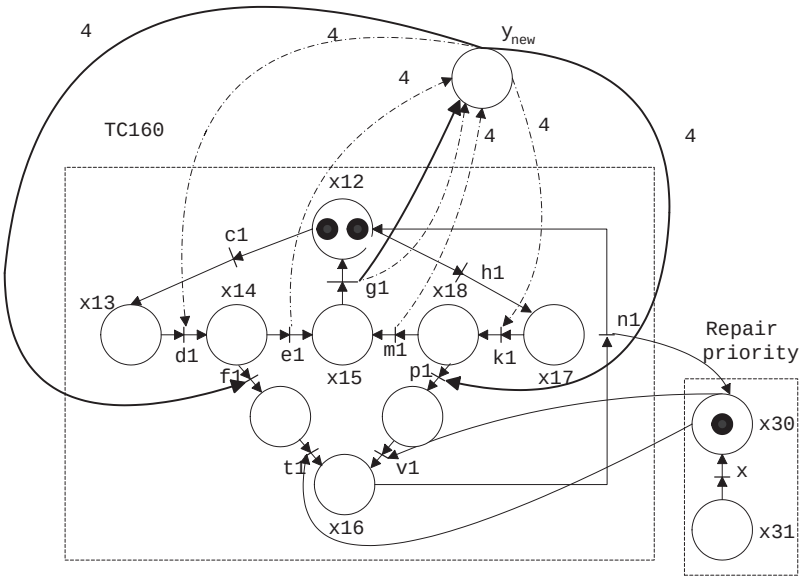

Figure 34: Fix and VDESI for P_3

References

[1] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, (126):183–235, 1994.

[2] B.A. Brandin and W.M. Wonham. Supervisory Control of Timed Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 39:329–342, 1994.

[3] P.E. Caines and Y.-J. Wei. The Hierarchical Lattice of a Finite Machine. *Systems & Control Letters*, 1996.

[4] S.-L. Chen. *Control of Discrete-event Systems of Vectors and Mixed Strucutral Type*. PhD thesis, University of Toronto, 1996.

[5] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming, North-Holland*, 8(3):231–274, 1987.

[6] Y. Li and W.M. Wonham. Control of vector discrete-event systems: I - The base model. *IEEE Trans. on Automatic Control*, 38(8):1214–1227, 1993.

[7] Y. Li and W.M. Wonham. Control of vector discrete-event systems: II - Controller synthesis. *IEEE Trans. on Automatic Control*, 39(3):512–531, 1994.

[8] P.J. Ramadge and W.M. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM Journal of Control and Optimization*, 25(1):637–659, 1987.

[9] P.J. Ramadge and W.M. Wonham. The Control of Discrete Event Systems. *Proceeding of the IEEE*, 77:81–98, 1989.

[10] K. Rudie, N. Shimkin, and S.D. O'Young. Timed Discrete-Event Systems: A Manufacturing Application. *Proceedings of the 1994 Conference on Information Science and Systems*, pages 374–381, 1994.

[11] H.A. Simon. *The Sciences of The Artificial, 2nd Edition*. The MIT Press, 1981.

[12] W.M. Wohnam. *Lecture Notes of DES1 class*. University of Toronto, available at: http://www.control.utoronto.ca/people/profs/wonham/wonham.html, 1999.

[13] H. Zhong and W.M. Wonham. On the Consistency of Hierarchical Supervion in Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 35:1125–1134, 1990.