NOTES ON CONTROL OF DISCRETE-EVENT SYSTEMS

ECE 1636F/1637S 2002-03

W.M. Wonham
Systems Control Group
Edward S. Rogers Sr. Dept. of
Electrical & Computer Engineering
University of Toronto

Revised 2002.07.01

Foreword

These notes are based on the author's lectures at the University of Toronto during the sessions 1987-88 through 2001-02, as well as at Washington University (St. Louis) in May 1988, the Indian Institute of Technology (Kanpur) in February 1989, Bilkent University (Ankara) in May 1989, the Universidade Federal de Santa Catarina (Florianopolis) in February 1993, the Centro de Investigacion y de Estudios Avanzados (Guadalajara, Mexico) in February 1997, the University of Stuttgart in May 1998, and the Banaras Hindu University in February 2000 and December 2001. The material on control theory originated with the U. of T. doctoral theses of Peter Ramadge (1983), Feng Lin (1987), Yong Li (1991), Hao Zhong (1992), Bertil Brandin (1993), Shulin Chen (1996), Kai Wong (1994), and others, together with joint publications with the author. The software package *TCT* (for untimed DES) has been developed with the help of Karen Rudie, Pablo Iglesias, Jimmy Wong, and Pok Lee; while the package *TTCT* for timed DES is under development by Christian Meder.

I am indebted to Bibiana Pang and Linda Espeut for typing and editorial assistance.

W.M.W.
2002.07.01

Contents

Introduction

The control of discrete-event systems (DES) is a research area of current vitality, stimulated by the hope of discovering general principles common to a wide range of application domains. Among the latter are manufacturing systems, traffic systems, database management systems, communication protocols, and logistic (service) systems. The contributing specialities are notably control, computer and communication science and engineering, together with industrial engineering and operations research.

With this variety of supporting disciplines, it is no surprise that the DES research area embraces a corresponding variety of problem types and modelling approaches. It is fair to say that no single control-theoretic paradigm is dominant, nor is it necessarily desirable that matters should be otherwise.

From a formal viewpoint, a DES can be thought of as a dynamic system, namely an entity equipped with a state space and a state-transition structure. In particular, a DES is discrete in time and (usually) in state space; it is asynchronous or event-driven: that is, driven by events other than, or in addition to, the tick of a clock; and it may be nondeterministic: that is, capable of transitional 'choices' by internal chance or other mechanisms not necessarily modelled by the system analyst.

The present course notes are devoted to a simple, abstract model of controlled DES that has proved to be tractable, appealing to control specialists, and expressive of a range of control-theoretic ideas. As it was introduced in the control literature by P.J. Ramadge and the author in 1982, it will be referred to as RW. RW supports the formulation of various control problems of standard types, like the synthesis of controlled dynamic invariants by state feedback, and the resolution of such problems in terms of naturally definable control-theoretic concepts and properties, like reachability, controllability and observability. RW is automaton-based, or dually language-based, depending on whether one prefers an internal structural or external behavioral description at the start. Briefly, a DES is modelled as the generator of a formal language, the control feature being that certain events (transitions) can be disabled by an external controller. The idea is to construct this controller so that the events it currently disables depend in a suitable way on the past behavior of the generating DES. In this way the DES can be made to behave optimally with respect to a variety of criteria, where 'optimal' means in 'minimally restrictive fashion'. Among the criteria are 'safety' specifications like the avoidance of prohibited regions of state space, or the observation of service priorities; and 'liveness' specifications, at least in the weak sense that distinguished target states always remain reachable.

As a practical matter, in the present state of RW software technology, DES and their specifications and controllers must be representable by finite transition structures (FTS), although there is no intrinsic restriction to FTS in the theory itself. In the FTS setup the computations for many of the control solutions entail only polynomial effort in the model's state size. However, complex controlled DES are directly modelled as product structures of simpler components; so each time a new component is adjoined (with state space size N, say) the state size of the product FTS is multiplied by N; and thus the size of the model increases exponentially with the number of components. The situation is actually worse in the case of control with partial observations: in one natural version of this problem, the computational effort is exponential (rather than polynomial) in the model size itself, for instance owing to the necessity of converting from a nondeterministic FTS to its deterministic counterpart.

While exponential (or worse) complexity is not inevitably disastrous (after all, salesmen continue to travel) it is surely a strong incentive to refine the approach. For this, two well known and universal systemic strategies can be invoked, each of them already familiar in control theory. The first is to create suitable architecture: that is, to exploit horizontal and vertical modularity, or in this context decentralized and hierarchical control. The second is to exploit internal regularity of algebraic or arithmetic structure if it happens to be present.

Thus a specialization of the RW base model to 'vector' DES (VDES) allows the exploitation of vector-additive arithmetic state structure: for instance, when dealing with sets of similar objects of which the number in a given state may be incremented or decremented by various events (machines in a factory workcell, entering a busy state or a breakdown state). For modelling and analysis in this domain Petri nets have been widely utilized, especially by computer and communications specialists, but there seems little need to adopt the arcane terminology of nets to treat what after all are just standard control problems in a specialized state framework. Of course, insights from the extensive literature on Petri nets may be exploited to advantage.

Taking a different approach, one may seek to generalize the RW model in directions of greater realism and modelling flexibility. For instance, a generalization to 'timed transition models' (TTMs) incorporates real-time features along with modelling enhancements like program variables, their transformations, and transition guards. Another, deep and rather technical, generalization in the spirit of temporal logic (due to both Peter Ramadge and John Thistle) brings in languages over sets of infinite strings and addresses issues of 'eventuality', or liveness in the long run.

While the present notes do not cover all the topics just listed, they provide an introduction to, and preparation for research in, control of DES in the style described. Two software packages are available as project tools: CTCT, for untimed DES, and TTCT, for both timed and untimed systems. These are linked to the website

http://www.control.utoronto.ca/DES

Introductory exercise: Elevator modelling and simulation

Part 1: Write a simple simulation program for a single-cage elevator in a 5-story building. The elevator should respond to both external and internal calls, in a "reasonable" way which matches your experience. No special background knowledge is assumed about DES or elevators.

Simulation runs are generated by allowing calls to occur randomly. The elevator state can be taken to be the floor at which the cage is currently located, together with the current pattern of unserviced calls and such other information that you deem relevant. Cage transitions between adjacent floors can be assumed instantaneous.

The display may be quite simple, say in the form of a table, as shown. Here the cage is at Floor 3, and there are unserviced calls to Floors 1 and 5 that have originated inside the cage, along with external up-calls at Floors 0, 1, 4 and down-calls at Floors 2, 4.

FLOOR	CAGE-LOCATION	INCALLS	EXCALLS$_{\mathrm{UP}}$	EXCALLS$_{\mathrm{DN}}$
5		x		
4			x	x
3	x			
2				x
1		x	x	
0			x	

In the presence of unserviced calls the cage location should change by one level at each stage, following which new x's in the CALLS columns may appear and old ones disappear.

Include a brief description of your approach and your control logic.

Part 2: Develop an automaton model of the system in Part 1, including a complete specification of the state set and transition structure. For instance, the state set could take the form

$$Q = F \times H \times (U0 \times \ ... \ \times U4) \times (D1 \times \ ... \ \times D5) \times (I0 \times \ ... \ \times I5)$$

where $F = \{0, 1, ..., 5\}$ is the floor set, $H = \{up, rest, down\}$ is the cage heading set, and Ui, Di, Ii represent 2-state switches ('buttons') with state sets $\{set, reset\}$ for external up-calls, external down-calls, and inside-cage-calls. Thus $ui = set \in Ui$ indicates the presence of an up-call at floor i; ui will be switched back to $reset$ when the call is serviced. The state size is $|Q| = 6 \times 3 \times 2^5 \times 2^5 \times 2^6 = 1179648$. Write $f \in F$ for the current floor, $h \in H$ for the current heading, u, d, i for the button vectors (thus $u = (u0, ..., u4) \in U0 \times \ ... \ \times U4$), and $calls = (u, d, i)$. Then

$$h_{\text{next}} = \delta_H((calls)_{\text{next}}, f, h)$$

$$f_{\text{next}} = \delta_F(h_{\text{next}}, f)$$

for suitable functions δ_H, δ_F. Define $calls_{\text{next}}$ as a suitable (in part, random) function of the current values $(calls, f, h)$, so the computation sequence is

$$(calls, f, h) \longmapsto calls_{\text{next}}$$

$$(calls_{\text{next}}, f, h) \longmapsto h_{\text{next}}$$

$$(h_{\text{next}}, f) \longmapsto f_{\text{next}}$$

Part 3: Check systematically that your simulation code from Part 1 is an implementation of your automaton model in Part 2. If the automaton model were developed first, would it be helpful in writing the code?

Part 4: Discuss possible performance specifications for your elevator (e.g. "Every call is eventually serviced."). Sketch a proof that your automaton model satisfies them.

TCT: General Information

TCT is a program for the synthesis of supervisory controls for untimed discrete-event systems. Generators and recognizers are represented as standard DES in the form of a 5-tuple

[Size, Init, Mark, Voc, Tran]

Size is the number of states (the standard state set is {0,...,Size-1}), Init is the initial state (always taken to be 0), Mark lists the marker states, Voc the vocal states, and Tran the transitions. A vocal state is a pair $[I, V]$ representing positive integer output V at state I. A transition is a triple $[I, E, J]$ representing a transition from the exit state I to the entrance state J and having event label E. E is an odd or even nonnegative integer, depending on whether the corresponding event is controllable or uncontrollable.

$$\text{exit } I \quad \circ \xrightarrow{\text{event } E} \circ \quad J \text{ entrance}$$

All DES transition structures must be deterministic: distinct transitions from the same exit state must carry distinct labels.

Synthesis procedures currently available are the following.

create prompts the user to define a new discrete-event system (DES).

selfloop augments an existing DES by adjoining selfloops at each state with event labels in a LIST provided by the user.

trim for DES1 constructs the trim (reachable and coreachable) substructure DES2.

sync forms the reachable synchronous product of DES1 and DES2 to create DES3. Not for use with vocalized DES.

meet forms the meet (reachable cartesian product) of DES1 and DES2 to create DES3. DES3 need not be coreachable. Not for use with vocalized DES.

supcon for a controlled generator DES1, forms a trim recognizer for the supremal controllable sublanguage of the marked ("legal") language generated by DES2 to create DES3. This structure provides a proper supervisor for DES1. Not for use with vocalized DES.

mutex forms DES3 from the shuffle of DES1 and DES2, by excluding state pairs specified in LIST = $\{[I1, J1], [I2, J2], ...\}$, plus all state pairs from which LIST is reachable along an uncontrollable path; then taking the reachable substructure of the result. DES3 is thus reachable and controllable, but need not be coreachable. For the corresponding control data, compute DES = **sync**(DES1, DES2), then DAT = **condat**(DES, DES3). If DES3 is trim, it provides a proper supervisor for the mutual exclusion problem; if not, a solution is SUP = **supcon**(DES,DES3). Not for use with vocalized DES.

condat returns control data DAT for the supervisor DES2 of the controlled system DES1. If DES2 represents a controllable language (with respect to DES1), as when DES2 has been previously computed with **supcon**, then **condat** will display the events that are to be disabled at each state of DES2. In general **condat** can be used to test whether a given language DES2 is controllable: just check that the disabled events tabled by **condat** are themselves controllable (have odd-numbered labels). To **show** DAT call SA. **condat** is not for use with vocalized DES.

minstate reduces DES1 to a minimal state transition structure DES2 that generates the same closed and marked languages, and the same string mapping induced by vocalization (if any). DES2 is reachable but not necessarily coreachable.

complement for a generator DES1 and a LIST of event labels, forms a generator DES2 of the marked language complementary to the marked language of DES1, with respect to the extended alphabet comprising the event labels of DES1 plus those in the auxiliary LIST. The closed behavior of DES2 is all strings over the extended alphabet. The string mapping induced by vocalization (if any) is unchanged.

project for a generator DES1 and a LIST of event labels, forms a generator DES2 of the closed and marked languages of DES1 with the LISTed events either erased or retained, according to whether the user specifies NULL or IMAGE. In decentralized control, DES2 could be an observer's local model of DES1. Not for use with vocalized DES.

convert returns DES2 corresponding to a specified mapping of event labels in DES1; unmapped labels are unchanged. Can be used with **project** to construct an arbitrary zero-memory output map having as domain the language represented by DES1. Not for use with vocalized DES.

vocalize returns a transition structure DES2 having the same closed and marked behaviors as DES1, but with specified state outputs corresponding to selected state/event input pairs.

outconsis returns a transition structure DES2 having the same closed and marked behaviors as DES1, but which is output-consistent in the sense that nonzero state outputs are unambiguously controllable or uncontrollable. A vocal state with output V in 10...99 may be split into siblings with outputs respectively $V1$ or $V0$.

hiconsis returns a transition structure DES2 having the same closed and marked behaviors as DES1, but hierarchically consistent in the sense that high-level controllable events may be disabled without side effects. This may require additional vocalization together with change in the control status of existing state outputs. This procedure incorporates and extends **outconsis**.

higen returns a transition structure DES2 over the state-output alphabet of DES1, representing the closed and marked state-output (or 'high-level') behaviors of DES1. For instance, starting with a 'low-level' vocalized model GLO, the sequence

$$\begin{gathered}\mathrm{OCGLO} = \textbf{outconsis}(\mathrm{GLO}) \\ \mathrm{HCGLO} = \textbf{hiconsis}(\mathrm{OCGLO}) \\ \mathrm{HCGHI} = \textbf{higen}(\mathrm{HCGLO})\end{gathered}$$

returns a DES pair (HCGLO, HCGHI) that is hierarchically consistent: controllable languages in HCGHI can be synthesized, via the state-output map, as controllable languages in HCGLO.

allevents returns a one-state transition structure DES2 self-looped with the events of DES1.

supnorm returns a trim transition structure DES3 which represents the supremal sublanguage of the legal language DES1, that is normal with respect to the marked behavior of the plant generator DES2 and the projection specified by NULL_EVENT_LIST (the list of unobservable events). Not for use with vocalized DES.

For supervisor synthesis, project DES2 to get PDES2, and DES3 to get PDES3, with respect to NULL_EVENT_LIST. The local supervisor is DES4 = **supcon**(PDES2,PDES3). The global supervised behavior is represented by

DES5 = **meet**(DES2,**selfloop**(DES4,NULL_EVENT_LIST))

In general DES5 may fail to be nonblocking: trim to check.

nonconflict tests whether DES1, DES2 are nonconflicting, namely whether all reachable states of the product DES are coreachable. Not for use with vocalized DES.

isomorph tests whether DES1 and DES2 are identical up to renumbering of states; if so, their state correspondence is displayed.

UTILITIES

bfs returns DES2 with state set of DES1 recoded by breadth - first search from state 0.

edit allows the user to modify an existing DES.

show SE displays an existing DES, SA a DAT (condat) table, SX a TXT (text) file. Tables can be scanned with Page keys. MAKEIT.TXT keeps a record of user files as they are generated.

file FE (resp. FA) converts a DES (resp. DAT) file to an ASCII text file (PDS resp. PDT) or Postscript file (PSS resp. PST) for printing. Some printers may only recognize a Postscript file with suffix .PS; in that case, rename the .PSS/.PST files with due care to avoid duplication.

user file directory
lists the current user subdirectory.

Chapter 1

Algebraic Preliminaries

1.1 Posets

Partially ordered sets or "posets" play a basic role in system theory; for instance, such structures are needed to support certain concepts of optimization. Posets lead to lattices, of which the lattice of subsets (of a fixed set) and the lattice of equivalence relations (on a fixed set) will be of key importance in the theory of formal languages.

Let X be a set. A *binary relation* on X is a subset of $X \times X$, the cartesian product of X with itself. Let $\leq$ be a binary relation on X. We use infix notation (as usual) and, for $x, y \in X$, write $x \leq y$ to mean that the ordered pair $(x, y) \in X \times X$ belongs to the relation $\leq$. The relation $\leq$ is a *partial order (p.o.)* on X if it is

reflexive:	$(\forall x \in X)\ x \leq x$
transitive:	$(\forall x, y, z \in X)\ x \leq y \ \&\ y \leq z \Rightarrow x \leq z$
antisymmetric:	$(\forall x, y \in X)\ x \leq y \ \&\ y \leq x \Rightarrow x = y$.

Elements $x, y \in X$ are *comparable* if either $x \leq y$ or $y \leq x$. A p.o. is a *total ordering* if every two elements of X are comparable. In a p.o. in general, it needn't be the case that two arbitrary elements $x, y \in X$ are comparable; if x, y are not comparable, we may write $x <> y$.

If $\leq$ is a partial order on X, the pair $(X, \leq)$ is a *poset* (or *partially ordered set*). If $\leq$ is understood, one speaks of "the poset X".

Examples

1. Let $X = \mathbb{R}$ (the real numbers), or $X = \mathbb{N} := \{0, 1, 2, ...\}$ (the natural numbers), or $X = \mathbb{Z} := \{..., -1, 0, +1, ...\}$ (the integers), with $\leq$ the usual ordering.

2. Let $X = \mathbb{N}^+ := \{1, 2, 3, ...\}$ and define $x \leq y$ iff $x|y$ (x divides y), namely $(\exists k \in \mathbb{N}^+)y = kx$.

3. Let $X = \mathbb{Z} \times \mathbb{Z}$. Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$ belong to X. Define $x \leq y$ iff $x_1 \leq y_1$ and $x_2 \leq y_2$. Thus (7,-2) $\leq$ (9,-1), but (7,2) <> (-10,3).

4. Let A be a set and let $X = Pwr(A)$ be the set of all subsets of A (the *power set*) of A. Thus $x, y, ... \in X$ are subsets of A. Define $x \leq y$ iff $x \subseteq y$.

5. With n fixed, let $X = S^{n \times n}$ be the set of $n \times n$ symmetric matrices with real elements. For $P, Q \in X$ define $P \leq Q$ iff the matrix $Q - P$ is positive semidefinite.

6. Let X, Y be posets. Define a relation $\leq$ on $X \times Y$ by the recipe:

$$(x_1, y_1) \leq (x_2, y_2) \text{ iff } x_1 \leq x_2 \text{ in } X \text{ and } y_1 \leq y_2 \text{ in } Y$$

Exercise 1.1.1: Verify that the definition in Example 5 really does turn X into a poset. By considering P and Q as quadratic forms, interpret the relation $P \leq Q$ in geometric terms. What is the picture if $P <> Q$?

Exercise 1.1.2: In Example 6 check that $(X \times Y, \leq)$ is actually a poset. It is the *product poset* of X and Y. $\diamond$

From now on we assume that X is a poset. Let $x, y \in X$. An element $a \in X$ is a *lower bound* for x and y if $a \leq x$ and $a \leq y$. An element $l \in X$ is a *meet* (or *greatest lower bound*) for x and y iff

$$\begin{aligned} l \leq x \quad \& \quad l \leq y \quad & \text{[i.e. } l \text{ is a lower bound for } x \text{ and } y] \\ \& \quad (\forall a \in X) a \leq x \quad \& \quad a \leq y \quad & \Rightarrow \quad a \leq l \end{aligned}$$

[i.e. l beats every other lower bound for x and y]

Exercise 1.1.3: Check that if l, l' are both meets for x and y then $l = l'$: a meet, if it exists, is unique. If it exists, the meet of x, y is denoted by $x \wedge y$. $\diamond$

Dually, an element $b \in X$ is an *upper bound* for x, y iff $x \leq b$ and $y \leq b$. An element $u \in X$ is a *join* (or *least upper bound*) of x and y if

$$x \leq u \quad \& \quad y \leq u \quad \& \quad (\forall b \in X) x \leq b \quad \& \quad y \leq b \quad \Rightarrow \quad u \leq b.$$

If the join of x and y exists it is unique, and is written $x \vee y$.

Examples

1. Let $X = Pwr(A)$ and $x, y \in X$. Then $x \wedge y = x \cap y$ (set intersection) and $x \vee y = x \cup y$ (set union). Thus the meet and join always exist.

2. Let $X = \mathbb{Z} \times \mathbb{Z}$, and let $x = (x_1, x_2)$, $y = (y_1, y_2) \in X$. Then

$$\begin{aligned} x \wedge y &= (\min(x_1, y_1), \min(x_2, y_2)), \\ x \vee y &= (\max(x_1, y_1), \max(x_2, y_2)). \end{aligned}$$

Again the meet and join always exist.

3. Let $X = S^{n \times n}$. In general $P \vee Q$ and $P \wedge Q$ do not exist.

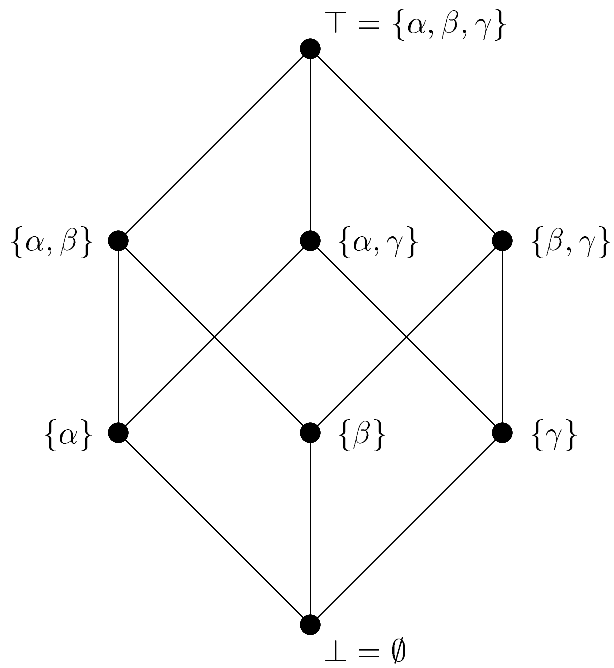
Exercise 1.1.4: Explain this situation with a 2×2 counterexample, and draw the picture.
Hint: Consider $P = 0$, $Q = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

Exercise 1.1.5: Investigate the existence of meet and join for the poset $(\mathbb{N}^+, \cdot|\cdot)$ defined earlier. $\diamond$

The following extensions of our notation are often useful. Write $x \geq y$ for $y \leq x$; $x < y$ for $x \leq y$ & $x \neq y$; $x > y$ for $x \geq y$ & $x \neq y$. Notice that, in general, the negation of $x \leq y$ is (either $x > y$ or $x <> y$). Also let $\perp$ stand for *bottom element* (if it exists): namely $\perp \in X$ and $\perp \leq x$ for all $x \in X$. Similarly let $\top$ stand for *top element*: $\top \in X$ and $\top \geq x$ for all $x \in X$. $\diamond$

Hasse Diagrams

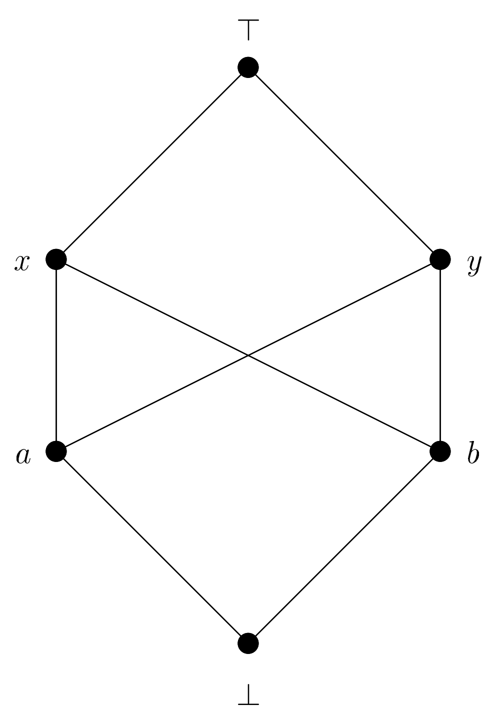
A *Hasse diagram* for the poset $(X, \leq)$ is a directed graph with nodes corresponding to elements $x \in X$ and edges to pairs (x, y) with $x < y$. Edges are drawn as rising lines and are usually displayed only for 'neighboring' x, y. For $A = \{\alpha, \beta, \gamma\}$ and $X = Pwr(A)$ the Hasse diagram is shown below.



As another example, consider

$$X = \{\top, x, y, a, b, \bot\}$$

as displayed.



Here a, b are both lower bounds for x, y, but x and y have no greatest lower bound: $x \wedge y$ doesn't exist. However, $a \wedge b$ exists and is $\perp$. Dually $a \vee b$ doesn't exist, but $x \vee y = \top$.

Exercise 1.1.6: Investigate the existence of $\perp$ and $\top$ in $(\mathbb{N}, \leq)$, $(\mathbb{N}^+, \cdot|\cdot)$, $(\mathbb{Z}, \leq)$ and $(Pwr\ A, \subseteq)$.

Exercise 1.1.7: Define the poset $X = (\mathbb{N}, \cdot|\cdot)$ according to $x \leq y$ iff $x|y$, i.e. $(\exists k \in \mathbb{N})$ $y = kx$. Thus $x|0$ $(x \in \mathbb{N})$ but *not* $0|x$ if $x \neq 0$. Show that $\perp = 1$ and $\top = 0$.

1.2 Lattices

A *lattice* is a poset L in which the meet and join of any two elements always exist; in other words the binary operations $\vee$ and $\wedge$ define functions

$$\wedge : L \times L \to L, \quad \vee : L \times L \to L.$$

It is easy to see that, if $x, y, z \in L$ and if $\star$ denotes either $\wedge$ or $\vee$ consistently throughout, then

$$\begin{aligned} x \star x &= x && (\star \text{ is } \textit{idempotent}) \\ x \star y &= y \star x && (\star \text{ is } \textit{commutative}) \\ (x \star y) \star z &= x \star (y \star z) && (\star \text{ is } \textit{associative}) \end{aligned}$$

So for any k one can write $x_1 \star x_2 \star \ldots \star x_k$, say, without ambiguity, namely the meet and join are defined for arbitrary nonempty finite subsets of elements of L.

In addition one has the easily verified relationships

$$\begin{array}{ll} x \wedge (x \vee y) = x \vee (x \wedge y) = x & \text{(traditionally called “absorption”)} \\ x \leq y \text{ iff } x \wedge y = x \text{ iff } x \vee y = y & \text{(“consistency”)} \end{array}$$

Exercise 1.2.1: Verify the above relationships.

Exercise 1.2.2: In any lattice

$$\begin{array}{ll} y \leq z \Rightarrow (x \wedge y \leq x \wedge z) \ \& \ (x \vee y \leq x \vee z) & \\ & [\wedge \text{ and } \vee \text{ are “isotone”}] \\ x \wedge (y \vee z) \geq (x \wedge y) \vee (x \wedge z) & \\ x \vee (y \wedge z) \leq (x \vee y) \wedge (x \vee z) & \\ & [\text{distributive inequalities}] \\ x \leq z \Rightarrow x \vee (y \wedge z) \leq (x \vee y) \wedge z & [\text{modular inequality}] \end{array}$$

Exercise 1.2.3: Investigate the lattices $X = Pwr(A)$ and $X = \mathbb{Z} \times \mathbb{Z}$ to see whether the distributive inequalities, or the right side of the modular inequality, can be strengthened to equality. $\diamond$

If in a given lattice, the distributive inequalities are actually always equalities, the lattice is *distributive*; if the right side of the modular inequality is actually always equality, the lattice is *modular*. Clearly every distributive lattice is modular.

Let $(L, \wedge, \vee)$, or simply L, be a lattice and let S be a nonempty, and possibly infinite, subset of L. To generalize the notion of meet (greatest lower bound) of the elements of S, define $l = \inf(S)$ to mean that l is an element of L with the properties:

$$(\forall y \in S) l \leq y \qquad \& \qquad (\forall z)((\forall y \in S) z \leq y) \Rightarrow z \leq l$$

Notice that it is not required that l belong to S. Similarly, the notion of join is generalized by defining an element $u = \sup(S)$ in dual fashion. If S is finite, then $\inf(S)$ and $\sup(S)$ reduce to the meet and join as defined above for a finite number of elements of L, and hence always exist because L is a lattice; but if S is an infinite subset, it need not be true that $\inf(S)$ or $\sup(S)$ exist. The lattice L is *complete* if, for any nonempty subset S of L, both $\inf(S)$ and $\sup(S)$ exist (as elements of L). Thus one easily verifies that $L = (Pwr(A), \cap, \cup)$ is complete, but that $L = (\mathbb{Z} \times \mathbb{Z}, \wedge, \vee))$ is not complete.

Exercise 1.2.4: Let V be a finite-dimensional linear vector space and let $X = \mathcal{S}(V)$ be the set of linear subspaces of V. For subspaces x and y of V, define $x \leq y$ iff $x \subseteq y$ (subspace inclusion). Verify that $(X, \leq)$ is a complete lattice, where $\wedge$ is subspace intersection and $\vee$ is subspace addition (i.e. vector addition extended to subspaces). Show that X is modular but not distributive.

Exercise 1.2.5: $L = (\mathbb{Q}[0, 1], \inf, \sup)$, the rational numbers in [0,1] with the usual real-analysis definitions of inf and sup, is not complete; while $L = (\mathbb{R}[0, 1], \inf, \sup)$, the real numbers in [0,1], is complete.

Exercise 1.2.6 If L and M are lattices, show that the product poset $L \times M$ is a lattice as well. It is the *product lattice* of L and M. Show that $L \times M$ is complete iff L, M are both complete. $\diamond$

Whether or not L is complete, if $\sup(L)$ (or $\top$) happens to exist then the empty subset $S = \emptyset \subseteq L$ can be brought within the scope of our definition of $\inf(S)$ by the convention

$$\inf(\emptyset) = \sup(L)$$

Similarly, if $\inf(L)$ (or $\perp$) exists then one may define

$$\sup(\emptyset) = \inf(L)$$

These odd-looking conventions are, in fact, forced by "empty set logic", as can easily be checked.

Exercise 1.2.7: Adjoin to the (incomplete) lattice $(\mathbb{Z}, \leq)$ two new symbols $-\infty$, $+\infty$ to form $\bar{\mathbb{Z}} := \mathbb{Z} \cup \{-\infty, +\infty\}$. Extend $\leq$ to $\bar{\mathbb{Z}}$ according to

$$\begin{array}{rl} x < +\infty & \text{if} \quad x \in \mathbb{Z} \cup \{-\infty\} \\ -\infty < x & \text{if} \quad x \in \mathbb{Z} \cup \{+\infty\} \end{array}$$

Show that $(\bar{\mathbb{Z}}, \leq)$ is complete and identify $\bot$, $\top$.

Exercise 1.2.8: Show that, if $\inf(S)$ exists (in L) for every subset $S \subseteq L$, then L is complete. **Hint:** Let $S^+ := \{x \in L | (\forall y \in S) x \geq y\}$. Show that $\sup(S) = \inf(S^+)$. $\diamond$

Let $L = (X, \leq)$ be a lattice and $Y \subseteq X$. We say that $M := (Y, \leq)$ is a *sublattice* of L if Y is closed under the meet and join operations of L.

Exercise 1.2.9: Referring to Exercise 1.2.5, show that $\mathbb{Q}[0, 1]$ is an incomplete sublattice of the complete lattice $\mathbb{R}[0, 1]$.

Exercise 1.2.10: For $L = (X, \leq)$ a lattice and $Y \subseteq X$ an arbitrary subset, show that there is a (unique) smallest subset $Z \subseteq X$ such that $Y \subseteq Z$ and $M = (Z, \leq)$ is a sublattice of L. M is the *sublattice of L generated by Y*. **Hint:** First show that the intersection of an arbitrary nonempty collection of sublattices of L is a sublattice.

1.3 Equivalence Relations

Let X be a nonempty set, and $E \subseteq X \times X$ a binary relation on X. E is an *equivalence relation* if

$$\begin{array}{ll} (\forall x \in X) x E x & (E \text{ is } \textit{reflexive}) \\ (\forall x, x' \in X) x E x' \Rightarrow x' E x & (E \text{ is } \textit{symmetric}) \\ (\forall x, x', x'' \in X) x E x' \ \&\ x' E x'' \Rightarrow x E x'' & (E \text{ is } \textit{transitive}) \end{array}$$

Instead of xEx' we shall often write $x \equiv x' (\mod E)$.

For $x \in X$ let $[x]$ denote the subset of elements x' that are equivalent to x:

$$[x] := \{x' \in X | x' E x\} \subseteq X$$

The subset $[x]$ is the *coset* (or *equivalence class*) of x with respect to the equivalence relation E. By reflexivity $x \in [x]$, i.e. every coset is nonempty. The following proposition states that any two cosets either coincide or are disjoint.

Proposition 1.3.1

$$(\forall x, y \in X) \quad \text{either} \quad [x] = [y] \quad \text{or} \quad [x] \cap [y] = \emptyset$$

Proof

Let $x, y \in X$ and $u \in [x] \cap [y]$, so that uEx and uEy. We claim that $[x] \subseteq [y]$. If $x' \in [x]$ then $x'Ex$. Since xEu (by symmetry) we have $x'Eu$ (by transitivity) and so (again by transitivity) $x'Ey$, namely $x' \in [y]$, proving the claim. Similarly $[y] \subseteq [x]$, and therefore $[x] = [y]$. □

Let $\mathcal{P}$ be a family of subsets of X indexed by α in some index set A:

$$\mathcal{P} = \{C_\alpha | \alpha \in A\}, \quad C_\alpha \subseteq X$$

The family $\mathcal{P}$ is a *partition* of X if

$(\forall \alpha \in A)\ C_\alpha \neq \emptyset$	(each subset C_α is nonempty)
$(\forall x \in X)(\exists \alpha \in A)\ x \in C_\alpha$	(each x belongs to some C_α)
$(\forall \alpha, \beta \in A)\ \alpha \neq \beta \Rightarrow C_\alpha \cap C_\beta = \emptyset$	(subsets with distinct indices are pairwise disjoint)

The subsets C_α are the *cells* of $\mathcal{P}$.

Thus for the equivalence relation E we have proved that the collection of distinct cosets $[x]$ (each of them indexed, say, by some representative member x) is a partition of X. Conversely for a partition $\mathcal{P}$ of X, as above, we may define an equivalence relation E on X by the recipe

$$xEy \quad \text{iff} \quad (\exists \alpha \in A) x \in C_\alpha \quad \& \quad y \in C_\alpha;$$

namely x and y are equivalent iff they belong to the same cell of $\mathcal{P}$. It is easily checked that E as just defined really is an equivalence relation on X. With this correspondence in mind we shall often speak of equivalence relations and partitions interchangeably.

Let $\mathcal{E}(X)$, or simply $\mathcal{E}$, be the class of all equivalence relations on (or partitions of) X. We shall assign a p.o. to $\mathcal{E}$ in such a way that $\mathcal{E}$ becomes a complete lattice, as follows:

$$(\forall E_1, E_2 \in \mathcal{E}) E_1 \leq E_2 \quad \text{iff} \quad (\forall x, y \in X) x E_1 y \Rightarrow x E_2 y$$

In other words $E_1 \leq E_2$ iff, whenever $x \equiv y \pmod{E_1}$ then $x \equiv y \pmod{E_2}$; that is, every coset of E_1 is a subset of some (and therefore exactly one) coset of E_2. If $E_1 \leq E_2$ one may say that E_1 *refines* E_2, or E_1 is *finer* than E_2, or E_2 is *coarser* than E_1.

Exercise 1.3.1: Verify that $\leq$ really does define a p.o. on $\mathcal{E}$; that is, $\leq$ is reflexive, transitive and antisymmetric. $\diamond$

Proposition 1.3.2

In the poset $(\mathcal{E}, \leq)$ the meet $E_1 \wedge E_2$ of elements E_1 and E_2 always exists, and is given by

$$\begin{aligned}(\forall x, x' \in X) x &\equiv x' \pmod{E_1 \wedge E_2} \\ \text{iff } x &\equiv x' \pmod{E_1} \quad \& \quad x \equiv x' \pmod{E_2}\end{aligned}$$

Proof (Outline)

Write $E := E_1 \wedge E_2$ as just defined. Then E really is an equivalence relation on X, that is $E \in \mathcal{E}$. Next, $E \leq E_1$ and $E \leq E_2$. Finally if $F \in \mathcal{E}$ and $F \leq E_1$, $F \leq E_2$ then $F \leq E$. $\Box$

Exercise 1.3.2: Supply all the details in the above proof. $\diamond$

The meet may be described by saying that $E_1 \wedge E_2$ is the coarsest partition that is finer than both E_1 and E_2. The situation is sketched in Fig. 1.1.

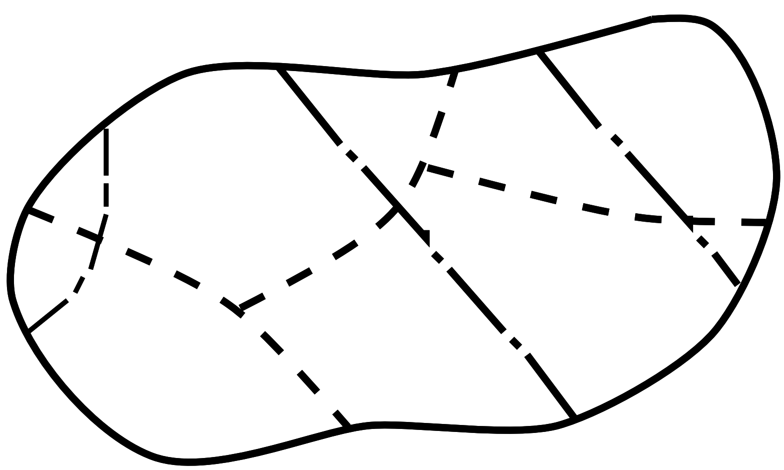


Fig. 1.1
Meet of Two Partitions

—— - —— - —— boundaries of E-cells
. . . . boundaries of F-cells
Any boundary line is a boundary of an $(E \wedge F)$-cell

The definition of join is more complicated to state than the definition of meet.

Proposition 1.3.3

In the poset $(\mathcal{E}, \leq)$ the join $E_1 \vee E_2$ of elements E_1, E_2 always exists and is given by

$$(\forall x, x' \in X) x \equiv x' (\text{mod } E_1 \vee E_2)$$

$$\text{iff } (\exists \text{ integer } k \geq 1)\ (\exists x_0, x_1, ..., x_k \in X) x_0 = x \ \&\ x_k = x'$$

$$\&\ (\forall i)\ 1 \leq i \leq k \quad \Rightarrow \quad [x_i \equiv x_{i-1} (\text{mod } E_1) \text{ or } x_i \equiv x_{i-1} (\text{mod } E_2)] \qquad \square$$

Exercise 1.3.3: Prove Proposition 1.3.3. ◇

The definition of join amounts to saying that x and x' can be chained together by a sequence of auxiliary elements $x_1, ..., x_{k-1}$, where each link in the chain represents either equivalence (mod E_1) or equivalence (mod E_2). In case $k = 1$, either $x \equiv x'$ (mod E_1) or $x \equiv x'$ (mod E_2). The join may be described by saying that $E_1 \vee E_2$ is the finest partition that is coarser than both E_1 and E_2. The situation is sketched in Fig. 1.2.

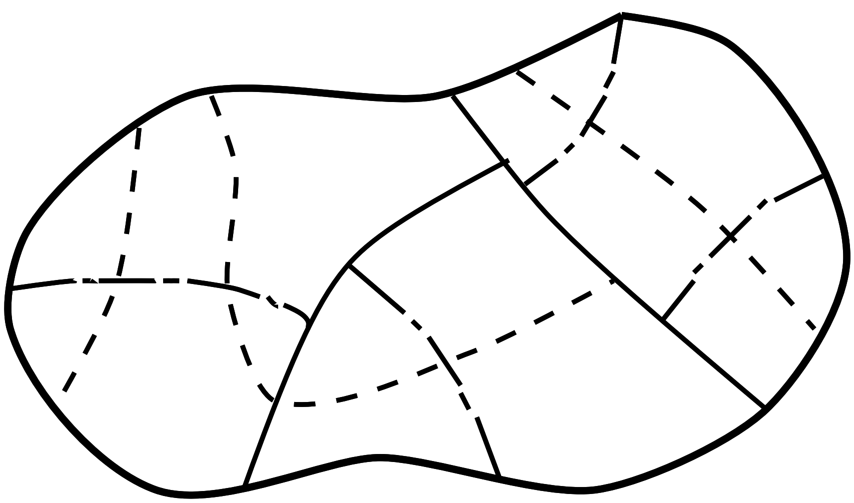


Fig. 1.2
Join of Two Partitions

—— - —— - —— boundaries of E-cells
. . . . boundaries of F-cells
———————— common boundaries of E-cells and F-cells, forming boundaries of $(E \vee F)$-cells

We have now established that $(\mathcal{E}, \leq)$ is a lattice, the *lattice of equivalence relations* (or *partitions*) on X. Finally we show that the lattice $\mathcal{E}$ is complete.

Proposition 1.3.4

Let $\mathcal{F} \subseteq \mathcal{E}$ be a nonempty collection of equivalence relations on X. Then $\inf(\mathcal{F})$ exists; in fact

$$(\forall x, x' \in X) x(\inf(\mathcal{F}))x' \text{ iff } (\forall F \in \mathcal{F}) xFx'$$

Also $\sup(\mathcal{F})$ exists; in fact

$$\begin{aligned} &(\forall x, x' \in X) x(\sup(\mathcal{F}))x' \\ &\quad \text{iff } (\exists \text{ integer } k \geq 1)\ (\exists F_1, ..., F_k \in \mathcal{F})\ (\exists x_0, ..., x_k \in X) \\ &\qquad x_0 = x \ \&\ x_k = x' \ \&\ (\forall i) 1 \leq i \leq k \Rightarrow x_i \equiv x_{i-1} (\text{mod } F_i) \end{aligned}$$

Proof (Outline)

As defined above, $\inf(\mathcal{F})$ and $\sup(\mathcal{F})$ are indeed equivalence relations on X, and have the properties required by the definitions of inf and sup given in Sect. 1.2. □

Exercise 1.3.4: Supply the details. ◇

The definition of $\inf(\mathcal{F})$ says that x and x' are equivalent with respect to each of the equivalence relations in the collection $\mathcal{F}$. The definition of $\sup(\mathcal{F})$ says that x and x' can be chained together by a finite sequence of elements, where each link of the chain represents equivalence (mod F) for some $F \in \mathcal{F}$.

In particular $\bot = \inf(\mathcal{E})$ and $\top = \sup(\mathcal{E})$ exist and are given by

$$x \equiv x' (\text{mod } \bot) \text{ iff } x = x', \quad x \equiv x' (\text{mod } \top) \text{ iff } \textit{true}.$$

Thus $\bot$ is the finest possible partition (each singleton is a cell), while $\top$ is the coarsest possible partition (there is only one cell: the whole set X).

We have now shown that the lattice of equivalence relations on X is complete.

To conclude this section we note that the elements of $\mathcal{E}(X)$ may be crudely interpreted as information structures on X. Thus if $E \in \mathcal{E}(X)$, and some element $y \in X$ is "known exactly", we may interpret the statement "$x \equiv y(\mod E)$" to mean that "x is known to within the coset that contains y". [As a metaphor, consider an ideal voltmeter that is perfectly accurate, but only reads out to a precision of 0.01 volt. If a voltage may be any real number and if, say, a reading of 1.23 volts means that the measured voltage v satisfies $1.225 \leq v < 1.235$, then the meter determines a partition of the real line into cells (intervals) of length 0.01. What is "known" about any measured voltage is just that it lies within the cell corresponding to the reading.] On this basis a given partition represents more information than a coarser one. The element $\perp \in \mathcal{E}$ stands for "perfect information" (or "zero ignorance"), while $\top$ stands for "zero information" ("complete ignorance"). If $E, F \in \mathcal{E}(X)$ then $E \wedge F$ represents "the information contributed by E and F when present together, or cooperating." [Let X be the state space of an electrical network, and let E and F represent an ideal ammeter and an ideal voltmeter.] Dually $E \vee F$ might be interpreted as "the information that E and F produce in common." [With X as before, suppose the state $x \in X$ of the network can be perturbed, either by a shove or a kick. Assume that shoves and kicks are known to define partitions S and K belonging to $\mathcal{E}(X)$: a shove can only perturb x to a state x' in the same cell of S, while a kick can only perturb x to a state x' in the same cell of K. If initially x is measured with perfect accuracy, and the network is subsequently perturbed by some arbitrary sequence of shoves and kicks, then the best available information about the final state y is that $y \equiv x(\text{mod } S \vee K)$.]

Exercise 1.3.5: In a certain Nation, regarded as a network of villages, it is always possible to make a two-way trip from any village to zero or more other villages (i.e. perhaps only the same village) by at least one of the modes: canoe, footpath, or elephant. Show that, to a Traveller restricted to these modes, the Nation is partitioned into Territories that are mutually inaccessible, but within each of which every village can be reached from any other.

Exercise 1.3.6: For $X = \{1, 2, 3\}$ present $\mathcal{E}(X)$ as a list of partitions of X and draw the Hasse diagram. Repeat for $X = \{1, 2, 3, 4\}$.

Exercise 1.3.7: Let $E, F, G \in \mathcal{E}(X)$. Investigate the validity of proposed distributive identities

$$(?) \qquad E \wedge (F \vee G) = (E \wedge F) \vee (E \wedge G)$$

and

$$(?) \qquad E \vee (F \wedge G) = (E \vee F) \wedge (E \vee G)$$

If either one is valid (i.e. holds for arbitrary E, F, G), prove it; otherwise provide a counterexample. **Hint:** Examine the first Hasse diagram of Exercise 1.3.6.

Exercise 1.3.8: Investigate whether $\mathcal{E}(X)$ is modular. That is, either prove that $\mathcal{E}(X)$ is modular, or show by a counterexample that it isn't. **Hint:** Examine the second Hasse diagram of Exercise 1.3.6.

Exercise 1.3.9: Given two elements $E, F \in \mathcal{E}(X)$, say that elements $x, x' \in X$ are *indistinguishable*, and write xIx', if either $x \equiv x' \pmod{E}$ or $x \equiv x' \pmod{F}$, or possibly both. It can be checked that the binary relation I is reflexive and symmetric but not necessarily transitive. Such a relation is a *tolerance relation*. Provide examples of tolerance relations that are not transitive. If R is any binary relation on X (i.e. $R \subseteq X \times X$), the *transitive closure* of R is the smallest (in the sense of subset inclusion in $X \times X$) transitive binary relation that contains R. Show that the transitive closure of I is an element of $\mathcal{E}(X)$ and compute this element in terms of E and F. **Hint:** First show that the transitive closure of R is given by

$$\begin{aligned} R^* := \{(x, x')|(\exists k)(\exists x_0, ..., x_k)x &= x_0, \\ x' &= x_k, \quad (\forall i = 1, ..., k)x_{i-1}Rx_i\} \end{aligned}$$

Exercise 1.3.10: For $|X| = n$ let $p_n := |\mathcal{E}(X)|$ $(n = 1, 2, ...)$ and let $p_0 := 1$. Show that

$$p_{n+1} = \sum_{k=0}^{n} \binom{n}{k} p_k$$

Deduce that (i) $p_{2n} \geq n!2^n$, and (ii) $p_n \leq n!$. Write a program to compute p_n $(1 \leq n \leq 10)$, and calculate p_n approximately for $n = 20, 30, ...$ (e.g. $p_{50} \simeq 1.86 \times 10^{47}$).

1.4 Equivalence Kernel and Canonical Factorization

Let X and Y be sets. In this section we shall write π, ρ etc. for elements of $\mathcal{E}(X)$. It will be convenient to think of the elements of $\mathcal{E}(X)$ as partitions of X. Let $\pi \in \mathcal{E}(X)$ and let $\bar{X}$ denote the set of (distinct) cells of π. Alternatively $\bar{X}$ can be taken to be a set of labels or indices for these cells. Often $\bar{X}$ is written X/π, read "X mod π". Denote by P_π the surjective function mapping any x to its coset (or coset label):

$$P_\pi : X \to X/\pi : x \mapsto [x]$$

where $[x]$ is the coset of x mod π. We call P_π the *canonical projection* associated with π.

Let $f : X \to Y$ be a function with domain X and codomain Y. With f we often associate the induced function $f_* : Pwr(X) \to Pwr(Y)$ taking subsets of X into their f-images in Y:

$$f_*(A) := \{f(x) \mid x \in A\}, \qquad A \in Pwr(X), \qquad \text{i.e. } A \subseteq X$$

Thus $f_*(A) \subseteq Y$. In particular $f_*(X)$ is the *image of f*, denoted $\mathrm{Im}(f)$. Usually we do not distinguish notationally between f and f_*, simply writing $f(A)$ for $f_*(A)$.

With $f : X \to Y$ we also associate the *inverse image function* $f^{-1} : Pwr(Y) \to Pwr(X)$ according to

$$f^{-1}(B) := \{x \in X \mid f(x) \in B\}, \qquad B \in Pwr(Y) \quad (\text{i.e. } B \subseteq Y)$$

Thus $f^{-1}(B) \subseteq X$. Notice that f^{-1} is always well-defined, even if f does not have an 'inverse' in the ordinary sense. If f happens to be bijective, then the ordinary inverse of f (strictly, its induced map on subsets of Y) coincides with the inverse image function f^{-1}.

Next, with $f : X \to Y$ associate an equivalence relation in $\mathcal{E}(X)$ called the *equivalence kernel* of f and denoted by ker f, as follows:

$$(\forall x, x' \in X) x \equiv x' (\text{mod } \ker f) \text{ iff } f(x) = f(x')$$

For instance, ker $P_\pi = \pi$. The cosets of ker f are just the subsets of X on which f assumes its distinct values in Y, and are sometimes called the *fibers* of f. [For illustration consider the function $f : \mathbb{S}^2 \to \mathbb{R}$ that maps points on the earth's surface into their elevation above sea level in m. Then $f^{-1}(\{100\})$ is the coset of ker f consisting of those points whose elevation is 100 m.] The partition corresponding to ker f consists of the subfamily of (distinct) nonempty subsets of X formed from the family of subsets

$$\{f^{-1}(\{y\}) | y \in Y\}$$

Intuitively, f "throws away more or less information according to whether its kernel is coarser or finer." [Consider the ideal voltmeter as a function $f : \mathbb{R} \to \mathbb{R}$. Compare the kernels of two voltmeters, respectively reading out to the nearest 0.01 v. and 0.001 v.]

Exercise 1.4.1: Let $f : X \to Y$. Show that

(i) $(\forall B \subseteq Y) f(f^{-1}(B)) \subseteq B$;

equality holds for B iff $B \subseteq \text{Im}(f)$; and equality holds for all $B \subseteq Y$ iff Im $f = Y$, i.e. f is surjective.

(ii) $(\forall A \subseteq X) f^{-1}(f(A)) \supseteq A$;

equality holds for A iff ker $f \leq \{A, X - A\}$; and equality holds for all A iff ker $f = \perp$, i.e. f is injective.

(iii) Let $\{A_\alpha\}$, $\{B_\beta\}$ be arbitrary families of subsets of X, Y respectively. Then

$$f(\cup_\alpha A_\alpha) = \cup_\alpha f(A_\alpha), \quad f(\cap_\alpha A_\alpha) \subseteq \cap_\alpha f(A_\alpha)$$

$$f^{-1}(\cup_\beta B_\beta) = \cup_\beta f^{-1}(B_\beta), \quad f^{-1}(\cap_\beta B_\beta) = \cap_\beta f^{-1}(B_\beta)$$

Illustrate strict inclusion in the second of these distribution relations. ◇

Let $f : X \to Y$, and let $P_f : X \to X/\ker f$ be the canonical projection. Then there is a unique function $g : X/\ker f \to Y$ such that $f = g \circ P_f$. ($\circ$ denotes composition of functions). Indeed if $z \in X/\ker\, f$ and $z = P_f(x)$ for some $x \in X$, define $g(z) := f(x)$. This definition is unambiguous because if, also, $z = P_f(x')$ then $P_f(x') = P_f(x)$ and therefore $f(x') = f(x)$ since $\ker P_f = \ker f$. In this way we obtain the *canonical factorization* of f through its equivalence kernel. The situation is displayed in the commutative diagram below.

$$\begin{tikzcd} X \arrow[r, "P_f"] \arrow[dr, "f"'] & X/\ker f \arrow[d, dotted, "g"] \\ & Y \end{tikzcd}$$

In a canonical factorization $f = g \circ P_f$ the left factor g is always injective. For suppose $z, z' \in X/\ker f$ and $g(z) = g(z')$. If $z = P_f(x)$ and $z' = P_f(x')$ then

$$f(x) = g \circ P_f(x) = g(z) = g(z') = g \circ P_f(x') = f(x'),$$

namely $x \equiv x' \pmod{\ker f}$, so $x \equiv x' \pmod{\ker P_f}$, i.e.

$$z = P_f(x) = P_f(x') = z'$$

as claimed.

If $\pi \in \mathcal{E}(X)$ we may write $P_\pi : X \to X/\pi$ for the canonical projection; thus $\ker P_\pi = \pi$.

The following propositions offer variations on the foregoing theme; their proofs will be left as an exercise.

Proposition 1.4.1

Suppose $f : X \to Y$ and let $\rho \in \mathcal{E}(X)$ with $\ker f \geq \rho$. There exists a unique map $g : X/\rho \to Y$ such that $f = g \circ P_\rho$.

$$\begin{tikzcd} X \arrow[r, "P_\rho"] \arrow[dr, "f"'] & X/\rho \arrow[d, dotted, "g"] \\ & Y \end{tikzcd} \qquad \ker f \geq \rho$$

Proposition 1.4.2

Suppose $f : X \to Y$ and $g : X \to Z$ and let $\ker f \geq \ker g$. Then there exists a map $h : Z \to Y$ such that $f = h \circ g$. Furthermore h is uniquely defined on the image $g(X)$ of X in Z; that is, the restriction $h|g(X)$ is unique.

$$\begin{tikzcd} & X \arrow[dl, "g"'] \arrow[dr, "f"] & \\ Z \arrow[rr, dotted, "h"] & & Y \end{tikzcd} \qquad \ker f \geq \ker g$$

In this situation f is said to *factor through* g. Intuitively, "g preserves enough information to calculate f (via h)."

Proposition 1.4.3

If $\pi, \rho \in \mathcal{E}(X)$ and $\pi \leq \rho$, there is a unique function $f : X/\pi \to X/\rho$ such that $P_\rho = f \circ P_\pi$.

$$\begin{tikzcd} & X \arrow[dl, "P_\pi"'] \arrow[dr, "P_\rho"] & \\ X/\pi \arrow[rr, dotted, "f"] & & X/\rho \end{tikzcd} \qquad \rho \geq \pi$$

Exercise 1.4.2: Prove Propositions 1.4.1 - 1.4.3. In Proposition 1.4.3, by drawing a picture of X interpret $\ker f \in \mathcal{E}(X/\pi)$ in terms of the partitions π, ρ of X.

Exercise 1.4.3: (Parable of the Stones): To illustrate the benefits of functional factorization, consider a large field of $N >> 1$ stones. Each stone is suited for making exactly one of k possible tools: hammer, arrowhead, ... One fine day, the toolmaker organizes his tribe to sort all the stones in the field into k piles, one for each tool. Under suitable probabilistic assumptions, and taking into account search and sort times, prove that the 'sorted architecture' is k times more efficient than no architecture, given that the toolmaker will make $N(\to \infty)$ tools over his lifetime.

Example 1.4.1: Congruences of a dynamic system

A *dynamic system* on a set X is a map $\alpha : X \to X$ with the following interpretation. The elements $x \in X$ are the system 'states', and α is the 'state transition function'. Select $x_0 \in X$ as 'initial' state and let the system evolve successively through states $x_1 = \alpha(x_0), x_2 = \alpha(x_1), ...$ Write α^k for the k-fold composition of α (with $\alpha^0 =$ identity, $\alpha^1 = \alpha$). The sequence $\{\alpha^k(x_0)|k \in \mathbb{N}\} \in X^{\mathbb{N}}$ is the *path* of (X, α) with initial state x_0.

Let $\pi \in \mathcal{E}(X)$ with canonical projection $P_\pi : X \to \bar{X} := X/\pi$. We say that π is a *congruence* for α if there exists a map $\bar{\alpha} : \bar{X} \to \bar{X}$ such that $\bar{\alpha} \circ P_\pi = P_\pi \circ \alpha$, namely the following diagram commutes.

$$\begin{tikzcd}
X \arrow[r, "\alpha"] \arrow[d, "P_\pi"'] & X \arrow[d, "P_\pi"] \\
\bar{X} \arrow[r, dashed, "\bar{\alpha}"] & \bar{X}
\end{tikzcd}$$

Proposition 1.4.3

π is a congruence for α iff

$$\ker P_\pi \leq \ker(P_\pi \circ \alpha),$$

namely

$$(\forall x, x')(x, x') \in \pi \Rightarrow (\alpha(x), \alpha(x')) \in \pi$$

Proof

Immediate from Prop. 1.4.1, with the identifications $(Y, f, \rho, g) = (\bar{X}, P_\pi \circ \alpha, \pi, \bar{\alpha})$. □

If π is a congruence for α, $\bar{\alpha}$ is the map *induced* by α on $\bar{X}$. The condition says that α 'respects' the partition corresponding to π, in the sense that cells are mapped under α consistently into cells. Thus the dynamic system $(\bar{X}, \bar{\alpha})$ can be regarded as a consistent aggregated (or 'high-level' or 'lumped') model of (X, α).

Exercise 1.4.4: With (X, α) fixed, let $\mathcal{C}(X) \subseteq \mathcal{E}(X)$ be the set of all congruences for α. Show that $\mathcal{C}(X)$ is a complete sublattice of $\mathcal{E}(X)$ that contains the elements $\top$, $\bot$ of $\mathcal{E}(X)$.

Exercise 1.4.5: Let $X = X_1 \times X_2$ and write $x = (x_1, x_2)$. Suppose

$$\alpha : X \times X : (x_1, x_2) \mapsto (\alpha_1(x_1, x_2), \alpha_2(x_2)),$$

i.e. $\alpha : x \mapsto \hat{x}$ is coordinatized as

$$\hat{x}_1 = \alpha_1(x_1, x_2), \quad \hat{x}_2 = \alpha_2(x_2)$$

for suitable maps α_1, α_2. Let $(x, x') \in \pi$ iff $x_2 = x'_2$. Show that π is a congruence for α and find a coordinatization (i.e. a concrete representation) of $(\bar{X}, \bar{\alpha})$.

Exercise 1.4.6: Consider a clock with hour, minute and second hands. Identify the corresponding congruences. **Hint:** For the second hand alone, consider

$$\alpha : \mathbb{R} \to \mathbb{R} : t \mapsto t + 60$$

where t is real-valued time in units of seconds. Let $P : \mathbb{R} \to \mathbb{R}/60\mathbb{N} = \bar{\mathbb{R}}$, and identify $\bar{\alpha} : \bar{\mathbb{R}} \to \bar{\mathbb{R}}$ so the appropriate diagram commutes. Generalize the picture to include the hour and minute hands.

Exercise 1.4.7: Let A be a set of maps $\alpha : X \to X$. Show that if $\pi \in \mathcal{C}(X)$ is a congruence for every $\alpha \in A$ then π is a congruence for every composition $\alpha_k \circ \alpha_{k-1} \circ ... \circ \alpha_1$ of maps in A.

Exercise 1.4.8: Given a finite set X with $|X| = n$, construct $\alpha : X \to X$ 'randomly', for instance by assigning each evaluation $\alpha(x)$ independently by a uniform probability distribution over X. Discuss the probability that (X, α) admits at least one nontrivial congruence (i.e. other than $\perp$, $\top$), especially as $n \to \infty$. **Hint:** Show that (X, α) admits a nontrivial congruence if α is not bijective. For fixed n calculate the fractional number of such α and recall Stirling's formula.

Exercise 1.4.9: Let X be a finite-dimensional linear vector space (over $\mathbb{R}$, say) and assume $\alpha : X \to X$ is linear. Describe $\mathcal{C}(X)$ (as in Exercise 1.4.4).

Exercise 1.4.10: With X a set let $\alpha : X \to Pwr(X)$. The pair (X, α) is a *nondeterministic dynamic system.* A *path* of (X, α) with initial state x_0 is either an infinite sequence $\{x_k | k \geq 0\}$ with $x_k \in \alpha(x_{k-1})$ $(k \geq 1)$ or a finite sequence $\{x_k | 0 \leq k \leq n\}$ with $x_k \in \alpha(x_{k-1})$ $(1 \leq k \leq n)$ and $\alpha(x_n) = \emptyset$. Let $\pi \in \mathcal{E}(X)$, $P_\pi : X \to \bar{X} := X/\pi$. If $S \subseteq X$ write $P_\pi S = \{P_\pi x | x \in S\} \subseteq \bar{X}$. We say that π is a *quasi-congruence* for (X, α) if

$$(\forall x, x' \in X) P_\pi x = P_\pi x' \Rightarrow P_\pi \circ \alpha(x) = P_\pi \circ \alpha(x')$$

Show that $\perp \in \mathcal{E}(X)$ is a quasi-congruence and that the quasi-congruences for (X, α) form a complete upper semilattice under the join operation of $\mathcal{E}(X)$, namely if $\pi_\lambda \in \mathcal{E}(X)$ is a quasi-congruence for each $\lambda \in \Lambda$ (some index set) then so is

$$\sup\{\pi_\lambda | \lambda \in \Lambda\}$$

Find an example showing that if π_1, π_2 are quasi-congruences, $\pi_1 \wedge \pi_2$ need not be. Show also that $\top \in \mathcal{E}(X)$ need not be a quasi-congruence.

Exercise 1.4.11: Let $\mathcal{L}$ be a complete lattice. A function $\psi : \mathcal{L} \to \mathcal{L}$ is *monotone* if, for all $\omega, \omega' \in \mathcal{L}$, $\omega \leq \omega'$ implies $\psi(\omega) \leq \psi(\omega')$. An element $\omega \in \mathcal{L}$ is a *fixpoint* of ψ if $\omega = \psi(\omega)$.

Show that if ψ is monotone it has at least one fixpoint. **Hint:** Let

$$\Omega^{\uparrow} := \{\omega \in \mathcal{L} | \omega \leq \psi(\omega)\}$$

Note that $\perp \in \Omega^{\uparrow}$. Define $\omega^* := \sup \Omega^{\uparrow}$, and show that ω^* is a fixpoint of ψ.

Now let Ω be the set of all fixpoints of ψ. Clearly $\Omega \subseteq \Omega^{\uparrow}$.

Show that ω^* is the greatest fixpoint of ψ, namely $\omega^* \in \Omega$ and thus $\omega^* = \sup \Omega$.

Dually let

$$\Omega^{\downarrow} := \{\omega \in \mathcal{L} | \omega \geq \psi(\omega)\}$$

Note that $\top \in \Omega^{\downarrow}$, and define $\omega_* := \inf \Omega^{\downarrow}$. Show that ω_* is the least fixpoint of ψ, namely $\omega_* \in \Omega$ and thus $\omega_* = \inf \Omega$.

Suppose that $\omega_1, \omega_2 \in \Omega$. Is it true that $\omega_1 \vee \omega_2$, $\omega_1 \wedge \omega_2 \in \Omega$? In each case either prove the positive statement or provide a counterexample.

For a sequence $\{\omega_n \in \mathcal{L} | n \in \mathbb{N}\}$ write $\omega_n\!\downarrow$ if the ω_n are nonincreasing, i.e. $\omega_n \geq \omega_{n+1}$ for all n. The function ψ is *downward continuous* if, whenever $\omega_n \in \mathcal{L}$ and $\omega_n\!\downarrow$, then

$$\psi(\inf_n \omega_n) = \inf_n \psi(\omega_n)$$

Show that if ψ is downward continuous, it is necessarily monotone.

In case ψ is downward continuous show that

$$\omega^* = \inf\{\psi^k(\top) | k \in \mathbb{N}\}$$

Exercise 1.4.12: Let $\mathcal{L}$ be a complete lattice. For a sequence $\{\omega_n \in \mathcal{L} \mid n \in \mathbb{N}\}$ write $\omega_n\!\uparrow$ if the ω_n are nondecreasing, i.e. $\omega_n \leq \omega_{n+1}$ for all n. Say the function $\psi : \mathcal{L} \to \mathcal{L}$ is *upward continuous* if, whenever $\omega_n \in \mathcal{L}$ and $\omega_n\!\uparrow$, then

$$\psi(\sup_n \omega_n) = \sup_n \psi(\omega_n)$$

Dualize the results of Exercise 1.4.9 for upward continuity.

Exercise 1.4.13: Let $\psi : \mathcal{E}(\mathbb{N}) \to \mathcal{E}(\mathbb{N})$. Show that ψ monotone and upward (resp. downward) continuous does not imply ψ downward (resp. upward) continuous.

Exercise 1.4.14: (Yingcong Guan) In $\mathcal{E}(\mathbb{N})$ define

$$\begin{aligned} \omega_0 &= \top \\ \omega_1 &= \{(0,1),(2,3,...)\} \\ \omega_n &= \{(0,1),(2),...,(n),(n+1,n+2,...)\}, \qquad n \geq 2 \end{aligned}$$

Write $\|\pi\|$ for the number of cells of $\pi \in \mathcal{E}(\mathbb{N})$. Define $\psi : \mathcal{E}(\mathbb{N}) \to \mathcal{E}(\mathbb{N})$ according to

$$\begin{aligned} \psi(\omega_n) &= \omega_{n+1}, \qquad n \geq 0 \\ \psi(\pi) &= \begin{cases} \omega_n & \text{if} \quad \pi \neq \omega_n \text{ and } \|\pi\| = n+1 \qquad (n \geq 1) \\ \perp & \text{if} \quad \|\pi\| = \infty \end{cases} \end{aligned}$$

Show that ψ is monotone, investigate the fixpoint(s) of ψ, and calculate $\psi(\inf_n \omega_n)$ and $\inf_n \psi(\omega_n)$.

Exercise 1.4.15: Let $\alpha : X \to X$ be an arbitrary function. Recall that an element $\omega \in \mathcal{E}(X)$ is a *congruence* for α if, whenever $x \equiv x' \pmod{\omega}$ then $\alpha(x) \equiv \alpha(x') \pmod{\omega}$, i.e. α 'respects' the partition induced by ω. Define $\omega \cdot \alpha \in \mathcal{E}(X)$ according to:

$$\omega \cdot \alpha = \ker(P_\omega \circ \alpha)$$

Thus $x \equiv x' \pmod{\omega \cdot \alpha}$ iff $\alpha(x) \equiv \alpha(x') \pmod{\omega}$. Then ω is a congruence for α iff $\omega \leq \omega \cdot \alpha$.

(i) Let $\gamma : X \to Y$ be a function from X to a set Y. We define the *observer* for the triple (X, α, γ) to be the equivalence relation

$$\omega_o := \sup\{\omega \in \mathcal{E}(X) | \omega \leq (\ker \gamma) \wedge (\omega \cdot \alpha)\}$$

Define $\psi : \mathcal{E}(X) \to \mathcal{E}(X)$ according to $\psi(\omega) := (\ker \gamma) \wedge (\omega \cdot \alpha)$. Show that ψ is monotone and that ω_o is the greatest fixpoint of ψ. Thus the observer is the coarsest congruence for α that is finer than ker γ. Prove that ψ is downward continuous, and that consequently

$$\omega_o = \inf\{(\ker \gamma) \cdot \alpha^{i-1} | i = 1, 2, ...\}$$

where α^j is the j-fold composition of α with itself.

(ii) We interpret the observer as follows. Consider a dynamic system with state $x \in X$ and discrete-time *transition function* α. When started in state $x(0)$ at $t = 0$, the system generates the sequence of states, or *trajectory*, given by

$$x(t+1) = \alpha[x(t)], \qquad t = 0, 1, 2, ...$$

With γ an *output map*, the system generates the corresponding sequence of outputs, or *observations*

$$y(t) = \gamma[x(t)], \qquad t = 0, 1, 2, ...$$

Thus ω_o represents the information available about the initial state $x(0)$ after observation of the entire output sequence $\{y(t)|t = 0, 1, 2, ...\}$: the observations cannot resolve the uncertainty about $x(0)$ more finely than ω_o. On this basis the pair (γ, α) is said to be *observable* if $\omega_o = \perp$, namely the observation sequence determines the initial state uniquely.

(iii) Calculate ω_o when α, γ are defined as follows:

$$X = \{1, 2, 3, 4, 5, 6, 7\}$$

x	1	2	3	4	5	6	7
$\alpha(x)$	2	3	4	5	1	7	5
$\gamma(x)$	r	r	r	g	g	r	g

Here r, g stand for 'red' and 'green'. What conclusions can be drawn about the system from your result for ω_o?

(iv) Calculate ω_o for the following. Let $X = \mathbb{N} \times \mathbb{N}$, and for $(i, j) \in X$,

$$\begin{aligned} \alpha(i,j) &= (i+1, j) \\ \gamma(i,j) &= \begin{cases} 0, & i \le j \\ 1, & i > j \end{cases} \end{aligned}$$

Sketch the cells of ω_o in the $\mathbb{N}^2$ plane.

(v) Very often in practice one is more interested in the current state $x(t)$, as inferred from the observation set $O(t) := \{y(0), y(1), ..., y(t)\}$, than in the initial state as estimated in the long run. Define

$$\begin{aligned} \omega_t &= \ker\gamma \wedge \ker(\gamma \circ \alpha) \wedge ... \wedge \ker(\gamma \circ \alpha^t) \\ \bar{X}_t &= X/\omega_t \end{aligned}$$

After observations $O(t)$, the uncertainty in $x(0)$ is represented by an element of $\bar{X}_t$ (i.e. cell of ω_t in X), inducing a map $P_t : Y^{t+1} \to \bar{X}_t$. The corresponding uncertainty in $x(t)$ is just the subset

$$\{\alpha^t(x')|x' \in P_t(O(t))\} \subseteq X$$

Discuss how to compute this subset in recursive fashion, 'on-line'.

(vi) Even if the system is observable, it need not be true that much useful information can be gained about $x(t)$ from $O(t)$, even for large t. From this viewpoint consider the 'chaotic' system defined as follows.

$$X := \{x \in \mathbb{R} | 0 \leq x < 1\}$$

For $x \in \mathbb{R}$, $x \geq 0$, write $x = \mathrm{integ}(x) + \mathrm{fract}(x)$, where $\mathrm{integ}(x) \in \mathbb{N}$ and $0 \leq \mathrm{fract}(x) < 1$; and define $\alpha : X \to X$, $\gamma : X \to \mathbb{N}$ according to

$$\alpha(x) := \mathrm{fract}(10x), \qquad \gamma(x) := \mathrm{integ}(10x)$$

1.5[1] Sprays and Covers

In this section we show how the canonical factorization of a function through its equivalence kernel can be generalized to the setting of relations.

Let X and Y be sets. A *relation* from X to Y is a subset $f \subseteq X \times Y$. If $(x, y) \in f$, y is said to be *f-related* to x. A relation f from X to Y is a *spray* if

$$(\forall x \in X)(\exists y \in Y)(x, y) \in f,$$

namely each element of X has at least one f-related element of Y. In this case we write $f : X \dashv Y$. A spray from X to Y is *functional* if for every $x \in X$ there is just one $y \in Y$ such that $(x, y) \in f$; in that case the notation $f : X \dashv Y$ is thought of as 'sharpened' to the conventional notation $f : X \to Y$. Let $f : X \dashv Y$. The *cover* of f, denoted by cov (f), is the family of subsets of X given by

$$\{f_y | y \in Y\}, \quad f_y := \{x \in X | (x, y) \in f\}$$

The subsets $f_y \subseteq X$ are the *cells* of cov (f). It is clear that for a spray f there holds

$$X = \cup\{f_y | y \in Y\},$$

namely cov (f) is a 'covering' of X in the usual sense. A spray $f : X \dashv Y$ is *surjective* if

$$(\forall y \in Y)(\exists x \in X)(x, y) \in f,$$

namely for all $y \in Y$, $f_y \neq \emptyset$. If $f : X \dashv Y$ and $g : Y \dashv Z$ then their *composition* $g \circ f : X \dashv Z$ is the spray defined by

$$g \circ f := \{(x, z) \in X \times Z | (\exists y \in Y)(x, y) \in f \quad \& \quad (y, z) \in g\};$$

[1]Not needed in the sequel.

namely for all $z \in Z$

$$(g \circ f)_z = \{x \in X | (\exists y \in Y) x \in f_y \quad \& \quad y \in g_z\}$$

The following proposition describes how a given spray may be factored through another.

Proposition 1.5.1

Let $f : X {-\!\!\dashv}\, Z$ and $g : X {-\!\!\dashv}\, Y$ be sprays. There exists a spray $h : Y {-\!\!\dashv}\, Z$ such that $h \circ g = f$ iff

(1) $(\forall y \in Y)(\exists z \in Z) g_y \subseteq f_z$, and

(2) $(\forall z \in Z)(\exists Y' \subseteq Y) f_z = \cup\{g_y | y \in Y'\}$

Thus f factors through g iff every member of cov (g) is contained in some member of cov (f), and every member of cov (f) is a union of certain members of cov (g). The situation is displayed in the following commutative diagram of sprays.

$$\begin{tikzcd}
X \arrow[r, "g", -|] \arrow[dr, "f"', -|] & Y \arrow[d, "h", dotted, -|] \\
& Z
\end{tikzcd}$$

Proof:

(If) Suppose (1) and (2) hold. Define $h : Y {-\!\!\dashv}\, Z$ by the rule: given $y \in Y$, let $(y, z) \in h$ iff $g_y \subseteq f_z$; at least one such z exists by (1), and so h is a spray. To verify that $f = h \circ g$, suppose first that $(x, z) \in f$, namely $x \in f_z$. By (2) there is $W \subseteq Y$ such that

$$f_z = \cup\{g_w | w \in W\}$$

Since $x \in f_z$ there is $w \in W$ with $x \in g_w$. Since $g_w \subseteq f_z$, there follows $(w, z) \in h$. Since $(x, w) \in g$ this gives $(x, z) \in h \circ g$ as required. For the reverse statement let $(x, z) \in h \circ g$, so for some $y \in Y$ there hold $x \in g_y$ and $y \in h_z$. By definition of h, $g_y \subseteq f_z$, hence $x \in f_z$, namely $(x, z) \in f$ as required.

(Only if) Let $f \circ g$. To verify (1) let $y \in Y$ and choose $z \in Z$ with $y \in h_z$; since h is a spray such z exists. There follows $x \in f_z$ for all $x \in g_y$, namely $g_y \subseteq f_z$. As for (2), let

$z \in Z$, and $x \in f_z$. By definition of $h \circ g$, there is $y \in Y$ such that $x \in g_y$ and $y \in h_z$. If $x' \in g_y$ then $(x', z) \in h \circ g = f$, namely $x' \in f_z$ and therefore $g_y \subseteq f_z$. For each $x \in f_z$ let $y(x) \in Y$ be chosen as just described; then

$$f_z = \cup\{g_{y(x)} | x \in f_z\},$$

a covering of f_z as required. $\square$

Note that the role of condition (1) is just to guarantee that the relation h is actually a spray, namely for all y there is some z with $(y, z) \in h$.

For given sprays f and g as in Proposition 1.5.1, it is not true in general that the spray h is uniquely determined by the requirement that $h \circ g = f$, even if g is surjective. For example, let $X = \{x\}$, $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2\}$, with $f = \{(x, z_1), (x, z_2)\}$ and $g = \{(x, y_1), (x, y_2)\}$. Then $f = h \circ g = h' \circ g$, where $h = \{(y_1, z_1), (y_2, z_2)\}$ and $h' = \{(y_1, z_2), (y_2, z_1)\}$. However, as subsets of $Y \times Z$, the sprays $h : Y \multimap Z$ are partially ordered by inclusion, and with respect to this ordering the family of sprays h that satisfy the condition $f = h \circ g$ has (if the family is nonempty) a unique maximal element, say h^*, defined as in the proof of Proposition 1.5.1:

$$(y, z) \in h^* \text{ iff } g_y \subseteq f_z$$

We now examine coverings by subsets in their own right. A *cover* of X is a family $\mathcal{C}$ of subsets of X (the *cells* of $\mathcal{C}$) such that

$$X = \cup\{X' | X' \in \mathcal{C}\}$$

Let $\mathbf{C}(X)$ be the collection of all covers of X. If $\mathcal{C} \in \mathbf{C}(X)$, let Y be any set in bijective correspondence with $\mathcal{C}$, with $y(X')$ the element of Y that indexes the cell $X' \in \mathcal{C}$. Clearly $\mathcal{C}$ defines a spray $f : X \multimap Y$, with $f_{y(X')} = X'$ and cov $(f) = \mathcal{C}$. In case no cell of $\mathcal{C}$ is empty, the spray f is surjective. If $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}(X)$ write $\mathcal{C}_1 \preceq \mathcal{C}_2$ to denote that the following two conditions hold:

(1) $(\forall X' \in \mathcal{C}_1)(\exists X'' \in \mathcal{C}_2) X' \subseteq X''$

(2) $(\forall X'' \in \mathcal{C}_2)(\exists \mathcal{C}_1' \subseteq \mathcal{C}_1) X'' = \cup\{X' | X' \in \mathcal{C}_1'\}$

Thus every cell of $\mathcal{C}_1$ is contained in some cell of $\mathcal{C}_2$, and every cell of $\mathcal{C}_2$ is a union of some subfamily of cells of $\mathcal{C}_1$. Clearly $\preceq$ is reflexive and transitive on $\mathbf{C}(X)$. If $\mathcal{C}_1 \preceq \mathcal{C}_2$ and $\mathcal{C}_2 \preceq \mathcal{C}_1$, then we shall say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent*, and write $\mathcal{C}_1 \equiv \mathcal{C}_2$. It is easily checked that

$$\bar{\mathbf{C}}(X) := \mathbf{C}(X)/\equiv$$

is a poset with respect to the induced partial order, which we denote again by $\preceq$. However, it is not true that the poset $\bar{\mathbf{C}}(X)$ is a lattice.

To illustrate the notation, we remark that Proposition 1.5.1 can be stated in the form: there exists h such that $f = h \circ g$ iff cov $(f) \succeq$ cov (g).

1.6[2] Dynamic Invariance

Let $\alpha : Q \multimap Q$ and $f : Q \multimap X$ be sprays. We think of α as representing a (nondeterministic) 'dynamic action' on Q, and f as representing a lumping or aggregation of Q into (possibly overlapping) subsets indexed by X. It is thus of interest to know whether an induced dynamic action $\bar{\alpha} : X \multimap X$ can be defined on X, such that $\bar{\alpha} \circ f = f \circ \alpha$, namely the following diagram commutes:

$$\begin{tikzcd} Q \arrow[r, "\alpha", -|] \arrow[d, "f"', -|] & Q \arrow[d, "f", -|] \\ X \arrow[r, "\bar{\alpha}", dashed, -|] & X \end{tikzcd}$$

By Proposition 1.5.1, such $\bar{\alpha}$ exists iff

$$\text{cov } (f \circ \alpha) \succeq \text{cov } (f)$$

In this case we shall say that cov (f) is *α-invariant*. We also know that $\bar{\alpha}$ can be defined in such a way that it is 'maximal', namely to each $x \in X$, $\bar{\alpha}$ assigns the largest possible subset $A_x \subseteq X$ with the property that $(x, x') \in \bar{\alpha}$ iff $x' \in A_x$.

As an aid to intuition it is convenient to paraphrase the statement $q \in (f \circ \alpha)_x$ by saying that there is a 'route' from q to x via α and f. The first defining condition for cov $(f) \preceq$ cov $(f \circ \alpha)$ implies that for every $x \in X$ there is $x' \in X$ such that, for every $q \in f_x$ there is a route to x' via α and f. In the special but important case where $\alpha : Q \to Q$ is functional, this means precisely that $\alpha(f_x) \subseteq f_{x'}$. The second defining condition states that, for every $x' \in X$, the subset of all q that are routed to x' (via α and f) can be indexed by some subset $X' \subseteq X$, in the sense that

$$(f \circ \alpha)_{x'} = \cup \{ f_x | x \in X' \}$$

Intuitively, every transition $q \xrightarrow{\alpha} q'$ in the detailed dynamic model $\alpha : Q \multimap Q$ is represented by some transition $x \xrightarrow{\bar{\alpha}} x'$ in the aggregated model $\bar{\alpha} : X \multimap X'$. In the absence of this condition, certain transitions might be left unrepresented because their exit state q belongs to

[2]Not needed in the sequel.

no cell of cov (f) with the property that all of its elements $\tilde{q}$ are α-related to some particular cell of cov (f).

The following describes a tracking property of the aggregated model that holds when the detailed dynamic action α is functional.

Proposition 1.6.1

Suppose $\alpha : Q \to Q$ is functional, $\bar{\alpha} \circ f = f \circ \alpha$, and let $(q, q') \in \alpha$. If $q \in f_x$ then

$$q' \in \cap\{f_{x'} | (x, x') \in \bar{\alpha}\}$$

Proof:

Let $(x, x') \in \bar{\alpha}$. Since $(q, x) \in f$ we have

$$(q, x') \in \bar{\alpha} \circ f = f \circ \alpha$$

so for some q'', $(q, q'') \in \alpha$ and $(q'', x') \in f$. Since α is functional, $q'' = q'$ and so $(q', x') \in f$, i.e. $q' \in f_{x'}$. $\square$

It should be emphasized that even if α is functional, in general $\bar{\alpha}$ will not be. Nevertheless, because the spray diagram commutes, the aggregated model does tend to restrict the spread of uncertainty about the location of the entrance state q' under a transition $q \overset{\alpha}{\longmapsto} q'$.

The considerations of this section are all naturally extendible to the case of several transition sprays. In fact let $\mathcal{S}$ be a family of state transition sprays $s : Q \multimap Q$ that is closed under composition, namely $s' \circ s \in \mathcal{S}$ if $s, s' \in \mathcal{S}$ (so that $\mathcal{S}$ is a semigroup under composition). Assume that $f : Q \multimap X$ has the property that cov (f) is s-invariant for each $s \in \mathcal{S}$. Then to each s there corresponds an induced spray $\bar{s} : X \multimap X$ with the property $\bar{s} \circ f = f \circ s$. The operation sending s to $\bar{s}$ will be made well-defined by requiring that $\bar{s}$ be maximal in the sense described above. Let

$$\bar{\mathcal{S}} := \{\bar{s} | s \in \mathcal{S}\}$$

Since

$$(\bar{s} \circ \overline{s'}) \circ f = \bar{s} \circ (\overline{s'} \circ f) = \bar{s} \circ (f \circ s') = (\bar{s} \circ f) \circ s' = (f \circ s) \circ s' = f \circ (s \circ s'),$$

and since $\bar{s} \circ \overline{s'}$ can be shown to be maximal for $s \circ s'$, the operation $s \mapsto \bar{s}$ determines a morphism $\mathcal{S} \to \bar{\mathcal{S}}$ of semigroups.

Exercise 1.6.1: Can Exercise 1.4.11 be generalized to a (nondeterministic) transition relation and output relation? Investigate.

1.7 Notes and References

Most of the material in this chapter is standard. For Sects. 1.1-1.4 see especially Mac Lane & Birkhoff [1993], and Davey & Priestley [1990]. Sects. 1.5-1.6 originate here, but are not used in the sequel.

Chapter 2

Linguistic Preliminaries

2.1 Languages

Let Σ be a finite set of distinct symbols $\sigma, \tau, \ldots$. We refer to Σ as an *alphabet*. Let Σ^+ denote the set of all finite symbol sequences, of the form $\sigma_1\sigma_2...\sigma_k$ where $k \geq 1$ is arbitrary and the $\sigma_i \in \Sigma$. It is convenient also to bring in the empty sequence (sequence with no symbols), denoted by the new symbol ϵ, where $\epsilon \notin \Sigma$. We then write

$$\Sigma^* := \{\epsilon\} \cup \Sigma^+$$

An element of Σ^* is a *string* or *word* over the alphabet Σ; ϵ is the *empty string*.

Next we define the operation of *catenation* of strings:

$$\mathrm{cat} : \Sigma^* \times \Sigma^* \to \Sigma^*$$

according to

$$\mathrm{cat}(\epsilon, s) = \mathrm{cat}(s, \epsilon) = s, \qquad s \in \Sigma^*$$

$$\mathrm{cat}(s, t) = st, \qquad s, t \in \Sigma^+$$

Thus ϵ is the unit element of catenation. Evidently $\mathrm{cat}(\cdot, \cdot)$ is associative, for clearly

$$\mathrm{cat}(\mathrm{cat}(s, t), u) = \mathrm{cat}(s, \mathrm{cat}(t, u))$$

if $s, t, u \in \Sigma^+$, and the other possibilities are easily checked.

With catenation as the 'product' operation, the foregoing relationships turn Σ^* into a *multiplicative monoid* (or *multiplicative semigroup with identity*).

Notice that a symbol sequence like

$$\sigma_1\sigma_2\epsilon\sigma_3\epsilon\epsilon\sigma_4 \qquad (\sigma_i \in \Sigma)$$

is not (syntactically) an element of Σ^*. It will be taken as a convenient abbreviation for

$$\mathrm{cat}(\mathrm{cat}(\mathrm{cat}(\mathrm{cat}(\mathrm{cat}(\sigma_1\sigma_2, \epsilon), \sigma_3), \epsilon), \epsilon), \sigma_4)$$

As such it evaluates, of course, to the Σ^*-element $\sigma_1\sigma_2\sigma_3\sigma_4$. Also, brackets may sometimes be inserted in a string for clarity of exposition.

The *length* $|s|$ of a string $s \in \Sigma^*$ is defined according to

$$|\epsilon| = 0; \quad |s| = k, \quad \text{if} \quad s = \sigma_1 ... \sigma_k \in \Sigma^+$$

Thus $|\mathrm{cat}(s,t)| = |s| + |t|$.

A *language* over Σ is any subset of Σ^*, i.e. an element of the power set $Pwr(\Sigma^*)$; thus the definition includes both the empty language $\emptyset$, and Σ^* itself. Note the distinction between $\emptyset$ (the language with no strings) and ϵ (the string with no symbols). For instance the language $\{\epsilon\}$ is nonempty, but contains only the empty string.

2.2 Nerode Equivalence and Right Congruence

Let $L \subseteq \Sigma^*$ be an arbitrary language. We would like to construct, if possible, a decision procedure to test any given string $s \in \Sigma^*$ for membership in L. We might visualize a machine into which s is fed as input and which emits a beep just in case $s \in L$. To this end we first construct a partition of Σ^* that is finer than the partition $\{L, \Sigma^* - L\}$ and which has a certain invariance property with respect to L.

The *Nerode equivalence relation on* Σ^* *with respect to* L (or $\mod L$) is defined as follows. For $s, t \in \Sigma^*$,

$$s \equiv_L t \quad \text{or} \quad s \equiv t \pmod L$$

iff

$$(\forall u \in \Sigma^*) su \in L \quad \text{iff} \quad tu \in L$$

In other words $s \equiv_L t$ iff s and t can be continued in exactly the same ways (if at all) to form a word of L.

We write $\|L\|$ for the *index* (cardinality of the set of equivalence classes) of the Nerode equivalence relation $\equiv_L$. Since Σ^* is countable, $\|L\|$ is at most countable infinity (cardinality of the integers). If $\|L\| < \infty$, the language L is said to be *regular*.

Let $R \in \mathcal{E}(\Sigma^*)$ be an equivalence relation on Σ^*. R is a *right congruence* on Σ^* if

$$(\forall s, t, u \in \Sigma^*) sRt \quad \Rightarrow \quad (su)R(tu)$$

In other words, R is a right congruence iff the cells of R are 'respected' by the operation of right catenation. As elements of $\mathcal{E}(\Sigma^*)$ the right congruences on Σ^* inherit the partial order $\leq$ on $\mathcal{E}(\Sigma^*)$.

Proposition 2.2.1

Nerode equivalence is a right congruence.

Proof

Let $s \equiv t \pmod{L}$ and $u \in \Sigma^*$. It must be shown that $su \equiv tu \pmod{L}$. Let $v \in \Sigma^*$ and $(su)v \in L$. Then $s(uv) \in L$, so $t(uv) \in L$, i.e. $(tu)v \in L$. Similarly $(tu)v \in L$ implies $(su)v \in L$, hence $su \equiv tu \pmod{L}$ as claimed. $\square$

Proposition 2.2.2

Nerode equivalence is finer than the partition $\{L, \Sigma^* - L\}$.

Proof

Let $s \equiv t \pmod{L}$. Then $s \in L$ iff $s\epsilon \in L$, iff $t\epsilon \in L$, iff $t \in L$. $\square$

Proposition 2.2.3

Let R be a right congruence on Σ^* such that $R \leq \{L, \Sigma^* - L\}$. Then

$$R \leq \equiv_L$$

Proof

Let sRt. We show that $s \equiv t \pmod{L}$. Let $su \in L$. Now $sRt \Rightarrow (su)R(tu)$. Since $R \leq \{L, \Sigma^* - L\}$ and $su \in L$, it follows that $tu \in L$. Similarly if $tu \in L$ then $su \in L$. $\square$

We summarize Propositions 2.2.1 - 2.2.3 as

Theorem 2.2.1

Let $L \subseteq \Sigma^*$. The Nerode equivalence $\pmod{L}$ is the coarsest right congruence on Σ^* that is finer than $\{L, \Sigma^* - L\}$. $\square$

Exercise 2.2.1: Let $E, F \in \mathcal{E}(\Sigma^*)$ with $E \leq F$. Let

$$\mathcal{R} = \{R \mid R \text{ is a right-congruence on } \Sigma^*, \text{ with } E \leq R \leq F\}$$

Assuming that $\mathcal{R} \neq \emptyset$, show that $\mathcal{R}$ is a complete sublattice of $\mathcal{E}(\Sigma^*)$. (Recall that a *sublattice* L of a lattice M is a subset of M that is a lattice under the operations of meet and join inherited from M.)

Exercise 2.2.2: In the notation of Ex. 2.2.1, assume that $E, F \in \mathcal{R}$. Consider the statement

$$(\forall G) G \in \mathcal{R} \Rightarrow (\exists H) H \in \mathcal{R} \quad \& \quad G \wedge H = E \quad \& \quad G \vee H = F$$

Either prove it's always true or show by counterexample that it can be false. $\diamond$

For $s \in \Sigma^*$ we say $t \in \Sigma^*$ is a *prefix* of s, and write $t \leq s$, if $s = tu$ for some $u \in \Sigma^*$. Thus $\epsilon \leq s$ and $s \leq s$ for all $s \in \Sigma^*$. If $L \subseteq \Sigma^*$ the (*prefix*) *closure* of L is the language $\bar{L}$ consisting of all prefixes of strings of L:

$$\bar{L} = \{t \in \Sigma^* | t \leq s \text{ for some } s \in L\}$$

Clearly $L \subseteq \bar{L}$. If $L = \emptyset$ then $\bar{L} = \emptyset$; if $L \neq \emptyset$ then $\epsilon \in \bar{L}$. For a string s we write $\bar{s}$ instead of $\overline{\{s\}}$ for its set of prefixes. A language L is *closed* if $L = \bar{L}$.

Exercise 2.2.3: If $A \subseteq \Sigma^*$ is closed and $B \subseteq \Sigma^*$ is arbitrary, show that $A - B\Sigma^*$ is closed.

The closure of a language L is often relevant to control problems because it embodies the 'evolutionary history' of words in L. Notice that if $s, t \in \Sigma^* - \bar{L}$ then

$$(\forall w \in \Sigma^*) sw \notin L \quad \& \quad tw \notin L$$

so that $s \equiv t \pmod{L}$. In other words the subset $\Sigma^* - \bar{L}$ of Σ^* is, if nonempty, a single Nerode cell, which we call the *dump cell*; a string that enters the dump cell can never exit from it. On the other hand if $s \in \bar{L}$ and $s \equiv t \pmod{L}$ then $sw \in L$ for some $w \in \Sigma^*$ and so $tw \in L$, i.e. $t \in \bar{L}$. If $s \in \bar{L} - L$ and $s \equiv t \pmod{L}$ then $s = s\epsilon \notin L$, so $t = t\epsilon \notin L$ but (as just proved) $t \in \bar{L}$. These remarks are summarized in

Proposition 2.2.4

Nerode equivalence $\equiv_L$ refines the partition

$$\{L, \bar{L} - L, \Sigma^* - \bar{L}\}$$

of Σ^*. $\square$

2.3 Canonical Recognizers

The fact that Nerode equivalence is invariant under right catenation allows us to construct abstractly an automaton that tracks the Nerode cells which a string in Σ^* visits as it evolves symbol-by-symbol. Because Nerode equivalence is as coarse as can be for this purpose, the corresponding automaton is said to be *canonical*.

Thus let $L \subseteq \Sigma^*$ and write

$$X := \Sigma^* / \equiv_L$$

with

$$P_L : \Sigma^* \to X : s \mapsto [s]$$

the canonical projection. Write

$$\text{cat} : \Sigma^* \times \Sigma \to \Sigma^* : (s, \sigma) \mapsto s\sigma$$

for catenation,

$$id_\Sigma : \Sigma \to \Sigma : \sigma \mapsto \sigma$$

for the identity on Σ, and

$$P_L \times id_\Sigma : \Sigma^* \times \Sigma \to X \times \Sigma : (s, \sigma) \mapsto ([s], \sigma)$$

Proposition 2.3.1

There exists a unique map

$$\xi : X \times \Sigma \to X$$

such that

$$\xi \circ (P_L \times id_\Sigma) = P_L \circ \text{cat},$$

namely the following diagram commutes.

$$\begin{tikzcd}
\Sigma^* \times \Sigma \arrow[r, "\text{cat}"] \arrow[d, "P_L \times id_\Sigma"'] & \Sigma^* \arrow[d, "P_L"] \\
X \times \Sigma \arrow[r, dashed, "\xi"] & X
\end{tikzcd}$$

Proof

By Prop. 1.4.2, for the existence of ξ it is enough to check that

$$\ker(P_L \times id_\Sigma) \leq \ker(P_L \circ \mathrm{cat})$$

Uniqueness will follow by the fact that $P_L \times id_\Sigma$ is surjective. Let

$$((s,\sigma),(s',\sigma')) \in \ker(P_L \times id_\Sigma)$$

namely

$$(P_L \times id_\Sigma)(s,\sigma) = (P_L \times id_\Sigma)(s',\sigma')$$

or

$$([s],\sigma) = ([s'],\sigma'),$$

that is,

$$s \equiv_L s', \qquad \sigma = \sigma'$$

Since $\equiv_L$ is a right congruence,

$$s\sigma \equiv_L s'\sigma'$$

namely

$$[s\sigma] = [s'\sigma']$$

or

$$P_L(\mathrm{cat}(s,\sigma)) = P_L(\mathrm{cat}(s',\sigma'))$$

so

$$(P_L \circ \mathrm{cat})(s,\sigma) = (P_L \circ \mathrm{cat})(s',\sigma')$$

or finally

$$((s,\sigma),(s',\sigma')) \in \ker(P_L \circ \mathrm{cat})$$

as required. □

The elements $x \in X$ are the *states* of L; X is the *state set* of L; ξ is the *transition function* of L; and the triple (X, Σ, ξ) is the *transition structure* of L. It is convenient to extend ξ to a map

$$\hat{\xi} : X \times \Sigma^* \to X$$

as follows. Define

$$\begin{aligned}
\hat{\xi}(x,\epsilon) &= x, \quad x \in X \\
\hat{\xi}(x,\sigma) &= \xi(x,\sigma) \quad x \in X,\ \sigma \in \Sigma \\
\hat{\xi}(x,s\sigma) &= \xi(\hat{\xi}(x,s),\sigma) \quad x \in X,\ s \in \Sigma^*,\ \sigma \in \Sigma
\end{aligned}$$

It is easily checked (say, by induction on length of strings) that $\hat{\xi}$ is well defined. From now on we omit the $\hat{}$ and write ξ in place of $\hat{\xi}$.

If $x = [t]$ then by definition of ξ, $\xi(x,\sigma) = [t\sigma]$. Assuming inductively that $\xi(x,s) = [ts]$ we have

$$\begin{aligned}
\xi(x,s\sigma) &= \xi(\xi(x,s),\sigma) && \text{by definition of } \xi \text{ (i.e. } \hat{\xi}) \\
&= \xi([ts],\sigma) && \text{by the inductive assumption} \\
&= [ts\sigma] && \text{by definition of } \xi
\end{aligned}$$

so that $\xi(x,u) = [tu]$ for all $u \in \Sigma^*$. From this we get the composition property: for all $x \in X$ and $s, u \in \Sigma^*$,

$$\begin{aligned}
\xi(x,su) &= [tsu] \\
&= \xi(x',u),\ x' = [ts] \\
&= \xi(\xi(x,s),u)
\end{aligned}$$

We distinguish an element x_o and a subset X_m of X as follows. Let

$$x_o = [\epsilon], \quad X_m = \{[s] | s \in L\}$$

The state x_o is the *initial state* of L, and if $x \in X_m$, x is a *marker state* of L. We have by definition

$$\xi(x_o,s) = [\epsilon s] = [s], \quad s \in \Sigma^*$$

In particular, if $s \in L$ then

$$\xi(x_o,s) = [s] \in X_m$$

Thus one can think of X_m as the subset of states of L that 'mark' precisely the strings of L: imagine that a beep sounds just when such a state is reached. These definitions are displayed in the diagram below.

$$\begin{tikzcd}
\epsilon \in \Sigma^* \arrow[d, mapsto] \arrow[d, shift left=3, "P_L"] & L \arrow[l] \arrow[d, "P_L"] \\
x_0 \in X & X_m \arrow[l]
\end{tikzcd}$$

Here the horizontal arrows are the natural subset injections.

Finally the *dump state* of L is the (single) state corresponding to the dump cell $\Sigma^* - \bar{L}$, when the latter is nonempty.

In general, then, a language L can be visualized as shown in Fig. 2.1. The shaded divisions demarcate the cosets of $\equiv_L$. The marker states are the cosets contained in L. The initial state (coset) x_o belongs to L if the empty string $\epsilon \in L$; otherwise (provided $L \neq \emptyset$) x_o belongs to $\bar{L}$. In case $L = \emptyset$ then $\bar{L} = \emptyset$, so

$$X = \{\Sigma^*\}, \quad x_o = \Sigma^*, \quad X_m = \emptyset.$$

On the other hand if $L = \Sigma^*$ then $\bar{L} = \Sigma^*$ and

$$X = \{\Sigma^*\}, \quad x_o = \Sigma^*, \quad X_m = \{\Sigma^*\}$$

In general if L is closed then

$$X_m = X - \{\Sigma^* - L\}$$

namely all states except the dump state are marked.

Fig. 2.2 displays alternative 'high-level' transition graphs for L, showing in a general way how transitions may occur in the two cases where (a) the initial state is not a marker state (namely the empty string ϵ does not belong to L), and (b) the initial state is a marker state. If L were nonempty and closed then in the corresponding graph (b) the right-hand state (identified with $\bar{L} - L$) and its associated transitions would be deleted.

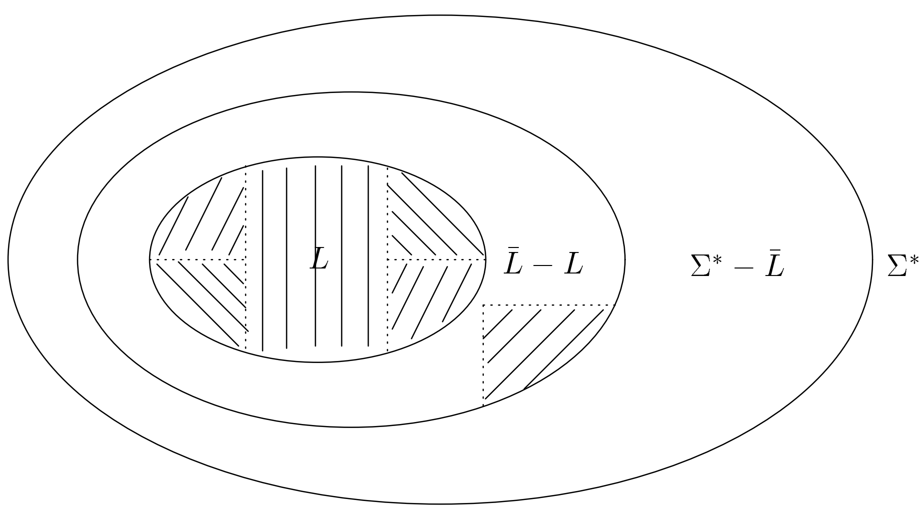


Fig. 2.1
Subset diagram for a typical language $L \subseteq \Sigma^*$
$\Sigma^* = L \dot{\cup} (\bar{L} - L) \dot{\cup} (\Sigma^* - \bar{L})$

(a) $\epsilon \notin L$

(b) $\epsilon \in L$

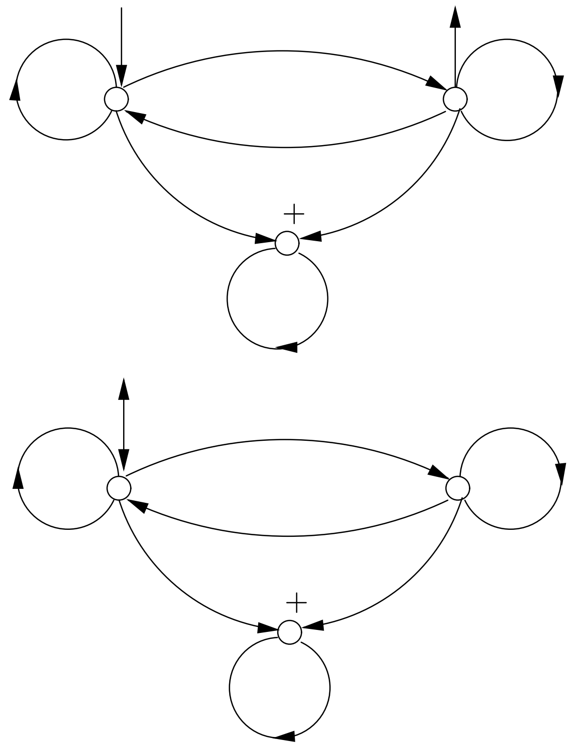


Fig. 2.2
'High-level' state transition graphs for a language L
Case (a): $\epsilon \in \bar{L} - L$; $x_o \notin X_m$
Case (b): $\epsilon \in L$; $x_o \in X_m$

The 5-tuple

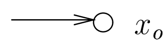
$$\mathbf{R} = (X, \Sigma, \xi, x_o, X_m)$$

will be called a *canonical recognizer* for L. While its existence has now been established abstractly there is, of course, no implication in general that $\mathbf{R}$ can actually be implemented by some constructive procedure. Naturally this issue is fundamental in the applications.

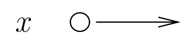
In general, a recognizer for L (see below, Sect. 2.4) will be called *canonical* if its state set X is in bijective correspondence with the cosets of $\equiv_L$. Subject to this requirement, X may be chosen according to convenience, for example as a subset of the integers. In another common representation, X is identified with the nodes of a directed graph $\mathcal{G}$ whose edges are labelled with symbols $\sigma \in \Sigma$; namely (x, σ, x') is a labelled edge

$$x \circ \xrightarrow{\ \sigma\ } \circ\, x'$$

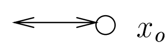
of $\mathcal{G}$, if and only if $\xi(x, \sigma) = x'$. Such a graph $\mathcal{G}$ is a *state transition graph* for $\mathbf{R}$ (or L). In $\mathcal{G}$ we attach to x_o an entering arrow, as in



and to state $x \in X_m$ an exiting arrow, as in



If x_o happens also to be a marker state we may attach a double arrow, as in

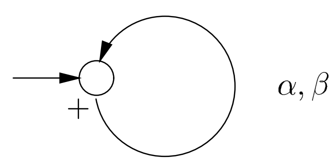


The dump state will be labelled $+$.

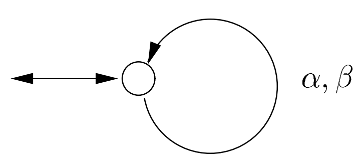
Examples

Let $\Sigma = \{\alpha, \beta\}$.

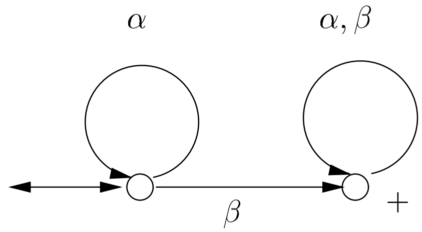
1. $L = \emptyset$



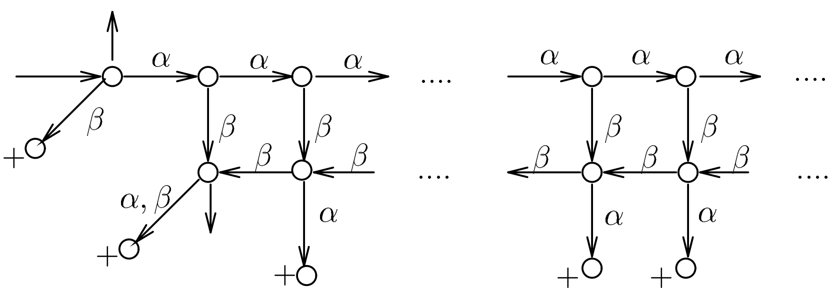
2. $L = \Sigma^*$



3. $L = \{\alpha^n | n = 0, 1, 2, ...\}$

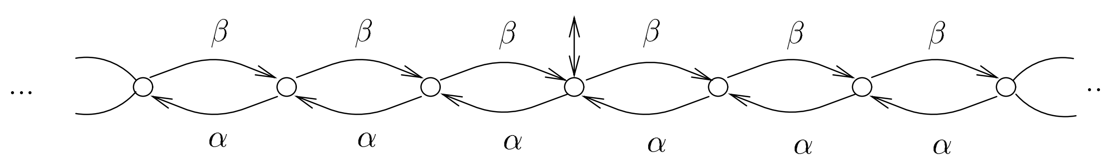


4. $L = \{\alpha^n \beta^n | n = 0, 1, 2, ...\}$.



In the above transition graph, the nodes labelled $+$ should be merged to a single 'dump' node self-looped with $\{\alpha, \beta\}$.

5. $L = \{s | \#\alpha(s) = \#\beta(s)\}$, where $0 \leq \#\sigma(s) =$ number of σ's in the string s.



We conclude this section with a useful extension of the concept of canonical recognizer, obtained by starting with an arbitrary partition θ of Σ^* in place of the binary partition $\{L, \Sigma^* - L\}$. Let T be a new alphabet, not necessarily disjoint from Σ, with $|T| = \|\theta\|$ (here we allow the possibility that T is countably infinite); and label the cells of θ in bijective correspondence with T. Thus $T \cong \Sigma^*/\theta$. There is then a well-defined map that we shall denote again by θ, taking strings $s \in \Sigma^*$ to their labels in T; namely $\theta : \Sigma^* \to T$, as shown below. An element $\tau \in T$ can be thought of as signalling the presence of a string $s \in \Sigma^*$ in the corresponding cell of θ (or $\ker \theta$); for this reason T will be called the *output* alphabet.

$$\begin{tikzcd} \Sigma^* \arrow[r] \arrow[rr, bend right, "\theta"'] & \Sigma^*/\theta \arrow[r, phantom, "\simeq"] & T \end{tikzcd}$$

It is straightforward to construct (abstractly) an automaton A that maps any string

$$s = \sigma_1\sigma_2...\sigma_k \in \Sigma^*$$

into the corresponding sequence of output elements

$$t = \theta(\epsilon)\theta(\sigma_1)\theta(\sigma_1\sigma_2)...\theta(\sigma_1\sigma_2...\sigma_k) \in T^*$$

For the state space X it suffices to take the set of cosets of the coarsest right-congruence ω on Σ^* that is finer than θ. There is then a unique output map $\lambda : X \to T$ such that $\lambda(x[s]) = \theta(s)$, where $x[s]$ is the coset of $s \pmod{\omega}$. The transition function $\xi : X \times \Sigma \to X$ is defined as before, together with the initial state $x_o = x[\epsilon]$. The 6-tuple

$$\mathbf{A} = (X, \Sigma, \xi, x_o, T, \lambda)$$

is sometimes called a *Moore automaton.* Evidently the previous construction of a canonical recognizer for L is recovered on taking $\theta = \{L, \Sigma^* - L\}$, $T = \{0, 1\}$, $\theta(s) = 1$ iff $s \in L$, and $X_m = \{x|\lambda(x) = 1\}$.

Exercise 2.3.1: Let $\bar{K} = L$ and let κ, λ be the Nerode right congruences for K, L respectively. Show that $\kappa \leq \lambda$. In other words, closing a language coarsens its right congruence.

Exercise 2.3.2: Verify in detail that Examples 1-5 above indeed display the canonical recognizers for the indicated languages. Suggestion: For each node n of the given transition graph, let $\Sigma^*(n) \subseteq \Sigma^*$ be the subset of strings that correspond to paths through the graph from n_o to n. Show that the $\Sigma^*(n)$ are precisely the Nerode equivalence classes for the given language L: namely every string in Σ^* belongs to $\Sigma^*(n)$ for some n, for every n any pair of strings in $\Sigma^*(n)$ are Nerode equivalent, and no two strings in distinct subsets $\Sigma^*(n)$, $\Sigma^*(n')$ are equivalent.

Exercise 2.3.3: As in the construction of a Moore automaton, let θ be a given partition of Σ^* and let T label the cells of θ. Construct a recognizer that generates a new output symbol from T only when the θ-cell membership of the input string in Σ^* actually changes, as the string evolves symbol-by-symbol. Show that the number of states of this recognizer need be no more than twice that of the original Moore automaton. Suggestion: start by augmenting T with a 'silent symbol' $\tau_o \notin T$, corresponding to "no change in θ-cell membership". Let $T_o = T \cup \{\tau_o\}$. Examine the relationship between Nerode cells when the output alphabet is T and when it is T_o. Provide a concrete example displaying both the original and the new Moore automata.

Exercise 2.3.4: Let Σ, T be alphabets, let $L \subseteq \Sigma^*$, and let $P : \Sigma^* \to T^*$ be a map with the properties

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(s\sigma) &= \text{either } P(s) \text{ or } P(s)\tau, \text{ some } \tau \in T \end{aligned}$$

Notice that P is *prefix-preserving* in the sense that

$$(\forall s, s' \in \Sigma^*) s \leq s' \quad \Rightarrow \quad P(s) \leq P(s')$$

With τ_o a 'silent symbol' as in Ex. 2.3.3, define $T_o = T \cup \{\tau_o\}$, and then $Q : \Sigma^* \to T_o$ according to

$$\begin{aligned} Q(\epsilon) &= \tau_o, \\ Q(s\sigma) &= \begin{cases} \tau_o & \text{if } P(s\sigma) = P(s) \\ \tau & \text{if } P(s\sigma) = P(s)\tau \end{cases} \end{aligned}$$

Evidently Q maps a string $s \in \Sigma^*$ either into τ_o, or into the last symbol of $P(s)$ in T upon its fresh occurrence. Let $\nu \in \mathcal{E}(\Sigma^*)$ be the equivalence relation defined by $s \equiv s' \pmod{\nu}$ if and only if $Q(s) = Q(s')$ and

$$(\forall u \in \Sigma^*)(\forall t \in T^*) P(su) = P(s)t \quad \Leftrightarrow \quad P(s'u) = P(s')t$$

Show that ν is the coarsest right congruence that is finer than $\ker Q$. Then show how to construct (abstractly) a Moore automaton $\mathbf{A}$ that recognizes the language L and, for each string $s \in \bar{L}$, produces the output $Q(s)$ in T_o. $\mathbf{A}$ is both a recognizer for L and a *realization* of the restriction of P to $\bar{L}$. Create a simple but nontrivial example for which your construction can be carried out explicitly.

2.4 Automata

Let

$$\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$$

be a 5-tuple with Σ as before, Y a nonempty set, $y_o \in Y$, $Y_m \subseteq Y$, and

$$\eta : Y \times \Sigma \to Y$$

a function. $\mathbf{A}$ is an *automaton over the alphabet* Σ. As before, η is the *state transition function*, y_o is the *initial state* and Y_m is the subset of *marker states*; again we extend η to a function

$$\eta : Y \times \Sigma^* \to Y$$

by induction on length of strings.

The language $L \subseteq \Sigma^*$ *recognized* by $\mathbf{A}$ is

$$L := \{s \in \Sigma^* | \eta(y_o, s) \in Y_m\}$$

$\mathbf{A}$ is said to be a *recognizer* for L.

A state $y \in Y$ is *reachable* if $y = \eta(y_o, s)$ for some $s \in \Sigma^*$; and $\mathbf{A}$ is *reachable* if y is reachable for all $y \in Y$. Evidently a state that is not reachable can play no role in the recognition process. If $Y_{rch} \subseteq Y$ is the subset of reachable states then the *reachable subautomaton* $\mathbf{A_{rch}}$ of $\mathbf{A}$ is defined as

$$\mathbf{A_{rch}} = (Y_{rch}, \Sigma, \eta_{rch}, y_o, Y_{m,rch})$$

where

$$\eta_{rch} = \eta | Y_{rch} \times \Sigma, \quad Y_{m,rch} = Y_m \cap Y_{rch}$$

Clearly $\mathbf{A_{rch}}$ recognizes $L \subseteq \Sigma^*$ iff $\mathbf{A}$ does.

Define an equivalence relation λ on Y according to

$$y_1 \equiv y_2 \pmod{\lambda}$$

iff

$$(\forall s \in \Sigma^*) \eta(y_1, s) \in Y_m \quad \Leftrightarrow \quad \eta(y_2, s) \in Y_m$$

That is, two states of $\mathbf{A}$ are λ-equivalent if the same input strings map each of them into the subset of marker states of $\mathbf{A}$. As usual we write $s \equiv s' \pmod{L}$ for Nerode equivalence of strings with respect to L. Now we can state

Proposition 2.4.1

(i) $(\forall t, t' \in \Sigma^*) \eta(y_o, t) \equiv \eta(y_o, t') \pmod{\lambda} \Leftrightarrow t \equiv t' \pmod{L}$

(ii) $(\forall y, y' \in Y) y \equiv y' \pmod{\lambda} \Leftrightarrow (\forall s \in \Sigma^*) \eta(y, s) \equiv \eta(y', s) \pmod{\lambda}$

(iii) $(\forall y, y' \in Y) y \in Y_m \ \&\ y' \equiv y \pmod{\lambda} \Rightarrow y' \in Y_m$

$\square$

Here (iii) states that λ refines the partition $\{Y_m, Y - Y_m\}$.

Define $X := Y/\lambda$ and let $P : Y \to X$ be the canonical projection. Let $X_m := PY_m$ and $x_o := Py_o$. For $x = Py$ define

$$\xi(x, \sigma) = P\eta(y, \sigma)$$

Then ξ is well defined (Exercise 2.4.2) and extends inductively to $X \times \Sigma^*$. Properties of the extended map are summarized in

Proposition 2.4.2

(i) $(\forall s \in \Sigma^*)\xi(x, s) = P\eta(y, s)$ for $x = Py$

(ii) $(\forall s \in \Sigma^*)s \in L \Leftrightarrow \xi(x_o, s) \in X_m$

(iii) $(\forall s, s' \in \Sigma^*)\xi(x_o, s) = \xi(x_o, s') \Leftrightarrow s \equiv s' \pmod L$

$\square$

Thus the 5-tuple

$$\mathbf{B} = (X, \Sigma, \xi, x_o, X_m)$$

is an automaton over Σ. The foregoing definitions and relationships are displayed in the commutative diagrams below.

$$\begin{tikzcd} Y \times \Sigma^* \arrow[r, "\eta"] \arrow[d, "P \times id"'] & Y \arrow[d, "P"] \\ X \times \Sigma^* \arrow[r, "\xi"'] & X \end{tikzcd} \qquad \begin{tikzcd} y_o \in Y \arrow[d, mapsto, "P"] & Y_m \arrow[l] \arrow[d, "P"] \\ x_o \in X & X_m \arrow[l] \end{tikzcd}$$

Informally, "P projects $\mathbf{A}$ onto $\mathbf{B}$." The automaton $\mathbf{B}$ is reachable if $\mathbf{A}$ is reachable. By Proposition 2.4.2(iii) the reachable states of $\mathbf{B}$ can be identified with the cosets of the Nerode equivalence relation on Σ^* with respect to L. We therefore have the following.

Theorem 2.4.1

If $\mathbf{B} = (X, \Sigma, \xi, x_o, X_m)$ is reachable then $\mathbf{B}$ is a canonical recognizer for L. $\square$

Let $\mathbf{A} = (Y, \Sigma, \eta, y_o, Y_m)$ as before. Define the *complementary automaton*

$$\mathbf{A_{co}} = (Y, \Sigma, \eta, y_o, Y - Y_m)$$

Clearly $\mathbf{A}$ recognizes $L \subseteq \Sigma^*$ iff $\mathbf{A_{co}}$ recognizes the complementary language $L_{co} := \Sigma^* - L$. It is easy to see that $s \equiv s' \pmod L$ iff $s \equiv s' \pmod{L_{co}}$, and thus $\|L_{co}\| = \|L\|$.

Similarly if $\mathbf{A_1}$, $\mathbf{A_2}$ are automata over Σ then in obvious notation the *product automaton* $\mathbf{A_1} \times \mathbf{A_2}$ is defined to be

$$\mathbf{A_1} \times \mathbf{A_2} = (Y_1 \times Y_2, \Sigma, \eta_1 \times \eta_2, (y_{1o}, y_{2o}), Y_{1m} \times Y_{2m})$$

where $\eta_1 \times \eta_2 : Y_1 \times Y_2 \times \Sigma \to Y_1 \times Y_2$ is given by

$$(\eta_1 \times \eta_2)((y_1, y_2), \sigma) = (\eta_1(y_1, \sigma), \eta_2(y_2, \sigma))$$

If $\mathbf{A_i}$ recognizes $L_i \subseteq \Sigma^*$ then it is easily seen that $\mathbf{A_1} \times \mathbf{A_2}$ recognizes $L_1 \cap L_2$.

Exercise 2.4.1: Prove Proposition 2.4.1.

Exercise 2.4.2: Show that ξ as described above exists and is unique by first verifying $\ker(P \times id) \leq \ker(P \circ \eta)$ on $Y \times \Sigma$. Then extend ξ to $X \times \Sigma^*$ and prove Prop. 2.4.2.

Exercise 2.4.3: Consider the automaton

$$\mathbf{A} = (Y, \Sigma, \eta, y_0, Y_m)$$

with

$$\begin{aligned} Y &= \{0, 1, 2, 3, 4\} \\ \Sigma &= \{\alpha, \beta\} \\ y_0 &= 0 \\ Y_m &= \{0, 1, 2\} \end{aligned}$$

and transitions

$$\begin{array}{ll} [0, \alpha, 1] & [2, \beta, 3] \\ [0, \beta, 4] & [3, \alpha, 3] \\ [1, \alpha, 2] & [3, \beta, 3] \\ [1, \beta, 4] & [4, \alpha, 4] \\ [2, \alpha, 2] & [4, \beta, 4] \end{array}$$

Construct the automaton $\mathbf{B}$ as above, and tabulate P. Use *TCT* **minstate** to check your result.

Exercise 2.4.4: Given recognizers for L_1 and $L_2 \subseteq \Sigma^*$, construct a recognizer for $L_1 \cup L_2$. **Hint:** use

$$L_1 \cup L_2 = [(L_1)_{co} \cap (L_2)_{co}]_{co}$$

Exercise 2.4.5: Let $L_1, L_2 \subseteq \Sigma^*$ and let \$ be a symbol not in Σ. Let $L \subseteq (\Sigma \cup \{\$\})^*$ be the language $L_1 \$ L_2$, consisting of strings $s_1 \$ s_2$ with $s_1 \in L_1$ and $s_2 \in L_2$. Given recognizers for L_1 and L_2, construct a recognizer for L.

Exercise 2.4.6: In the regular case where $\|L_1\| = n_1$ and $\|L_2\| = n_2$ are both finite, derive tight upper bounds on $\|L_1 \cap L_2\|$, $\|L_1 \cup L_2\|$ and $\|L_1 \$ L_2\|$: that is, show by examples that your bounds cannot be improved for any (n_1, n_2).

2.5 Generators

In Sect. 2.4 we saw that a language can be represented concretely by specifying a corresponding recognizer. For many purposes, a similar but more flexible and economical representation is provided by a *generator*, namely a transition structure in which, in general, only a proper subset of the totality of events can occur at each state. For example, a generator might simply be a recognizer from which the dump state (if any) and all transitions to it have been dropped. Let

$$\mathbf{G} = (Y, \Sigma, \eta, y_o, Y_m)$$

In the present case the transition function $\eta : Y \times \Sigma \to Y$ is defined at each $y \in Y$ only for a subset of the elements $\sigma \in \Sigma$, namely η is a *partial function* (pfn) and we write

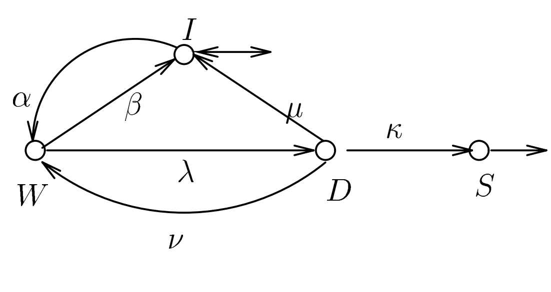
$$\eta : Y \times \Sigma \to Y \quad \text{(pfn)}$$

The notation $\eta(y, \sigma)!$ will mean that $\eta(y, \sigma)$ is defined. Much as before, η is extended to a partial function $\eta : Y \times \Sigma^* \to Y$ by the rules

$$\begin{aligned} \eta(y, \epsilon) &= y \\ \eta(y, s\sigma) &= \eta(\eta(y, s), \sigma) \end{aligned}$$

provided $y' := \eta(y, s)!$ and $\eta(y', \sigma)!$.

The state transition graph of a simple generator is displayed below. The generator represents a 'machine' with possible states (I)dle, (W)orking, (D)own and (S)crapped. Starting in I, the machine may take a workpiece (event α), thereby moving to W. From W the machine may either complete its work cycle, returning to I (event β), or else break down (event λ), moving to D. It remains at D until it is either repaired (events μ, ν) or scrapped (event κ). Repair event μ corresponds to loss of workpiece and return to I, completing a cycle of working, breakdown and repair, while ν corresponds to saving the workpiece to continue work at W. The initial state is I and the marker states both I and S. This process may be thought of as repeating an arbitrary finite number of times.



Exercise 2.5.1: Let $\Sigma = \{\alpha, \beta\}$, $L = \Sigma^*\alpha\beta\alpha\Sigma^*$. Thus L consists of all finite strings of the form $s_1\alpha\beta\alpha s_2$, where s_1 and s_2 are arbitrary strings over Σ.

(i) Give an alternative verbal description of L, regarding the occurrence of $\alpha\beta\alpha$ as the signal of an 'emergency'.

(ii) Design a (deterministic) finite-state recognizer for L. **Hint:** This can be done using just 4 states.

Exercise 2.5.2: Develop a finite-state operational (not electrical) model of an ordinary household telephone, as seen by a single subscriber able to place or receive a call. Note: This can become surprisingly complicated, so start with something simple and then refine it in stages, up to a model on the order of 10 to 20 states. ◇

In general one may think of **G** as a device that 'generates' strings by starting at the initial state y_o and executing only transitions for which its transition function η is defined; if more than one transition is defined to exit from a given state y, the device may be supposed to choose just one of these possibilities, on any particular occasion, by some quasi-random internal mechanism that is unmodelled by the system analyst. In this sense the generating action is 'possibilistic'. It may be thought of as carried out in repeated 'trials', each trial generating just one of the possible strings s for which $\eta(y_o, s)$ is defined. In this account, 'choose' needn't always be interpreted literally: most machines do not choose to start work autonomously, but are forced by some external agent. The generation model is independent of (but consistent with) causative factors, which should be examined in context.

The set of words $s \in \Sigma^*$ such that $\eta(y_o, s)!$ is the *closed behavior* of **G**, denoted by $L(\mathbf{G})$, while the subset of words $s \in L(\mathbf{G})$ such that $\eta(y_o, s) \in Y_m$ is the *marked behavior* of **G**, denoted by $L_m(\mathbf{G})$. Clearly $L(\mathbf{G})$ is closed and contains $L_m(\mathbf{G})$. A generator **G** is to be thought of as representing both its closed and marked behaviors. As in the case of an automaton, a state $y \in Y$ is *reachable* if there is a string $s \in \Sigma^*$ with $\eta(y_o, s)!$ and $\eta(y_o, s) = y$; **G** itself is *reachable* if y is reachable for all $y \in Y$. A state $y \in Y$ is *coreachable* if there is $s \in \Sigma^*$ such that $\eta(y, s) \in Y_m$; and **G** is *coreachable* if y is coreachable for every $y \in Y$. **G** is *nonblocking* if every reachable state is coreachable, or equivalently $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$; the latter condition says that any string that can be generated by **G** is a prefix of (i.e. can always be completed to) a marked string of **G**. Finally **G** is *trim* if it is both reachable and coreachable. Of course **G** trim implies **G** nonblocking, but the converse is false: a nonblocking generator might have nonreachable states (that might or might not be coreachable). The 'machine' illustrated above is trim; if the state S were not marked, the resulting generator would still be reachable, but not coreachable. In practice one usually models a generator **G** to be reachable; however, it may be quite realistic for **G** not to be coreachable, for instance to have a 'dump' or 'dead-end' state from which no marker state is accessible. As a rule it is therefore advisable not to overlook non-coreachable states, or inadvertently remove them from the model.

The following counterpart of Proposition 2.4.1 can be used to reduce a (reachable) generator **G** to a minimal state version having the same closed and marked behaviors. This time we need Nerode equivalence relations on Σ^* for both $L(\mathbf{G})$ ($\equiv_c$, say) and $L_m(\mathbf{G})$ ($\equiv_m$).

Define $\lambda \in \mathcal{E}(Y)$ according to $y \equiv y' \pmod{\lambda}$ provided

(i) $(\forall s \in \Sigma^*)\ \eta(y,s)! \Leftrightarrow \eta(y',s)!$

(ii) $(\forall s \in \Sigma^*)\ \eta(y,s)!\ \&\ \eta(y,s) \in Y_m \Leftrightarrow \eta(y',s)!\ \&\ \eta(y',s) \in Y_m$

Proposition 2.5.1

(i) $(\forall s, s' \in \Sigma^*)\ \eta(y_o,s) \equiv \eta(y_o,s')(\mod \lambda) \Leftrightarrow s \equiv_c s'\ \&\ s \equiv_m s'$

(ii) $(\forall y, y' \in Y)\ y \equiv y'(\mod \lambda) \Leftrightarrow (\forall s \in \Sigma^*)\eta(y,s) \equiv \eta(y',s)(\mod \lambda)$

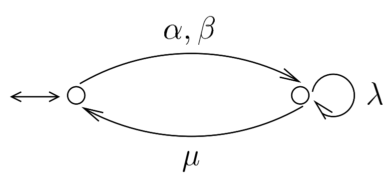
(iii) $(\forall y, y' \in Y)\ y \in Y_m\ \&\ y' \equiv y(\mod \lambda) \Rightarrow y' \in Y_m$.

Exercise 2.5.3: Prove Prop. 2.5.1 and provide a nontrivial application. ◇

Reduction of a generator **G** to a minimal (reachable) version by projection (mod λ) is implemented in *TCT* as the procedure **minstate**.

Exercise 2.5.4: Let $L \subseteq \Sigma^*$ and **G** be its minimal-state recognizer. Show how to construct a recognizer **H** whose current state encodes both the current state of **G** and the last previous state of **G**. Generalize your result to encode the list of n most recent states of **G** (in temporal order). Alternatively, construct a recognizer **K** whose current state encodes the list of n most recent events of **G**, in order of occurrence. Illustrate your results using **G** as shown below.

G



Exercise 2.5.5: Let $\Sigma = \{0, 1\}$ and let $L \subseteq \Sigma^*$ be those strings in which a 1 occurs in the third-to-last place. Design a recognizer for L. ◇

Occasionally it will be useful to bring in a nondeterministic generator, namely one in which more than one transition defined at a given exit state may carry the same label $\sigma \in \Sigma$. Formally a *nondeterministic generator* is a 5-tuple

$$\mathbf{T} = (Y, \Sigma, \tau, y_o, Y_m)$$

as before, but with the difference that the transition function τ now maps pairs (y, σ) into subsets of Y:

$$\tau : Y \times \Sigma \to Pwr(Y)$$

Notice that τ may be considered a total function because of the possible evaluation $\tau(y, \sigma) = \emptyset$. We extend τ to a function on strings by the rules

$$\begin{aligned} \tau(y, \epsilon) &= \{y\} \\ \tau(y, s\sigma) &= \cup\{\tau(y', \sigma) | y' \in \tau(y, s)\} \end{aligned}$$

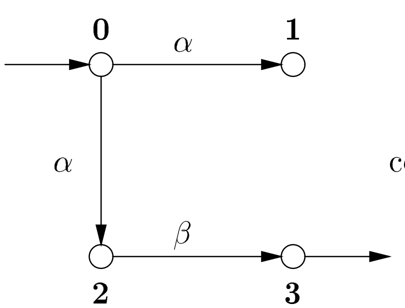
We define the *closed behavior* $L(\mathbf{T})$ of the nondeterministic generator $\mathbf{T}$ to be the set of all strings $s \in \Sigma^*$ for which $\tau(y_o, s) \neq \emptyset$. The *marked behavior* $L_m(\mathbf{T})$ of $\mathbf{T}$ is the set of all strings in $L(\mathbf{T})$ for which at least one particular realization (path through the transition graph) corresponds to a sequence of states starting at y_o and ending in Y_m:

$$L_m(\mathbf{T}) = \{s \in \Sigma^* | \tau(y_o, s) \cap Y_m \neq \emptyset\}$$

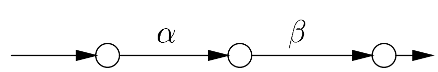
If the nondeterministic generator $\mathbf{T}$ is given, then a deterministic generator $\mathbf{T_{det}}$ that generates the same languages $L(\mathbf{T})$ and $L_m(\mathbf{T})$ can be constructed by taking as the states of $\mathbf{T_{det}}$ the nonempty subsets of Y, i.e. $Y_{det} = Pwr(Y) - \{\emptyset\}$; usually only a small fraction of the new states turn out to be reachable and therefore worthy of retention. This process of converting a nondeterministic to a deterministic generator is known as the *subset construction*. While a nondeterministic generator can always be converted in this way to a deterministic one, in some applications the use of a nondeterministic generator may result in greater convenience or economy of description.

A word of warning. Conversion by the subset construction may obliterate blocking situations in the nondeterministic model. For instance

$\mathbf{T}$

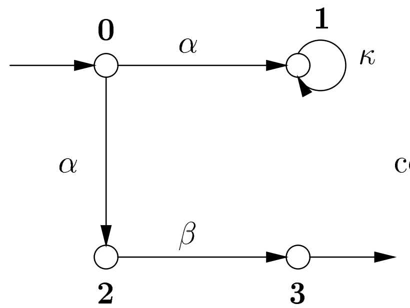


converts to $\mathbf{T}_{\text{det}}$

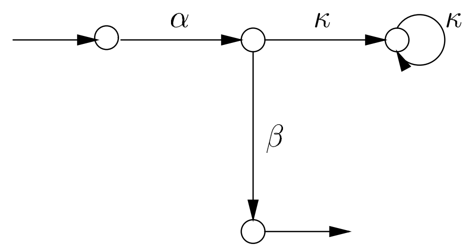


ignoring the fact that $\mathbf{T}$ can block at state 1. A solution is first to enhance $\mathbf{T}$ by selflooping all non-coreachable states with a new event label $\kappa \notin \Sigma$. Thus

$\mathbf{T}'$



converts to $\mathbf{T}'_{\text{det}}$



Exercise 2.5.6: (Subset construction) Supply the details of the subset construction. Namely let

$$\mathbf{H} = (Y, \Sigma, \tau, y_o, Y_m)$$

be a nondeterministic generator, and let

$$\mathbf{G} = (X, \Sigma, \xi, x_o, X_m)$$

be the deterministic generator defined by

$$\begin{array}{ll} X = Pwr(Y) - \{\emptyset\}, & \xi(x, \sigma) = \cup\{\tau(y, \sigma)|y \in x\} \\ x_o = \{y_o\}, & X_m = \{x|x \cap Y_m \neq \emptyset\} \end{array}$$

Here ξ is a partial function on $X \times \Sigma$ with $\xi(x, \sigma)!$ iff the defining evaluation is nonempty. Show that $L(\mathbf{G}) = L(\mathbf{H})$ and $L_m(\mathbf{G}) = L_m(\mathbf{H})$. Check the two examples above in detail.

Exercise 2.5.7: Implementation of the subset construction is unattractive in that it may require exponential computational effort in the state size of $\mathbf{T}$. Why? Can you exhibit a 'worst case'?

2.6 Regular Expressions

In discussing small examples we may use, in addition to state transition graphs, a representation for regular languages known as regular expressions. These may be combined by *regular algebra* to represent complex languages in terms of simpler ones.

If $s \in \Sigma^*$ we may write s for the language $\{s\} \subseteq \Sigma^*$. Let $L, M \subseteq \Sigma^*$. New languages $L + M$, LM and L^* are defined as follows.

$$\begin{array}{rcl} L + M & := & \{s|s \in L \text{ or } s \in M\} \\ LM & := & \{st|s \in L \text{ and } t \in M\} \\ L^* & := & \epsilon + \cup_{k=1}^{\infty}\{s_1 ... s_k|s_1, ..., s_k \in L\} \end{array}$$

Thus

$$L^* = \epsilon + L + L^2 + ... = \cup_{k=0}^{\infty} L^k, \quad L^o := \epsilon$$

If $\Sigma = \{\alpha, \beta\}$ we may sometimes write $\Sigma = \alpha + \beta$. In accordance with the definition of catenation and the properties of ϵ we have

$$\epsilon s = s\epsilon = s, \quad \epsilon\epsilon = \epsilon, \quad \epsilon^* = \epsilon$$

and for the empty language $\emptyset$,

$$\emptyset + L = L, \quad \emptyset L = L\emptyset = \emptyset, \quad \emptyset^* = \epsilon.$$

A *regular expression over* Σ is a formal expression obtained by a finite number of applications of the operations listed above, to elements in the list: elements of $\Sigma, \epsilon, \emptyset$, and all expressions so obtained. Since $\Sigma, \{\epsilon\}, \emptyset$ are subsets of Σ^*, a regular expression represents a subset of Σ^*. It is shown in the literature (Kleene's Theorem) that the subsets represented by the regular expressions over Σ are exactly the regular sublanguages of Σ^*, namely the sublanguages whose canonical recognizers have a finite state set.

Regular algebra admits numerous identities that are useful for simplifying regular expressions. They may be proved by comparing the corresponding subsets of Σ^*. While we are not likely to undertake complicated manipulations of regular expressions, the following catalog of identities is provided for reference. Here $L, M, N \subseteq \Sigma^*$ are arbitrary languages over Σ.

$$L\epsilon = \epsilon L = L, \quad L + L = L, \quad L + M = M + L$$

$$(L + M) + N = L + (M + N) \quad \text{(so we write } L + M + N\text{)}$$

$$(LM)N = L(MN) \quad \text{(so we write } LMN\text{)}, \quad (L + M)N = LN + MN$$

$$L^* = \epsilon + LL^*, \quad L^*L^* = L^*, \quad (L^*)^* = L^*, \quad LL^* = L^*L$$

$$(L^* + M^*)^* = (L^*M^*)^* = (L^*M)^*L^* = (L + M)^*$$

$$(LM)^*L = L(ML)^*, \quad (L^*M)^* = \epsilon + (L + M)^*M$$

The following result is fundamental for the solution of systems of equations.

Proposition 2.6.1

(i) If $L = M^*N$ then $L = ML + N$

(ii) If $\epsilon \notin M$ then $L = ML + N$ implies $L = M^*N$. $\square$

Part (ii) is known as Arden's rule. Taken with (i) it says that if $\epsilon \notin M$ then $L = M^*N$ is the unique solution of $L = ML + N$; in particular if $L = ML$ (with $\epsilon \notin M$) then $L = \emptyset$.

Exercise 2.6.1: Show by counterexample that the restriction $\epsilon \notin M$ in Arden's rule cannot be dropped.

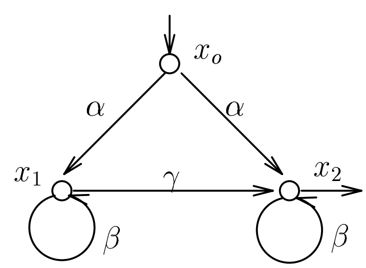
Exercise 2.6.2: Prove Arden's rule. **Hint:** If $L = ML + N$ then for every $k \geq 0$

$$L = M^{k+1}L + (M^k + M^{k-1} + .. + M + \epsilon)N$$

$\diamond$

As an application of Arden's rule it will now be shown how to find a regular expression for the language generated by a finite nondeterministic transition structure. In the example displayed below we write by definition

$$L_m(\mathbf{G}) = \{s \in \Sigma^* | \tau(x_o, s) \cap X_m \neq \emptyset\}$$



Step 1. Write a formal linear equation representing the transitions at each state of $\mathbf{G}$:

$$\begin{aligned} x_o &= \alpha x_1 + \alpha x_2 \\ x_1 &= \beta x_1 + \gamma x_2 \\ x_2 &= \beta x_2 + \epsilon \end{aligned}$$

Note that the 'forcing' term ϵ is added on the right side if $x \in X_m$.

Step 2. Consider the states x_i as tokens for the 'unknown' regular languages

$$X_i = \{s \in \Sigma^* | \tau(x_i, s) \cap X_m \neq \emptyset\}$$

Thus it is easy to see that the X_i satisfy exactly the regular-algebraic equations just written. Solve these equations using Arden's rule:

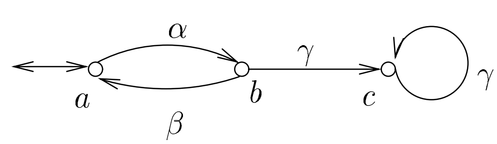
$$\begin{aligned} X_2 &= \beta^* \\ X_1 &= \beta^* \gamma X_2 = \beta^* \gamma \beta^* \\ X_o &= \alpha \beta^* \gamma \beta^* + \alpha \beta^* \\ &= \alpha \beta^* (\epsilon + \gamma \beta^*) \end{aligned}$$

Step 3. Since x_o is the initial state we obtain

$$L_m(\mathbf{G}) = X_o = \alpha \beta^* (\epsilon + \gamma \beta^*)$$

As a second example consider the transition graph below, with states labelled a, b, c. We have

$$a = \alpha b + \epsilon, \quad b = \beta a + \gamma c, \quad c = \gamma c$$



These equations give

$$c = \emptyset, \quad b = \beta a, \quad a = (\alpha \beta)^*$$

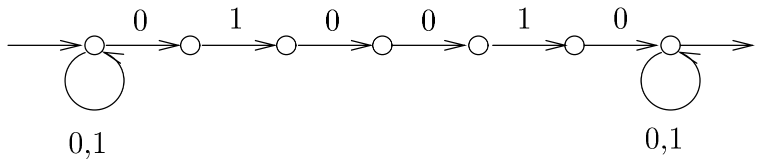
Exercise 2.6.3:

Consider the problem of designing a recognizer for the language

$$L = \Sigma^* 0\ 1\ 0\ 0\ 1\ 0 \Sigma^*$$

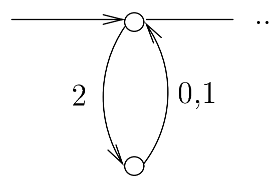
where $\Sigma = \{0, 1\}$. In other words, we want a bell to beep (and go on beeping with each new input symbol) as soon as the string indicated occurs for the first time in an arbitrary sequence of 0's and 1's. It is easy to specify a nondeterministic recognizer, as shown below.

NDR



By contrast, it's quite difficult to design a deterministic recognizer 'by hand': try it and see. To do this using TCT, modify the initial selfloop to obtain

XDR



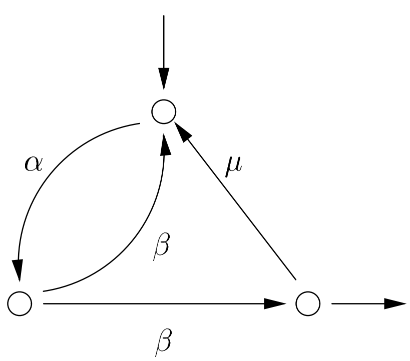
where 2 is a dummy symbol to make **XDR** deterministic. Compute

$$\mathbf{DR} = \mathbf{project}(\mathbf{XDR}, [2])$$

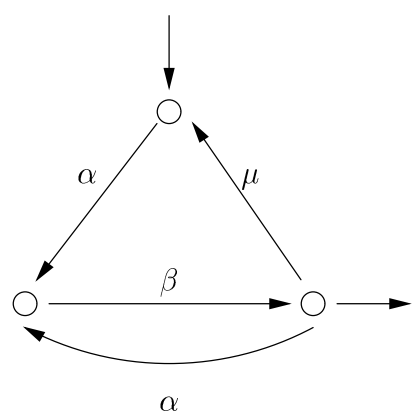
Draw the state diagram and convince yourself that **DR** does the job.

Exercise 2.6.4: Consider the generators

$\mathbf{T_1}$:



$\mathbf{T_2}$



By application of the subset construction to $\mathbf{T_1}$, show that $\mathbf{T_1}$ and $\mathbf{T_2}$ determine the same regular language. Using Arden's rule, obtain corresponding regular expressions

$$\begin{aligned} L_1 &= (\alpha\beta(\varepsilon + \mu))^* \alpha\beta \\ L_2 &= (\alpha(\beta\alpha)^*\beta\mu)^*\alpha(\beta\alpha)^*\beta \end{aligned}$$

and prove by "regular algebra" that indeed $L_1 = L_2$. **Hint:** First prove the identity

$$(L^*M)^*L^* = (L + M)^*$$

and then reduce L_2 to L_1.

2.7[1] Causal Output Mapping and Hierarchical Aggregation

In this section we develop in greater detail some of the ideas in Sects. 2.2, 2.3 and 2.5. For easier reading, some definitions are repeated.

Let $\mathbf{G} = (Q, \Sigma, \delta, q_o)$ be a *generator*: namely Q is a nonempty *state set*, $q_o \in Q$ is the *initial state*, Σ is a nonempty set, the *alphabet of transition labels*, and $\delta : Q \times \Sigma \to Q$ is the (partial) *transition function*. The action of δ may sometimes be written by juxtaposition: $\delta(q, \sigma) = q\sigma$; and if this convention is understood then $\mathbf{G}$ may be written $(Q, \Sigma, \_, q_o)$. Also δ is extended by iteration in the natural way to a partial function on the set Σ^* of all finite strings s of elements in Σ. We write $\delta(q, s)!$ or $qs!$ to mean that the action of δ is defined at $(q, s) \in Q \times \Sigma^*$. If $\epsilon \in \Sigma^*$ is the *empty string* then $q\epsilon := \delta(q, \epsilon) := q$. The *closed behavior* of $\mathbf{G}$ is the subset of strings

$$L(\mathbf{G}) = \{s \in \Sigma^* | \delta(q_o, s)!\}$$

In this section marked behavior plays no role.

For brevity write $L(\mathbf{G}) =: L$. Note that L contains ϵ, and that L is *prefix-closed*, namely if $s \in L$ and $s' \leq s$ (s' is a prefix of s) then $s' \in L$ as well. Now let T be a second alphabet and suppose that $P : L \to T^*$ is a (total) map with the properties

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(s\sigma) &= \begin{cases} \text{either } P(s) \\ \text{or} \\ P(s)\tau, \ \text{some } \tau \in T \end{cases} \qquad s \in \Sigma^*, \quad \sigma \in \Sigma \end{aligned}$$

We call P the *reporter* of $\mathbf{G}$. P is *causal* (or *nonanticipative*), in the sense that it is prefix preserving: if $s \leq s'$ then $P(s) \leq P(s')$. The pair $(\mathbf{G}, P)$ may be called a *generator with reporter*.

The simplest way to visualize the behavior of $(\mathbf{G}, P)$ is via the *reachability tree* of $L(\mathbf{G})$, a tree in which each node n is identified with a string s of L by a bijection $n : L \to$ Nodes: the root node is $n(\epsilon)$, and for each $s \in L$ the children of $n(s)$ are exactly the nodes $\{n(s\sigma) | s\sigma \in L\}$. Notice that it is possible to drop the distinction between nodes and strings, taking as one particular version of $\mathbf{G}$ the 4-tuple

$$\mathcal{T}(L) := \{L, \Sigma, \_, \epsilon\}$$

In the reachability tree the action of P can be indicated as follows. Bring in an element $\tau_o \notin T$ and write $T_o = T \cup \{\tau_o\}$. Define the *tail map* $\omega_o : L \to T_o$ according to

$$\begin{aligned} \omega_o(\epsilon) &= \tau_o \\ \omega_o(s\sigma) &= \begin{cases} \tau_o & \text{if } P(s\sigma) = P(s) \\ \tau & \text{if } P(s\sigma) = P(s)\tau \end{cases} \end{aligned}$$

[1] Not needed in the sequel.

Thus ω_o identifies the last output symbol "reported" by P, with τ_o interpreted as the "silent output symbol". Using $\mathcal{T}(L)$ to represent $\mathbf{G}$, we can now represent $(\mathbf{G}, P)$ by the 6-tuple

$$\mathcal{T}(L,P) := \{L, \Sigma, \_\_, \epsilon, T_o, \omega_o\}$$

In graphical representation the edges of the tree (transitions $s \xrightarrow{\sigma} s\sigma$ of L) are labelled by the corresponding element σ of Σ, while the nodes are labelled by the corresponding output in T_o.

Define the *output language* of $(\mathbf{G}, P)$ to be $PL \subseteq T^*$, and let $\mathcal{T}(PL)$ be the reachability tree of PL. Write ϵ for the empty string in T^* as well. Clearly PL contains ϵ and is prefix-closed. Again, ignoring the distinction between nodes of the tree and strings of PL, we have that P induces a surjection $\hat{P}$ from $\mathcal{T}(L,P)$ to $\mathcal{T}(PL)$:

$$\hat{P} : L \to PL : s \mapsto Ps$$

It is convenient to introduce the modified tail map

$$\hat{\omega}_o : L \times \Sigma \to T_o : (s,\sigma) \mapsto \omega_o(s\sigma)$$

and define the product map

$$\hat{P} \times \hat{\omega}_o : L \times \Sigma \to PL \times T_o : (s,\sigma) \mapsto (\hat{P}(s), \hat{\omega}_o(s,\sigma))$$

Then we have the obvious commutative diagram

$$\begin{tikzcd}
L \times \Sigma \arrow[r] \arrow[d, "\hat{P} \times \hat{\omega}_o"] & L \arrow[d, "\hat{P}"] \\
PL \times T_o \arrow[r] & PL
\end{tikzcd}$$

Here the horizontal arrows represent the transition action, with the extended definition for the bottom arrow that $(t, \tau_o) \mapsto t$. One can think of the diagram as a display of how transitions in the tree $\mathcal{T}(L)$ are "tracked" by transitions in $\mathcal{T}(PL)$. Notice that by composition of transition functions the diagram can be iterated arbitrarily far to the right:

$$\begin{tikzcd}
L \times \Sigma \arrow[r] \arrow[d] & L & \times\Sigma \arrow[r] \arrow[d] & L & \times\Sigma \arrow[r] \arrow[d] & \cdots \\
PL \times T_o \arrow[r] & PL & \times T_o \arrow[r] & PL & \times T_o \arrow[r] & \cdots
\end{tikzcd}$$

thus extending the tracking feature to strings of L.

While the reachability trees of L and PL are useful for purposes of visualization, more efficient representations are available in principle, which are often more convenient in practical applications. These are obtained by aggregating nodes of the tree (i.e. strings of the language) by means of suitable equivalence relations. For any language $K \subseteq A^*$ over an alphabet A, write

$$K/s := \{u \in A^* | su \in K\}, \quad s \in A^*$$

The *Nerode equivalence relation* (Nerode(K)) on A^* is defined by

$$s \equiv s' \pmod{\text{Nerode}(K)} \text{ iff } K/s = K/s'$$

For brevity write $(\bmod\ K)$ for (mod Nerode(K)). In more detail the definition can be stated:

$$s \equiv s' \pmod{K} \text{ iff } (\forall u \in A^*) su \in K \quad \Leftrightarrow \quad s'u \in K$$

Nerode(K) is a *right congruence* on A^*, namely

$$(\forall u \in A^*) s \equiv s' \pmod{K} \quad \Rightarrow \quad su \equiv s'u \pmod{K}$$

as is quickly seen from the identity

$$K/(su) = (K/s)/u$$

If $[s]$ is the coset of $s \pmod{K}$ it then makes sense to define

$$[s]\alpha = [s\alpha] \quad s \in A^*, \quad \alpha \in A$$

Setting $Z = \{[s] | s \in L\}$, $z_o = [\epsilon]$ and with transitions (as just described) written by juxtaposition, we obtain the generator

$$\mathbf{N} = (Z, A, \_\_, z_o)$$

with $L(\mathbf{N}) = K$. We refer to $\mathbf{N}$ as the *Nerode generator* of K. Its states are the cells (equivalence classes) of Nerode(K). It can easily be shown that the Nerode generator of K is "universal" in the sense that any other generator for K (over A), say

$$\tilde{\mathbf{N}} = (\tilde{Z}, A, \_\_, \tilde{z}_o)$$

can be mapped onto $\mathbf{N}$ in accordance with the commutative diagram

$$\begin{tikzcd}
\tilde{Z} \times A \arrow[r] \arrow[d, "\pi \times id_A"] & \tilde{Z} \arrow[d, "\pi"] & \ni & \tilde{z}_o \arrow[d, mapsto] \\
Z \times A \arrow[r] & Z & \ni & z_o
\end{tikzcd}$$

where $\pi : \tilde{Z} \to Z$ is a suitable surjection and id_A is the identity map on A.

Let $\mathcal{N}(PL) = (X, T, \_\_, x_o)$ be the Nerode generator for the language PL discussed above.

In order to find a suitably economical representation of the pair (L, P) we must incorporate into the new state structure both the information required for the generation of L and the additional information required to specify P. To this end, define an equivalence relation $\mathrm{Fut}(P)$ on L as follows:

$$s \equiv s' \pmod{\mathrm{Fut}(P)} \text{ iff } (\forall u \in \Sigma^*) P(su)/Ps = P(s'u)/Ps'$$

or in more detail

$$s \equiv s' \pmod{\mathrm{Fut}(P)} \text{ iff } (\forall u \in \Sigma^*)(\forall w \in T^*) P(su) = (Ps)w \Leftrightarrow P(s'u) = P(s')w$$

Thus $\mathrm{Fut}(P)$ aggregates strings whose corresponding outputs share a common future. It is well to note that equivalence $\mathrm{Fut}(P)$ does not imply that corresponding outputs share a common present, namely that the output map ω_o determined by P takes the same value on equivalent strings. In the poset of equivalence relations on L the equivalence kernel of ω_o is in general not comparable with $\mathrm{Fut}(P)$.

The identity

$$P(suv)/P(su) = [P(suv)/Ps]/[P(su)/Ps]$$

shows that $\mathrm{Fut}(P)$ is actually a right congruence on L. Since the meet of right congruences is again a right congruence, we may define the right congruence

$$\mathrm{Mealy}(L, P) := \mathrm{Nerode}(L) \wedge \mathrm{Fut}(P)$$

Now let $\mathbf{G} = (Q, \Sigma, \_\_, q_o)$ be specifically the generator of $L = L(\mathbf{G})$ based on $\mathrm{Mealy}(L, P)$, i.e. $q \in Q$ stands for a coset (mod $\mathrm{Mealy}(L, P)$) in L. We define the *Mealy output map* λ of $\mathbf{G}$ according to

$$\lambda : Q \times \Sigma \to T_o : (q, \sigma) \mapsto \omega_o(s\sigma), \quad \text{any } s \in q$$

From the definitions it easily follows that λ is well defined. The 5-tuple

$$\mathcal{M}(L, P) := (Q, \Sigma, \_\_, q_o, \lambda)$$

will be called the *Mealy* generator for (L, P). One can verify that any other generator for (L, P) of the same type, say

$$\tilde{\mathcal{M}}(L, P) = (\tilde{Q}, \Sigma, \_\_, \tilde{q}_o, \tilde{\lambda})$$

maps onto $\mathcal{M}(L, P)$ in the sense that for a suitable surjection π the following diagram commutes:

$$
\begin{tikzcd}
 & \tilde{Q}\times\Sigma \arrow[dl, "\tilde{\lambda}"'] \arrow[r] \arrow[dd, "\pi\times id"] & \tilde{Q} \ni \tilde{q}_o \arrow[dd, "\pi", mapsto] \\
T_o & & \\
 & Q\times\Sigma \arrow[ul, "\lambda"] \arrow[r] & Q \ni q_o
\end{tikzcd}
$$

In particular such a diagram exists with $\tilde{\mathcal{M}}$ taken to be the Mealy description

$$(L, \Sigma, \_, \epsilon, \hat{\omega}_o)$$

The situation now is that we have obtained two "economical" descriptions: the Mealy generator $\mathcal{M}(L, P)$ of (L, P), and the Nerode generator $\mathcal{N}(PL)$ of PL. However, while the items P and PL can certainly be recovered from $\mathcal{M}(L, P)$, the state set of $\mathcal{M}(L, P)$ is a little too coarse to allow tracking in $\mathcal{N}(PL)$ of transitions in $\mathcal{M}(L, P)$. The problem is just that $s \equiv s'$ (mod Mealy(L, P)) does not imply $Ps \equiv Ps'$ (mod PL). The cure is to refine equivalence (mod Mealy(L, P)) as follows. Define

$$\text{Hier}(L, P) := \text{Mealy}(L, P) \wedge \text{Nerode}(PL) \circ P$$

where $s \equiv s'$ (mod Nerode$(PL) \circ P$) is defined to mean $Ps \equiv Ps'$ (mod Nerode(PL)).

Proposition 2.7.1

Hier(L, P) is a right congruence on L.

Proof

In the proof write Hier etc. for brevity. Suppose $s \equiv s'$ (mod Hier), let $u \in \Sigma^*$, and let $P(su) = (Ps)w$. Since $s \equiv s'$ (mod Mealy), it follows both that $su \equiv s'u$ (mod Mealy) and that $P(s'u) = (Ps')w$. Since $Ps \equiv Ps'$ (mod PL) we therefore have $P(su) \equiv P(s'u)$ (mod PL) and thus Nerode $\circ P(su) \equiv$ Nerode $\circ P(s'u)$. $\square$

With Hier(L, P) as the basis of our new description of (L, P), let the corresponding generator of Mealy type be

$$\mathcal{H}(L, P) = (Y, \Sigma, \_, y_o, \eta)$$

Just as above we have that $(L, \Sigma, \_, \epsilon, \hat{\omega}_o)$ projects onto $\mathcal{H}$ according to the diagram

$$
\begin{tikzcd}
 & L\times\Sigma \arrow[dl, "\hat{\omega}_o"'] \arrow[r] \arrow[dd, "\rho\times id"] & L \ni \epsilon \arrow[dd, "\rho", mapsto] \\
T_o & & \\
 & Y\times\Sigma \arrow[ul, "\eta"] \arrow[r] & Y \ni y_o
\end{tikzcd}
$$

It will be seen that the output map η can be identified with the map g of the following proposition, which states that there is a natural connection between $\mathcal{H}(L, P)$ and $\mathcal{N}(PL)$ that admits step-by-step transition tracking.

Proposition 2.7.2

There exist a surjection $f : Y \to X$ and a map $g : Y \times \Sigma \to T_o$ such that the following diagram commutes, where

$$f \times g : Y \times \Sigma \to X \times T_o : (y, \sigma) \quad \longmapsto \quad (f(y), g(y, \sigma))$$

$$\begin{tikzcd}
Y \times \Sigma \arrow[r] \arrow[d, "f \times g"] & Y \ni y_o \arrow[d, "f"] \arrow[d, mapsto, shift left=4] \\
X \times T_o \arrow[r] & X \ni x_o
\end{tikzcd}$$

For the bottom arrow we recall the extended definition $(x, \tau_o) \mapsto x$.

Proof

Let $\rho : L \to Y$ be the natural projection (mod Hier), $\nu : PL \to X$ the natural projection (mod PL), and consider the diagram

$$\begin{tikzcd}
L \arrow[r, "\rho"] \arrow[d, "P"] & Y \arrow[d, dotted, "f"] \\
PL \arrow[r, "\nu"] & X
\end{tikzcd} \qquad (1)$$

By definition of Hier, $\rho(s) = \rho(s')$ implies $\nu \circ P(s) = \nu \circ P(s')$, namely $\ker(\rho) \leq \ker(\nu \circ P)$, which shows that f exists as displayed. That f is surjective follows because P and ν are surjective, hence $\nu \circ P$ is surjective. Furthermore $\nu \circ P(\epsilon) = \nu(\epsilon) = x_o$ and $\rho(\epsilon) = y_o$ by definition of generator, so $f(y_o) = x_o$.

To complete the proof we need another version of the tail map, namely $\omega : L \times \Sigma \to T^*$, defined according to

$$\omega(s, \sigma) = \begin{cases} \epsilon & \text{if } \hat{\omega}_o(s, \sigma) = \tau_o \\ \tau & \text{if } \hat{\omega}_o(s, \sigma) = \tau \in T \end{cases}$$

With the usual definition $t = t\epsilon$ for catenation by ϵ, we have the identity

$$P(s\sigma) = P(s)\omega(s,\sigma) \quad (2)$$

Next consider the diagram

$$\begin{tikzcd}
L \times \Sigma \arrow[rr, "\rho \times id"] \arrow[dr, "\hat{\omega}_o"'] & & Y \times \Sigma \arrow[dl, dotted, "g"] \\
& T_o &
\end{tikzcd} \qquad (3)$$

where $id := id_\Sigma$. We have $(\rho \times id)(s,\sigma) = (\rho \times id)(s',\sigma')$ iff $\rho(s) = \rho(s')$ and $\sigma = \sigma'$, which implies $s \equiv s'$ (mod Fut(P)). But then $P(s\sigma) = P(s)\omega(s,\sigma)$ iff $P(s'\sigma) = P(s')\omega(s,\sigma)$, so that $\omega_o(s\sigma) = \omega_o(s'\sigma) \in T_o$, namely $\hat{\omega}_o(s,\sigma) = \hat{\omega}_o(s',\sigma)$. This shows that

$$\ker(\rho \times id) \leq \ker(\hat{\omega}_o)$$

proving the existence of g as displayed. To check that $f(y\sigma) = f(y)g(y,\sigma)$ we assume that $y = \rho(s)$ and compute

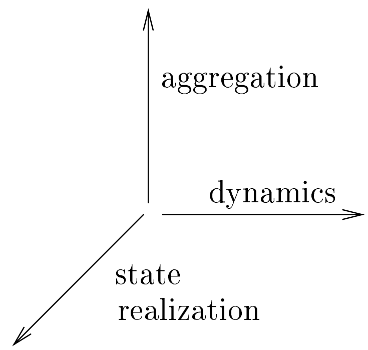
$$\begin{aligned}
f(y\sigma) &= f(\rho(s)\sigma) \\
&= f \circ \rho(s\sigma) \quad \{\ker\rho := \text{ Hier is a right congruence}\} \\
&= \nu \circ P(s\sigma) \quad \{\text{commutativity of (1)}\} \\
&= \nu[P(s)\omega(s\sigma)] \quad \{\text{identity (2) for } \omega\} \\
&= \nu[P(s)]\omega(s\sigma) \quad \{\ker\nu := \text{ Nerode is a right congruence}\} \\
&= f[\rho(s)]\omega(s\sigma) \quad \{\text{commutativity of (1)}\} \\
&= f[\rho(s)]\hat{\omega}_o(s,\sigma) \quad \{\text{definitions of } \omega\text{, transition function}\} \\
&= f(y)(g \circ (\rho \times id))(s,\sigma) \quad \{\text{commutativity of (3)}\} \\
&= f(y)g(y,\sigma) \quad \{y = \rho(s)\}
\end{aligned}$$

□

The results so far can all be displayed in the commutative cube below.

$$\begin{tikzcd}
& PL \times T_o \arrow[rr] \arrow[dl, "\nu \times id_{T_o}"'] & & PL \arrow[dl, "\nu"'] \\
X \times T_o \arrow[rr] & & X & \\
& L \times \Sigma \arrow[uu, "\hat{P} \times \hat{\omega}_o"] \arrow[rr] \arrow[dl, "\rho \times id_\Sigma"] & & L \arrow[uu, "\hat{P}"'] \arrow[dl, "\rho"] \\
Y \times \Sigma \arrow[uu, "f \times g"] \arrow[rr] & & Y \arrow[uu, "f"'] &
\end{tikzcd}$$

In the cube, unlabelled arrows represent transition action. The bottom face can be thought of as representing "fast" dynamics, originating with the generating action $L \times \Sigma \to L$, while the top face represents the "slow" dynamics that result from hierarchical aggregation. The rear face of the cube represents fine-grained behavioral descriptions in terms of strings, while the front face carries the corresponding more economical state descriptions. The scheme is summarized below.



As it stands, the scheme is purely passive (it is nothing more than a "clock with two hands"); the dynamic action is purely deterministic, and there is no way for an "agent" to intervene. However, the scheme admits an interesting elaboration that incorporates the action of a controller: this is the subject of *hierarchical control theory*, to be considered in Chapt. 5.

To conclude this section we note for completeness' sake a slightly more fine-grained state realization of (L, P) in which the next output symbol corresponding to a state transition $q \xrightarrow{\sigma} q'$ in $\mathcal{M}(L, P)$ or $\mathcal{H}(L, P)$ becomes a function of the entrance state q' alone (as distinct from being a function of the pair (q, σ) – of course q' may be the entrance state for several other transitions too). Such a representation is more convenient than the Mealy description for graphical representation and certain data processing operations. For this, refine the equivalence $\mathrm{Fut}(P)$ to include the present:

$$\mathrm{Pfut}(P) := \mathrm{Fut}(P) \wedge \ker \omega_o$$

In detail,

$$s \equiv s' \pmod{\mathrm{Pfut}(P)} \text{ iff } s \equiv s' \pmod{\mathrm{Fut}(P)} \quad \& \quad \omega_o(s) = \omega_o(s')$$

It is easy to see that $\mathrm{Pfut}(P)$ is a right congruence on L, so we may define the right congruence

$$\mathrm{Moore}(L, P) := \mathrm{Nerode}(L) \wedge \mathrm{Pfut}(P)$$

All the previous considerations now apply with $\mathrm{Mealy}(L, P)$ replaced by Moore (L, P). It can be checked that the finer granularity of state description for $\mathcal{H}(L, P)$ is reflected in the property that the output map g of Prop. 2.7.2 now factors through the transition function (say $\delta : Y \times \Sigma \to Y$) of $\mathcal{H}(L, P)$: namely $\ker(g) \geq \ker(\delta)$, hence there exists a map $\phi : Q \to T_o$ such that $g = \phi \circ \delta$.

Exercise 2.7.1: Show how to include marked behavior in the foregoing discussion by 'marking' states of $\mathbf{G}$ with an auxiliary selfloop.

2.8 Notes and References

Most of the material in this chapter is standard. For Sects. 2.1-2.6 see especially Hopcroft & Ullman [1979]. Exercise 2.6.3 is adapted from Carroll & Long [1989], p. 123. Our distinction between "automaton" and "generator" is perhaps non-standard, but is helpful in control theory. Sect. 2.7 originates here, but is not used in the sequel.

Chapter 3

Supervision of Discrete-Event Systems: Basics

3.1 Introduction

Discrete-event systems encompass a wide variety of physical systems that arise in technology. These include manufacturing systems, traffic systems, logistic systems (for the distribution and storage of goods, or the delivery of services), database management systems, communication protocols, and data communication networks. Typically the processes associated with these systems may be thought of as discrete (in time and state space), asynchronous (event-driven rather than clock-driven), and in some sense generative (or nondeterministic). The underlying primitive concepts include events, conditions and signals.

Our approach in these notes will be to regard the discrete-event system to be controlled, i.e. the 'plant' in traditional control terminology, as the generator of a formal language. By adjoining control structure, it will be possible to vary the language generated by the system within certain limits. The desired performance of such a controlled generator will be specified by stating that its generated language must be contained in some specification language. It is often possible to meet this specification in an 'optimal', that is, minimally restrictive, fashion. The control problem will be considered fully solved when a controller that forces the specification to be met has been shown to exist and to be constructible. In accordance with widely accepted control methodology, we take the state description of a system (and, in this case, a language) to be fundamental.

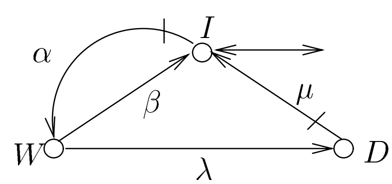
In parallel with the formal theory, we shall provide a guide to the software package TCT, which can be used for developing small-scale examples on a personal computer.

3.2 Representation of Controlled Discrete-Event Systems

The formal structure of a DES to be controlled is that of a *generator* in the sense of Sect. 2.5. As usual, let

$$\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$$

Here Σ is a finite alphabet of symbols that we refer to as *event labels*, Q is the *state set* (at most countable), $\delta : Q \times \Sigma \to Q$ is the (partial) *transition function*, q_o is the *initial state*, and $Q_m \subseteq Q$ is the subset of *marker states*. The *transition graph* shown below represents a primitive 'machine' named **MACH**, with 3 states, labelled I, W, D for 'idle', 'working' and 'broken down'.



In a transition graph the initial state is labelled with an entering arrow ($\to\!\circ$), while a state labelled with an exiting arrow ($\circ\!\to$) will denote a marker state. If the initial state is also a marker state, it may be labelled with a double arrow ($\leftrightarrow$). Formally a transition or *event* of $\mathbf{G}$ is a triple of the form (q, σ, q') where $\delta(q, \sigma) = q'$. Here $q, q' \in Q$ are respectively the *exit state* and the *entrance state*, while $\sigma \in \Sigma$ is the *event label*. The *event set* of $\mathbf{G}$ is just the set of all such triples.

For the alphabet Σ we have the partition

$$\Sigma = \Sigma_c \cup \Sigma_u$$

where the disjoint subsets Σ_c and Σ_u comprise respectively the *controllable* events and the *uncontrollable* events. In a transition graph a controllable event may be indicated by an optional tick on its transition arrow ($\circ\!\!\not\to\!\circ$). For **MACH**, $\Sigma_c = \{\alpha, \mu\}$ and $\Sigma_u = \{\beta, \lambda\}$. The mode of operation of a DES, of which **MACH** is typical, may be pictured as follows. Starting from state I, **MACH** executes a sequence of events in accordance with its transition graph. Each event is instantaneous in time. The events occur at quasi-random (unpredictable) time instants. Upon occurrence of an event, the event label is 'emitted' to some external agent. In this way **MACH** generates a string of event labels over the alphabet Σ. At a state such as W from which more than one event may occur, **MACH** will be considered to select just one of the possibilities, in accordance with some mechanism that is hidden from the system analyst and is therefore unmodelled. Such a mechanism could be 'forcing' by an external agent. In this sense the operation of **MACH** is nondeterministic. However, it will be assumed that the labelling of events is 'deterministic' in the sense that distinct events exiting from a given state always carry distinct labels. In general it may happen that two or more events exiting from distinct states may carry the same label. The marker states serve to distinguish those strings that have some special significance, for instance represent a completed work cycle or sequence of work cycles. The controllable event labels, in this case $\{\alpha, \mu\}$, label transitions that may be enabled or disabled by an external agent. A controllable event can occur only if it is enabled. Thus if the event (labelled) α is enabled, but not otherwise, **MACH** can execute the transition (I, α, W) to W from I; if **MACH** is at D, enablement of μ may be interpreted as the condition that **MACH** is under repair, and so may (eventually) execute the transition (D, μ, I). For brevity we shall often refer to 'the event σ', meaning any or all events (transitions) that happen to be labelled by σ.

The *TCT* procedure **create** allows the user to create and file a new DES. In response to the prompt, the user enters the DES name, number of states or *size*, the list of marker states and list of transitions (event triples). The *TCT* standard state set is the integer set $\{0, 1, ..., \text{size} - 1\}$, with 0 as the initial state. Event labels must be entered as integers between 0 and 999, where controllable events are odd and uncontrollable events are even. For instance **MACH** could be created as displayed below.

Example 3.2.1 (*TCT* procedure **create**)

Name? MACH
\# States? 3
% *TCT* selects standard state set {0,1,2}
Marker state(s)? 0
% User selects event labels {0,1,2,3}:
% events labelled 1 or 3 are controllable

Transitions?
0 1 1
1 0 0
1 2 2
2 3 0

◇

The *TCT* procedure **SE** (DES_name) displays an existing DES in approximately the format indicated above.

We recall from Chapter 2 that the languages associated with a DES **G** are the *closed behavior*

$$L(\mathbf{G}) = \{s \in \Sigma^* | \delta(q_o, s)!\}$$

and the *marked behavior*

$$L_m(\mathbf{G}) = \{s \in \Sigma^* | \delta(q_o, s) \in Q_m\}$$

Note that $\emptyset \subseteq L_m(\mathbf{G}) \subseteq L(\mathbf{G})$, and always $\epsilon \in L(\mathbf{G})$ (provided $\mathbf{G} \neq \mathbf{EMPTY}$, the DES with empty state set). The *reachable (state) subset* of $\mathbf{G}$ is

$$Q_r = \{q \in Q | (\exists s \in \Sigma^*) \delta(q_o, s) = q\};$$

$\mathbf{G}$ is *reachable* if $Q_r = Q$. The *coreachable subset* is

$$Q_{cr} = \{q \in Q | (\exists s \in \Sigma^*) \delta(q, s) \in Q_m\};$$

$\mathbf{G}$ is *coreachable* if $Q_{cr} = Q$. $\mathbf{G}$ is *trim* if it is both reachable and coreachable.

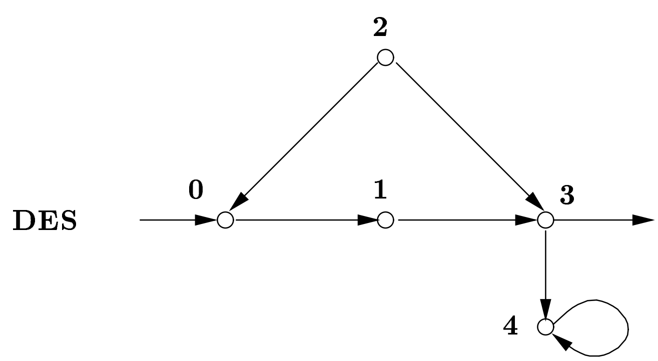
The *TCT* procedure **trim** returns the trimmed version of its argument:

$$\mathbf{DES_{new}} = \mathbf{trim}(\mathbf{DES})$$

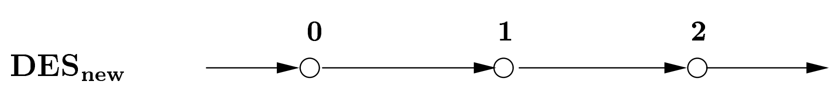
possibly after state recoding, as illustrated below.

Example 3.2.2 (*TCT* procedure **trim**)

$$\mathbf{DES_{new}} = \mathbf{trim}(\mathbf{DES})$$



$$Q_r = \{0, 1, 3, 4\}, \quad Q_{cr} = \{0, 1, 2, 3\}, \quad Q_{new} = Q_r \cap Q_{cr} = \{0, 1, 3\}$$



Note that state 3 in Q_{new} has been recoded as 2. $\diamond$

The DES $\mathbf{G}$ is *nonblocking* if every reachable state is coreachable, i.e.

$$\bar{L}_m(\mathbf{G}) = L(\mathbf{G})$$

In particular $\mathbf{G}$ is nonblocking if it is trim. If $K \subseteq \Sigma^*$ then $\mathbf{G}$ *represents* K if $\mathbf{G}$ is nonblocking and $L_m(\mathbf{G}) = K$. Then $L(\mathbf{G}) = \bar{K}$, although $\mathbf{G}$ might possibly be non-coreachable. Normally, if $\mathbf{G}$ is intended to represent K, it is taken to be both reachable and coreachable (i.e. trim).

3.3 Synchronous Product, Shuffle, and Meet

In this section we describe a way of combining several DES into a single, more complex DES. The technique will be standard for the specification of control problems involving the coordination or synchronization of several DES together. We define the operations required on languages, and then the counterpart *TCT* operations on their generators.

Let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, where it is allowed that $\Sigma_1 \cap \Sigma_2 \neq \emptyset$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$. Define

$$P_i : \Sigma^* \to \Sigma_i^* \quad (i = 1, 2)$$

according to

$$\begin{aligned} P_i(\epsilon) &= \epsilon \\ P_i(\sigma) &= \begin{cases} \epsilon & \text{if} \quad \sigma \notin \Sigma_i \\ \sigma & \text{if} \quad \sigma \in \Sigma_i \end{cases} \\ P_i(s\sigma) &= P_i(s)P_i(\sigma) \quad s \in \Sigma^*, \sigma \in \Sigma \end{aligned}$$

Clearly $P_i(st) = P_i(s)P_i(t)$, i.e. P_i is *catenative*. The action of P_i on a string s is just to erase all occurrences of σ in s such that $\sigma \notin \Sigma_i$. P_i is the *natural projection* of Σ^* onto Σ_i^*. For $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ we define the *synchronous product* $L_1 \| L_2 \subseteq \Sigma^*$ according to

$$L_1 \| L_2 = P_1^{-1} L_1 \cap P_2^{-1} L_2$$

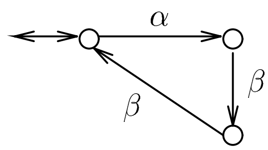
Thus $s \in L_1 \| L_2$ iff $P_1(s) \in L_1$ and $P_2(s) \in L_2$. If $L_1 = L_m(\mathbf{G_1})$ and $L_2 = L_m(\mathbf{G_2})$, one can think of $\mathbf{G_1}$ and $\mathbf{G_2}$ as generating $L_1 \| L_2$ 'cooperatively' by agreeing to synchronize those events with labels σ which they possess in common.

The *TCT* procedure **sync** returns $\mathbf{G} = \mathbf{sync}(\mathbf{G_1}, \mathbf{G_2})$ where

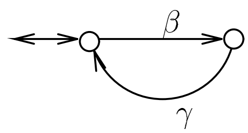
$$L_m(\mathbf{G}) = L_m(\mathbf{G_1}) \| L_m(\mathbf{G_2}), \quad L(\mathbf{G}) = L(\mathbf{G_1}) \| L(\mathbf{G_2})$$

Example 3.3.1 (Synchronous product)

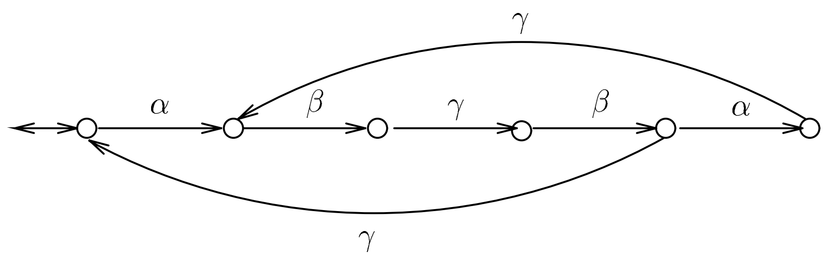
$\Sigma_1 = \{\alpha, \beta\}$ $\qquad L_m(\mathbf{G_1})$



$\Sigma_2 = \{\beta, \gamma\}$ $\qquad L_m(\mathbf{G_2})$



$L_m(\mathbf{G_1}) \| L_m(\mathbf{G_2})$



Exercise 3.3.1: Show that, in general,

$$L_1 \| L_2 = (L_1 \| (\Sigma_2 - \Sigma_1)^*) \cap (L_2 \| (\Sigma_1 - \Sigma_2)^*)$$

Notation: If μ, ν are binary relations on a set X, then $\mu \circ \nu$ is the relation given by

$$x(\mu \circ \nu)x' \quad \text{iff} \quad (\exists x'')x\mu x'' \quad \& \quad x''\nu x'$$

Exercise 3.3.2: With $\Sigma_1 \cup \Sigma_2 \subseteq \Sigma$ and $P_i : \Sigma^* \to \Sigma_i^*$ $(i = 1, 2)$ natural projections, show that $P_1 P_2 = P_2 P_1$, and that

$$\begin{aligned} \ker P_1 \vee \ker P_2 &= \ker(P_1 P_2) \\ &= (\ker P_1) \circ (\ker P_2) \end{aligned}$$

Exercise 3.3.3: Show that synchronous product is associative, namely

$$(L_1 \| L_2) \| L_3 = L_1 \| (L_2 \| L_3)$$

where L_i is defined over Σ_i and the Σ_i bear no special relationship to one another. $\diamond$

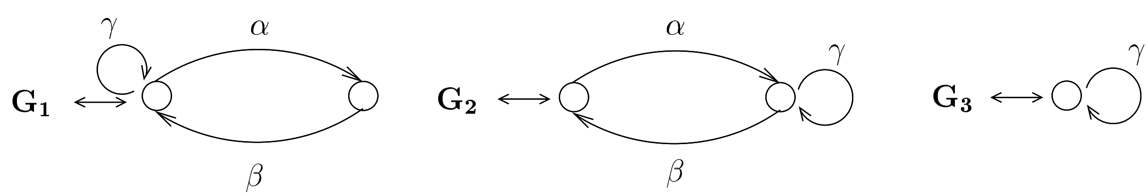
It is well to note a subtlety in the definition of synchronous product. While it must be true that Σ_i includes all the event labels that explicitly appear in $\mathbf{G}_i$, it may be true that some label in Σ_i does not appear in $\mathbf{G}_i$ at all. If $\sigma \in \Sigma_1 \cap \Sigma_2$ does not actually appear in $\mathbf{G}_2$ but may appear in $\mathbf{G}_1$, then **sync** will cause $\mathbf{G}_2$ to block σ from appearing anywhere in $\mathbf{G}$. Thus if, in Example 3.3.1, Σ_2 is redefined as $\{\alpha, \beta, \gamma\}$, then α is blocked, with the result that now

$$L_m(\mathbf{G}_1) \| L_m(\mathbf{G}_2) = \{\epsilon\}$$

Thus in general $L_1 \| L_2$ depends critically on the specification of Σ_1, Σ_2. Currently *TCT* implements **sync** by taking for Σ_i exactly the events that appear in $\mathbf{G}_i$.

Exercise 3.3.4: Nonassociativity of *TCT* sync

The *TCT* implementation **sync** of synchronous product need not respect associativity, since the events appearing in **sync** $(\mathbf{G_1}, \mathbf{G_2})$ may form a proper subset of $\Sigma_1 \cup \Sigma_2$. Consider



Check that

$$\mathbf{sync}((\mathbf{sync}(\mathbf{G}_1, \mathbf{G}_2), \mathbf{G}_3)$$

$$\neq \mathbf{sync}(\mathbf{G}_1, \mathbf{sync}(\mathbf{G}_2, \mathbf{G}_3))$$

and explain why. $\diamond$

TCT will warn the user if some event in $\Sigma_1 \cup \Sigma_2$ fails to appear in the synchronous product: such an event is "blocked". This remedy was deemed preferable to maintaining a separate event list for a DES throughout its history. For more on this issue see Exercise 3.3.7 below.

Exercise 3.3.5: For alphabets Σ_0, Σ_1, Σ_2 with $\Sigma_0 \subseteq \Sigma_1 \cup \Sigma_2$, let

$$L_1 \subseteq \Sigma_1^*, \qquad L_2 \subseteq \Sigma_2^*$$

and let

$$P_0 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_0^*$$

be the natural projection. Show that

$$P_0(L_1 \| L_2) \subseteq (P_0 L_1) \| (P_0 L_2)$$

always, and that equality holds provided

$$\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_0,$$

namely "every shared event is observable under P_0". Here consider $P_0 L_i \subseteq (\Sigma_0 \cap \Sigma_i)^*$.

$\diamond$

Assume now that $\Sigma_1 \cap \Sigma_2 = \emptyset$. With L_1, L_2 as before the *shuffle product* $L_1 \| L_2$ is defined to be the synchronous product for this special case. Thus the shuffle product of two languages L_1, L_2 over disjoint alphabets is the language consisting of all possible interleavings ('shuffles') of strings of L_1 with strings of L_2. One can think of $\mathbf{G_1}$ and $\mathbf{G_2}$ as generating $L_1 \| L_2$ by independent and asynchronous generation of L_1 and L_2 respectively.

In these notes we may write **shuffle** to stand for the *TCT* procedure **sync** when the component alphabets Σ_1 and Σ_2 are disjoint.

Under the latter assumption, the *TCT* procedure **selfloop** with arguments $\mathbf{G}_1, \Sigma_2$ returns $\mathbf{G} = \mathbf{selfloop}(\mathbf{G_1}, \Sigma_2)$, where

$$L(\mathbf{G}) = L(\mathbf{G_1}) \| \Sigma_2^* = P_1^{-1} L(\mathbf{G_1}) \subseteq (\Sigma_1 \cup \Sigma_2)^*, \quad L_m(\mathbf{G}) = L_m(\mathbf{G_1}) \| \Sigma_2^*$$

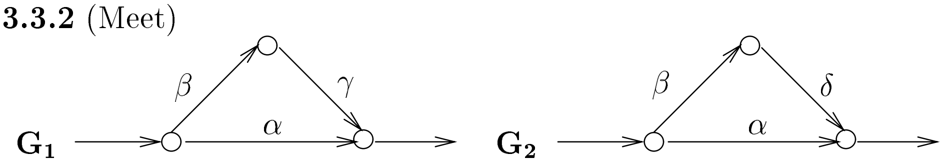
As its name suggests, **selfloop** forms $\mathbf{G}$ by attaching a transition (q, σ, q) at each state q of $\mathbf{G_1}$ for each label σ of Σ_2.

For DES $\mathbf{G_1}$ and $\mathbf{G_2}$, the *TCT* procedure **meet** returns a reachable DES $\mathbf{G} = \mathbf{meet}(\mathbf{G_1}, \mathbf{G_2})$ such that

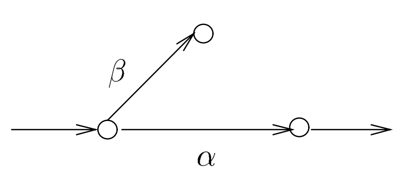
$$L_m(\mathbf{G}) = L_m(\mathbf{G_1}) \cap L_m(\mathbf{G_2}), \quad L(\mathbf{G}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$$

Thus **meet** is really the special case of **sync** corresponding to $\Sigma_1 = \Sigma_2$, namely all events are considered shared and synchronization is total. In particular, **meet** will block any event whose label does not occur in *both* $\mathbf{G_1}$ and $\mathbf{G_2}$. Note that $\bar{L}_m(\mathbf{G})$ may be a proper sublanguage of $L(\mathbf{G})$, even when each of $\mathbf{G_1}$ and $\mathbf{G_2}$ is trim.

Example 3.3.2 (Meet)



$\mathbf{G} = \mathbf{meet}(\mathbf{G_1}, \mathbf{G_2})$



While $L_m(\mathbf{G}) = \{\alpha\} = L_m(\mathbf{G_1}) \cap L_m(\mathbf{G_2})$ and $L(\mathbf{G}) = \{\epsilon, \alpha, \beta\} = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$, nevertheless $\bar{L}_m(\mathbf{G}) \subsetneqq L(\mathbf{G})$ even though each of $\mathbf{G_1}$ and $\mathbf{G_2}$ is trim. $\diamond$

Exercise 3.3.6: The *TCT* procedure **meet** is implemented for DES **G1** and **G2** as follows. Let $\mathbf{Gi} = (Q_i, \Sigma, \delta_i, q_{oi}, Q_{mi})$, $i = 1, 2$. First define the product $\mathbf{G1} \times \mathbf{G2} = (Q, \Sigma, \delta, q_o, Q_m)$, where $Q = Q_1 \times Q_2$, $\delta = \delta_1 \times \delta_2$, $q_o = (q_{o1}, q_{o2})$, and $Q_m = Q_{m1} \times Q_{m2}$, with

$$(\delta_1 \times \delta_2)((q_1, q_2), \sigma) := (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$$

whenever $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$. In other words the product is defined like the product of two automata, due account being taken of the fact that the component transition functions are partial functions, hence the product transition function is a partial function as well. *TCT* now generates $\mathbf{G} = \mathbf{meet}(\mathbf{G1}, \mathbf{G2})$ as the reachable sub-DES of $\mathbf{G1} \times \mathbf{G2}$, and will number the states from 0 to Size-1 (in some arbitrary fashion) as usual. Note that one can think of $\mathbf{G}$ as a structure that is capable of tracking strings that can be generated by **G1** and **G2** in common, when each starts at its initial state. Calculate by hand the meet of two DES and check your result using *TCT*.

Exercise 3.3.7: With reference to the $\mathbf{G}_i$ of Exercise 3.3.4, adopt the *TCT* specification

$$\Sigma_1 = \Sigma_2 = \{\alpha, \beta, \gamma\}, \ \Sigma_3 = \{\gamma\}$$

and write $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$. Show that the synchronous product of languages,

$$L_m(\mathbf{G_1}) \parallel L_m(\mathbf{G_2}) \parallel L_m(\mathbf{G_3})$$

is represented by

$$\mathbf{meet}\ (\mathbf{G}_1', \mathbf{G}_2', \mathbf{G_3}') \qquad (*)$$

where $\mathbf{G_i'} =$ **selfloop** $(\mathbf{G}_i, \Sigma - \Sigma_i)$. Here

$$\mathbf{meet}(\mathbf{F}, \mathbf{G}, \mathbf{H}) := \mathbf{meet}(\mathbf{meet}(\mathbf{F}, \mathbf{G}), \mathbf{H})$$

is always independent of the order of arguments. Thus $(*)$, and its generalization to k arguments, provides a correct, order-independent (hence, associative) implementation of synchronous product, as long as *all* the relevant $\mathbf{G_i}$ are specified in advance.

Which of the two results in Exercise 3.3.4 agrees with $(*)$? Explain.

Exercise 3.3.8: Let $L \subseteq \Sigma^*$. Writing

$$L = \left[L \cup \left(\Sigma^* - \bar{L}\right)\right] \cap \bar{L}$$

show that the two languages intersected on the right are represented by generator DES in which, respectively, all events are enabled (i.e. can occur) at each state, and every state is marked. The former places no constraints on local choice but does distinguish 'successful' (marked) strings from others, whereas the latter declares every string to be 'successful' but constrains event choice. For $\Sigma = \{\alpha, \beta\}, L = \alpha(\beta\alpha)^*$, illustrate by drawing the state diagrams and check using **meet**.

Exercise 3.3.9: The *prioritized synchronous product* of languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ can be defined informally as follows. Let L_i be represented by generator $\mathbf{G_i} = (Q_i, \Sigma_i, \delta_i, \_, \_)$. Let $\alpha \in \Sigma_1 \cap \Sigma_2$. To assign $\mathbf{G_1}$ 'priority' over $\mathbf{G_2}$ with respect to α declare that, in $\mathbf{G_1} \times \mathbf{G_2}$,

$$\delta((q_1, q_2), \alpha) = \begin{cases} (\delta_1(q_1, \alpha), \delta_2(q_2, \alpha)) \text{ if } \delta_1(q_1, \alpha)! \ \& \ \delta_2(q_2, \alpha)! \\ (\delta(q_1, \alpha), q_2) \text{ if } \delta_1(q_1, \alpha)! \ \& \ \textit{not } \delta_2(q_2, \alpha)! \\ \text{undefined, otherwise} \end{cases}$$

In other words, $\mathbf{G_1}$ may execute α whenever it can; $\mathbf{G_2}$ synchronizes with $\mathbf{G_1}$ if it can, but otherwise exercises no blocking action and makes no state change. The definition of δ may be completed in the evident way, corresponding to a three-fold partition of $\Sigma_1 \cap \Sigma_2$ into events prioritized for $\mathbf{G_1}$ (resp. $\mathbf{G_2}$), or non-prioritized (as in ordinary synchronous product). For simplicity assume that only $\alpha \in \Sigma_1 \cap \Sigma_2$ is prioritized, say for $\mathbf{G_1}$. Denote the required product by $\mathbf{psync}(\mathbf{G_1}, \mathbf{G_2})$. To implement this in *TCT*, extend $\mathbf{G_2}$ to $\mathbf{G_2'}$ according to

$$\delta_2'(q_2, \alpha) = \begin{cases} \delta_2(q_2, \alpha) \text{ if } \delta_2(q_2, \alpha)! \\ q_2 \text{ if } \textit{not } \delta_2(q_2, \alpha)! \end{cases}$$

Then

$$\mathbf{psync}(\mathbf{G_1}, \mathbf{G_2}) = \mathbf{sync}(\mathbf{G_1}, \mathbf{G_2'})$$

Illustrate this construction using Example 3.3.1, assigning $\mathbf{G_1}$ priority with respect to β, and compare the result with that of $\mathbf{sync}(\mathbf{G_1}, \mathbf{G_2})$. ◇

Remark: Since *TCT* recodes the states of a product structure generated by **sync** or **meet** into sequential format, information about component states is discarded. To retain such information one can use auxiliary selfloops as 'flags'. For instance, to display which states of **G3 = sync(G1,G2)** correspond to states (1,2) or (6,2) of **G1** $\times$ **G2**, first modify **G1** and **G2** by selflooping states 1 and 6 of **G1** and state 2 of **G2** with a new flag event σ_{flag}. After recoding, the selected product states will appear selflooped in **G3**. ◇

For use in the next example we introduce *TCT* **project**. For a DES **G** over Σ, let $\Sigma_o \subseteq \Sigma$, $\Sigma_{\text{null}} := \Sigma - \Sigma_o$, and $P : \Sigma^* \to \Sigma_0^*$ the natural projection. Then **project** $(\mathbf{G}, \Sigma_{\text{null}})$ returns a (minimal) DES **PG** over Σ_o such that

$$L_m(\mathbf{PG}) = PL_m(\mathbf{G}), \quad L(\mathbf{PG}) = PL(\mathbf{G})$$

Example 3.3.3: KANBAN

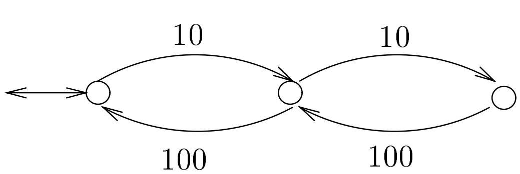
This example illustrates the usefulness of synchronous product (*TCT* **sync**) in building up complex systems. **KANBAN** is an instance of a Kanban production system. We consider just two workcells, say **CELL1** and **CELL2**, indexing from output (right-hand) end of the system to input (left-hand) end. **CELLi** consists of an output hopper **HOPi**, and input bulletin-board **BBi** for kanbans (cards to signal request for input), a feeder queue **Qi** for processor machine **Mi**, and **Mi** itself. Information (via kanbans) circulates in the same order. We model each storage item **HOPi**, **BBi**, **Qi** as a 2-slot buffer. **CELLi** = **sync (HOPi,BBi,Qi,Mi)** (9,14) and **KANBAN** = **sync(CELL1,CELL2)** (81,196), where integer pairs (n, m) denote the number n of states and m of transitions in the DES.

Customer requests for product are sensed at **CELL1** and modelled by event 100. Requests are either blocked, if **HOP1** is empty, or satisfied instantaneously by decrementing **HOP1**. When **HOP1** is decremented, a card is transferred to **BB1**, incrementing **BB1** (doubly synchronized event 100). A card in **BB1** represents a signal to **CELL2** that an input part is required for processing. If **HOP2** is empty, nothing happens. Otherwise, **HOP2** and **BB1** are decremented and **Q1** is incremented, by the 4-tuply synchronized event 12. Parts are taken from **Q1** by **M1** for processing (event 11) and the finished parts deposited in **HOP1** (event 10). In **CELL2** the action is similar: event 12 increments **BB2**; if **BB2** is nonempty a raw part can be taken in by **CELL2**, causing **BB2** to be decremented and **Q2** incremented (doubly synchronized event 22). **M2** deposits its output part in **HOP2** (doubly synchronized event 20).

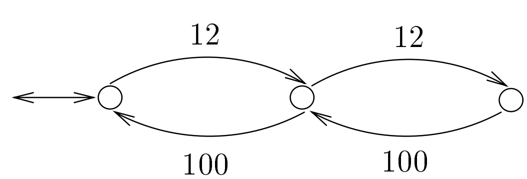
To display the overall input/output structure, let

$$\textbf{PKANBAN} = \textbf{project}\ (\textbf{KANBAN},[10,11,12,20,21]) \quad (5,8)\ .$$

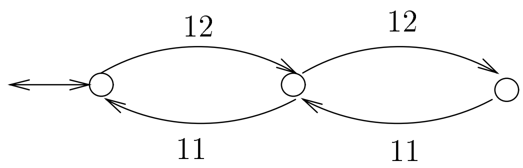
The result is just a 4-slot buffer that is incremented by event 22 (raw part input) and decremented by event 100 (customer request filled). Notice that no more than 4 parts will be in progress in the system (WIP ≤ 4) at one time. If the system is initialized with **BB1**, **BB2** both full and **HOP1**, **Q1**, **HOP2**, **Q2** all empty, then initially requests are blocked and production must begin at the input end.



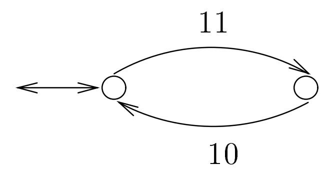
HOP1



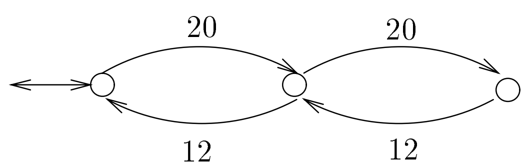
BB1



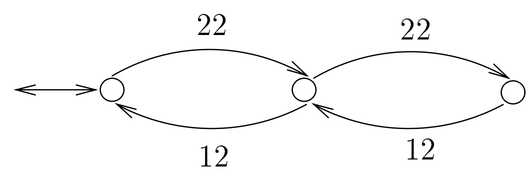
Q1



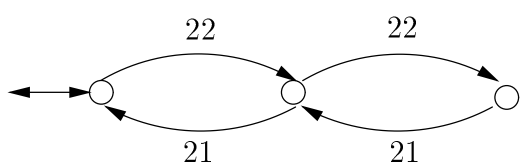
M1



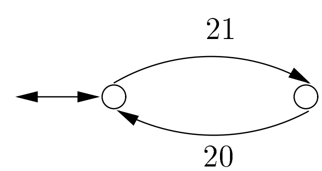
HOP2



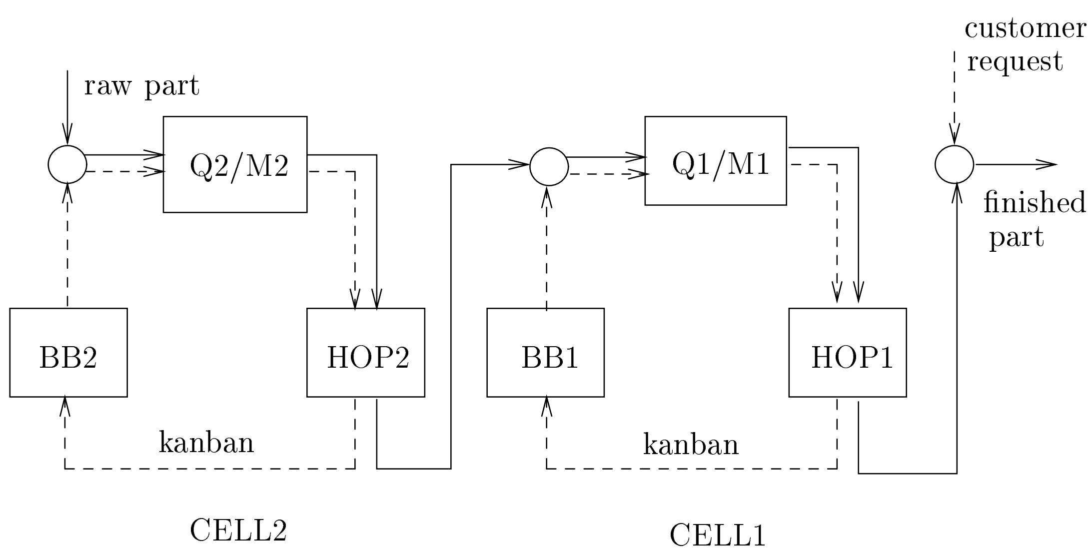
BB2



Q2



M2



KANBAN

Exercise 3.3.10: Consider the DES **TOY** defined as follows.

#states: 3; state set: 0...2; initial state: 0; marker state: 0.

transition table:

[0,0,1], [0,1,1], [0,2,2], [1,1,2], [1,2,0], [1,3,1], [2,1,0], [2,2,1], [2,3,2].

(i) Construct **PTOY** = **project**(**TOY**,[0]) by hand and check your answer using *TCT*.

(ii) If you did things right, you should find that **PTOY** has 5 states, i.e. the state size of the projected DES is larger (cf. also Exercise 2.5.5). Since it can be shown using **minstate** that both **TOY** and **PTOY** have the minimal possible number of states, this increase in state size is a modelling "reality", and not just a result of inefficient representation. Provide an intuitive explanation for what's going on. It may help to give **TOY** some physical interpretation.

(iii) Call the preceding DES **TOY_3** and generalize to **TOY_N** on state set $\{0, 1, ..., N-1\}$, with $3N$ transitions:

$$\begin{aligned} &[0,0,1] \\ [k,1,k+1]\,,&\quad k=0,1,...,N-1 \\ [k+1,2,k]\,,&\quad k=0,1,...,N-1 \\ [k,3,k]\,,&\quad k=1,2,...,N-1 \end{aligned}$$

where the indexing is mod(N). Let

$$\mathbf{PTOY\_N} = \mathbf{project}(\mathbf{TOY\_N}, [0])$$

Find a formula for the state size of **PTOY_N** and verify it computationally for $N = 3, 4, ..., 20$. For $N = 20$ the answer is 786431. **Hint:** To calculate the result of **project** by hand, first replace each transition with label in the NULL list by a 'silent' transition labelled, say, λ, where $\lambda \notin \Sigma$. Next apply a variant of the subset construction to obtain a deterministic model that is λ-free: the initial 'state' is the subset reachable from 0 on paths labelled by λ only; the next 'state', following σ, say (with $\sigma \in \Sigma-$NULL), is the subset reachable on paths of form $\lambda^*\sigma\lambda^*$; and so on. ◇

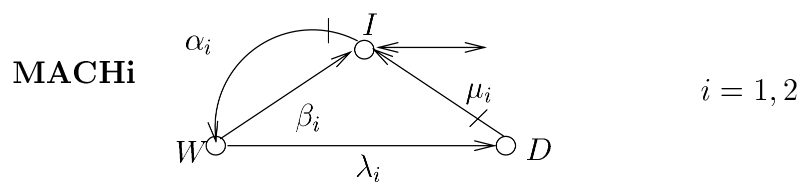
For later reference we define the *TCT* procedure **complement**. Assume that the DES $\mathbf{G}$ contains in its description exactly the symbols of some alphabet Σ, and let $T \supseteq \Sigma$. Then $\mathbf{G_{co}}$ = **complement**($\mathbf{G}, T - \Sigma$) has the properties

$$L_m(\mathbf{G_{co}}) = T^* - L_m(\mathbf{G}), \quad L(\mathbf{G_{co}}) = T^*$$

If $T = \Sigma$ we write simply $\mathbf{G_{co}}$ = **complement**($\mathbf{G}$, __). In terms of transition structures, **complement** forms $\mathbf{G_{co}}$ by adjoining a (non-marker) dump state q_+ to the state set of $\mathbf{G}$, and complementary transitions from each state of $\mathbf{G_{co}}$ to q_+ as required to render the new transition function a total function; the subsets of marker and non-marker states are then interchanged.

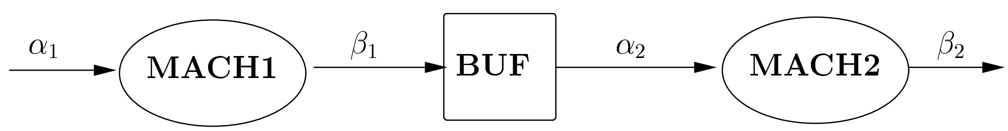
Example 3.3.4: Small Factory

To conclude this section we show how the foregoing procedures can be used to build up the specifications for a control problem, to be known as Small Factory. We bring in two 'machines' **MACH1**, **MACH2** as shown.



Define **FACT** = **shuffle**(**MACH1**, **MACH2**). Small Factory consists of the arrangement shown below, where **BUF** denotes a buffer with one slot.

Small Factory



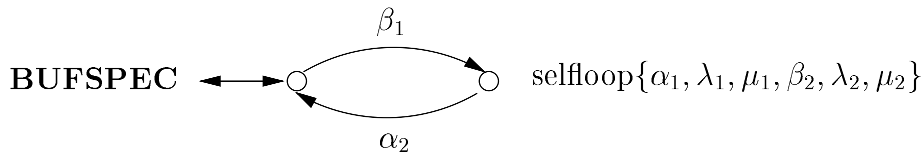
Small Factory operates as follows. Initially the buffer is empty. With the event α_1, **MACH1** takes a workpiece from an infinite input bin and enters W. Subsequently **MACH1** either breaks down and enters D (event λ_1), or successfully completes its work cycle, deposits the workpiece in the buffer, and returns to I (event β_1). **MACH2** operates similarly, but takes its workpiece from the buffer and deposits it when finished in an infinite output bin. If a machine breaks down, then on repair it returns to I (event μ).

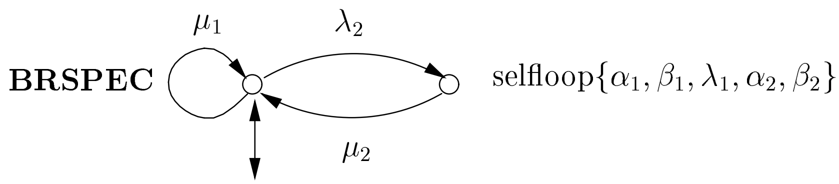
The informal specifications for admissible operation are the following:

1. The buffer must not overflow or underflow.

2. If both machines are broken down, then **MACH2** must be repaired before **MACH1**.

To formalize these specifications we bring in two language generators as the DES **BUFSPEC** and **BRSPEC**, as shown below.





$L(\mathbf{BUFSPEC})$ expresses the requirement that β_1 and α_2 must occur alternately, with β_1 occurring first, while $L(\mathbf{BRSPEC})$ requires that if λ_2 occurs then μ_1 may not occur (again) until μ_2 occurs. The assignment of the initial state as a marker state in each of these DES is largely a matter of convenience. In each case selfloops must be adjoined to account for all events that are irrelevant to the specification but which may be executed in the plant. For the combined specification we form

$$\mathbf{SPEC} = \mathbf{meet}(\mathbf{BUFSPEC}, \mathbf{BRSPEC})$$

It is clear that **SPEC** is trim.

Temporarily denote by **G** the (as yet unknown) DES that would represent '**FACT** under control'. In general, for a given DES **G** and given specification DES **SPEC** as above, we shall say that **G** *satisfies* **SPEC** if

$$L_m(\mathbf{G}) \subseteq L_m(\mathbf{SPEC})$$

Typically **G** and **SPEC** will both be trim, and then it follows on taking closures,

$$L(\mathbf{G}) \subseteq L(\mathbf{SPEC}).$$

The first condition could be checked in *TCT* by computing

$$\mathbf{COSPEC} = \mathbf{complement}(\mathbf{SPEC}, \_\_)$$

and then verifying that $\mathbf{trim}(\mathbf{meet}(\mathbf{G}, \mathbf{COSPEC})) = \mathbf{EMPTY}$, where **EMPTY** is the DES with empty state set (it suffices to check that $\mathbf{meet}(\mathbf{G}, \mathbf{COSPEC})$ has empty marker set). The results for Small Factory will be presented in a later section.

3.4 Controllability and Supervision

Let

$$\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$$

be a (nonempty) controlled DES, with $\Sigma = \Sigma_c \cup \Sigma_u$ as in Section 3.2. A particular subset of events to be enabled can be selected by specifying a subset of controllable events. It is convenient to adjoin with this all the uncontrollable events as these are automatically enabled. Each such subset of events is a *control pattern*; and we introduce the set of all control patterns

$$\Gamma = \{\gamma \in Pwr(\Sigma) \mid \gamma \supseteq \Sigma_u\}$$

A *supervisory control for* $\mathbf{G}$ is any map $V : L(\mathbf{G}) \to \Gamma$. The pair $(\mathbf{G}, V)$ will be written $V/\mathbf{G}$, to suggest '$\mathbf{G}$ under the supervision of V'. The *closed behavior* of $V/\mathbf{G}$ is defined to be the language $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ described as follows.

(i) $\epsilon \in L(V/\mathbf{G})$

(ii) If $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\mathbf{G})$ then $s\sigma \in L(V/\mathbf{G})$

(iii) No other strings belong to $L(V/\mathbf{G})$.

We always have $\{\epsilon\} \subseteq L(V/\mathbf{G}) \subseteq L(\mathbf{G})$, with either bound a possibility depending on V and $\mathbf{G}$. Clearly $L(V/\mathbf{G})$ is nonempty and closed.

The *marked behavior* of $V/\mathbf{G}$ is

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G})$$

Thus the marked behavior of $V/\mathbf{G}$ consists exactly of the strings of $L_m(\mathbf{G})$ that 'survive' under supervision by V. We always have $\emptyset \subseteq L_m(V/\mathbf{G}) \subseteq L_m(\mathbf{G})$.

We say that V is *nonblocking* (for $\mathbf{G}$) if

$$\bar{L}_m(V/\mathbf{G}) = L(V/\mathbf{G})$$

Our main objective is to characterize those languages that qualify as the marked behavior of some supervisory control V. To this end we define a language $K \subseteq \Sigma^*$ to be *controllable* (*with respect to* $\mathbf{G}$) if

$$(\forall s, \sigma) s \in \bar{K} \quad \& \quad \sigma \in \Sigma_u \quad \& \quad s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \bar{K}$$

In other words, K is controllable if and only if no $L(\mathbf{G})$-string that is already a prefix of K, when followed by an uncontrollable event in $\mathbf{G}$, thereby exits from the prefixes of K: the prefix closure $\bar{K}$ is invariant under the occurrence in $\mathbf{G}$ of uncontrollable events.

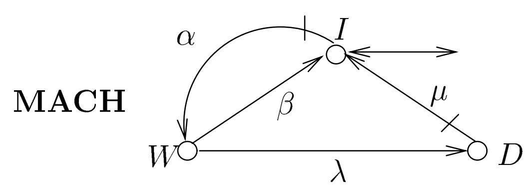
For a more concise statement, we use the following notation. If $S \subseteq \Sigma^*$ and $\Sigma_o \subseteq \Sigma$, let $S\Sigma_o$ denote the set of strings of form $s\sigma$ with $s \in S$ and $\sigma \in \Sigma_o$. Then K is controllable iff

$$\bar{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{K}$$

It is clear that $\emptyset$, $L(\mathbf{G})$ and Σ^* are always controllable with respect to $\mathbf{G}$.

Note that the controllability condition constrains only $\bar{K} \cap L(\mathbf{G})$, since if $s \notin L(\mathbf{G})$ then $s\sigma \notin L(\mathbf{G})$, i.e. the condition $s \in \bar{K} - L(\mathbf{G})$ & $s\sigma \in L(\mathbf{G})$ is always false.

Example 3.4.1 (Controllable and uncontrollable languages)



$$\Sigma_c = \{\alpha, \mu\}, \quad \Sigma_u = \{\beta, \lambda\}$$

With respect to **MACH**, $L' = \{\alpha\lambda\mu\}$ is not controllable, since $\alpha\beta$ consists of a prefix α of L' followed by an uncontrollable event β such that $\alpha\beta$ belongs to $L(\mathbf{MACH})$ but not to $\bar{L}'$. On the other hand $L'' = \{\alpha\beta, \alpha\lambda\}$ is controllable, since none of its prefixes $s \in \bar{L}'' = \{\epsilon, \alpha, \alpha\beta, \alpha\lambda\}$ can be followed by an uncontrollable event σ such that $s\sigma$ belongs to $L(\mathbf{MACH}) - \bar{L}''$.

Exercise 3.4.1: Assume that a language K is controllable with respect to a DES $\mathbf{G}$. Show that if $s \in \bar{K}$, $w \in \Sigma_u^*$, and $sw \in L(\mathbf{G})$, then $sw \in \bar{K}$. Suggestion: use structural induction on w. $\diamond$

Let $K \subseteq L \subseteq \Sigma^*$. The language K is L-*closed* if $K = \bar{K} \cap L$. Thus K is L-closed provided it contains every one of its prefixes that belong to L.

We can now present our first main result.

Theorem 3.4.1

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. There exists a nonblocking supervisory control V for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$ if and only if

(i) K is controllable with respect to $\mathbf{G}$, and

(ii) K is $L_m(\mathbf{G})$-closed.

Proof

(If) We have $\bar{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{K}$ together with $\bar{K} \subseteq \bar{L}_m(\mathbf{G}) \subseteq L(\mathbf{G})$. Furthermore $\epsilon \in \bar{K}$ since $K \neq \emptyset$. For $s \in \bar{K}$ define $V(s) \in \Gamma$ according to

$$V(s) = \Sigma_u \cup \{\sigma \in \Sigma_c | s\sigma \in \bar{K}\}$$

We claim that $L(V/\mathbf{G}) = \bar{K}$. First we show that $L(V/\mathbf{G}) \subseteq \bar{K}$. Suppose $s\sigma \in L(V/\mathbf{G})$, i.e., $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\mathbf{G})$. Assuming inductively that $s \in \bar{K}$ we have that $\sigma \in \Sigma_u$ implies $s\sigma \in \bar{K}\Sigma_u \cap L(\mathbf{G})$, so that $s\sigma \in \bar{K}$ (by controllability); whereas $\sigma \in \Sigma_c$ implies that $s\sigma \in \bar{K}$ by definition of $V(s)$. For the reverse inclusion, suppose $s\sigma \in \bar{K}$; thus $s\sigma \in L(\mathbf{G})$. Assuming inductively that $s \in L(V/\mathbf{G})$ we have that $\sigma \in \Sigma_u$ automatically implies that $\sigma \in V(s)$, so that $s\sigma \in L(V/\mathbf{G})$; while $\sigma \in \Sigma_c$ and $s\sigma \in \bar{K}$ imply that $\sigma \in V(s)$, so $s\sigma \in L(V/\mathbf{G})$. The claim is proved. Finally

$$\begin{aligned} L_m(V/\mathbf{G}) &= L(V/\mathbf{G}) \cap L_m(\mathbf{G}) \quad \text{(by definition)} \\ &= \bar{K} \cap L_m(\mathbf{G}) \\ &= K \quad \text{(since } K \text{ is } L_m(\mathbf{G})\text{-closed)} \end{aligned}$$

and $\bar{L}_m(V/\mathbf{G}) = \bar{K} = L(V/\mathbf{G})$, so $V/\mathbf{G}$ is nonblocking for $\mathbf{G}$.

(Only if) Let V be a supervisory control for $\mathbf{G}$ with $L_m(V/\mathbf{G}) = K$. Assuming that V is nonblocking for $\mathbf{G}$ we have $L(V/\mathbf{G}) = \bar{K}$, so

$$K = L(V/\mathbf{G}) \cap L_m(\mathbf{G}) = \bar{K} \cap L_m(\mathbf{G})$$

i.e. K is $L_m(\mathbf{G})$-closed. To show that K is controllable let $s \in \bar{K}$, $\sigma \in \Sigma_u$, $s\sigma \in L(\mathbf{G})$. Then $s \in L(V/\mathbf{G})$ and $\sigma \in V(s)$. So $s\sigma \in L(V/\mathbf{G}) = \bar{K}$, i.e.

$$\bar{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{K}$$

as required. □

Corollary

Let $K \subseteq L(\mathbf{G})$ be nonempty and closed. There exists a supervisory control V for $\mathbf{G}$ such that $L(V/\mathbf{G}) = K$ if and only if K is controllable with respect to $\mathbf{G}$. □

For brevity we refer to a nonblocking supervisory control (for $\mathbf{G}$, understood) as an NSC. It is useful to introduce a slight generalization of NSC in which the supervisory action includes marking as well as control. For this, let $M \subseteq L_m(\mathbf{G})$. Define a *marking nonblocking supervisory control for the pair* $(M, \mathbf{G})$, or *MNSC*, as a map $V : L(\mathbf{G}) \to \Gamma$ exactly as before; but now for the marked behavior of $V/\mathbf{G}$ we define

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M.$$

One may think of the marking action of the MNSC V as carried out by a recognizer for M that monitors the closed behavior of $V/\mathbf{G}$, sounding a beep exactly when a string in M has been generated. As a sublanguage of $L_m(\mathbf{G})$, these strings could be thought of as representing a subset of the 'tasks' that $\mathbf{G}$ (or its underlying physical referent) is supposed to accomplish. For instance in Small Factory, one might define a 'batch' to consist of 10 fully processed workpieces. M might then be taken as the set of strings that represent the successful processing of N integral batches, $N \geq 0$, with both machines returned to the I(dle) state and the buffer empty.

The counterpart result to Theorem 3.4.1 actually represents a simplification, as the condition of $L_m(\mathbf{G})$-closedness can now be dropped.

Theorem 3.4.2

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. There exists an MNSC V for $(K, \mathbf{G})$ such that

$$L_m(V/\mathbf{G}) = K$$

if and only if K is controllable with respect to $\mathbf{G}$.

Proof

(If) With V defined as in the proof of Theorem 3.4.1, it may be shown as before that $L(V/\mathbf{G}) = \bar{K}$. Then

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap K = \bar{K} \cap K = K$$

so that $\bar{L}_m(V/\mathbf{G}) = \bar{K} = L(V/\mathbf{G})$, namely V is nonblocking for $\mathbf{G}$.

(Only if) We have $\bar{K} = \bar{L}_m(V/\mathbf{G}) = L(V/\mathbf{G})$. Then the proof that K is controllable is unchanged from that of Theorem 3.4.1. □

3.5 Supremal Controllable Sublanguages and Optimal Supervision

Let $\mathbf{G} = (\_, \Sigma, \_, \_, \_)$ be a controlled DES with $\Sigma = \Sigma_c \cup \Sigma_u$; the items $(\_)$ of $\mathbf{G}$ will be immaterial to the discussion of this section. Let $E \subseteq \Sigma^*$; later E will play the role of a specification language for the supervisory control of $\mathbf{G}$. We introduce the set of all sublanguages of E that are controllable with respect to $\mathbf{G}$:

$$\mathcal{C}(E) = \{K \subseteq E \mid K \text{ is controllable with respect to } \mathbf{G}\}$$

As a subset of the sublanguages of E, $\mathcal{C}(E)$ is a poset with respect to inclusion. It will be shown that the supremum in this poset always exists in $\mathcal{C}(E)$.

Proposition 3.5.1

$\mathcal{C}(E)$ is nonempty and is closed under arbitrary unions. In particular, $\mathcal{C}(E)$ contains a (unique) supremal element [which we denote by $\sup \mathcal{C}(E)$].

Proof

Since the empty language is controllable, it is a member of $\mathcal{C}(E)$. Let $K_\alpha \in \mathcal{C}(E)$ for all α in some index set A, and let $K = \cup\{K_\alpha | \alpha \in A\}$. Then $K \subseteq E$. Furthermore, $\bar{K} = \cup\{\bar{K}_\alpha | \alpha \in A\}$ and $\bar{K}\Sigma_u = \cup\{\bar{K}_\alpha \Sigma_u | \alpha \in A\}$. Therefore

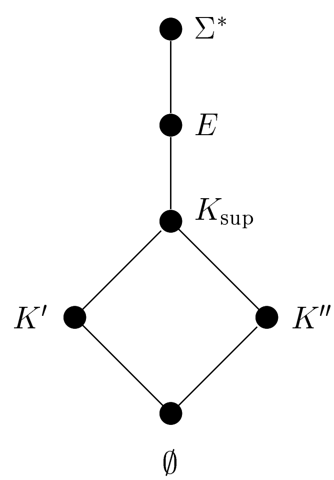
$$\begin{aligned} \bar{K}\Sigma_u \cap L(\mathbf{G}) &= [\cup(\bar{K}_\alpha \Sigma_u)] \cap L(\mathbf{G}) \\ &= \cup[\bar{K}_\alpha \Sigma_u \cap L(\mathbf{G})] \\ &\subseteq \cup \bar{K}_\alpha \\ &= \bar{K} \end{aligned}$$

Finally we have for the supremal element

$$\sup \mathcal{C}(E) = \cup\{K | K \in \mathcal{C}(E)\}$$

□

It may be helpful to keep in mind the following Hasse diagram, where $K_{\text{sup}} = \sup \mathcal{C}(E)$.



We remark that $\mathcal{C}(E)$ is not generally closed under intersection, so it is not a sublattice of the lattice of sublanguages of E. To see what goes wrong, let $K_1, K_2 \in \mathcal{C}(E)$. We must determine whether or not

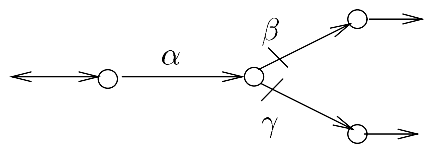
$$\overline{K_1 \cap K_2}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K_1 \cap K_2} \quad (?)$$

But $\overline{K_1 \cap K_2} \subseteq \bar{K}_1 \cap \bar{K}_2$ always, and quite possibly with strict inclusion. It follows that the left side of (?) is included in

$$\begin{aligned}(\bar{K}_1 \cap \bar{K}_2)\Sigma_u \cap L(\mathbf{G}) &= (\bar{K}_1\Sigma_u \cap L(\mathbf{G})) \cap (\bar{K}_2\Sigma_u \cap L(\mathbf{G})) \\ &\subseteq \bar{K}_1 \cap \bar{K}_2 \\ &\neq \overline{K_1 \cap K_2}\end{aligned}$$

in general. The situation may be described by saying that $\mathcal{C}(E)$ is only a complete *upper semilattice* with join operation (union) that of the lattice of sublanguages of E.

Example 3.5.1 (Controllability need not be preserved by intersection)

G 

Here $\Sigma = \{\alpha, \beta, \gamma\}$, $\Sigma_c = \{\beta, \gamma\}$. The languages $K_1 = \{\epsilon, \alpha\beta\}$, $K_2 = \{\epsilon, \alpha\gamma\}$ are controllable, but $K_1 \cap K_2 = \{\epsilon\}$ is not controllable, since the event α is uncontrollable and $\alpha \notin \overline{K_1 \cap K_2}$. On the other hand, $\bar{K}_1 \cap \bar{K}_2 = \{\epsilon, \alpha\}$ is controllable. $\diamond$

It is easy to see that the intersection of an arbitrary collection of closed controllable languages is always closed and controllable. Proof of the following observation may now be left to the reader.

Proposition 3.5.2

With respect to a fixed controlled DES **G** with alphabet Σ, the closed controllable sublanguages of an arbitrary language $E \subseteq \Sigma^*$ form a complete sublattice of the lattice of sublanguages of E. $\square$

To summarize, if K_α $(\alpha \in A)$ are controllable then $\cup K_\alpha$ and $\cap \bar{K}_\alpha$ are controllable, but generally $\cap K_\alpha$ is not.

Together with $E \subseteq \Sigma^*$ now fix $L \subseteq \Sigma^*$ arbitrarily. Consider the collection of all sublanguages of E that are L-closed:

$$\mathcal{F}(E) = \{F \subseteq E | F = \bar{F} \cap L\}$$

It is straightforward to verify that $\mathcal{F}(E)$ is nonempty ($\emptyset$ belongs) and is closed under arbitrary unions and intersections. Thus we have the following.

Proposition 3.5.3

$\mathcal{F}(E)$ is a complete sublattice of the lattice of sublanguages of E. □

Again let $E, L \subseteq \Sigma^*$. We say that E is *L-marked* if $E \supseteq \bar{E} \cap L$, namely any prefix of E that belongs to L must also belong to E.

Proposition 3.5.4

Let $E \subseteq \Sigma^*$ be $L_m(\mathbf{G})$-marked. Then $\sup \mathcal{C}(E \cap L_m(\mathbf{G}))$ is $L_m(\mathbf{G})$-closed.

Proof

We have $E \supseteq \bar{E} \cap L_m(\mathbf{G})$, from which there follows in turn

$$\begin{aligned} \bar{E} \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G}) \\ \bar{E} \cap \bar{L}_m(\mathbf{G}) \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G}) \\ \overline{E \cap L_m(\mathbf{G})} \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G}) \end{aligned}$$

so that $F := E \cap L_m(\mathbf{G})$ is $L_m(\mathbf{G})$-closed. Let $K = \sup \mathcal{C}(F)$. If K is not $L_m(\mathbf{G})$-closed, i.e. $K \subsetneqq \bar{K} \cap L_m(\mathbf{G})$, there is a string $s \in \bar{K} \cap L_m(\mathbf{G})$ with $s \notin K$. Let $J = K \cup \{s\}$. Since $\bar{J} = \bar{K}$ we have that J is controllable. Also $K \subseteq F$ implies that

$$\bar{K} \cap L_m(\mathbf{G}) \subseteq \bar{F} \cap L_m(\mathbf{G}) = F$$

so that $s \in F$ and thus $J \subseteq F$. Therefore $J \in \mathcal{C}(F)$ and $J \supsetneqq K$, contradicting the fact that K is supremal. □

Now we can present the main result of this section.

Theorem 3.5.1

Let $E \subseteq \Sigma^*$ be $L_m(\mathbf{G})$-marked, and let $K = \sup \mathcal{C}(E \cap L_m(\mathbf{G}))$. If $K \neq \emptyset$, there exists a nonblocking supervisory control (NSC) V for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$.

Proof

K is controllable and, by Proposition 3.5.4, $L_m(\mathbf{G})$-closed. The result follows by Theorem 3.4.1. □

The result may be paraphrased by saying that K is (if nonempty) the minimally restrictive solution of the problem of supervising $\mathbf{G}$ in such a way that its behavior belongs to E

and control is nonblocking. In this sense the supervisory control provided by Theorem 3.5.1 is optimal.

As might be expected, if we place part of the burden of 'marking action' on the supervisory control itself we may relax the prior requirement on E. By an application of Theorem 3.4.2 the reader may easily obtain the following.

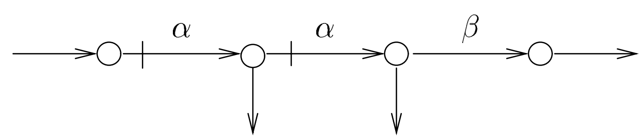
Theorem 3.5.2

Let $E \subseteq \Sigma^*$ and let $K = \sup \mathcal{C}(E \cap L_m(\mathbf{G}))$. If $K \neq \emptyset$ there exists a marking nonblocking supervisory control (MNSC) V for $(K, \mathbf{G})$ such that $L_m(V/\mathbf{G}) = K$. □

Example 3.5.2

Let $\mathbf{G}$ be the controlled DES displayed below:

G



Here

$$\Sigma = \{\alpha, \beta\}, \quad \Sigma_c = \{\alpha\}, \quad L(G) = \{\epsilon, \alpha, \alpha^2, \alpha^2\beta\},$$

$$L_m(G) = \{\alpha, \alpha^2, \alpha^2\beta\}$$

For the 'specification' language we take $E = \{\alpha, \beta, \alpha^2\}$. Then

$$E \cap L_m(\mathbf{G}) = \{\alpha, \alpha^2\}, \quad \bar{E} = \{\epsilon, \alpha, \alpha^2, \beta\}$$

$$\bar{E} \cap L_m(\mathbf{G}) = \{\alpha, \alpha^2\}, \quad \sup \mathcal{C}(E \cap L_m(\mathbf{G})) = \{\alpha\},$$

$$\overline{\{\alpha\}} = \{\epsilon, \alpha\}, \quad \overline{\{\alpha\}} \cap L_m(\mathbf{G}) = \{\alpha\}$$

From these results we see that E is $L_m(\mathbf{G})$-marked, and that indeed $\sup \mathcal{C}(E \cap L_m(\mathbf{G}))$ is $L_m(\mathbf{G})$-closed as asserted by Proposition 3.5.4. For the supervisory control we may take $V(\epsilon) = \{\alpha, \beta\}$, $V(\alpha) = \{\beta\}$, and $V(s) = \{\beta\}$ otherwise. Then it is clear that

$$L(V/\mathbf{G}) = \{\epsilon, \alpha\}, \quad L_m(V/\mathbf{G}) := L(V/\mathbf{G}) \cap L_m(\mathbf{G}) = \{\alpha\}$$

namely V is nonblocking for $\mathbf{G}$, as expected. ◇

Exercise 3.5.1: For **MACH** defined in Sect. 3.4, consider the languages

$$E_1 := \{s \in L_m(\mathbf{MACH}) | \#\beta(s) \geq 5\},$$

"production runs with at least 5 items produced"; and

$$E_2 := \{s \in L_m(\mathbf{MACH}) | \#\lambda(s) \leq 10\}$$

"runs with at most 10 breakdowns"; and

$$E_3 := E_1 \cap E_2$$

In each case calculate the supremal controllable sublanguage and describe the corresponding control action.

3.6 Implementation of Supervisory Controls by Automata

While theoretically convenient, the abstract definition of a supervisory control as a map $L(G) \to \Gamma$ does not in itself provide a concrete representation for practical implementation. As a first step in this direction we show how such a representation may be derived in the form of an automaton.[1] Let V be a marking nonblocking supervisory control (MNSC) for the controlled DES $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$, with

$$L_m(V/\mathbf{G}) = K, \quad L(V/\mathbf{G}) = \bar{K} \tag{0}$$

In case V is nonmarking (V is an NSC), the discussion will specialize in an evident way. Now let **KDES** be a reachable automaton over Σ that represents K, namely

$$L_m(\mathbf{KDES}) = K, \qquad L(\mathbf{KDES}) = \bar{K}$$

Obviously

$$K = L_m(\mathbf{KDES}) \cap L_m(\mathbf{G}), \qquad \bar{K} = L(\mathbf{KDES}) \cap L(\mathbf{G}) \tag{1}$$

With K as in (0), now let **SDES** be any DES such that

$$K = L_m(\mathbf{SDES}) \cap L_m(\mathbf{G}), \qquad \bar{K} = L(\mathbf{SDES}) \cap L(\mathbf{G}) \tag{2}$$

If (2) holds we say that **SDES** *implements* V. Notice that, in general, **SDES** need not represent $L_m(V/\mathbf{G})$ in order to implement V: the conditions (2) allow **SDES** to represent a superlanguage of $L_m(V/\mathbf{G})$ that may be simpler (admit a smaller state description) than $L_m(V/\mathbf{G})$ itself. This flexibility is due to the fact that closed-loop behavior is a consequence of constraints imposed by the plant **G** (i.e. the structure of $L(\mathbf{G})$) as well as by the supervisory control V.

[1] In this section we prefer the term 'automaton' for this representation, rather than 'generator', but allow the transition function to be a partial function.

In any case we have

Proposition 3.6.1

Let $E \subseteq \Sigma^*$ and let

$$K := \sup \mathcal{C}(E \cap L_m(\mathbf{G})) \neq \emptyset$$

Let V be an MNSC such that $L_m(V/\mathbf{G}) = K$ (which exists by Theorem 3.5.2). Let **KDES** represent K. Then **KDES** implements V. □

For the converse, let $S \subseteq \Sigma^*$ be an arbitrary language over Σ such that

$$S \text{ is controllable with respect to } \mathbf{G} \tag{3a}$$

$$S \cap L_m(\mathbf{G}) \neq \emptyset \tag{3b}$$

$$\overline{S \cap L_m(\mathbf{G})} = \bar{S} \cap L(\mathbf{G}) \tag{3c}$$

We note that $K := S \cap L_m(\mathbf{G})$ is controllable with respect to $\mathbf{G}$; in fact

$$\begin{aligned}
\bar{K}\Sigma_u \cap L(\mathbf{G}) &= (\overline{S \cap L_m(\mathbf{G})})\Sigma_u \cap L(\mathbf{G}) \\
&= (\bar{S} \cap L(\mathbf{G}))\Sigma_u \cap L(\mathbf{G}) \\
&\subseteq \bar{S}\Sigma_u \cap L(\mathbf{G}) \\
&\subseteq \bar{S} \cap L(\mathbf{G}) \qquad \text{(by (3a))} \\
&= \bar{K}
\end{aligned}$$

Let V be an MNSC such that $L_m(V/\mathbf{G}) = K$ (which exists by Theorem 3.5.2) and let **SDES** represent S. It is easy to see that **SDES** implements V; in fact

$$\begin{aligned}
L_m(\mathbf{SDES}) \cap L_m(\mathbf{G}) &= S \cap L_m(\mathbf{G}) = K \\
L(\mathbf{SDES}) \cap L(\mathbf{G}) &= \bar{S} \cap L(\mathbf{G}) = \bar{K}
\end{aligned}$$

Thus we have

Proposition 3.6.2

Let **SDES** be any nonblocking DES over Σ such that $S := L_m(\mathbf{SDES})$ satisfies conditions (3a) and (3c). Let $\emptyset \neq K := S \cap L_m(\mathbf{G})$ and let V be an MNSC such that $L_m(V/\mathbf{G}) = K$. Then **SDES** implements V. In particular

$$L_m(V/\mathbf{G}) = L_m(G) \cap L_m(\mathbf{SDES}), \qquad L(V/\mathbf{G}) = L(\mathbf{G}) \cap L(\mathbf{SDES}) \qquad □$$

If (3a) and (3c) hold and **SDES** represents S we say that **SDES** is a *supervisor* for $\mathbf{G}$. It is thus convenient to include under "supervisor" the trivial case where $L_m(\mathbf{G}) \cap L_m(\mathbf{SDES}) = \emptyset$.

If, in addition, it is trim, **SDES** will be called a *proper* supervisor for **G**. It is also convenient to extend the usage of "controllability" to DES: thus if **SDES** represents the language S and S is controllable with respect to **G**, we shall say that **SDES** is *controllable* with respect to **G**. To summarize, **SDES** is declared to be a *proper supervisor* for **G** if

(i) **SDES** is trim (reachable and coreachable);

(ii) **SDES** is controllable with respect to **G**;

(iii) $\overline{L_m(\mathbf{SDES}) \cap L_m(\mathbf{G})} = L(\mathbf{SDES}) \cap L(\mathbf{G})$.

As an illustration of Proposition 3.6.2, let **EDES** be an arbitrary DES over Σ and let

$$K := \sup \mathcal{C}(L_m(\mathbf{G}) \cap L_m(\mathbf{EDES}))$$

where the right side may possibly be empty. Let **KDES** represent K. Then **KDES** is a proper supervisor for **G**.

The relationships discussed above are displayed in Fig. 3.6.1. The *TCT* procedures **supcon** and **condat** are introduced in the following section. In general a 'simplified' supervisor language S, or its generator **SDES**, can be obtained only by intelligent guesswork, or a heuristic reduction procedure like SupReduce (Sect. 3.10).

Let $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ implement V. We may interpret **S** as a state machine that accepts as 'forcing inputs' the sequence of symbols of Σ output by **G** and executes corresponding state transitions in accordance with its transition function ξ. In this interpretation, control action is exercised by **S** on **G** implicitly, by way of a state-output function

$$\psi : X \to \Gamma$$

defined according to

$$\psi(x) := \{\sigma \in \Sigma | \xi(x, \sigma)!\}$$

The control action of **S** may be visualized as follows. Immediately upon entering the state $x \in X$, and while resident there, **S** disables in **G** just those (controllable) events $\sigma \in \Sigma_c$ such that $\sigma \notin \psi(x)$. In other words the next possible event that can be generated by **G** is any event, but only events, in the set

$$\{\sigma \in \Sigma | \xi(x, \sigma)! \ \& \ \delta(q, \sigma)!\}$$

where $q \in Q$ is the current state of **G**. The actual mechanism of disablement, which would involve instantaneous transfer of information from **S** to **G**, will be left unspecified in our interpretation. As a metaphor, one might consider the switching of signals between red and green (no amber!) in an idealized road traffic network.

To formalize the closed-loop supervisory control system that results from this construction, we denote by **S**/**G** the product generator (cf. Section 2.4)

$$\mathbf{S}/\mathbf{G} = (X \times Q, \Sigma, \xi \times \delta, (x_o, q_o), X_m \times Q_m)$$

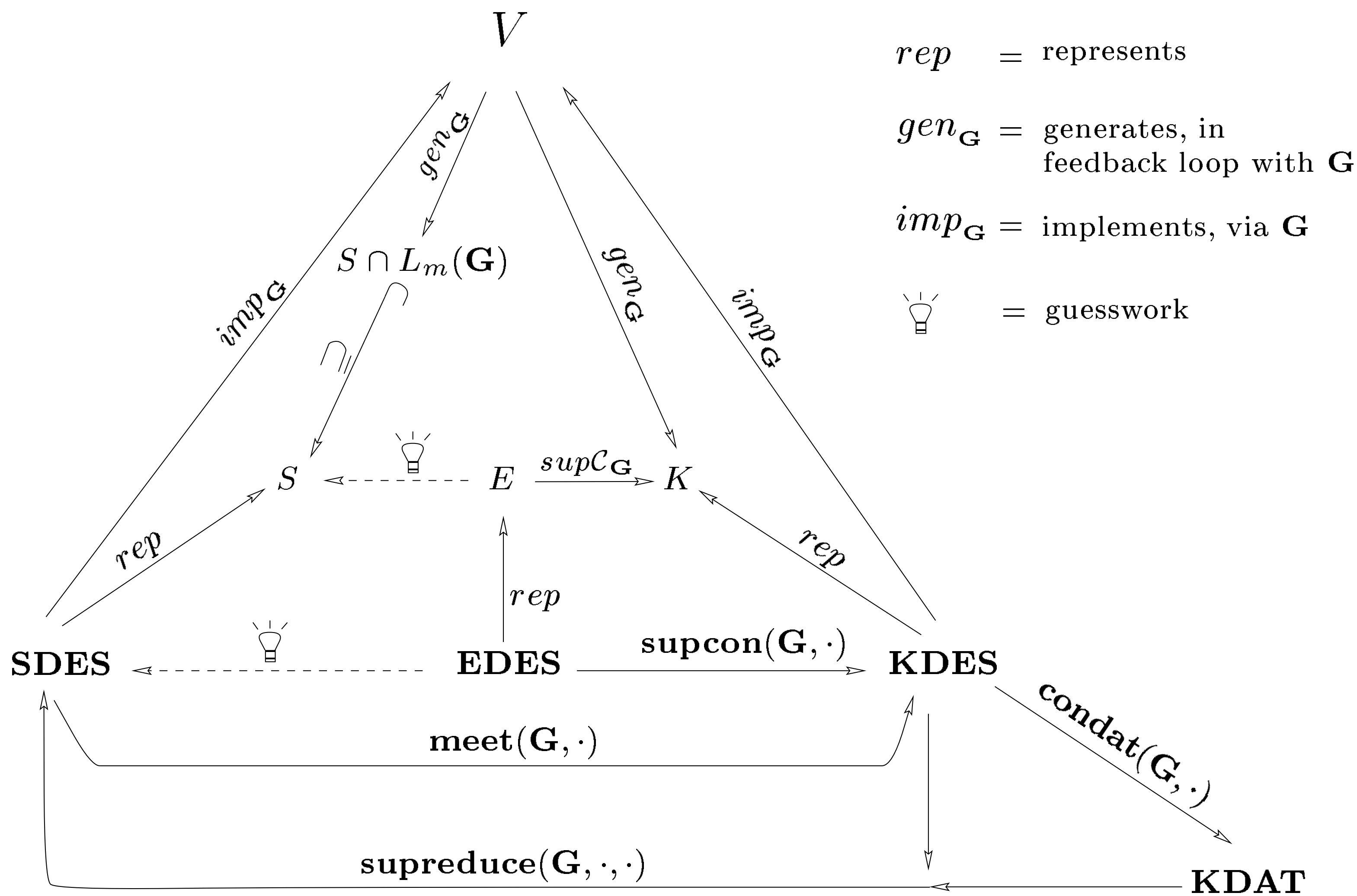


Fig 3.6.1. Scheme for supervisory control implementation

where

$$\xi \times \delta : X \times Q \times \Sigma \to X \times Q : (x, q, \sigma) \mapsto (\xi(x, \sigma), \delta(q, \sigma))$$

provided $\xi(x, \sigma)!$ and $\delta(q, \sigma)!$.

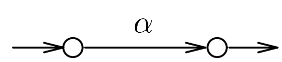
Exercise 3.6.1: From the foregoing discussion verify that

$$L_m(\mathbf{S}/\mathbf{G}) = L_m(V/\mathbf{G}), \quad L(\mathbf{S}/\mathbf{G}) = L(V/\mathbf{G})$$

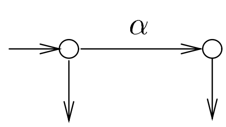
In this sense our use of the term 'implements' is justified.

Example 3.6.1 (Supervisor)

Referring to Example 3.5.2 and applying the foregoing result for MNSC, we may take for $\mathbf{S}$ the recognizer for $\sup \mathcal{C}(E \cap L_m(\mathbf{G})) = \{\alpha\}$ with (partial) transition function displayed below:



Alternatively, since in this example the specification language E is $L_m(\mathbf{G})$-marked, all marking action for $\sup \mathcal{C}(E \cap L_m(\mathbf{G}))$ could be left to $\mathbf{G}$ itself, namely we could take for $\mathbf{S}$ the recognizer with $X_m = X$ corresponding to the closed language $\{\epsilon, \alpha\}$:



Which style is selected will depend on computational convenience. Currently *TCT* runs more efficiently the smaller the subset of marker states, so the first approach would be preferred.

◇

The condition (3c) provides the basis for a useful definition. In general let K, L be arbitrary sublanguages of Σ^*. Then K, L are *nonconflicting* if

$$\overline{K \cap L} = \bar{K} \cap \bar{L}$$

Thus K and L are nonconflicting just in case every string that is both a prefix of K and a prefix of L can be extended to a string belonging to K and L in common. In *TCT* the boolean function **nonconflict(G1,G2)** = *true* just in case every reachable state of the product structure **meet(G1,G2)** is coreachable, namely $\overline{L_m(\mathbf{G1}) \cap L_m(\mathbf{G2})} = L(\mathbf{G1}) \cap L(\mathbf{G2})$. Thus to check whether two languages L_1 and L_2 are nonconflicting it is equivalent to check that **G1** and **G2** satisfy **nonconflict**, where $L_i = L_m(\mathbf{Gi})$ and $\bar{L}_i = L(\mathbf{Gi})$, namely **G1**, **G2** represent L_1, L_2 respectively.

The next result follows immediately from the definitions.

Proposition 3.6.3

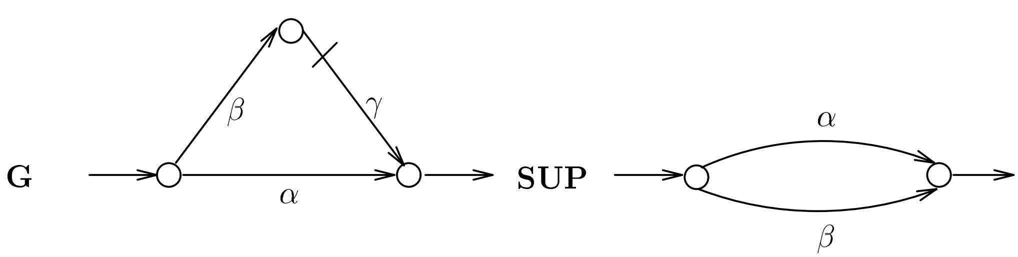
Let **SUP** be an arbitrary DES over Σ and assume $\bar{L}_m(\mathbf{G}) = L(\mathbf{G})$. Then **SUP** is a proper supervisor for **G** if and only if (i) $L_m(\mathbf{SUP})$ is controllable with respect to **G**, (ii) **SUP** is trim, and (iii) $L_m(\mathbf{SUP})$ and $L_m(\mathbf{G})$ are nonconflicting. $\square$

In case $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{G})$ the 'nonconflicting' condition is automatic.

Notice that condition (iii) for **SUP** is a nontrivial property that may require separate verification whenever **SUP** is not, or is not already known (perhaps by construction) to be, the trim (or at least nonblocking) generator of a controllable sublanguage of $L_m(\mathbf{G})$.

The following example illustrates how the conclusion of Proposition 3.6.3 fails when $L_m(\mathbf{SUP})$ and $L_m(\mathbf{G})$ conflict.

Example 3.6.2



Here $\Sigma = \{\alpha, \beta, \gamma\}$ with $\Sigma_c = \{\gamma\}$. We have $L_m(\mathbf{SUP}) \cap L_m(\mathbf{G}) = \{\alpha\}$, whereas $L(\mathbf{SUP}) \cap L(\mathbf{G}) = \{\epsilon, \alpha, \beta\}$. $\diamond$

Taken literally, the interpretation of supervisory control action proposed earlier in this section assigns to the supervisor a role that is purely passive, consisting only in the disablement of events whose occurrence or nonoccurrence is otherwise actively 'decided' by the plant. While often physically plausible, this is not the only interpretation that is possible or even desirable. It should be borne in mind that, from a formal point of view, the theory treats nothing but event synchronization among transition structures: the issue as to which (if any) among two or more synchronized transition structures actively 'causes' a given shared event to occur is not formally addressed at all. The system modeller is free to ascribe causal action as he sees fit: a machine transition from Idle to Working is (in the theory) nothing but that; it is consistent with the theory to suppose that the transition is ‘caused’ by spontaneous internal machine volition, or by internal volition on the part of the supervisor, or indeed by some external agent that may or may not be explicitly modelled, say a human operator or aliens on Mars. This feature provides the modeller with considerable flexibility. In the exercises of later sections, the reader is invited to test the model against whatever interpretation he deems appropriate; of course the desired interpretation may very well guide the modelling process. See, for instance, Section 3.8, which touches on the related issue of forced events.

3.7 Design of Supervisors Using *TCT*

We now indicate how the results of Sections 3.5 and 3.6 can be applied to supervisor design. Let the controlled DES **G** be given, along with an upper bound $E \subseteq \Sigma^*$ on admissible marked behavior. As before we refer to E as the specification language. It will be assumed that $E = L_m(\mathbf{E})$, where the DES **E** along with **G** has been created by *TCT*. Our objective is to design an optimal (i.e. minimally restrictive) proper supervisor **S** for **G** subject to $L_m(\mathbf{S}/\mathbf{G}) \subseteq E$. In accordance with Theorem 3.5.2 and the discussion in Section 3.6 the most direct method is to compute a trim recognizer for the language $K := \sup \mathcal{C}(E \cap L_m(\mathbf{G}))$. The *TCT* procedure **supcon** computes a trim representation **KDES** of K according to

$$\mathbf{KDES} = \mathbf{supcon}(\mathbf{G}, \mathbf{E})$$

To complete the description of **S**, the *TCT* procedure **condat** returns the control pattern (specifically, the minimal set of controllable events that must be disabled) at each state of **S**:

$$\mathbf{KDAT} = \mathbf{condat}(\mathbf{G}, \mathbf{KDES})$$

In outline the procedure **supcon** works as follows. Let $Pwr(\Sigma^*)$ be the power set of Σ^*, i.e. the set of all sublanguages of Σ^*. Define the operator

$$\Omega : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$$

according to

$$\Omega(Z) = E \cap L_m(\mathbf{G}) \cap \sup\{T \subseteq \Sigma^* | T = \bar{T}, \quad T\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{Z}\}$$

With K as defined above, it can be shown that K is the largest fixpoint of Ω. In the present regular (finite-state) case, this fixpoint can be computed by successive approximation. Let

$$K_o = E \cap L_m(\mathbf{G}), \quad K_{j+1} = \Omega(K_j) \quad (j = 0, 1, 2, ...)$$

It can be shown that

$$K = \lim K_j \quad (j \to \infty)$$

Furthermore the limit is attained after a finite number of steps that is of worst case order $\|L_m(\mathbf{G})\| \cdot \|E\|$. In *TCT* the operator Ω is implemented by a simple backtracking operation on the product transition structure $\mathbf{meet}(\mathbf{G}, \mathbf{E})$.

As an example, we consider Small Factory, as described in Section 3.3. The result for

$$\mathbf{FACTSUP} = \mathbf{supcon}(\mathbf{FACT}, \mathbf{SPEC})$$

is displayed in Figs. 3.7.1 and 3.7.2 (see Appendix 3.7.1). By tracing through the transition graph the reader may convince himself that the specifications are satisfied; and the theory guarantees that **FACTSUP** represents the freest possible behavior of **FACT** under the stated constraints. We also tabulate the control patterns as displayed by[2]

$$\mathbf{FACTSUP} = \mathbf{condat}(\mathbf{FACT}, \mathbf{SUPER})$$

Only controllable events that are strictly required to be disabled appear in the table.

In practice it is rarely necessary to implement an optimal supervisor by explicit representation of the language $\sup \mathcal{C}(L_m(\mathbf{G}) \cap E)$. Often common sense and intuition will lead to an optimal supervisor with a much smaller transition structure[3]. For justification of such a proposed supervisor, we may apply the *TCT* analog of Proposition 3.6.3.

Proposition 3.7.1

Let **SUP** be a DES over Σ, such that

(i) **condat(G,SUP)** lists only controllable (i.e. odd-numbered) events as requiring disablement;

(ii) **SUP** = **trim (SUP)**;

(iii) **nonconflict(G,SUP)** = *true.*

Then **SUP** is a proper supervisor for **G**. □

In analogy to the notation $V/\mathbf{G}$, denote by $\mathbf{SUP}/\mathbf{G}$ the closed-loop controlled DES obtained by forming the **meet** of **SUP** with **G**. Then, with **SUP** a proper supervisor for **G**, we have

$$L_m(\mathbf{SUP}/\mathbf{G}) = L_m(\mathbf{meet}(\mathbf{G}, \mathbf{SUP})), \quad L(\mathbf{SUP}/\mathbf{G}) = L(\mathbf{meet}(\mathbf{G}, \mathbf{SUP}))$$

along with the guaranteed nonblocking property

$$\overline{L_m(\mathbf{SUP}/\mathbf{G})} = L(\mathbf{SUP}/\mathbf{G})$$

[2] *TCT* stores the result of **condat** as a .DAT file, whereas the result of **supcon** is a .DES file.

[3] See also Sect. 3.10.

For Small Factory we construct the candidate supervisor **SIMFTSUP** directly as shown in Fig. 3.7.2, where the natural decomposition of the control problem into the regimes of 'normal operation' and 'breakdown and repair' is clearly manifest. Evidently **SIMFTSUP** is trim. Controllability of the language $L_m(\mathbf{SIMFTSUP})$ is easily checked from the table for

$$\mathbf{SIMFTSUP} = \mathbf{condat}(\mathbf{FACT}, \mathbf{SIMFTSUP})$$

inasmuch as only controllable events are required to be disabled. To test whether **SIMFT-SUP** is nonblocking we apply the *TCT* procedure **nonconflict**. In the present case we find that **nonconflict(FACT,SIMFTSUP)** is *true*, and so conclude finally that **SIMFTSUP** really is a proper supervisor for **FACT**, as expected.

As yet there is no guarantee that **SIMFTSUP** is optimal. To verify that it is, we must first compute the closed-loop language

$$L_m(\mathbf{SIMFTSUP/FACT}) = L_m(\mathbf{SIMFTSUP}) \cap L_m(\mathbf{FACT})$$

as represented by, say,

$$\mathbf{SSM} = \mathbf{meet}(\mathbf{SIMFTSUP}, \mathbf{FACT})$$

and then check that $L_m(\mathbf{SSM}) = L_m(\mathbf{FACTSUP})$.

In general, suppose $M1 = L_m(\mathbf{G1})$ and $M2 = L_m(\mathbf{G2})$. *TCT* offers two ways of investigating equality of $M1$ and $M2$. A general method is to check the inclusions $M1 \subseteq M2$ and $M2 \subseteq M1$ according to

$$\mathbf{trim}(\mathbf{meet}(\mathbf{G1}, \mathbf{complement}(\mathbf{G2}, \_))) = \mathbf{EMPTY}$$
$$\mathbf{trim}(\mathbf{meet}(\mathbf{G2}, \mathbf{complement}(\mathbf{G1}, \_))) = \mathbf{EMPTY}$$

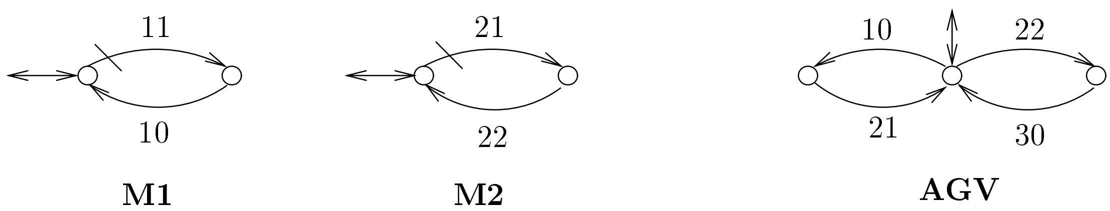
Alternatively, if **G1** and **G2** are seen to have the same state size, number of marker states and number of transitions, then it may already be plausible that $M1 = M2$, and it is sufficient to check that the *TCT* procedure

$$\mathbf{isomorph}(\mathbf{G1}, \mathbf{G2})$$

returns *true*.

In the present example, one can verify by either method that $L_m(\mathbf{SSM}) = L_m(\mathbf{SUPER})$, as hoped. The design and justification of **SIMFTSUP** are now complete.

Exercise 3.7.1: A workcell consists of two machines **M1**, **M2** and an automated guided vehicle **AGV** as shown. **AGV** can be loaded with a workpiece either from **M1** (event 10) or from **M2** (event 22), which it transfers respectively to **M2** (event 21) or to an output conveyor (event 30). Let **CELL** = **sync(M1,M2,AGV)**. By displaying an appropriate event sequence show that **CELL** can deadlock, i.e. reach a state from which no further transitions are possible. To prevent deadlock, define the legal language **TCELL** = **trim(CELL)**, then **SUPER** = **supcon(CELL,TCELL)**. Explain how **SUPER** prevents deadlocking event sequences. Also explain why **TCELL** itself cannot serve directly as a (proper) supervisor.



Exercise 3.7.2: A transmitter is modelled by the 3-state generator **T0** tabulated, where the event α denotes arrival of a message to be transmitted, σ denotes the start of a message transmission, τ denotes timeout in case the transmitted message is not acknowledged, and ρ denotes reset for transmission of a subsequent message. An unacknowledged message is retransmitted after timeout. New messages that arrive while a previous message is being processed are ignored.

The system **T** to be controlled consists of transmitters **T1** and **T2**, where **Ti** is modeled over the alphabet $\{\alpha i, \sigma i, \tau i, \rho i\}$; thus **T** = **shuffle(T1,T2)**. Only the events $\sigma 1, \sigma 2$ are controllable. It's required that the channel be utilized by at most one transmitter at a time.

A first trial solution implements the supervisory control V as the 2-state device **C**, which can be thought of as a model of the channel. **C** ensures that if, for instance, $\sigma 1$ occurs, then $\sigma 2$ cannot occur until $\rho 1$ occurs, namely **T1** has finished processing a message.

Verify that **T** and **C** conflict, and find a string that leads to a state of **T** that is non-coreachable. Explain why **C** is not a suitable controller.

Try the new channel model **NC**, verify that **T** and **NC** are nonconflicting, and that **NC** controls **T1** and **T2** according to specification. Provide *TCT* printouts and state transition graphs as appropriate.

T0 # states: 3; marker state: 0
transitions:
$(0, \alpha, 1),\ (1, \alpha, 1),\ (1, \sigma, 2),\ (2, \rho, 0),\ (2, \tau, 1),\ (2, \alpha, 2)$

C # states: 2; marker state: 0
transitions:
$(0, \sigma 1, 1),\ (0, \sigma 2, 1),\ (1, \rho 1, 0),\ (1, \rho 2, 0)$
[adjoin selfloops with events $\alpha i,\ \tau i,\ i = 1, 2$]

NC # states: 3; marker state: 0
transitions:
$(0, \sigma 1, 1),\ (1, \sigma 1, 1),\ (1, \rho 1, 0),\ (0, \sigma 2, 2),\ (2, \sigma 2, 2),\ (2, \rho 2, 0)$
[adjoin selfloops with events $\alpha i,\ \tau i,\ i = 1, 2$]

Exercise 3.7.3: **WORKCELL** is the synchronous product of **ROBOT**, **LATHE** and **FEEDER**. The latter is a mechanism that imports new parts for **WORKCELL** to process. There is a 2-slot input buffer **INBUF** to store new parts as they are imported, and a 1-slot buffer **SBBUF** associated with **LATHE**, to hold parts on standby. **ROBOT** transfers new parts from **INBUF** to **LATHE**. If **LATHE** is idle, **ROBOT** loads the new part; if busy, **ROBOT** places the new part in **SBBUF**; in each case, **ROBOT** then returns to idle. If **ROBOT** is idle and **LATHE** is idle and there is a part in **SBBUF**, then **ROBOT** can load it. There are other tasks unrelated to **LATHE**, which **ROBOT** can initiate and return from.

Specifications are the following. **SPEC1** says that **LATHE** can be loaded only if it is idle. **SPEC2** says that if a part is on standby (i.e. **SBBUF** is not empty) then **ROBOT** cannot transfer a new part from **INBUF**. **SPEC3** says that **LATHE** can move from idle to busy only after being loaded. **SPEC4** says that a part can be put on standby only if **LATHE** is busy. **SPEC5** says that **ROBOT** must give **LATHE** priority over its other tasks: namely **ROBOT** can initiate other tasks only when: either **LATHE** is busy, or both **INBUF** and **SBBUF** are empty. To set up **SPEC5**, compute **sync**(**LATHE,INBUF,SBBUF**), then selfloop with **ROBOT**'s "initiate_unrelated_task" event at just the appropriate states (recall the method of 'flags', Sect. 3.3). **SPEC5** automatically incorporates the usual overflow/underflow constraints on the buffers. Finally **SPEC** is the synchronous product of **SPEC1**,...,**SPEC5**, selflooped with any **WORKCELL** events not already included.

Create *TCT* models for the items described above, making your own detailed choices for controllable/uncontrollable events. Then compute **SUPER** = **supcon**(**WORKCELL,SPEC**), as well as **SUPER(.DAT)** = **condat**(**WORKCELL,SUPER**). Discuss any points of interest.

To examine the controlled behavior when **ROBOT**'s extraneous tasks are hidden from view, compute **PSUPER** = **project**(**SUPER**,appropriate_event_list), and discuss.

Exercise 3.7.4: Use *TCT* to re-solve Exercise 3.5.1, making sure your results are consistent.

Exercise 3.7.5: In Small Factory, compute

$$\mathbf{PSUPER} = \mathbf{project}(\mathbf{SUPER}, [10, 12, 13, 21, 22, 23])$$

and interpret the result.

Exercise 3.7.6: Using *TCT*, redo Small Factory using a buffer of capacity 2. Also design the corresponding simplified supervisor. Generalize your results to a buffer of arbitrary size N.

Exercise 3.7.7: Three cooks share a common store of 5 pots. For his favorite dish, COOK1 needs 2 pots, COOK2 4 pots, and COOK3 all 5 pots. The cooks may take pots from the store individually and independently, but only one pot at a time; a cook returns all his pots to the store simultaneously, but only when he has acquired and used his full complement. Design a supervisor that is maximally permissive and guarantees nonblocking. Assume that "take-pot" events are controllable and "return-pot" events uncontrollable.

Exercise 3.7.8: In the context of a DES problem where the alphabet is Σ, define the self-looped DES

$$\mathbf{ALL} = (\{0\}, \Sigma, \{\text{transitions } [0, \sigma, 0] | \sigma \in \Sigma\}, 0, \{0\})$$

Thus the closed and marked behaviors of **ALL** are both Σ^*. As usual let **PLANT** and **SPEC** be two DES over Σ. The corresponding supervisory control problem has solution **SUP** = **supcon(PLANT,SPEC)**. Show that this problem can always be replaced by an equivalent problem where the specification is **ALL**. **Hint:** First replace **SPEC** by **NEWSPEC**, where **NEWSPEC** merely adjoins a "dump state" to **SPEC**, if one is needed. This makes the closed behavior of **NEWSPEC** equal to Σ^*, while the marked behavior is that of **SPEC**. In *TCT*, **NEWSPEC** = **complement(complement(SPEC))** (why?). Now set

$$\begin{aligned} \mathbf{NEWPLANT} &= \mathbf{meet}(\mathbf{PLANT}, \mathbf{NEWSPEC}) \\ \mathbf{NEWSUP} &= \mathbf{supcon}(\mathbf{NEWPLANT}, \mathbf{ALL}) \end{aligned}$$

Show that **NEWSUP** and **SUP** define exactly the same languages (in particular, perhaps after application of **minstate**, **NEWSUP** and **SUP** are isomorphic).

While this maneuver offers no computational advantage, it can simplify theoretical discussion, as the specification **ALL** requires only that the closed-loop language be nonblocking.

Exercise 3.7.9: Let **G** be a DES defined over the alphabet Σ. With **ALL** defined as in Exercise 3.7.8, show that **G** is nonblocking if and only if **nonconflict(ALL,G)** = *true*. Using *TCT* check this result against examples of your own invention.

Exercise 3.7.10: Let $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ and let $K \subseteq \Sigma^*$. Suppose K is *represented* by a DES $\mathbf{KDES} = (X, \Sigma, \xi, x_o, X_m)$, in the sense that $K = L_m(\mathbf{KDES})$, $\bar{K} = L(\mathbf{KDES})$, i.e. the marked and closed behaviors of **KDES** are K and $\bar{K}$ respectively. Let $\mathbf{PROD} = \mathbf{G} \times \mathbf{KDES}$ as defined in Sect. 2.4, and let **RPROD** be the reachable sub-DES of **PROD**.

Show that K is controllable with respect to **G** if and only if, at each state (q, x) of **RPROD**,

$$\{\sigma \in \Sigma_u \mid \delta(q, \sigma)!\} \subseteq \{\sigma \in \Sigma_u \mid \xi(x, \sigma)!\} \ ,$$

namely any uncontrollable event that is state-enabled ('physically executable') by **G** is also control-enabled ('legally admissible') by **KDES**.

Illustrate the foregoing result with two simple examples, for the cases K controllable and uncontrollable respectively.

Exercise 3.7.11: The result of Exercise 3.7.10 is the basis of the *TCT* procedure **condat**. The result of **condat**, say **KDESDAT** = **condat(G,KDES)**, is a table of the states x of **KDES** along with all the events which must be disabled in **G** (by a supervisory control) when **RPROD** is at (q, x) in **G** $\times$ **KDES** for some $q \in Q$, in order to force the inclusion

$$\{\sigma \in \Sigma \mid \delta(q,\sigma)! \;\;\&\;\; \sigma \text{ control-enabled}\} \subseteq \{\sigma \in \Sigma \mid \xi(x,\sigma)!\}$$

Thus the set of disabled events tabulated by **condat** at x is

$$\{\sigma \in \Sigma \mid ((\exists q \in Q)(q,x) \;\text{ in } \mathbf{RPROD} \;\&\;\; \delta(q,\sigma)!) \;\;\&\;\; \textit{not} \;\; \xi(x,\sigma)!\}$$

For K to be controllable, this set must contain only events that are controllable (so they can be disabled): in other words, in the *TCT* event encoding, only events with odd-numbered labels. So to check controllability of K it's sufficient to scan the **condat** table: if only odd events are listed, K is controllable; if an even event occurs anywhere, K is uncontrollable.

Illustrate this remark by testing your two examples from Exercise 3.7.10 and supply the *TCT* printouts for **condat**.

Exercise 3.7.12 (Message passing): Investigate how to implement message- passing in our setup. For instance, suppose a supervisor **M0** wishes to enable an event 11 'remotely' in controlled module **M1** by sending a 'message' 0. **M1** should not execute 11 before receiving 0, but **M0** needn't wait for **M1** to execute 11 before completing other tasks (say, event 2). Also, if **M1** has just executed 11, it must not do so again until it has executed task 10 (say) and once again received a 0. For this, define

$$\begin{aligned}
\mathbf{M0} &= (Q, \Sigma, \delta, q_o, Q_m) \\
&= (\{0,1,2\}, \{0,1,2\}, \{[0,1,1],[1,0,2],[2,2,0]\}, 0, \{0\}) \\
\mathbf{M1} &= (\{0,1\}, \{10,11\}, \{[0,11,1],[1,10,0]\}, 0, \{0\})
\end{aligned}$$

To couple **M0** and **M1** as described define a 'mailbox'

$$\mathbf{MB} = (\{0,1\}, \{0,11\}, \{[0,0,1],[1,0,1],[1,11,0]\}, 0, \{0\})$$

Check that the synchronous product of **M0**, **M1**, and **MB** displays the required behavior. Show that the approach can be extended to two or more controlled modules **M1**, **M2**,... . For instance, create **M2** by relabeling events 10,11 in **M1** as 20,21; rename **MB** as **MB1**; create **MB2** by relabeling 11 in **MB1** as 21; and consider the new controlled module **M** = **sync**(**M1**,**M2**) and mailbox **MB** = **sync**(**MB1**,**MB2**). Notice that the message 0 can always be projected out of the final structure **sync**(**M0**,**M**,**MB**) if it is of no external interest.

Investigate this model, with particular attention to the growth in complexity as measured by the size of the final state set.

Other message-passing semantics are possible. For instance, suppose **M0** should not progress past state 2 until both enabled events 11 and 21 have occurred – **M0** waits for its message to be acted on. For this, remodel **M0** as

$$\begin{aligned}\mathbf{N0} \;=\; & (\{0,1,2,3,4\},\{1,2,11,21\},\\ & \{[0,1,1],[1,11,2],[1,21,3],[2,21,4],[3,11,4],[4,2,0]]\},0,\{0\})\end{aligned}$$

Because **N0** waits, there's logically no need for a mailbox at all. Check that the result has 20 states and 36 transitions, still complicated but much less so than before.

Now explore other variations on the theme.

Exercise 3.7.13: Show that

$$\begin{aligned}&\mathbf{supcon}(\mathbf{G},\mathbf{meet}(\mathbf{E_1},\mathbf{E_2}))\\ &\qquad = \mathbf{supcon}(\mathbf{supcon}(\mathbf{G},\mathbf{E_1}),\mathbf{E_2})\end{aligned}$$

and interpret.

Exercise 3.7.14: Carry through an original supervisor design problem of your own, along the lines of this section. If feasible, draw the transition graph of your supremal controllable sublanguage and discuss any features of special interest.

Appendix 3.7.1

EVENT CODING FOR SMALL FACTORY

TCT:	10	11	12	13	20	21	22	23
TEXT:	β_1	α_1	λ_1	μ_1	β_2	α_2	λ_2	μ_2

FACTSUP # states: 12 state set: 0 ... 11 initial state: 0

marker states: 0

transitions: 24

transition table:

[0,11,1]	[1,12,2]	[1,10,3]	[2,13,0]	[3,21,4]
[4,20,0]	[4,11,5]	[4,22,11]	[5,20,1]	[5,10,6]
[5,12,8]	[5,22,10]	[6,20,3]	[6,22,7]	[7,23,3]
[8,20,2]	[8,13,4]	[8,22,9]	[9,23,2]	[10,23,1]
[10,10,7]	[10,12,9]	[11,23,0]	[11,11,10]	

FACTSUP printed.

FACTSUP

Control Data are displayed by listing the supervisor states where disabling occurs, together with the events that must be disabled there.

Control Data:

0: 21	1: 21
2: 21	3: 11
6: 11	7: 11
9: 13	

FACTSUP printed.

SIMFTSUP # states: 3 state set: 0 ... 2 initial state: 0

marker states: 0

transitions: 16

transition table:

[0,13,0]	[0,11,0]	[0,12,0]	[0,20,0]	[0,10,1]
[0,22,2]	[1,21,0]	[1,23,1]	[1,12,1]	[1,20,1]
[1,22,1]	[2,23,0]	[2,10,1]	[2,11,2]	[2,12,2]
[2,20,2]				

SIMFTSUP printed.

SIMFTSUP

Control Data are displayed by listing the supervisor states where disabling occurs, together with the events that must be disabled there.

Control Data:

0: 21	1: 11
2: 13	

SIMFTSUP printed.

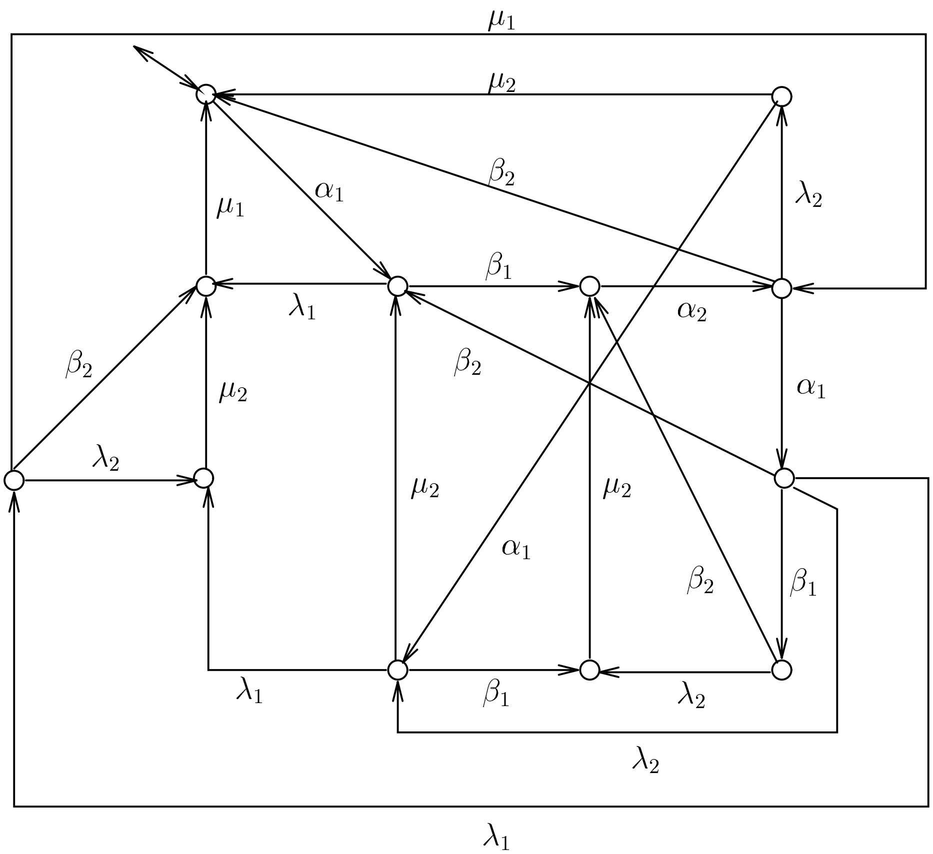


Fig. 3.7.1

$L(\mathbf{FACTSUP}/\mathbf{FACT})$
Supremal controllable sublanguage for Small Factory

SIMFTSUP

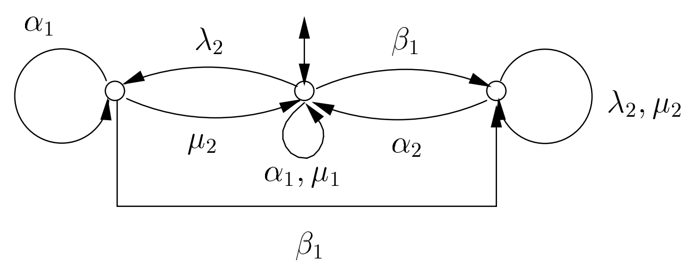


Fig. 3.7.2

selfloop$\{\lambda_1, \beta_2\}$
Simplified supervisor for Small Factory

3.8 Forced Events

In practice one often thinks of control as 'forcing' the occurrence of some desirable result. In the asynchronous world of discrete events, 'forcing' amounts to timely preemption: a tap or valve is closed in time to prevent overflow, a stirrer is switched on to prevent gelatification in a tank of fluid, a drainage pump is started when water in a mine reaches a defined level, a car is braked to forestall an impending collision, an industrial process is begun in time to deliver the product on schedule (before a deadline).

The crucial feature common to these examples is that the controlling agent denies permission for the occurrence of undesirable competing events; namely (directly or indirectly) such events are disabled. Enforcing the familiar specification that a buffer must not overflow or underflow is achieved by disabling the appropriate 'upstream' events in the causal (or just behavioral) sequence; meeting a deadline is achieved by 'disabling' the tick of a clock to ensure the occurrence of a desired event on schedule – how this can be modelled without violence to the physical requirement that 'time goes on' regardless of technology is explained in Chapter 9.

While in general 'forcing' is probably best placed in a temporal context (cf. Chapter 9) simple preemption can often capture the required action in the untimed framework considered so far. As a primitive example, suppose a tank T is filled by fluid flow through a valve V, which must be turned off to prevent overflow when the tank is full. V can be modelled as a one-state DES

$$\mathbf{V} = (Q, \Sigma, \delta, q_o, Q_m) = (\{0\}, \{\sigma\}, [0, \sigma, 0], 0, \{0\})$$

with σ controllable. The event σ is interpreted as the delivery of a defined unit of fluid to the tank. The tank itself is modelled like a buffer, with its content incremented by one unit when σ occurs. If the tank capacity is N units then the transition structure could be

$$\begin{aligned}\mathbf{T} \;&=\; (\{0, 1, ..., N+1\}, \{\sigma\}, \{[0, \sigma, 1], [1, \sigma, 2], ..., \\ &\qquad [N, \sigma, N+1], [N+1, \sigma, N+1]\}, 0, \{0\})\end{aligned}$$

where the state $N+1$ represents an overflow condition. To prevent overflow, let

$$\begin{aligned}\mathbf{TSPEC} \;&=\; (\{0, 1, ..., N\}, \{\sigma\}, \{[0, \sigma, 1], [1, \sigma, 2], ..., \\ &\qquad [N-1, \sigma, N]\}, 0, \{0\})\end{aligned}$$

thus disabling σ at state N. The closed behavior with respect to **TSPEC** is then simply $\overline{\sigma^N}$, as required. Notice that the model is consistent with the physical picture of (temporally) continuous flow through the valve, as there is no inconsistency in supposing that σ occurs one second after it is initially enabled, or reenabled after a subsequent occurrence. As soon as σ is disabled, flow stops. However, there is no logical necessity that σ be tied to a fixed interval of time or a unit flow. The situation is much like filling up the fuel tank of a car using a hose with a spring-loaded trigger valve: when the tank is full, the trigger is released 'automatically' (or by the user) and the valve closes.

More generally, the notion of forcing as timely preemption can be formalized as follows. Define a new subset $\Sigma_f \subseteq \Sigma$ of *forcible* events, and a subset $\Sigma_p \subseteq \Sigma$ of *preemptable* events, with $\Sigma_f \cap \Sigma_p = \emptyset$. Bring in a new controllable event $\tau \notin \Sigma$ which may be thought of as a 'timeout' event. Assume that a plant model $\mathbf{G}$ has been created as usual over Σ, and we wish to adjoin the feature that any event in Σ_f can be 'forced' to preempt any event in Σ_p. For this, examine each state in $\mathbf{G}$ where some event $\alpha \in \Sigma_f$ and some event $\beta \in \Sigma_p$ are both enabled, e.g. the state q as displayed in Fig. 3.8.1. Notice that there may exist events γ defined at q that are neither forcible nor preemptable.

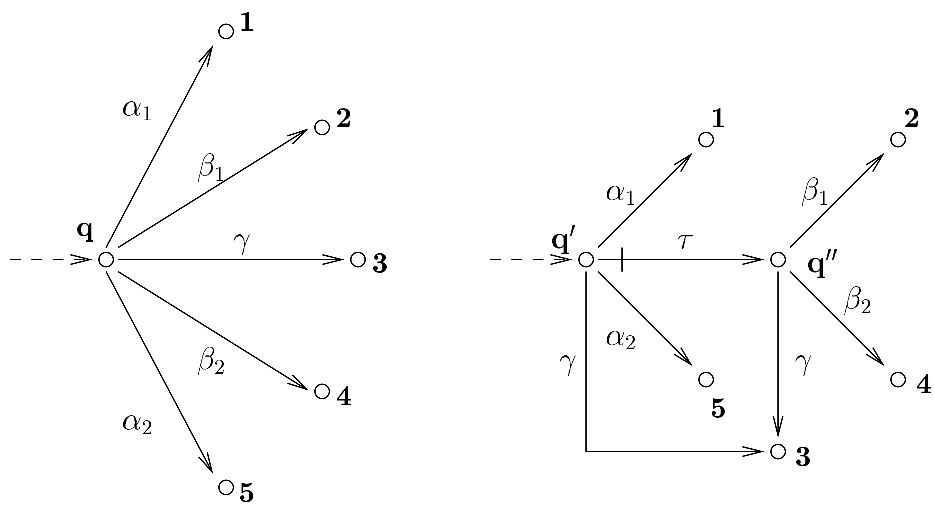


Fig. 3.8.1
Modelling forcing by α_1 or α_2
to preempt β_1 and β_2

Also, we impose no constraint as to whether an event in either Σ_f or Σ_p is controllable or not, although normally events in Σ_p will be uncontrollable. Now modify $\mathbf{G}$ (or in *TCT*, edit $\mathbf{G}$) at q as shown: split q into q' and q'', with a transition $[q', \tau, q'']$. If, say, α_1, α_2, β_1, β_2, γ are the events defined at q in $\mathbf{G}$, then define α_1, α_2, γ at q' and β_1, β_2, γ at q''. Selfloops should be treated as follows. If α was selflooped at q it should be selflooped at q'; a selfloop β at q is replaced by a transition $[q''\beta q']$; while a selfloop γ at q is replaced by selfloops γ at both q' and q''. In the new DES $\mathbf{G_{new}}$, say, the effect of disabling τ is to 'force' one of the events α_1, α_2, γ to preempt β_1, β_2. Observe that, unless γ is included in Σ_p, it could also preempt the other events α_1, α_2 defined at q'. Having modified $\mathbf{G}$ to $\mathbf{G_{new}}$, modify the specification DES $\mathbf{E}$ say, to $\mathbf{E_{new}}$, by selflooping each state of $\mathbf{E}$ with τ. We now have, in $(\mathbf{G_{new}}, \mathbf{E_{new}})$ a supervisory control problem of standard type, and proceed as usual to compute $\mathbf{supcon}(\mathbf{G_{new}}, \mathbf{E_{new}})$. This standard solution will 'decide' exactly when forcing (i.e. disablement of τ) is appropriate.

It is clear that our procedure could easily be automated in *TCT*, and that after the design has been completed, the event τ could be hidden by being projected out. Thus all the details except initial selection of the subsets Σ_f, Σ_p could be rendered invisible to the user if desired.

Example 3.8.1 (Forcing): Consider the two machines **M1**, **M2** and the 1-slot buffer **B** in Fig. 3.8.2, with *TCT* encoding of events. For the plant take **M** = **sync(M1,M2)** and for the specification **E** take **B** selflooped with $\{11, 20\}$.

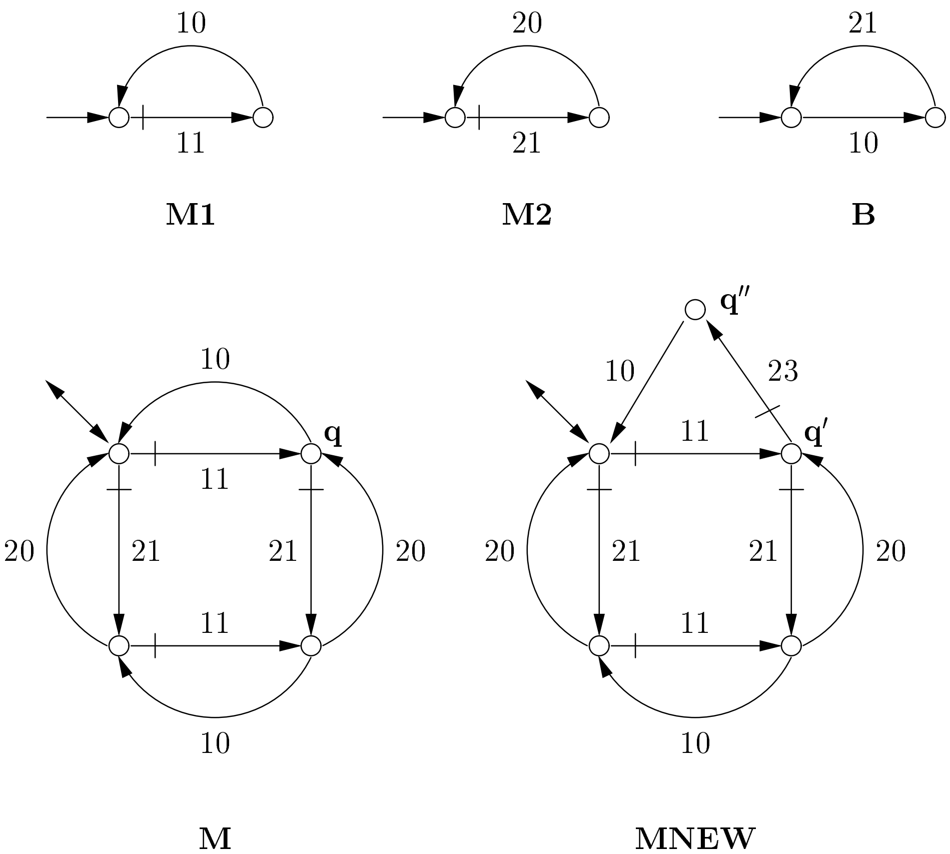


Fig. 3.8.2
Modelling forcing by event 21
to preempt event 10

The solution **SUP** = **supcon (M,E)** is displayed in Fig. 3.8.3. Now suppose that event 21 ('**M2** starts work') is forcible with respect to event 10 ('**M1** completes work') as preemptable. Construct **MNEW** by modifying the structure of **M** as shown in Fig. 3.8.2, at the one state q (in this case) where events 21 and 10 are both defined. The new controllable 'timeout' event 23 can be thought of as inserting a time delay invoked by disablement of this event, thus providing event 21 with the opportunity to preempt event 10. Construct **ENEW** = **selfloop**(**E**,[23]), and compute **SUPNEW** = **supcon(MNEW,ENEW)**. Finally, hide the auxiliary event 23 to obtain the solution **PSUPNEW** = **project**(**SUPNEW**,[23]), as displayed in Fig. 3.8.3. Notice that **PSUPNEW** generates a super-language of that of **SUP**; in general, controlled behavior with forcing will be less conservative than it is with the disablement feature alone. ◇

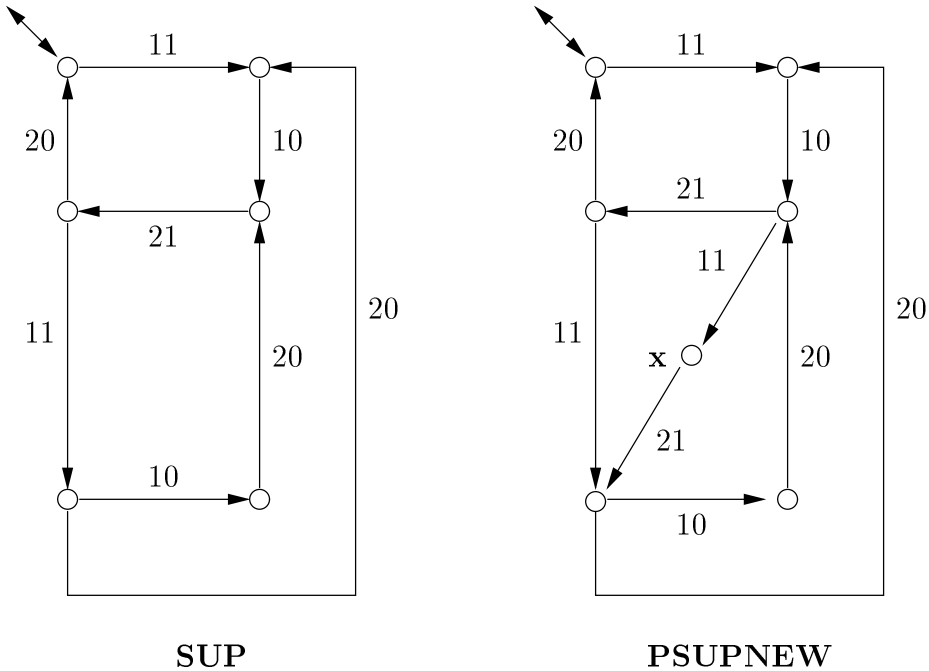


Fig. 3.8.3
In **PSUPNEW** event 21 preempts 10 at **x**

To summarize, forcing is really an issue not of synthesis but of modelling; more precisely, by declaring forcibility as a modelling assumption, we eliminate forcing as a synthesis issue, and the standard framework can be utilized without further change or the addition of any new mechanism. Nevertheless it is well to note that, once an instance of a *controllable* event is designated to be forced (e.g. event 21 at state x in **PSUPNEW**, Fig. 3.8.3), it is no longer available for disablement in any subsequent refinement of the control design. For instance, 21 at x could be relabelled as 22 (i.e. redefined as uncontrollable) as a safeguard against inadvertent disablement in a subsequent application of TCT.

Exercise 3.8.1 (Forced events): Provide examples of modelling intuitively 'forced' events as just described, carrying through a complete control design. For instance, consider a water supply tank for a country cottage, which is emptied incrementally by random household events, and filled by a pump. The pump is to be switched on when the water falls below a defined lower level and switched off when it rises to a defined upper level. Naturally a good design must ensure that the tank is never emptied by normal household usage.

3.9 Mutual Exclusion

Assume we are given DES

$$\begin{aligned} \mathbf{G1} &= (Q_1, \Sigma_1, \delta_1, q_{10}, Q_{1m}) \\ \mathbf{G2} &= (Q_2, \Sigma_2, \delta_2, q_{20}, Q_{2m}) \end{aligned}$$

with $\Sigma_1 \cap \Sigma_2 = \emptyset$. We may wish to control $\mathbf{G}_1$, $\mathbf{G}_2$ in such a way that designated state pairs $q_1 \in Q_1$, $q_2 \in Q_2$ are never occupied simultaneously. In such problems "$\mathbf{G}_i$ in q_i" typically means "$\mathbf{G}_i$ is using a single shared resource", for instance when two readers share a single textbook or two AGVs a single section of track.

Because such a constraint can be awkward to express linguistically, *TCT* provides a procedure to compute the required result directly. Thus

$$\mathbf{MXSPEC} = \mathbf{mutex}(\mathbf{G1}, \mathbf{G2}, \mathbf{LIST})$$

where

$$\mathbf{LIST} = [(q_1^{(1)}, q_2^{(1)}), ..., (q_1^{(k)}, q_2^{(k)})],$$

with $(q_1^{(i)}, q_2^{(i)}) \in Q_1 \times Q_2$, is the user's list of mutually exclusive state pairs. **MXSPEC** is reachable and controllable with respect to $\mathbf{G} = \mathbf{shuffle}(\mathbf{G1}, \mathbf{G2})$ but needn't be coreachable. If not, it may serve as a new specification for the plant $\mathbf{G}$, and the final result computed using **supcon** in the usual way.

Exercise 3.9.1: In **FACT** (Sect. 3.2) suppose power is limited, so at most one of **MACH1**, **MACH2** may be working at once. Compute a suitable supervisor. Repeat the exercise using the constraint that at most one machine at a time may be broken down.

Exercise 3.9.2: A cat and mouse share a maze of 5 interconnected chambers. The chambers are numbered 0,1,2,3,4 for the cat, but respectively 3,2,4,1,0 for the mouse. Adjoining chambers may be connected by one-way gates, each for the exclusive use of either the cat or the mouse. An event is a transition by either the cat or the mouse from one chamber to another via an appropriate gate; the animals never execute transitions simultaneously. Some gates are always open, corresponding to uncontrollable events; while others may be opened or closed by an external supervisory control, so passage through them is a controllable event. The cat and the mouse are initially located in their 'home' chambers, numbered 0. *TCT* models for the cat and mouse are printed below.

It is required to control the roamings of cat and mouse in such a way that (i) they never occupy the same chamber simultaneously, (ii) they can always return to their respective home chambers, and (iii) subject to the latter constraints they enjoy maximal freedom of movement.

```
CAT # states:  5 state set:  0 ...  4 initial state:  0

marker states:  0

vocal states:  none

# transitions:  8

transitions:

[ 0,201, 1] [ 1,205, 2] [ 1,207, 3] [ 2,200, 3]
[ 2,203, 0] [ 3,200, 2] [ 3,211, 4] [ 4,209, 1]

CAT printed

MOUSE # states:  5 state set:  0 ...  4 initial state:  0

marker states:  0

vocal states:  none

# transitions:  6

transitions:

[ 0,101, 1] [ 1,103, 2] [ 2,105, 0] [ 2,107, 3] [ 3,111, 4] [ 4,109, 2]

MOUSE printed
```

3.10 Remark on Supervisor Reduction

As indicated in Sect. 3.7 for Small Factory, the 'standard' supervisor

$$\mathbf{SUPER} = \mathbf{supcon}(\mathbf{PLANT}, \mathbf{SPEC})$$

computed by **supcon** (and representing the full optimal controlled behavior) can be much larger in state size than is actually required for the same control action. This is because the controlled behavior incorporates all the *a priori* transitional constraints embodied in the plant itself, as well as any additional constraints required by control action to enforce the specifications. The problem of finding a simplified proper supervisor, say **MINSUP**, equivalent in control action but of minimum state size, is of evident practical interest. Unfortunately, it is NP-hard [C82]. A reduction procedure called SupReduce has been developed, based on heuristic search for a suitable congruence on the state set of **SUPER**. SupReduce is of polynomial complexity in the state sizes of **PLANT** and **SPEC**. While of course it cannot guarantee a simplified supervisor of minimal size, SupReduce will often find a greatly reduced supervisor, say **SIMSUP**, and can also provide a lower bound on the size of **MINSUP**. **SIMSUP** is actually minimal if its size matches this bound. Some results found by SupReduce are reported in the examples of Sect. 8.14.

3.11 Notes and References

Supervisory control theory in the sense of this chapter originates with the doctoral thesis of P.J. Ramadge [T01] and related papers [J03,J05,C01-C05]. The Kanban Example 3.3.3 is adapted from Viswanadham & Narahari [1992], pp. 514-524. Prioritized synchronous product (Exercise 3.3.9) was introduced by Heymann [1990]. Exercise 3.3.10 is based on Wong [1998] and Exercise 3.7.2 on Cassandras [1993] (Example 2.17, p. 108); Exercise 3.7.3 was suggested by Robin Qiu.

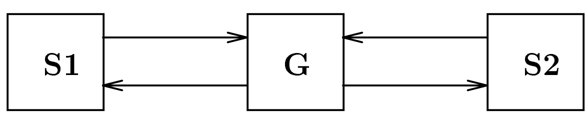
Computational methods for supervisor synthesis for DES of industrial size is an active area of current research. The reader is referred to the monographs of Germundsson [1995] and Gunnarsson [1997], as well as recent work by Zhang [T44,C84] and Leduc [T46,C85,C86,C87].

Chapter 4

Modular Supervision of Discrete-Event Systems

4.1 Introduction

In this chapter we discuss a modular approach to the synthesis of supervisors for discrete-event systems. In this approach the overall supervisory task is divided into two or more subtasks. Each of the latter is solved using the results of Chapter 3, and the resulting individual subsupervisors are run concurrently to implement a solution of the original problem. We refer to such a construction as a *modular synthesis*, and to the resultant supervisor as a *modular supervisor*. The 'architecture' is sketched below.



Such constructions represent a very general approach to complex problems – sometimes called 'divide and conquer'. In addition to being more easily synthesized, a modular supervisor should ideally be more readily modified, updated and maintained. For example, if one subtask is changed, then it should only be necessary to redesign the corresponding component supervisor: in other words, the overall modular supervisor should exhibit greater flexibility than its 'monolithic' counterpart.

Unfortunately, these advantages are not always to be gained without a price. The fact that the individual supervisory modules are simpler implies that their control action must be based on a partial or 'local' version of the global system state; in linguistic terms, a component supervisor processes only a 'projection' of the behavior of the DES to be controlled. A consequence of this relative insularity may be that different component supervisors, acting quasi-independently on the basis of local information, come into conflict at the 'global' level, and the overall system fails to be nonblocking. Thus a fundamental issue that always arises in the presence of modularity is how to guarantee the nonblocking property of the final synthesis.

4.2 Conjunction of Supervisors

Let $\mathbf{S_1}$ and $\mathbf{S_2}$ be proper supervisors for $\mathbf{G}$: that is, each of $\mathbf{S_1}$ and $\mathbf{S_2}$ is a trim automaton[1], is controllable with respect to $\mathbf{G}$ (equivalently, $L_m(\mathbf{S_1})$, $L_m(\mathbf{S_2})$ are controllable with respect to $\mathbf{G}$), and each of $\mathbf{S_1}/\mathbf{G}$, $\mathbf{S_2}/\mathbf{G}$ is nonblocking, namely

$$\bar{L}_m(\mathbf{S_1}/\mathbf{G}) = L(\mathbf{S_1}/\mathbf{G}), \quad \bar{L}_m(\mathbf{S_2}/\mathbf{G}) = L(\mathbf{S_2}/\mathbf{G})$$

Recalling from Sect. 2.4 the definitions of reachable subautomaton and of product automaton, we define the *conjunction* of $\mathbf{S_1}$ and $\mathbf{S_2}$, written $\mathbf{S_1} \wedge \mathbf{S_2}$, as the reachable subautomaton of the product:

$$\mathbf{S_1} \wedge \mathbf{S_2} = Rch(\mathbf{S_1} \times \mathbf{S_2}) = \mathbf{meet}(\mathbf{S_1}, \mathbf{S_2})$$

It is easily seen from the definition that the supervisory action of $\mathbf{S_1} \wedge \mathbf{S_2}$ is to enable an event σ just when σ is enabled by $\mathbf{S_1}$ and $\mathbf{S_2}$ simultaneously. To describe the action of $\mathbf{S_1} \wedge \mathbf{S_2}$ more fully we have the following.

Theorem 4.2.1

Under the foregoing conditions,

$$L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) = L_m(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2}/\mathbf{G})$$

Furthermore $\mathbf{S_1} \wedge \mathbf{S_2}$ is a proper supervisor for $\mathbf{G}$ if and only if it is trim and the languages $L_m(\mathbf{S_1}/\mathbf{G})$, $L_m(\mathbf{S_2}/\mathbf{G})$ are nonconflicting.

Proof

For the first statement we have

$$\begin{aligned}
L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) &= L_m(\mathbf{S_1} \wedge \mathbf{S_2}) \cap L_m(\mathbf{G}) \\
&= L_m(\mathbf{S_1} \times \mathbf{S_2}) \cap L_m(\mathbf{G}) \\
&= L_m(\mathbf{S_1}) \cap L_m(\mathbf{S_2}) \cap L_m(\mathbf{G}) \\
&= L_m(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2}/\mathbf{G})
\end{aligned}$$

Similarly, as $L(\mathbf{S_1} \wedge \mathbf{S_2}) = L(\mathbf{S_1}) \cap L(\mathbf{S_2})$ we have

$$L((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}) = L(\mathbf{S_1}/\mathbf{G}) \cap L(\mathbf{S_2}/\mathbf{G})$$

[1] As in Sect. 3.6 and our usage of 'generator', 'automaton' includes the case of partial transition function.

so that $(\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G}$ is nonblocking if and only if $L_m(\mathbf{S_1}/\mathbf{G})$ and $L_m(\mathbf{S_2}/\mathbf{G})$ are nonconflicting. Now $\mathbf{S_1}$ and $\mathbf{S_2}$ proper implies that each is controllable, so $L(\mathbf{S_1})$ and $L(\mathbf{S_2})$ are both controllable with respect to $\mathbf{G}$. By Proposition 3.5.2, $L(\mathbf{S_1}) \cap L(\mathbf{S_2})$ is controllable, therefore $L(\mathbf{S_1} \wedge \mathbf{S_2})$ is controllable. Thus $\mathbf{S_1} \wedge \mathbf{S_2}$ is a proper supervisor if and only if it satisfies the defining condition that it be trim, as claimed. □

Recall from Sect. 3.3 that in *TCT*

$$\mathbf{S}_1 \wedge \mathbf{S}_2 = \mathbf{meet}(\mathbf{S}_1, \mathbf{S}_2)$$

Obviously, if $\mathbf{S_1}$ and $\mathbf{S_2}$ satisfy all the conditions of Theorem 4.2.1 except that $\mathbf{S_1} \wedge \mathbf{S_2}$ happens not to be trim (i.e. fails to be coreachable), then $\mathbf{S_1} \wedge \mathbf{S_2}$ may be replaced by its trim version, to which the conclusions of the theorem will continue to apply. When designing with *TCT* the desirable situation is that $L_m(\mathbf{S_1})$ and $L_m(\mathbf{S_2})$ be nonconflicting (in *TCT*, $\mathbf{nonconflict}(\mathbf{S_1}, \mathbf{S_2})$ returns *true*); then $\mathbf{S_1} \wedge \mathbf{S_2} = \mathbf{meet}(\mathbf{S_1}, \mathbf{S_2})$ will indeed be trim.

Let the controlled DES $\mathbf{G}$ be arbitrary. The following results, which are almost immediate from the definitions, will find application when exploiting modularity.

Proposition 4.2.1

Let $K_1, K_2 \subseteq \Sigma^*$ be controllable with respect to $\mathbf{G}$. If K_1 and K_2 are nonconflicting then $K_1 \cap K_2$ is controllable with respect to $\mathbf{G}$. □

Proposition 4.2.2

Let $E_1, E_2 \subseteq \Sigma^*$. If $\sup \mathcal{C}(E_1)$, $\sup \mathcal{C}(E_2)$ are nonconflicting then

$$\sup \mathcal{C}(E_1 \cap E_2) = \sup \mathcal{C}(E_1) \cap \sup \mathcal{C}(E_2)$$

□

Exercise 4.2.1: Prove Propositions 4.2.1 and 4.2.2. ◇

To complete this section we provide a version of Theorem 4.2.1 adapted to *TCT*. For DES $\mathbf{G_1}$, $\mathbf{G_2}$, write $\mathbf{G_1} \approx \mathbf{G_2}$, "$\mathbf{G_1}$ and $\mathbf{G_2}$ are behaviorally equivalent", to mean

$$L_m(\mathbf{G_1}) = L_m(\mathbf{G_2}), \qquad L(\mathbf{G_1}) = L(\mathbf{G_2})$$

Theorem 4.2.2

Assume

(i) $\mathbf{S_1}$, $\mathbf{S_2}$ are controllable with respect to $\mathbf{G}$ [as confirmed, say, by **condat**],

(ii) **nonconflict** $(\mathbf{S_1} \wedge \mathbf{S_2}, \mathbf{G}) = true$, and

(iii) $\mathbf{S_1} \wedge \mathbf{S_2}$ is trim

Then $\mathbf{S_1} \wedge \mathbf{S_2}$ is a proper supervisor for $\mathbf{G}$, with

$$(\mathbf{S_1} \wedge \mathbf{S_2}) \wedge \mathbf{G} \approx (\mathbf{S_1} \wedge \mathbf{G}) \wedge (\mathbf{S_2} \wedge \mathbf{G})$$

□

Notice that condition (i) holds in particular if $\mathbf{S_1}$, $\mathbf{S_2}$ are proper supervisors for $\mathbf{G}$. Even in that case, however, condition (ii) is not automatic and must be checked. Finally, the result is easily extended to any collection $\mathbf{S_1}, \mathbf{S_2}, ..., \mathbf{S_k}$.

Corollary 4.2.2

Let $\mathbf{E_1}$, $\mathbf{E_2}$ be arbitrary DES and

$$\mathbf{S_i} = \mathbf{supcon}(\mathbf{G}, \mathbf{E_i}), \qquad i = 1, 2$$

If $\mathbf{nonconflict}(\mathbf{S_1}, \mathbf{S_2}) = true$, then

$$\mathbf{S_1} \wedge \mathbf{S_2} \approx \mathbf{supcon}(\mathbf{G}, \mathbf{E_1} \wedge \mathbf{E_2})$$

□

Exercise 4.2.2: Prove Theorem 4.2.2 and Corollary 4.2.2.

4.3 Naive Modular Supervision: "Deadly Embrace"

Before presenting successful examples of modular supervision we illustrate the possibility of blocking in a simple but classical situation. Consider two users of two shared resources (e.g. two professors sharing a single pencil and pad of paper). To carry out his task each user needs both resources simultaneously; but the resources may be acquired in either order. We model the generators **USER1**, **USER2** and the legal constraint languages **RESA**, **RESB** in the simple manner shown. Here

$$\Sigma_c = \{\alpha_1, \beta_1, \alpha_2, \beta_2\}, \quad \Sigma_u = \{\gamma_1, \gamma_2\}$$

The DES to be controlled is then

$$\mathbf{USER} = \mathbf{shuffle}(\mathbf{USER1}, \mathbf{USER2})$$

subject to the legal language

$$\mathbf{RES} = \mathbf{meet}(\mathbf{RESA}, \mathbf{RESB})$$

The optimal global supervisor is

$$\mathbf{USERSUP} = \mathbf{supcon}(\mathbf{USER}, \mathbf{RES})$$

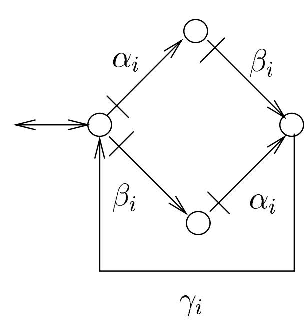
as displayed. Initially users and resources are all idle; as soon as one user acquires one resource, **USERSUP** disables the other user from acquiring any resource until the first user has completed his task. Notice, incidentally, that the validity of this proposed control depends crucially on the assumption of the shuffle model that independent events can never occur at the same time; if this assumption fails, the system will block if both users acquire their first resource simultaneously.

Let us now employ **RESA** and **RESB** as naive modular supervisors. Each is controllable and nonconflicting with respect to **USER**, hence is proper. The corresponding controlled languages are

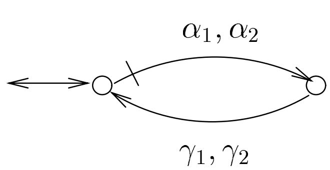
$$\mathbf{CONA} = \mathbf{meet}(\mathbf{USER}, \mathbf{RESA}), \quad \mathbf{CONB} = \mathbf{meet}(\mathbf{USER}, \mathbf{RESB});$$

however, **CONA** and **CONB** are conflicting! It is easy to see that concurrent operation of **CONA** and **CONB** could lead to blocking: because nothing prevents **USER1** from acquiring one resource (event α_1 or β_1), then **USER2** acquiring the other (resp. event β_2 or α_2), with the result that both users are blocked from further progress, a situation known as "deadly embrace". The example therefore illustrates the crucial role of marker states in system modeling and specification, as well as the importance of absence of conflict.

USERi

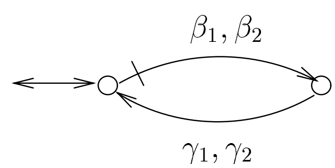


RESA



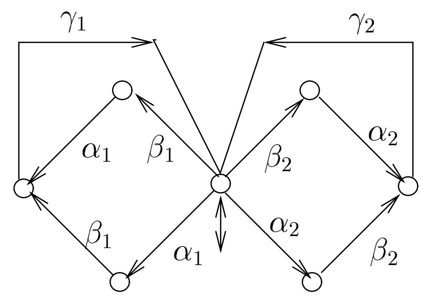
selfloop$\{\beta_1, \beta_2\}$

RESB



selfloop$\{\alpha_1, \alpha_2\}$

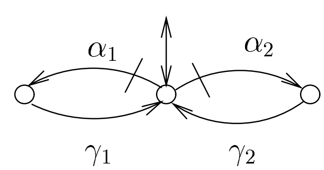
USERSUP



Exercise 4.3.1: Discuss control of this situation that guarantees nonblocking and also "fairness" according to some common-sense criterion of your invention: fairness should guarantee that neither user could indefinitely shut out the other. **Hint:** Use a queue.

Exercise 4.3.2: Replace **RESA** above with the more refined model

NRESA



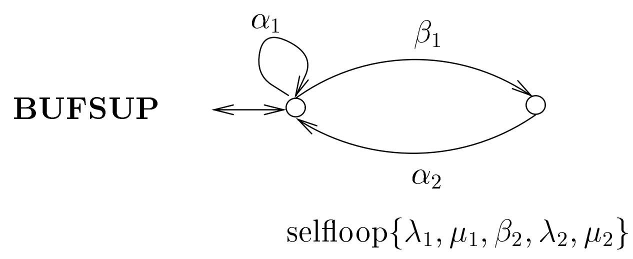
selfloop$\{\beta_1, \beta_2\}$

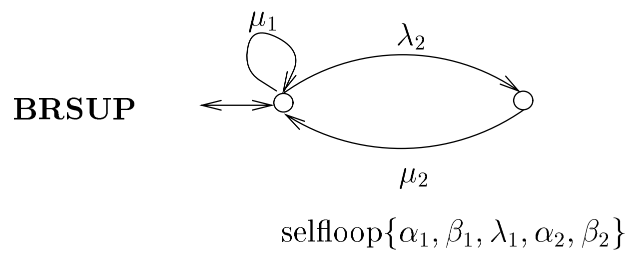
and similarly for **RESB**. Carry out the new design to get **NUSERSUP**. Verify that it is isomorphic to **USERSUP** and explain why this might be expected.

Exercise 4.3.3: As stated in the text, **USERSUP** depends for its validity on the assumption that events in independent agents never occur simultaneously. Discuss how event interleaving could be enforced, for practical purposes, by use of a queue. For this, require that **USER1**, **USER2** first request the use of a desired resource, while it is up to the supervisor to decide in what order competing requests are granted. Assume that simultaneous requests could be queued in random order.

4.4 Modular Supervision: Small Factory

We shall apply the results of Sect. 4.2 to the modular supervision of Small Factory (cf. Sects. 3.3, 3.7). As displayed below, introduce trim automata **BUFSUP** and **BRSUP** to enforce the buffer and the breakdown/repair specifications respectively.





By use of the *TCT* procedure **condat** it can be confirmed that **BUFSUP** and **BRSUP** are controllable with respect to **FACT**, and application of **nonconflict** to the pairs **FACT**, **BUFSUP** and **FACT**, **BRSUP** respectively shows by Proposition 3.6.3 that **BUFSUP** and **BRSUP** are nonblocking for **FACT**; so we may conclude that each is a proper supervisor for **FACT**. For our modular supervisor we now take the conjunction

$$\mathbf{MODSUP} = \mathbf{BUFSUP} \wedge \mathbf{BRSUP}$$

It is easy to check by hand that **BUFSUP**, **BRSUP** are nonconflicting, so

$$\mathbf{MODSUP} = \mathbf{trim}(\mathbf{meet}(\mathbf{BUFSUP}, \mathbf{BRSUP}))$$

namely **MODSUP** is trim; and by application of **condat** and **nonconflict** to the pair **FACT**, **MODSUP** we now conclude by Theorem 3.6.2 that **MODSUP** is a proper supervisor for **FACT**.

We note parenthetically that, on taking $\mathbf{G} = \mathbf{FACT}$, Proposition 4.2.1 holds with

$$K_1 = L_m(\mathbf{BUFSUP}), \quad K_2 = L_m(\mathbf{BRSUP})$$

while Proposition 4.2.2 holds with

$$E_1 = L_m(\mathbf{SPEC1}), \quad E_2 = L_m(\mathbf{SPEC2})$$

Finally it may be verified that **MODSUP** is actually optimal. Various approaches are possible: perhaps the most direct is to check that

$$L_m(\mathbf{FACT}) \cap L_m(\mathbf{MODSUP}) = L_m(\mathbf{SUPER})$$

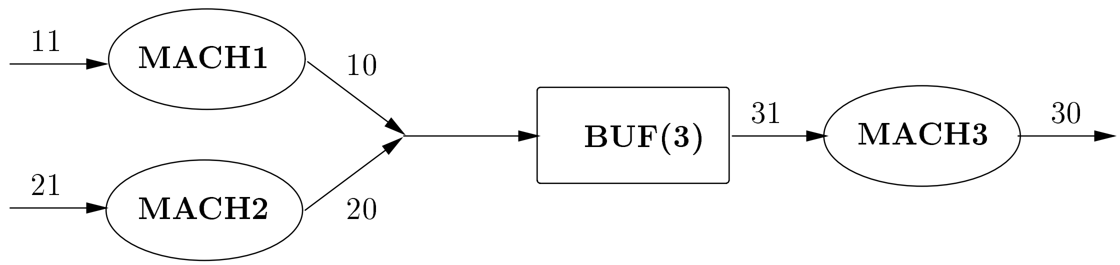
via the computation

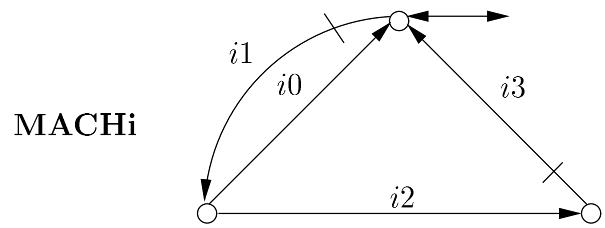
$$\mathbf{isomorph}(\mathbf{meet}(\mathbf{FACT}, \mathbf{MODSUP}), \mathbf{SUPER}) = true;$$

another possibility, using Proposition 4.2.2, is left to the reader to develop independently.

4.5 Modular Supervision: Big Factory

As another example of the foregoing ideas we consider Big Factory, as described below. Two machines as before operate in parallel to feed a buffer with capacity 3; a third machine empties the buffer.





The informal specifications are:

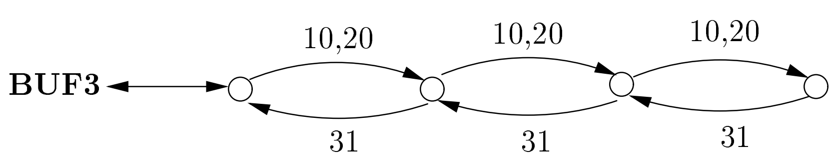
1. Buffer must not overflow or underflow.
2. **MACH1** and **MACH2** are repaired in order of breakdown.
3. **MACH3** has priority of repair over **MACH1** and **MACH2**.

As the plant we take

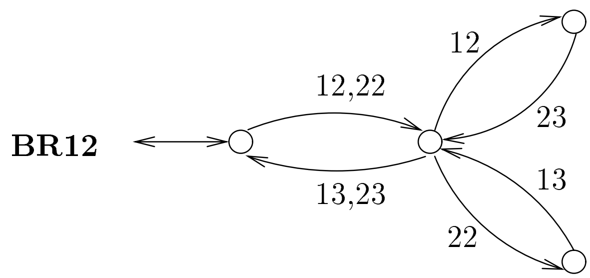
$$\mathbf{BFACT} = \mathbf{shuffle}(\mathbf{shuffle}(\mathbf{MACH1}, \mathbf{MACH2}), \mathbf{MACH3})$$

To formalize the specifications we construct the DES shown below:

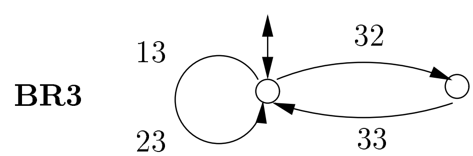
Buffer overflow/underflow:



Breakdown/repair of **MACH1,MACH2**:



Breakdown/repair of **MACH3**:



Each DES is understood to be selflooped with its complementary subalphabet.

We first consider 'monolithic' supervision. **BFACT** turns out to have 27 states and 108 transitions (written (27,108)). Combining the specification languages into their intersection, we define

$$\mathbf{BSPEC} = \mathbf{meet}(\mathbf{meet}(\mathbf{BUF3}, \mathbf{BR12}), \mathbf{BR3}) \quad (32, 248)$$

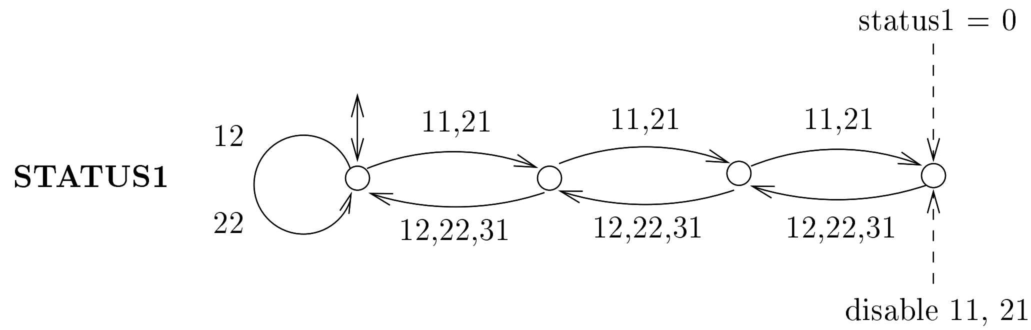
For the 'monolithic' supervisor we then obtain

$$\mathbf{BFACTSUP} = \mathbf{supcon}(\mathbf{BFACT}, \mathbf{BSPEC}) \quad (96, 302)$$

By the theory, the transition structure of the DES **BFACTSUP** is that of the supremal controllable sublanguage of $L_m(\mathbf{BFACT})$ that is contained in the specification language $L_m(\mathbf{BSPEC})$. Thus **BFACTSUP** is guaranteed to be the optimal (i.e. minimally restrictive) proper supervisor that controls **BFACT** subject to the three legal specifications. Nevertheless, **BFACTSUP** is a rather cumbersome structure to implement directly, and it makes sense to consider a modular approach.

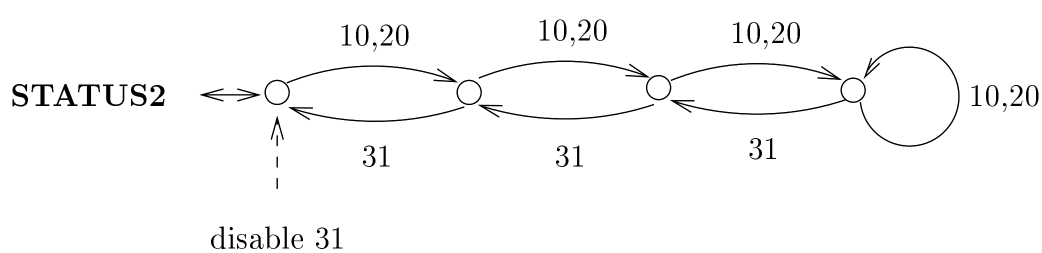
For prevention of buffer overflow alone, we compute

$$\text{status1} = \#\text{ empty buffer slots} - \#\text{ feeder machines at work}$$



STATUS1 disables 11 and 21 when status1 = 0, and is a proper supervisor for **BFACT**. For prevention of buffer underflow alone, we compute

status2 = # full slots in buffer



STATUS2 disables 31 when status2 = 0 and is also proper. For control of breakdown/repair, **BR12** and **BR3** are themselves proper supervisors. It can be verified that optimal (and proper) supervision of the buffer is enforced by

$$\mathbf{STATUS} = \mathbf{STATUS1} \wedge \mathbf{STATUS2}$$

while optimal (and proper) supervision of breakdown/repair is enforced by

$$\mathbf{BR} = \mathbf{BR12} \wedge \mathbf{BR3}$$

Finally, optimal (and proper) supervision with respect to all the legal specifications is enforced by

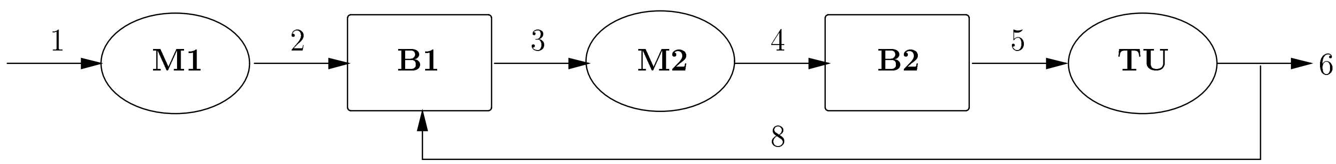
$$\mathbf{BFTMDSUP} = \mathbf{STATUS} \wedge \mathbf{BR}$$

Obviously **BFTMDSUP** is much simpler to implement than **BFACTSUP**, to which it is equivalent in supervisory action.

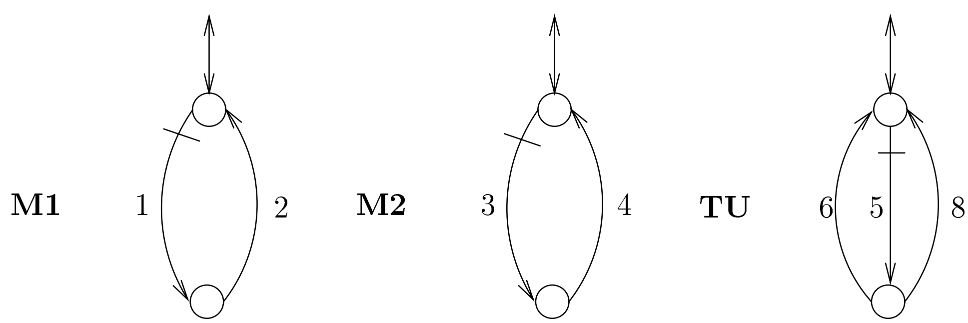
Exercise 4.5.1: Construct a 9-state supervisor that is equivalent in control action to **STATUS**. Check your result by *TCT* and supply the printouts.

4.6 Modular Supervision: Transfer Line

As a third example of modular control we consider an industrial 'transfer line' consisting of two machines **M1**, **M2** followed by a test unit **TU**, linked by buffers **B1** and **B2**, in the configuration shown. A workpiece tested by **TU** may be accepted or rejected; if accepted, it is released from the system; if rejected, it is returned to **B1** for reprocessing by **M2**. Thus the structure incorporates 'material feedback'. The specification is simply that **B1** and **B2** must be protected against underflow and overflow.



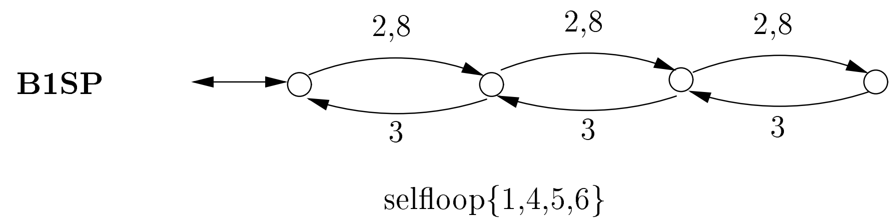
The component DES, displayed below, are taken to be as simple as possible.

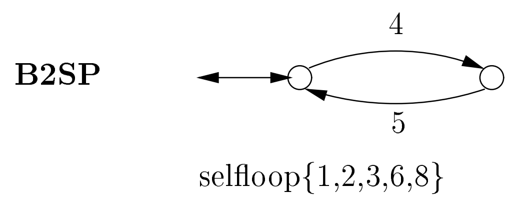


The DES representing the transfer line is

$$\mathbf{TL} = \mathbf{shuffle}(\mathbf{M1}, \mathbf{M2}, \mathbf{TU})$$

The capacities of **B1** and **B2** are assumed to be 3 and 1 respectively, and the specifications are modelled as **B1SP**, **B2SP** in the usual way.





Then the total specification is

BSP = meet(B1SP,B2SP)

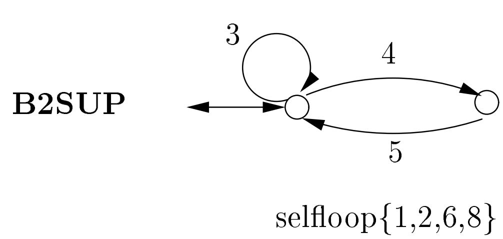
The centralized or 'monolithic' supervisor is computed as

CSUP = supcon(TL,BSP) (28,65)

and turns out to have 28 states and 65 transitions. The control data for **CSUP** is

CSUP = condat(TL,CSUP)

For modular supervision we may proceed as follows. A modular component supervisor for **B2** is simple enough: we construct **B2SUP** to disable event 5 in **TU** when **B2** is empty (to prevent underflow) and to disable event 3 in **M2** when **B2** is full (to prevent overflow).



For **B1** we separate the requirements of overflow and underflow into two subtasks, assigned to component supervisors **B1SUP1**, **B1SUP2**. To prevent underflow it suffices to adopt for **B1SUP2** the specification model **B1SP**, augmented by a harmless selfloop for events 2,8 at state 3 to render **B1SUP2** controllable. Then **B1SUP2** disables **M2** at state 0 (where **B1** is empty), but is indifferent to possible overflow at state 3. To prevent overflow we make a first, 'naive' attempt at designing **B1SUP1**, with result **XB1SUP1**, as follows. The entities feeding **B1** (potentially causing overflow) are **M1** and **TU**: define

FB1A = shuffle(M1,TU), FB1 = selfloop(FB1A,[3])

FB1 will be considered to be the controlled DES for the overflow specification

FB1SP = selfloop(B1SP,[1,5,6]),

leading to the proposed modular component supervisor

XB1SUP1A = supcon(FB1,FB1SP)

over the subalphabet {1,2,3,5,6,8}, and finally the global version

XB1SUP1 = selfloop(XB1SUP1A,[4]) (12,45)

over the full alphabet. It can be checked that each of **XB1SUP1** and **B1SUP2** is nonconflicting and controllable with respect to **TL**, and that **XB1SUP1** and **B1SUP2** are nonconflicting. Let

XB1SUP = meet(XB1SUP1,B1SUP2) (12,45)

(Verify that **XB1SUP1**, **XB1SUP** are isomorphic: why is this so?) From the theory or by direct computation, **XB1SUP** is controllable and nonconflicting with respect to **TL**. It remains to combine **XB1SUP** with **B2SUP**: to our chagrin, these components turn out to be conflicting! Let

XBSUP = trim(meet(XB1SUP,B2SUP))

Because of conflict, the closed behavior of **XBSUP** (equal to the closure of its marked behavior, by definition of the operation **trim**) is a proper sublanguage of the intersection of the closed behaviors of the trim DES **XB1SUP**, **B2SUP**; and from

XBSUP = condat(TL,XBSUP)

it is seen that **XBSUP** fails to be controllable as it calls for the disablement of events 4 and 8. The concurrent operation of **XB1SUP** and **B2SUP** will certainly result in satisfaction of the specifications **B1SP** and **B2SP**. However, each of these components admits the TL-string

$$s = 1 \ \ 2 \ \ 1 \ \ 2 \ \ 1 \ \ 2 \ \ 3 \ \ 4 \ \ 1 \ \ 2,$$

which leaves **B1** and **B2** both full. Following s, **B2SUP** disables **M2**, while **XB1SUP** disables **M1** and **TU**, and the system deadlocks; i.e. no further transitions are possible. This result illustrates that conflict and blocking can arise in seemingly innocent ways.

A correct modular supervisor for overflow of **B1** can be obtained by examining the overall feedback operation of the system. It is seen that any workpiece removed from **B1** by **M2** is a candidate for eventual return to **B1** by **TU**. Thus overflow of **B1** is prevented if and only if the number of empty slots in **B1** is maintained at least as great as the number of workpieces being processed by **M2** and **TU** or being stored in **B2**. In terms of event counts (#event) on the current string,

$$\#\text{ empty slots in } \mathbf{B1} = \text{cap}(\mathbf{B1}) + \#3 - \#2 - \#8$$

while

$$\#\text{ workpieces in } \{\mathbf{M2,B2,TU}\} = \#3 - \#6 - \#8$$

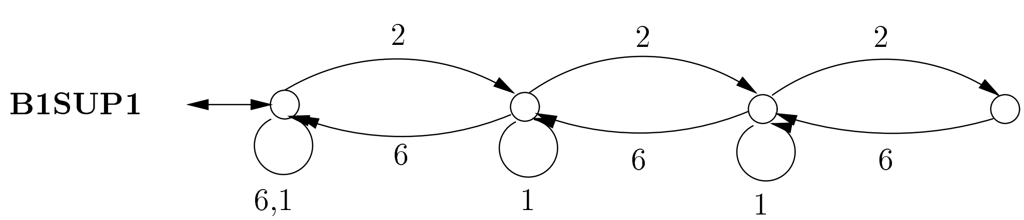
To maintain the desired inequality it is therefore required to disable **M1** if and only if

$$\#\text{ empty slots in } \mathbf{B1} \le \#\text{ workpieces in } \{\mathbf{M2,B2,TU}\}$$

i.e. (with cap(**B1**) = 3) $3 - \#2 \le -\ \#6$, or

$$\text{disable } \mathbf{M1} \quad \text{iff} \quad \#2 - \#6 \ge 3$$

Under these conditions event 2 can occur at most three times before an occurrence of event 6, so our new attempt at an overflow control for **B1** takes the form of **B1SUP1** as displayed. Here the harmless selfloop [0,6,0] has been adjoined to render **B1SUP1** controllable.



selfloop{3,4,5,8}

It can now be verified that **B1SUP1**, **B1SUP2** are nonconflicting, so that concurrent operation is represented by the proper supervisor

$$\mathbf{B1SUP} = \mathbf{meet(B1SUP1,B1SUP2)} \quad (16,100);$$

and that **B1SUP**, **B2SUP** are nonconflicting, with their concurrent operation represented by

$$\mathbf{BSUP} = \mathbf{meet(B1SUP,B2SUP)} \quad (32,156)$$

It can be checked that **BSUP** is nonconflicting and controllable with respect to **TL**. Thus the behavior of **TL** under modular supervision is given by

$$\mathbf{MSUP} = \mathbf{meet(TL,BSUP)} \quad (28,65)$$

Finally it can be checked that **MSUP** is isomorphic with **CSUP**, namely the modular supervisor **BSUP** is optimal.

Exercise 4.6.1: Improve the recycling logic of Transfer Line as follows. A failed workpiece is sent by **TU** to a new buffer **B3** (size 1), and **M2** takes its workpiece from either **B1** or **B3**. Define **M2** (or introduce a new specification) so that it takes from **B1** only if **B3** is empty, that is, a failed workpiece has priority over a new one. Design both centralized and modular supervisors for the improved system.

4.7 Reasoning About Nonblocking

In many applications the verification that the closed-loop languages implemented by individual modular controllers are nonconflicting can be achieved by exploiting plant structure and its relation to the task decomposition on which the modularity is based. For example, in Small Factory the overall supervisory task was decomposed into subtasks corresponding to 'normal operation' and 'breakdown and repair', of which the latter in a natural sense precedes the former: if either or both machines are broken down, then repair them before continuing with production. To verify that modular supervision is nonblocking, it suffices to show, roughly speaking, that at any state of the system **MODSUP/FACT** a breakdown and repair subtask (possibly null) can be completed first, followed by the completion of a normal operation subtask, in such a way that the system is brought to a marker state. The success of this maneuver depends on the fact that the subtasks of the modular decomposition are ordered in a natural sequence.

We present a simple formalization of this idea on which the reader may model his own versions in the context of more elaborate examples. Adopting the notation of Section 4.2, let

$$\mathbf{S_i} = (X_i, \Sigma, \xi_i, x_{oi}, X_{mi}) \quad i = 1, 2$$

For simplicity we assume that X_{m2} is a singleton $\{x_{m2}\}$. Now define

$$\Sigma_1 = \{\sigma \in \Sigma | (\forall x \in X_1)\xi_1(x, \sigma)!\}$$

In particular Σ_1 will include the events that are selflooped at each state of $\mathbf{S_1}$, these being the events to which the operation of $\mathbf{S_1}$ is indifferent, namely the events that are irrelevant to the execution of $\mathbf{S_1}$'s subtask; and these events will typically include those that are relevant to $\mathbf{S_2}$. Next define

$$\Sigma_2 = \{\sigma \in \Sigma | \xi_2(x_{m2}, \sigma) = x_{m2}\}$$

Thus Σ_2 is the subset of events that are selflooped at x_{m2} in $\mathbf{S_2}$, hence to which $\mathbf{S_2}$ is indifferent upon completion of its subtask. We impose two structural conditions on $\mathbf{S_1} \wedge \mathbf{S_2}$:

(i) $(\forall s \in L(\mathbf{S_2}/\mathbf{G}))(\exists t \in \Sigma_1^*)\ st \in L_m(\mathbf{S_2}) \cap L(\mathbf{G})$

(ii) $(\forall s \in L(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2}))(\exists t \in \Sigma_2^*)\ st \in L_m(\mathbf{S_1}/\mathbf{G})$

Condition (i) says that any string of $\mathbf{G}$ that is accepted (but not necessarily marked) by $\mathbf{S_2}$ can be completed to a marked string of $\mathbf{S_2}$ by means of a string that is accepted by $\mathbf{G}$ and $\mathbf{S_1}$. Condition (ii) states that any string that is accepted by $\mathbf{G}$ and $\mathbf{S_1}$ and marked by $\mathbf{S_2}$ can be completed to a marked string of both $\mathbf{S_1}$ and $\mathbf{G}$ by means of a string to which $\mathbf{S_2}$ is indifferent (with $\mathbf{S_2}$ resident in x_{m2}).

Theorem 4.7.1

Let $\mathbf{S_1}$ and $\mathbf{S_2}$ be proper supervisors for $\mathbf{G}$. Subject to conditions (i) and (ii) above, the supervisor $\mathbf{S_1} \wedge \mathbf{S_2}$ is nonblocking for $\mathbf{G}$.

Proof

Let $s \in L((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$. It must be checked that there exists $t \in \Sigma^*$ such that $st \in L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$. Since $s \in L(\mathbf{S_2}/\mathbf{G})$, by condition (i) there is $u \in \Sigma_1^*$ such that $su \in L_m(\mathbf{S_2}) \cap L(\mathbf{G})$. By definition of Σ_1 and the fact that $s \in L(\mathbf{S_1}/\mathbf{G})$ it follows that $su \in L(\mathbf{S_1}/\mathbf{G})$; therefore $su \in L(\mathbf{S_1}/\mathbf{G}) \cap L_m(\mathbf{S_2})$. By condition (ii) there is $v \in \Sigma_2^*$ such that $suv \in L_m(\mathbf{S_1}/\mathbf{G})$; and by definition of Σ_2 we also have $suv \in L_m(\mathbf{S_2})$. This shows that

$$suv \in L_m(\mathbf{S_1}) \cap L_m(\mathbf{G}) \cap L_m(\mathbf{S_2}) = L_m((\mathbf{S_1} \wedge \mathbf{S_2})/\mathbf{G})$$

and the result follows on setting $t = uv$. $\square$

As a straightforward illustration, we apply Theorem 4.7.1 to Small Factory. Set $\mathbf{G} =$ **FACT**, $\mathbf{S_1} =$ **BUFSUP** and $\mathbf{S_2} =$ **BRSUP**. Then we have

$$\Sigma_1 = \{\lambda_1, \mu_1, \beta_2, \lambda_2, \mu_2\}, \quad \Sigma_2 = \{\mu_1, \alpha_1, \beta_1, \lambda_1, \alpha_2, \beta_2\}$$

Let $s \in L(\mathbf{BRSUP/FACT})$. Call the states of **BRSUP** 'idle' and 'active'. If **BRSUP** is active (**MACH2** is broken down), then let $u_1 := \mu_2$ (repair **MACH2**), otherwise $u_1 := \epsilon$ (do nothing). Call the states of **MACHi** 'idle', 'working' and 'down'. If after s was generated **MACH1** was down then let $u_2 := \mu_1$ (repair **MACH1**), otherwise $u_2 := \epsilon$ (do nothing); and set $u := u_1u_2$. Then u is accepted by **BUFSUP**, and after su **BRSUP** is resident at its marker state 'idle'. Now use the fact that with each of **MACH1** and **MACH2** either idle or working there is a string v accepted by **BUFSUP** that returns both machines to idle and **BUFSUP** to its marker state (where the buffer is empty), while always keeping **BRSUP** idle. The string uv then suffices to show that **BUFSUP** $\wedge$ **BRSUP** is nonblocking.

Exercise 4.7.1: Apply Theorem 4.7.1 (or a suitable variation thereof) to show that the modular supervisor for Big Factory (Section 4.5) is nonblocking. Repeat the exercise for Transfer Line (Section 4.6).

Exercise 4.7.2: Consider a manufacturing cell consisting of a robot (**ROB**), input conveyor (**INCON**), input buffer (**INBUF**), machining station (**MS**), output buffer (**OUTBUF**), and output conveyor (**OUTCON**). The operations of **MS** are to download and initialize the machining program, accept a workpiece from **INBUF**, machine it to the specified dimensions, and place it in **OUTBUF**. The preconditions for the process to start are that **MS** should be idle and a workpiece should be available at **INBUF**. **ROB** transfers a workpiece from **INCON** to **INBUF**, provided a workpiece is available, **ROB** is free, and **INBUF** is empty. Similarly, **ROB** transfers a completed workpiece from **OUTBUF** to **OUTCON**. **INBUF** (resp. **OUTBUF**) can be in one of the states: empty (full), being loaded (being unloaded) by the robot, or full (empty). A workpiece follows the path: **INCON, INBUF, MS, OUTBUF, OUTCON**.

Develop a DES model for the workcell with plausible assignments of controllable and uncontrollable events. Investigate both centralized and modular supervision, subject (at least) to the specifications that the buffers never overflow or underflow, and that the supervised system is nonblocking.

Exercise 4.7.3: Four jobs **A1**, **A2**, **A3**, **A4** are to be done with two tools **T1**, **T2**. Each job is to be done exactly once. **A1** consists of an initial operation using **T1**, then a final operation using both **T1** and **T2**. **A2** consists of an initial operation using **T2**, then a final operation using both **T2** and **T1**. **A3** uses only **T1**; **A4** uses only **T2**. The four jobs can be done in any order; interleaving of several jobs at a time is permitted.

The jobs are identified with corresponding "agents"; thus "Ai does job i." Model **T1** on two states, with transitions [0,i11,1] (i=1,2,3) to mean "Ai acquires **T1**, and [1,i10,0] to mean "Ai releases **T1**." Similarly **T2** is modelled on two states with transitions [0,i21,1],[1,i20,0] (i=1,2,4). After a job is finished, the tool or tools are released, in any order, and finally a "job completed" signal is output (event i00, i=1,2,3,4). Thus **A1**, **A2** are each modelled on 6 states, **A3** and **A4** on three. The requirement that the ith job be done exactly once is modelled by the two-state automaton **Di** in which only state 1 is marked, and the appropriate event i?1 disabled there to prevent a repetition. **PLANT** is the synchronous product of **A1**, **A2**, **A3**, **A4**; SPEC is the synchronous product of **T1**, **T2**, **D1**, **D2**, **D3**, **D4**, **ALL**, where **ALL** is obtained from **allevents**.

Find the global supervisor by **supcon**, and then reduce it using **supreduce**. Compute various projections of interest: for instance, focus on tool usage but (by means of **convert**) blur the identity of agents, or focus on agents but blur the identity of tools.

To construct a modular supervisor, first note that **PLANT** and **T** (= **sync**(**T1**,**T2**,**ALL**)) conflict, since for instance if **A1** takes **T1** (event 111) and immediately afterwards **A2** takes **T2** (event 221) then deadlock occurs; and similarly for the event sequence 221,111. These sequences can be ruled out *a priori* by suitable specifications (conflict resolvers).

Exercise 4.7.4: Consider agents **A1**, **A2**, **A3**, each defined on two states with the initial state (only) marked.

A1 has transitions	$\{[0,\gamma,0],\ [0,\ \alpha,\ 1],\ [1,\ \alpha,0]\}$
A2	$\{[0,\gamma,\ 0],\ [0,\ \beta,\ 1],\ [1,\ \beta,0]\}$
A3	$\{[0,\gamma,\ 1],\ [1,\ \gamma,0]\}$

A1 and **A2** can be thought of as operating two switches which control **A3**. If both switches are RESET (state 0) then **A3** can make the transition [1,γ,0] and return home, but if either switch is SET then **A3** is blocked at state 1. Clearly **A1** and **A2** can cycle in such a way that, once **A3** has entered its state 1, it remains blocked forever. Despite this, the overall system **A** = **sync**(**A1**,**A2**,**A3**) is nonblocking in the DES sense. Suggest a possible cure.

Exercise 4.7.5: Dining Philosophers

In this famous problem (due to E.W. Dijkstra) five philosophers (**P1**, ..., **P5**), who spend their lives alternately eating and thinking, are seated at a round table at the center of which is placed a bowl of spaghetti. The table is set with five forks (**F1**, ..., **F5**), one between each pair of adjacent philosophers. So tangled is the spaghetti that a philosopher requires both forks, to his immediate right and left, in order to eat; and a fork may not be replaced on the table until its user has temporarily finished eating and reverts to thinking. No *a priori* constraint is placed on the times at which a philosopher eats or thinks.

Design modular supervisors which guarantee that (1) a fork is used by at most one philosopher at a time, and (2) every philosopher who wishes to eat can eventually do so – i.e. no one is starved out by the eating/thinking habits of others. **Hint:** Model each **P** on states [0] *Thinking*, [1] *Ready*, and [2] *Eating*, with the transition from [1] to [2] controllable; and each **F** on two states [0] *Free* and [1] *In_use*. You may assume that a philosopher can pick up and replace both his forks simultaneously. A fair way to prevent starvation could be to require that no philosopher may commence eating if either of his two neighbors has been ready longer. For this, equip each philosopher with a queue which he and his two neighbors enter when they are ready to eat. Note that the queue for **Pi** should not distinguish between **P(i-1)** and **P(i+1)**, but only enforce priority between **Pi** and one or both of them; it may be modelled on 9 states. Prove that, under your control scheme, anyone who is ready to eat is *guaranteed* eventually to be able to do so: this is a stronger condition than 'nonblocking', as it prohibits 'livelock' behavior such as **P2** and **P5** cycling in such a way as to lock out **P1**.

A *TCT* modular solution along these lines produced a combined on-line controller size of (1557,5370) with corresponding controlled behavior of size (341,1005).

4.8 Synchronization and Event Hiding

Individual DES can be combined into modules by synchronization followed by projection to achieve event hiding and thus encapsulation. However, care must be taken not to attempt to synchronize an uncontrollable specification with a generator, with respect to an uncontrollable event. The correct procedure would be to compute the supremal controllable sublanguage, and then hide the uncontrollable event. Also, care must be taken not to produce blocking or deadlock.

Example 4.8.1: Small Factory

Define **MACH1**, **MACH2**, **BUF2** [buffer with 2 slots] as usual. To plug **MACH1** into **BUF2** requires synchronizing on event 10. Since 10 is uncontrollable, one must compute the supremal controllable sublanguage. For this, take as the specification the (uncontrollable) synchronous product of **MACH1** and **BUF2**: call this **SPEC1**; and as the plant, **MACH1** self-looped with the buffer event 21: call this **SMACH1**.

SPEC1 = sync(MACH1,BUF2) (9,17)
SMACH1 = selfloop(MACH1,[21]) (3,7)
SUPM1B = supcon(SMACH1,SPEC1) (7,12)

MACH1, **BUF2** are now *controllably* synchronized on the shared event 10. Hiding this event, we get

HSUPM1B = **project**(**SUPM1B**,[10])

Thus **HSUPM1B** can be considered as a module with events 11,12,13,21. Let's suppose that the breakdown/repair logic is of no interest, and hide events 12,13. This gives the module

MACH3 = **project**(**HSUPM1B**,[12,13]) (3,5)

Now **MACH3** can be synchronized with **MACH2** on event 21, and events 21,22,23 hidden. This yields the final module, over events 11 ('**MACH1** goes to work') and 20 ('**MACH2** outputs a product').

MACH4 = **sync(MACH3,MACH2)** (9,20)
MACH5 = **project**(**MACH4**,[21,22,23]) (4,7)

This procedure may be compared with the more standard procedure of 'monolithic' design:

MACH6 = **shuffle(MACH1,MACH2)** (9,24)
SPEC2 = **selfloop**(**BUF2**,[11,12,13,20,22,23]) (3,22)
MACH7 = **supcon(MACH6,SPEC2)** (21,49)
MACH8 = **project**(**MACH7**,[10,12,13,21,22,23]) (4,7)

Of course **MACH8** is isomorphic with **MACH5**.

Example 4.8.2: Transfer Line

Systems with feedback loops should be encapsulated by working from inside a loop to the outside. For this system, **M1**, **M2**, **B1**, **B2**, **TU** are created as in Section 4.6. From the block diagram, it makes sense to synchronize **M2** with **B2**, then this result with **TU**.

SP1 = **sync(M2,B2)** (4,5)
SM2 = **selfloop**(**M2**,[5]) (2,4)
SUPM2B2 = **supcon(SM2,SP1)** (3,3)
M3 = **project**(**SUPM2B2**,[4]) (2,2)

M3TU = **sync(M3,TU)** (4,7)
M4 = **project**(**M3TU**,[5]) (3,6)

The module **M4** can now be synchronized with **B1**: since the synchronization event 3 is controllable, this is done just with synchronous product.

M5 = **sync(M4,B1)** (12,29)

Events 3,4,8 are now purely internal, so we define

M6 = **project**(**M5**,[3,4,8]) (6,10)

It remains to synchronize **M6** with **M1**, with respect to the uncontrollable event 2. From analysis of the feedback loop we know that to prevent deadlock it's necessary to enforce the condition

capacity(**B1**) + #3 - #2 - #8 $\geq$ #3 - #6 - #8

or

#2 - #6 $\leq$ 3

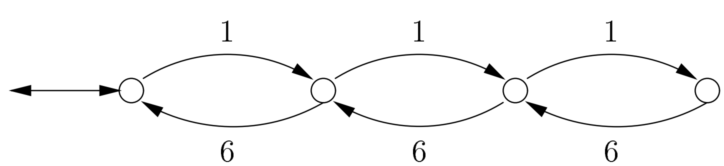
For this construct the transition structure **SP3**, with selfloop event 1. Then

M7 = **sync(M1,M6)** (12,21)
SUPM7SP3 = **supcon(M7,SP3)** (7,11)

and finally

M8 = **project**(**SUPM7SP3**,[2]) (4,6)

M8 displays the correct operation of transfer line with respect to the input event 1 and output event 6. It's equivalent to a buffer of capacity 3, as displayed.



4.9 Notes and References

Modular (specifically, decentralized) supervisory control theory, in the sense of this chapter, originated with the doctoral theses of P.J. Ramadge [T01], F. Lin [T08] and K. Rudie [T23], and related papers [J06, J07, J08, J20]. The Transfer Line of Sect. 4.6 is adapted from Al-Jaar & Desrochers [1988] and Desrochers & Al-Jaar [1995]. The robotics model of Exercise 4.7.2 was suggested by K.P. Valavanis, while the celebrated problem of the Dining Philosophers (Exercise 4.7.5) originated with E.W. Dijkstra [1971] and has been widely reproduced in the literature on concurrency and computer operating systems.

Chapter 5

Hierarchical Supervision of Discrete-Event Systems

5.1 Hierarchical Control Structure

Hierarchical structure is a familiar feature of the control of dynamic systems that perform a range of complex tasks. It may be described generally as a division of control action and the concomitant information processing according to scope. Commonly, the scope of a control action is defined by the extent of its temporal horizon, or by the depth of its logical dependence in a task decomposition. Generally speaking, the broader the temporal horizon of a control and its associated subtask, or the deeper its logical dependence on other controls and subtasks, the higher it is said to reside in the hierarchy. Frequently the two features of broad temporal horizon and deep logical dependency are found together.

In this chapter we formalize hierarchical structure in the control of discrete-event systems (DES), by means of a mild extension of the framework already introduced. While different approaches to hierarchical control might be adopted even within this restricted framework, the theory to be presented captures the basic feature of scope already mentioned, and casts light on an issue that we call *hierarchical consistency*.

In outline our setup will be the following. Consider a two-level hierarchy consisting of a low-level plant $\mathbf{G_{lo}}$ and controller $\mathbf{C_{lo}}$, along with a high-level plant $\mathbf{G_{hi}}$ and controller $\mathbf{C_{hi}}$. These are coupled as shown in Fig. 5.1.1.

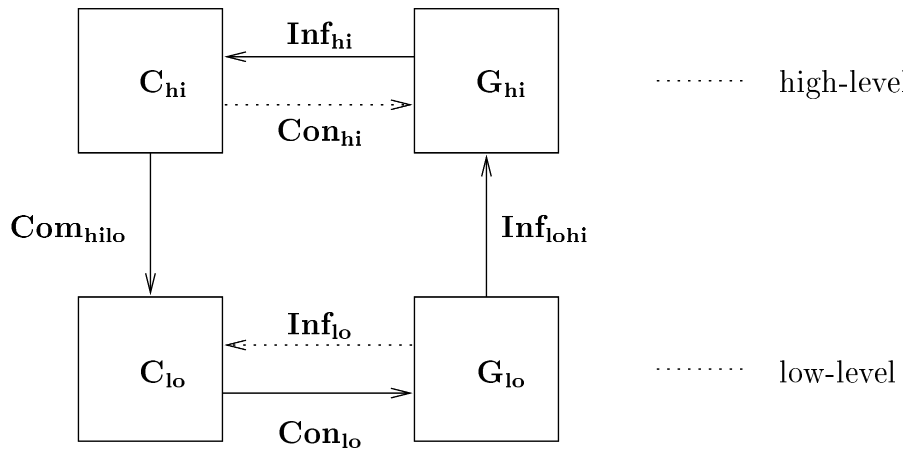


Fig. 5.1.1
Two-Level Control Hierarchy

Our viewpoint is that $\mathbf{G_{lo}}$ is the actual plant to be controlled in the real world by $\mathbf{C_{lo}}$, the operator; while $\mathbf{G_{hi}}$ is an abstract, simplified model of $\mathbf{G_{lo}}$ that is employed for decision-making in an ideal world by $\mathbf{C_{hi}}$, the manager. The model $\mathbf{G_{hi}}$ is refreshed or updated every so often via the information channel (or mapping) labelled $\mathbf{Inf_{lohi}}$ ('information-low-to-high') to $\mathbf{G_{hi}}$ from $\mathbf{G_{lo}}$. Alternatively one can interpret $\mathbf{Inf_{lohi}}$ as carrying information sent up by the operator $\mathbf{C_{lo}}$ to the manager $\mathbf{C_{hi}}$: in our model the formal result will be the same. Another information channel, $\mathbf{Inf_{lo}}$ ('low-level information'), provides conventional feedback from $\mathbf{G_{lo}}$ to its controller $\mathbf{C_{lo}}$, which in turn applies conventional control to $\mathbf{G_{lo}}$ via the control channel labelled $\mathbf{Con_{lo}}$ ('low-level control'). Returning to the high level, we consider that $\mathbf{G_{hi}}$ is endowed with control structure, according to which it makes sense for $\mathbf{C_{hi}}$ to attempt to exercise control over the behavior of $\mathbf{G_{hi}}$ via the control channel $\mathbf{Con_{hi}}$ ('high-level control'), on the basis of feedback received from $\mathbf{G_{hi}}$ via the information channel $\mathbf{Inf_{hi}}$ ('high-level information'). In actuality, the control exercised by $\mathbf{C_{hi}}$ in this way is only virtual, in that the behavior of $\mathbf{G_{hi}}$ is determined entirely by the behavior of $\mathbf{G_{lo}}$, through the updating process mediated by $\mathbf{Inf_{lohi}}$. The structure is, however, completed by the command channel $\mathbf{Com_{hilo}}$ linking $\mathbf{C_{hi}}$ to $\mathbf{C_{lo}}$. The function of $\mathbf{Com_{hilo}}$ is to convey the manager's high level control signals as commands to the operator $\mathbf{C_{lo}}$, which must translate (compile) these commands into corresponding low-level control signals which will actuate $\mathbf{G_{lo}}$ via $\mathbf{Con_{lo}}$. State changes in $\mathbf{G_{lo}}$ will eventually be conveyed in summary form to $\mathbf{G_{hi}}$ via $\mathbf{Inf_{lohi}}$. $\mathbf{G_{hi}}$ is updated accordingly, and then provides appropriate feedback to $\mathbf{C_{hi}}$ via $\mathbf{Inf_{hi}}$. In this way the hierarchical loop is closed. The forward path sequence $\mathbf{Com_{hilo}};\mathbf{Con_{lo}}$ is conventionally designated 'command and control', while the feedback path sequence $\mathbf{Inf_{lohi}};\mathbf{Inf_{hi}}$ could be referred to as 'report and advise'.

As a metaphor, one might think of the command center of a complex system (e.g. manufacturing system, electric power distribution system) as the site of the high-level plant model $\mathbf{G_{hi}}$, where a high-level decision maker (manager) $\mathbf{C_{hi}}$ is in command. The external (real) world and those (operators) coping with it are embodied in $\mathbf{G_{lo}}$, $\mathbf{C_{lo}}$. The questions to be addressed concern the relationship between the behavior required, or expected, by the manager $\mathbf{C_{hi}}$ of his high-level model $\mathbf{G_{hi}}$, and the actual behavior implemented by the operator $\mathbf{C_{lo}}$ in $\mathbf{G_{lo}}$ in the manner described, when $\mathbf{G_{lo}}$ and $\mathbf{Inf_{lohi}}$ are given at the start. It will turn out that a relationship of *hierarchical consistency* imposes rather stringent requirements on $\mathbf{Inf_{lohi}}$ and that, in general, it is necessary to refine the information conveyed by this channel before consistent hierarchical control structure can be achieved. This result accords with the intuition that for effective high-level control the information sent up by the operator to the manager must be timely, and sufficiently detailed for various critical low-level situations to be distinguished.

5.2 Two-Level Controlled Discrete-Event System

For $\mathbf{G_{lo}}$ we take the usual 5-tuple

$$\mathbf{G_{lo}} = (Q, \Sigma, \delta, q_o, Q_m)$$

Here Σ is the set of *event labels*, partitioned into *controllable* elements ($\Sigma_c \subseteq \Sigma$) and *uncontrollable* elements ($\Sigma_u \subseteq \Sigma$); Q is the *state set*; $\delta : Q \times \Sigma \to Q$ is the *transition function* (in general a partial function, defined at each $q \in Q$ for only a subset of events $\sigma \in \Sigma$: in that case we write $\delta(q, \sigma)!$); q_o is the *initial state*; and $Q_m \subseteq Q$ is the subset of *marker states.* The *uncontrolled behavior* of $\mathbf{G_{lo}}$ is the language

$$L_{lo} := L(\mathbf{G_{lo}}) \subseteq \Sigma^*$$

consisting of the (finite) strings $s \in \Sigma^*$ for which the (extended) transition map $\delta : Q \times \Sigma^* \to Q$ is defined.

In this section, as well as Sects. 5.3 - 5.5, we only consider the case $Q_m = Q$, namely all the relevant languages are prefix-closed. This assumption is made for simplicity in focussing on the basic issue of hierarchical consistency. The theory will be generalized to include marking and treat nonblocking in Sect. 5.7.

We recall that if $\mathbf{G}$ is a controlled DES over an alphabet $\Sigma = \Sigma_c \cup \Sigma_u$, and K is a closed sublanguage of Σ^*, then K is *controllable* (with respect to $\mathbf{G}$) if $K\Sigma_u \cap L(\mathbf{G}) \subseteq K$. To every closed language $E \subseteq \Sigma^*$ there corresponds the (closed) *supremal controllable sublanguage* $\sup \mathcal{C}(E \cap L(\mathbf{G}))$. In this chapter it will be convenient to use the notation

$$\sup \mathcal{C}(M) =: M^{\uparrow}$$

Let T be a nonempty set of labels of 'significant events'. T may be thought of as the events perceived by the manager which will enter into the description of the high-level plant model $\mathbf{G_{hi}}$, of which the derivation will follow in due course. First, to model the information channel (or mapping) $\mathbf{Inf_{lohi}}$ we postulate a map

$$\theta : L_{lo} \to T^*$$

with the properties

$$\theta(\epsilon) = \epsilon,$$

$$\theta(s\sigma) = \begin{cases} \text{either } \theta(s) \\ \text{or } \theta(s)\tau, \text{ some } \tau \in T \end{cases}$$

for $s \in L_{lo}$, $\sigma \in \Sigma$ (here and below, ϵ denotes the empty string regardless of alphabet). Such a map θ will be referred to as *causal*. A causal map is, in particular, *prefix-preserving*: if $s \leq s'$ then $\theta(s) \leq \theta(s')$. Intuitively, θ can be used to signal the occurrence of events that depend in some fashion on the past history of the behavior of $\mathbf{G_{lo}}$: for instance θ might produce a fresh symbol τ' whenever $\mathbf{G_{lo}}$ has just generated a positive multiple of 10 of some distinguished symbol σ', but 'remain silent' otherwise.

Exercise 5.2.1: Prove that $\theta : L_{lo} \to T^*$ is causal if and only if it commutes with prefix closure, namely for all sublanguages $K \subseteq L_{lo}$,

$$\overline{\theta(K)} = \theta(\bar{K})$$

$\diamond$

It is convenient to combine θ with $\mathbf{G_{lo}}$ in a unified description. This may be done in standard fashion by replacing the pair $(\mathbf{G_{lo}}, \theta)$ by a Moore generator having output alphabet

$$T_o = T \cup \{\tau_o\}$$

where τ_o is a new symbol ($\notin T$) interpreted as the 'silent output symbol'. To this end write temporarily

$$\tilde{\mathbf{G}}_{\mathbf{lo}} = (\tilde{Q}, \Sigma, T_o, \tilde{\delta}, \omega, \tilde{q}_o, \tilde{Q}_m)$$

Here the items written with a tilde play the same role as in $\mathbf{G_{lo}}$, while $\omega : \tilde{Q} \to T_o$ is the state output map. $\tilde{\mathbf{G}}_{\mathbf{lo}}$ is constructed so that

$$\tilde{\delta}(\tilde{q}_o, s)! \quad \text{iff} \quad \delta(q_o, s)! \quad s \in \Sigma^*$$

Thus $\tilde{\mathbf{G}}_{\mathbf{lo}}$ generates exactly the language L_{lo}. For ω define

$$\omega(\tilde{q}_o) = \tau_o$$

while if $\tilde{\delta}(\tilde{q}_o, s\sigma)!$ then

$$\begin{aligned} \omega(\tilde{\delta}(\tilde{q}_o, s\sigma)) &= \tau_o \quad \text{if } \theta(s\sigma) = \theta(s) \\ \omega(\tilde{\delta}(\tilde{q}_o, s\sigma)) &= \tau \quad \text{if } \theta(s\sigma) = \theta(s)\tau \end{aligned}$$

Thus ω outputs the silent symbol τ_o if θ outputs 'nothing new', and outputs the 'fresh' symbol $\tau \in T$ otherwise.

An abstract construction of $\tilde{\mathbf{G}}_{\mathbf{lo}}$ is straightforward, using the canonical identification of states with the cells (equivalence classes) of a suitable right congruence on strings. For $s, s' \in L_{lo}$ define

$$s \equiv s' \pmod{L_{lo}} \quad \text{iff} \quad (\forall u \in \Sigma^*) su \in L_{lo} \Leftrightarrow s'u \in L_{lo}$$

Next define $\hat{\omega} : L_{lo} \to T_o$ according to

$$\begin{aligned} \hat{\omega}(\epsilon) &= \tau_o \\ \hat{\omega}(s\sigma) &= \begin{cases} \tau_o & \text{if} \quad \theta(s\sigma) = \theta(s) \\ \tau & \text{if} \quad \theta(s\sigma) = \theta(s)\tau \end{cases} \end{aligned}$$

and let, for $s, s' \in L_{lo}$,

$$s \equiv s' \pmod{\theta} \quad \text{iff} \quad \hat{\omega}(s) = \hat{\omega}(s') \quad \text{and}$$

$$(\forall u \in \Sigma^*, t \in T^*)[su \in L_{lo} \quad \& \quad s'u \in L_{lo} \Rightarrow (\theta(su) = \theta(s)t \Leftrightarrow \theta(s'u) = \theta(s')t)]$$

It is readily shown that equivalence (mod θ) is a right congruence on L_{lo}. As equivalence (mod L_{lo}) is a right congruence too, so is their common refinement (i.e. their meet in the lattice of right congruences); and the cells of this refinement furnish the states of $\tilde{\mathbf{G}}_{\mathbf{lo}}$.

From this point on we shall assume that the starting point of our hierarchical control problem is the unified description $\tilde{\mathbf{G}}_{\mathbf{lo}}$. So we drop the tilde and write

$$\mathbf{G}_{\mathbf{lo}} = (Q, \Sigma, T_o, \delta, \omega, q_o, Q_m) \tag{2.1}$$

with $Q_m = Q$.

At this stage we temporarily define $\mathbf{G}_{\mathbf{hi}}$. For this we note that, in the absence of any control action, $\mathbf{G}_{\mathbf{lo}}$ generates the uncontrolled language L_{lo}. For now, $\mathbf{G}_{\mathbf{hi}}$ will be taken as the canonical recognizer (in the generator sense) for the image of L_{lo} under θ:

$$L(\mathbf{G}_{\mathbf{hi}}) = \theta(L_{lo}) \subseteq T^*$$

and we write $L(\mathbf{G}_{\mathbf{hi}}) =: L_{hi}$. As yet, however, the event label alphabet T of $\mathbf{G}_{\mathbf{hi}}$ needn't admit any natural partition into controllable and uncontrollable subalphabets; that is, $\mathbf{G}_{\mathbf{hi}}$ needn't possess any natural control structure. This defect will be remedied in the next section.

The following simple example will be used throughout. Following the integer labelling conventions of *TCT* we define the state set and alphabets of $\mathbf{G}_{\mathbf{lo}}$ in (2.1) according to

$$\begin{aligned} Q &= \{0,1,2,3,4\}, \quad q_o = 0 \\ \Sigma &= \{0,1,2,3,4\} \\ T &= \{\alpha, \beta\}, \qquad \tau_o = o \end{aligned}$$

In Σ the odd-numbered elements are controllable, the even-numbered elements uncontrollable. The state transition and output structure of $\mathbf{G_{lo}}$ is displayed in Fig. 5.2.1. along with a canonical recognizer for L_{hi}. Observe that, whether or not $\tau \in \{\alpha, \beta\}$ can be disabled as 'next output' by a supervisory controller that can disable the controllable elements of Σ, depends on the current state $q \in Q$ of $\mathbf{G_{lo}}$: for instance $\tau = \alpha$ can be disabled at $q = 2$ but not at $q = 0, 1, 3$ or 4. Thus $\mathbf{G_{hi}}$ does not yet possess natural control structure.

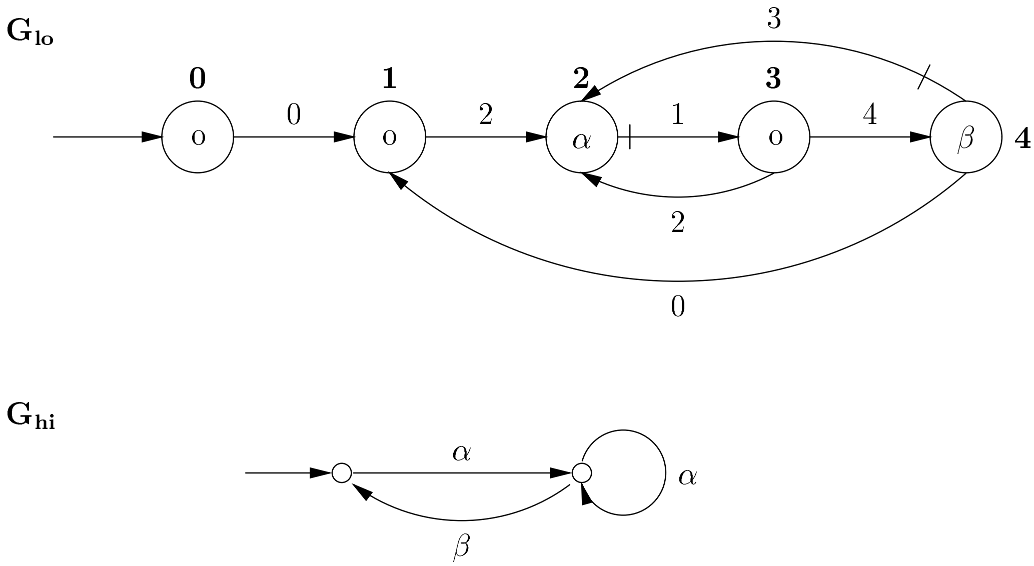


Fig. 5.2.1
Low and High-level DES

Exercise 5.2.2: Show that equivalence (mod θ) is a right congruence on L_{lo}.

5.3 High-Level Control Structure

In this section we indicate how to refine the descriptions of $\mathbf{G_{lo}}$ and $\mathbf{G_{hi}}$ in order to equip $\mathbf{G_{hi}}$ with control structure, so that a high-level controller $\mathbf{C_{hi}}$ that observes only the state of $\mathbf{G_{hi}}$ can make meaningful control decisions. By way of control structure we adopt the (usual) supervisory structure having the same type as in $\mathbf{G_{lo}}$. We shall refine the state structure of $\mathbf{G_{lo}}$, extend the high-level event alphabet T, and partition the extension into controllable and uncontrollable subsets.

Conceptually these operations are carried out as follows. Referring to the example above, consider a reachability tree for L_{lo} with initial state $q = 0$ as the root. The first few levels of the tree are displayed in Fig. 5.3.1. Each node of the tree is labelled with the corresponding value $\tau' \in T_o = \{o, \alpha, \beta\}$ of the output map ω, and is then called a τ'-*node*.

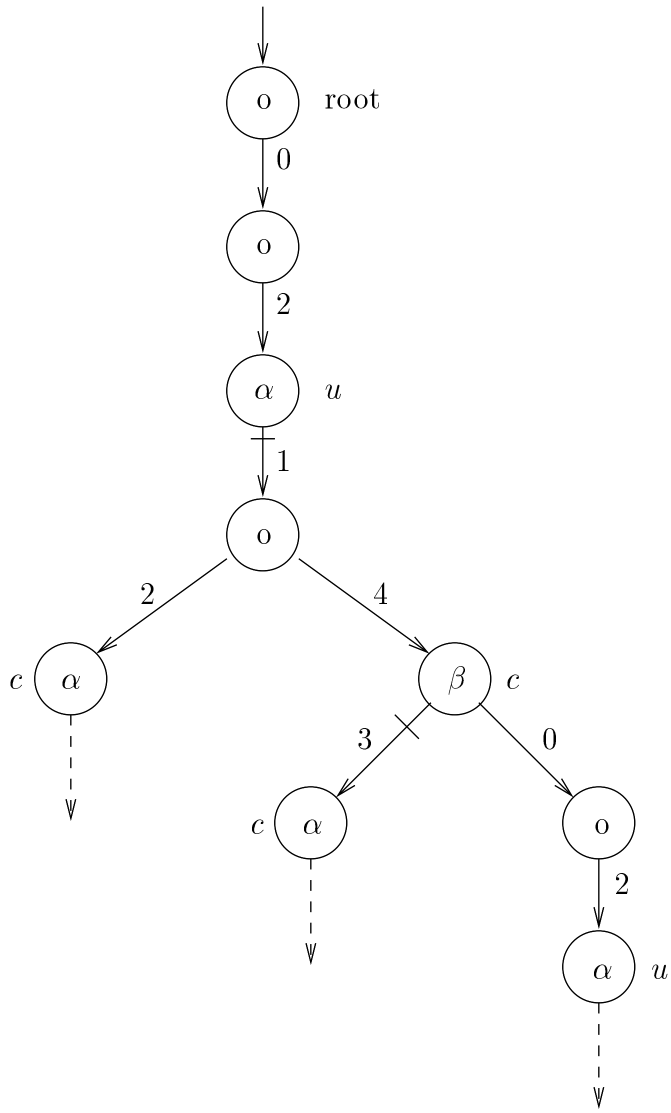


Fig. 5.3.1
Reachability Tree

In general it will be convenient to write $\hat{\omega} : L_{lo} \to T_o$ for the output map on strings defined by $\hat{\omega}(s) = \omega(\delta(q_o, s))$ whenever $s \in L_{lo}$, i.e. $\delta(q_o, s)!$. With a slight abuse of notation we also write $\hat{\omega} : \mathcal{N} \to T_o$ where $\mathcal{N}$ is the node set of the reachability tree of L_{lo}.

In the tree, τ'-nodes with $\tau' = \tau_o = o$ are *silent*; τ'-nodes with $\tau' \in T = \{\alpha, \beta\}$ are *vocal.* A *silent path* in the tree is a path joining two vocal nodes, or the root to a vocal node, all of whose intermediate nodes (if any) are silent. Schematically a silent path has the form

$$n \xrightarrow{\sigma} s \xrightarrow{\sigma'} s' \longrightarrow ... \longrightarrow s'' \xrightarrow{\sigma''} n'$$

where the starting node n is either vocal or the root node, and where the intermediate silent nodes $s, s', ..., s''$ may be absent. Thus to every vocal node n' there corresponds a unique silent path of which it is the terminal node. A silent path is *red* if at least one of its transition labels $\sigma \in \Sigma$ is controllable; otherwise it is *green*. Now color each vocal node red or green according to the color of its corresponding silent path. Create an extended output alphabet T_{ext} as follows. Starting with $T_{ext} = \{\tau_o\}$, for each $\tau \in T$ adjoin a new symbol $\tau_c \in T_{ext}$ if some τ-node in the tree is red; similarly adjoin $\tau_u \in T_{ext}$ if some τ-node in the tree is green. Now define $\hat{\omega}_{ext} : \mathcal{N} \to T_{ext}$ according to

$$\hat{\omega}_{ext}(n) = \tau_o \quad \text{if } n \text{ is silent}$$

$$\begin{aligned} \hat{\omega}_{ext}(n) = \tau_c \qquad & \text{if } \hat{\omega}(n) = \tau \in T \text{ and color}(n) = \text{red} \\ \hat{\omega}_{ext}(n) = \tau_u \qquad & \text{if } \hat{\omega}(n) = \tau \in T \text{ and color}(n) = \text{green} \end{aligned}$$

Define the extended tree to be the original tree with the node labelling determined by $\hat{\omega}_{ext}$. In Fig. 5.3.1, vocal nodes are labelled c or u accordingly. It is clear that $\hat{\omega}_{ext}$ in turn determines an extension

$$\theta_{ext} : L_{lo} \to T^*_{ext}$$

Evidently θ is recovered from θ_{ext} as follows: Define $P : T^*_{ext} \to T^*$ according to

$$P(\tau_c) = P(\tau_u) = \tau, \quad \tau \in T,$$

$$P(\epsilon) = \epsilon, \quad P(tt') = P(t)P(t'), \quad t, t' \in T^*_{ext}$$

The line just written expresses the property that P is *catenative*. So P just maps the new output symbols in any string back to where they came from. Then

$$\theta = P \cdot \theta_{ext}$$

Finally, define

$$\mathbf{G_{lo,ext}} = (Q_{ext}, \Sigma, T_{ext}, \delta_{ext}, \omega_{ext}, q_o, Q_{ext})$$

from the current transition structure $(Q, \Sigma, \delta, q_o, Q)$ and the map θ_{ext} in just the way $\tilde{\mathbf{G}}_{\mathbf{lo}}$ was defined (Sect. 5.2) in terms of $\mathbf{G_{lo}}$ and θ.

By the construction it is seen that $|T_{ext}| \leq 2|T| + 1$: the number of non-silent output symbols has at most doubled, as each old output symbol has now split into controllable and uncontrollable siblings (in some cases one sibling may be absent). It will be shown that the number of states has at most doubled as well. For this we extend the domain of the color map to include words of L_{lo}. Returning to the reachability tree of L_{lo}, color the silent nodes by the same rule as used previously for the vocal nodes (and color the root node green). For $s \in L_{lo}$ define node$(s) \in \mathcal{N}$ to be the node reached by s, and define color(s) = color(node(s)). For $s, s' \in L_{lo}$ define $s \equiv s'$ to mean (cf. Sect. 5.2)

$$s \equiv s' \pmod{L_{lo}} \quad \text{and} \quad s \equiv s' \pmod{\theta}$$

We claim that if $s \equiv s'$ and color(s) = color(s'), then for all $u \in \Sigma^*$ such that $su \in L_{lo}$ it is the case that $su \equiv s'u$ and color(su) = color$(s'u)$. In fact, the first statement follows by the observation (Sect. 5.2) that $\equiv$ is a right congruence on L_{lo}. Thus we know that $s'u \in L_{lo}$, and that for $v \leq u$, if $\theta(sv) = \theta(s)t$ for some $t \in T^*$ then $\theta(s'v) = \theta(s')t$. In other words, 'the output behaviors (under θ) of su (resp. $s'u$) coincide between s (resp. s') and su (resp. $s'u$)'. Since su and $s'u$ share the suffix u, it follows immediately by the definition of color on strings that color(su) = color$(s'u)$, and the second statement follows, as claimed. The upshot of this argument is that the common refinement of the right congruence $\equiv$, and the equivalence defined by equality of color, is again a right congruence on L_{lo}. It is, of course, the right congruence that provides the state structure of $\mathbf{G_{lo,ext}}$. Thus in passing from $\mathbf{G_{lo}}$ to $\mathbf{G_{lo,ext}}$ each state of $\mathbf{G_{lo}}$ is split at most once, into colored siblings. It follows that $|Q_{ext}| \leq 2|Q|$.

In the sections to follow it will be assumed that the foregoing construction has been carried out, namely our new starting point will be the Moore transition structure $\mathbf{G_{lo,ext}}$ as described. The property of $\mathbf{G_{lo,ext}}$ that each output $\tau \in T_{ext}$ is unambiguously controllable or uncontrollable in the sense indicated, will be summarized by saying that $\mathbf{G_{lo,ext}}$ is *output-control-consistent*. While we have not yet presented an algorithm (as distinct from a conceptual procedure) to pass from $\mathbf{G_{lo}}$ to $\mathbf{G_{lo,ext}}$, such an algorithm exists at least in case $|Q| < \infty$ (see Sect. 5.7). The result for our running example is displayed in Fig. 5.3.2, along with the extended high-level language

$$\theta_{ext}(L(\mathbf{G_{lo,ext}})) \subseteq T^*_{ext}$$

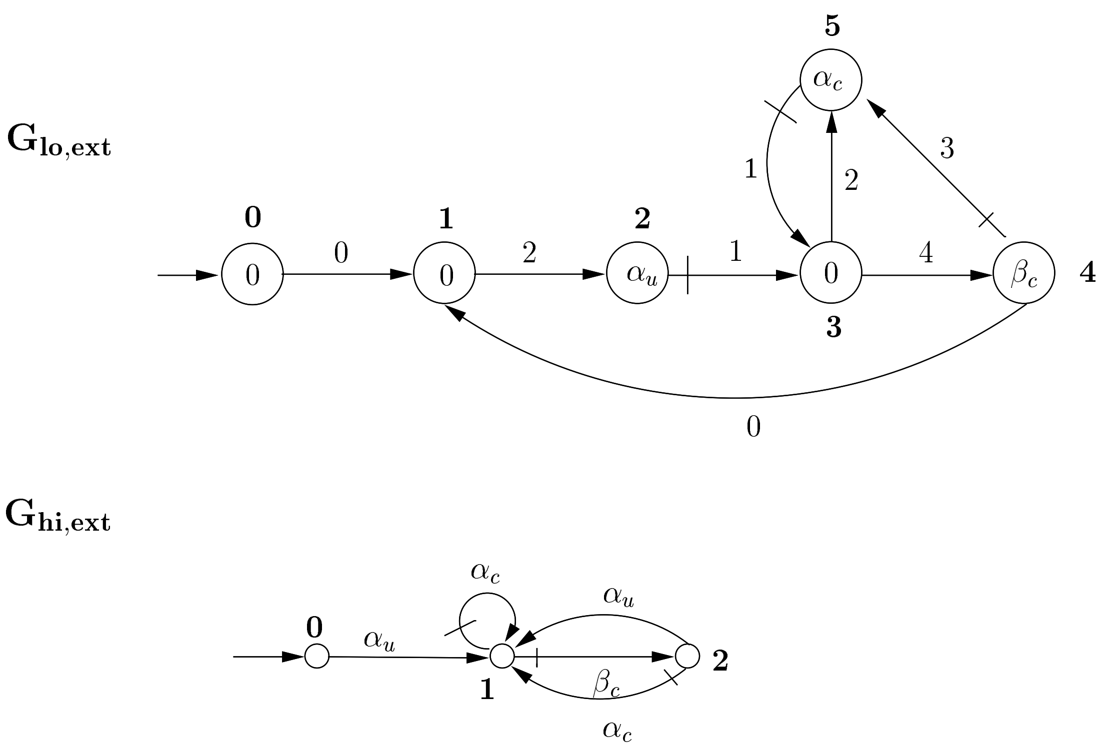


Fig. 5.3.2
Output Control Consistency

In *TCT*, a DES **GLO** with state outputs is referred to as "vocalized" and is set up using the "vocalize" option in **create**. State outputs τ can be numbered 10,...,99. The corresponding structure **GHI** is given by **higen(GLO)**. To extend **GLO** to be output-control-consistent, compute

$$\mathbf{OCGLO} = \mathbf{outconsis(GLO)}$$

with high-level result

$$\mathbf{OCGLOHI} = \mathbf{higen(OCGLO)}$$

In this process *TCT* will create event siblings $\tau_u = 100$, $\tau_c = 101$ from $\tau = 10$, and so forth.

For completeness we provide the formal, albeit cumbersome, definition of output control consistency of a Moore transition structure

$$\mathbf{G} = (Q, \Sigma, T_o, \delta, \omega, q_o, Q_m) \tag{3.1}$$

where the input alphabet $\Sigma = \Sigma_c \dot\cup \Sigma_u$ and the output alphabet $T_o = \{\tau_o\} \dot\cup T$ with $T = T_c \dot\cup T_u$. As before write $\hat\omega(s)$ for $\omega(\delta(q_o, s))$. Then $\mathbf{G}$ is *output-control-consistent* if, for every string $s \in L(\mathbf{G})$ of the form

$$s = \sigma_1\sigma_2...\sigma_k \quad \text{ or respectively } \quad s = s'\sigma_1\sigma_2...\sigma_k$$

(where $s' \in \Sigma^+$, $\sigma_i \in \Sigma$) with

$$\hat\omega(\sigma_1\sigma_2...\sigma_i) = \tau_o \quad (1 \le i \le k-1), \quad \hat\omega(s) = \tau \ne \tau_o$$

or respectively

$$\hat\omega(s') \ne \tau_o, \quad \hat\omega(s'\sigma_1\sigma_2...\sigma_i) = \tau_o \quad (1 \le i \le k-1), \quad \hat\omega(s) = \tau \ne \tau_o$$

it is the case that

if $\tau \in T_c$ then for some i $(1 \le i \le k)$, $\sigma_i \in \Sigma_c$
if $\tau \in T_u$ then for all i $(1 \le i \le k)$, $\sigma_i \in \Sigma_u$

To conclude this section we return to the output-control-consistent structure $\mathbf{G_{lo,ext}}$ and corresponding structure $\mathbf{G_{hi,ext}}$, as above, where from now on the subscript 'ext' will be dropped. While the usefulness of output-control-consistency will be demonstrated in the next section, the following exercise brings out some of its limitations.

Exercise 5.3.1: With $\mathbf{G_{lo}}$ output-control-consistent, construct examples where

(i) $K_{lo} \subseteq L_{lo}$ is controllable with respect to $\mathbf{G_{lo}}$, but $K_{hi} := \theta(K_{lo})$ is not controllable with respect to $\mathbf{G_{hi}}$.

(ii) $K_{hi} \subseteq L_{hi}$ is controllable with respect to $\mathbf{G_{hi}}$, but $K_{lo} := \theta^{-1}(K_{hi})$ is not controllable with respect to $\mathbf{G_{lo}}$. Furthermore there is no controllable sublanguage $K'_{lo} \subseteq L_{lo}$ such that $\theta(K'_{lo}) = K_{hi}$.

Exercise 5.3.2: With $\mathbf{G_{lo}}$ output-control-consistent, assume $K_{hi} \subseteq L_{hi}$, and $K_{lo} := \theta^{-1}(K_{hi})$ is controllable with respect to $\mathbf{G_{lo}}$. Show that K_{hi} is controllable with respect to $\mathbf{G_{hi}}$.

5.4 Hierarchical Control Action

In this section we relate supervisory control defined by the high-level controller (supervisor) $\mathbf{C_{hi}}$ to the appropriate low-level control exercised by $\mathbf{C_{lo}}$, thus defining the command and control path consisting of the command channel $\mathbf{Com_{hilo}}$ followed by the control channel $\mathbf{Con_{lo}}$ shown in Fig. 5.1.1.

High-level supervisory control is determined by a selection of high-level controllable events to be disabled, on the basis of high-level past history. That is, $\mathbf{C_{hi}}$ is defined by a map

$$\gamma_{hi} : L_{hi} \times T \to \{0, 1\}$$

such that $\gamma_{hi}(t, \tau) = 1$ for all $t \in L_{hi}$ and $\tau \in T_u$. As usual, if $\gamma_{hi}(t, \tau) = 0$ the event (labelled) τ is said to be *disabled*; otherwise τ is *enabled*; of course, only controllable events ($\tau \in T_c$) can be disabled. The result of applying this control directly on the generating action of $\mathbf{G_{hi}}$ would be to synthesize the closed-loop language

$$L(\gamma_{hi}, \mathbf{G_{hi}}) \subseteq T^*$$

say. In the standard theory, implementation of $\mathbf{C_{hi}}$ would amount to the construction of a suitable automaton (supervisor) over T as input alphabet, and the factorization of γ_{hi} through its state space (X, say) to create an equivalent state feedback control $\psi : X \times T \to \{0, 1\}$. However, in the hierarchical control loop direct implementation of $\mathbf{C_{hi}}$ is replaced by command and control. The action of $\mathbf{C_{hi}}$ on $\mathbf{G_{hi}}$ must be mediated via $\mathbf{Com_{hilo}}$ and $\mathbf{Con_{lo}}$ as already described. To this end, assuming γ_{hi} is given, define the high-level *disabled-event map*

$$\Delta_{hi} : L_{hi} \to Pwr(T_c)$$

($Pwr(\cdot)$ denotes power set) according to

$$\Delta_{hi}(t) = \{\tau \in T_c | \gamma_{hi}(t, \tau) = 0\}$$

Correspondingly we may define the low-level disabled-event map

$$\Delta_{lo} : L_{lo} \times L_{hi} \to Pwr(\Sigma_c)$$

according to

$$\begin{aligned}\Delta_{lo}(s,t) = \{\sigma \in \Sigma_c | (\exists s' \in \Sigma_u^*) s\sigma s' \in L_{lo} \\ \& \quad \hat{\omega}(s\sigma s') \in \Delta_{hi}(t) \\ \& \quad (\forall s'') s'' < s' \Rightarrow \hat{\omega}(s\sigma s'') = \tau_o\}\end{aligned}$$

Observe that the explicit t-dependence of Δ_{lo} factors through the subset evaluation $\Delta_{hi}(t)$; in other words, Δ_{lo} can be evaluated by examination of the structure of $\mathbf{G_{lo}}$ alone, once the subset $\Delta_{hi}(t)$ of high-level events to be disabled has been announced by $\mathbf{C_{hi}}$. The definition says that $\Delta_{lo}(s,t)$ is just the set of low-level controllable events that must be disabled immediately following the generation of s (in $\mathbf{G_{lo}}$) and of t (in $\mathbf{G_{hi}}$) in order to guarantee the nonoccurrence of any $\tau \in \Delta_{hi}(t)$ as the next event in $\mathbf{G_{hi}}$. Of course such a guarantee is actually provided only if, for the given pair (s,t), the set of uncontrollable strings leading to the next occurrence of τ is empty:

$$\begin{aligned}\{s' \in \Sigma_u^+ | ss' \in L_{lo} \quad \& \quad \hat{\omega}(ss') = \tau \\ \& \quad (\forall s'' \in \Sigma^+) s'' < s' \Rightarrow \hat{\omega}(ss'') = \tau_o\} = \emptyset\end{aligned}$$

As will be seen, the result of our construction in Sect. 5.3 of an output-control-consistent structure $\mathbf{G_{lo}}$ is that the required guarantee is provided when necessary.

When the hierarchical loop is closed through $\mathbf{Inf_{lohi}}$, a string $s \in L_{lo}$ is mapped to $t = \theta(s) \in L_{hi}$. Then the control implemented by C_{lo} will be given by

$$\gamma_{lo}(s,\sigma) = \begin{cases} 0 & \text{if } \sigma \in \Delta_{lo}(s,\theta(s)) \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

Now suppose that a nonempty closed 'legal' (or specification) language $E_{hi} \subseteq L_{hi}$ is specified to the high-level controller $\mathbf{C_{hi}}$. We assume that E_{hi} is controllable with respect to the high-level model structure; that is,

$$E_{hi} T_u \cap L_{hi} \subseteq E_{hi}$$

In accordance with standard theory, E_{hi} would be synthesized as the controlled behavior of $\mathbf{G_{hi}}$ by use of a suitable control law γ_{hi}. In the standard theory the determination of γ_{hi} is usually not unique; however, γ_{hi} must always satisfy

$$\gamma_{hi}(t,\tau) = 0 \quad \text{iff} \quad t \in E_{hi}, \quad t\tau \in L_{hi} - E_{hi}$$

Define E_{lo} to be the (maximal) behavior in $\mathbf{G_{lo}}$ that would be transmitted by $\mathbf{Inf_{lohi}}$ as behavior E_{hi} in the high-level model $\mathbf{G_{hi}}$:

$$E_{lo} := \theta^{-1}(E_{hi}) \subseteq L_{lo} \tag{4.2}$$

Since $L_{hi} = \theta(L_{lo})$ we have $\theta(E_{lo}) = E_{hi}$. Clearly E_{lo} is closed; but in general it will not be true that E_{lo} is controllable with respect to $\mathbf{G_{lo}}$. The main result of this section states that by use of the control (4.1) the closed-loop language $L(\gamma_{lo}, \mathbf{G_{lo}})$ synthesized in $\mathbf{G_{lo}}$ is made as large as possible subject to the constraint (4.2).

Theorem 5.4.1

Under the foregoing assumptions

$$L(\gamma_{lo}, \mathbf{G_{lo}}) = E_{lo}^{\uparrow}$$

Proof

It suffices to show the following:

1. $L(\gamma_{lo}, \mathbf{G_{lo}})$ is controllable with respect to $\mathbf{G_{lo}}$.
2. $L(\gamma_{lo}, \mathbf{G_{lo}}) \subseteq E_{lo}$.
3. For any $\mathbf{G_{lo}}$-controllable sublanguage $K \subseteq E_{lo}$ we have $K \subseteq L(\gamma_{lo}, \mathbf{G_{lo}})$.

In the proof we write $L(\mathbf{G_{lo}}) =: L_{lo}$, $L(\mathbf{G_{hi}}) =: L_{hi}$ and $L(\gamma_{lo}, \mathbf{G_{lo}}) =: K_{lo}$. Clearly K_{lo} is nonempty and closed.

1. By the definition of γ_{lo} we have

$$\gamma_{lo}(s, \sigma) = \begin{cases} 0 & \text{if } \sigma \in \Delta_{lo}(s, \theta(s)) \subseteq \Sigma_c \\ 1 & \text{otherwise} \end{cases}$$

Since the closed-loop behavior in $\mathbf{G_{lo}}$ is obtained by disabling a subset (possibly null) of controllable events following the generation of any string of L_{lo}, it follows that for all $s \in K_{lo}$, $\sigma \in \Sigma_u$ we have $\gamma_{lo}(s, \sigma) = 1$, and therefore

$$K_{lo}\Sigma_u \cap L_{lo} \subseteq K_{lo},$$

namely K_{lo} is controllable, as claimed.

2. Since $E_{lo} = \theta^{-1}(E_{hi})$ it suffices to show that

$$\theta(K_{lo}) \subseteq E_{hi}$$

and we proceed by induction on length of strings. Because both K_{lo} and E_{hi} are nonempty and closed we have

$$\epsilon \in \theta(K_{lo}) \cap E_{hi}$$

Assume that $t \in T^*$, $\tau \in T$, and $t\tau \in \theta(K_{lo})$. Clearly $t \in \theta(K_{lo})$ and $t\tau \in L_{hi}$. Invoking the inductive assumption yields $t \in E_{hi}$. Now if $\tau \in T_u$ then

$$t\tau \in E_{hi}T_u \cap L_{hi};$$

and by controllability of E_{hi}, $t\tau \in E_{hi}$. On the other hand if $\tau \in T_c$ then by the fact that $\mathbf{G_{lo}}$ is output-control-consistent there exist

$$s \in \Sigma^*, \quad \sigma \in \Sigma_c, \quad s' \in \Sigma_u^*$$

such that

$$s\sigma s' \in K_{lo}, \quad \theta(s) = t, \quad \theta(s\sigma s') = t\tau$$

By definition of γ_{lo}, $\sigma \notin \Delta_{lo}(s,t)$; therefore

$$\hat{\omega}(s\sigma s') = \tau \notin \Delta_{hi}(t)$$

so again $t\tau \in E_{hi}$.

3. Let $K \subseteq E_{lo}$ be nonempty, and controllable with respect to $\mathbf{G_{lo}}$. Since E_{lo} and K_{lo} are both closed, it can be assumed without loss of generality that K is closed. By induction on length of strings it will be shown that $K \subseteq K_{lo}$. First, $\epsilon \in K \cap K_{lo}$. Now let $s\sigma \in K$. Since K is closed, $s \in K$. Invoking the inductive assumption, $s \in K_{lo}$. Since $K \subseteq E_{lo} \subseteq L_{lo}$ we have $s\sigma \in L_{lo}$. Now if $\sigma \in \Sigma_u$ then $\gamma_{lo}(s,\sigma) = 1$ and therefore $s\sigma \in K_{lo}$. Suppose on the other hand that $\sigma \in \Sigma_c$. To show that $s\sigma \in K_{lo}$ it must be shown that $\gamma_{lo}(s,\sigma) = 1$, or equivalently

$$\sigma \notin \Delta_{lo}(s, \theta(s))$$

Assuming the contrary and setting $t := \theta(s)$ we have by definition of $\Delta_{lo}(s,t)$:

$$(\exists s' \in \Sigma_u^*) s\sigma s' \in L_{lo} \quad \& \quad \hat{\omega}(s\sigma s') \in \Delta_{hi}(t) \quad \& \quad (\forall s'')s'' < s' \Rightarrow \hat{\omega}(s\sigma s'') = \tau_o$$

Since $s\sigma \in K$ and K is controllable it results that $s\sigma s' \in K$. Let $\hat{\omega}(s\sigma s') = \tau$. Then $\theta(s\sigma s') = t\tau$. But $\tau \in \Delta_{hi}(t)$ implies $\gamma_{hi}(t,\tau) = 0$, so $t\tau \in L_{hi} - E_{hi}$. That is $t\tau \notin E_{hi}$, namely *not* $\theta(K) \subseteq E_{hi}$. But this contradicts the fact that $K \subseteq E_{lo} = \theta^{-1}(E_{hi})$. Therefore $\gamma_{lo}(s,\sigma) = 1$ after all, so that $s\sigma \in K_{lo}$ as required. □

Obviously the transmitted high-level behavior will satisfy the required legal constraint:

$$\theta(L(\gamma_{lo}, \mathbf{G_{lo}})) \subseteq E_{hi} \tag{4.3}$$

but in general the inclusion will be proper. That is, while the 'expectation' of the high-level controller $\mathbf{C_{hi}}$ on using the control γ_{hi} might ideally be the synthesis in $\mathbf{G_{hi}}$ of the controllable behavior E_{hi}, only a subset of this behavior can in general actually be realized. The reason is simply that a call by $\mathbf{C_{hi}}$ for the disablement of some high-level event $\tau \in T_c$ may require $\mathbf{C_{lo}}$ (i.e. the control γ_{lo}) to disable paths in $\mathbf{G_{lo}}$ that lead directly to outputs other than τ. However, this result is the best that can be achieved under the current assumptions about $\mathbf{G_{lo}}$.

The condition stated in Theorem 5.4.1 will be called *low-level hierarchical consistency.* Intuitively it guarantees that the updated behavior of $\mathbf{G_{hi}}$ will always satisfy the high-level legal constraint, and that the 'real' low-level behavior in $\mathbf{G_{lo}}$ will be as large as possible subject to this constraint. Nonetheless, the high-level behavior expected in $\mathbf{G_{hi}}$ by the manager may be larger than what the operator of $\mathbf{G_{lo}}$ can optimally report.

To conclude this section consider again the running example with $\mathbf{G_{lo}}$ (i.e. $\mathbf{G_{lo,ext}}$) and $\mathbf{G_{hi}}$ (i.e. $\mathbf{G_{hi,ext}}$) as displayed in Fig. 5.3.2.

First suppose that the transition graph of E_{hi} coincides with that of $\mathbf{G_{hi}}$ except that the (controllable) transition $[2, \alpha_c, 1]$ has been deleted. It is clear that E_{hi} is a controllable sublanguage of L_{hi}. The corresponding control law γ_{lo} requires merely the disablement of event 3 at state 4 in $\mathbf{G_{lo}}$ (in this simple example, state-based control with no additional memory is sufficient). It is evident that

$$\theta(L(\gamma_{lo}, \mathbf{G_{lo}})) = E_{hi} \ .$$

By contrast, suppose instead that E_{hi} is derived from $\mathbf{G_{hi}}$ by deletion of the selfloop $[1, \alpha_c, 1]$. Then γ_{lo} must disable event 1 at state 2 in $\mathbf{G_{lo}}$, with the unwanted side effect that state 4 in $\mathbf{G_{lo}}$, with output β_c, can never be reached. The manager is chagrined to find that the behavior reported by the operator is much less than he expected:

$$\begin{aligned} \theta(L(\gamma_{lo}, \mathbf{G_{lo}})) &= \{\varepsilon, \alpha_u\} \\ &\subsetneqq E_{hi} \end{aligned}$$

Exercise 5.4.1: Assume that $E_{hi} \subseteq L(\mathbf{G_{hi}})$ is nonempty and closed, but not necessarily controllable, and set $E_{lo} = \theta^{-1}(E_{hi})$. While it is always true that

$$\theta(E_{lo}^{\uparrow}) \subseteq \theta(E_{lo}) = E_{hi}$$

it may be true that

$$\theta(E_{lo}^{\uparrow}) \supsetneqq E_{hi}^{\uparrow}$$

Provide an example to illustrate this situation. In intuitive, 'real world' terms explain why, in general, this result might not be unexpected.

5.5 Hierarchical Consistency

Let $E_{hi} \subseteq L_{hi}$ be closed and controllable and let $\mathbf{G_{lo}}$ be output-control-consistent. It was noted in the previous section that the inclusion

$$\theta((\theta^{-1}(E_{hi}))^{\uparrow}) \subseteq E_{hi} \tag{5.1}$$

may turn out to be strict. Intuitively, the behavior E_{hi} 'expected' by the manager in $\mathbf{G_{hi}}$ may be larger than what the operator can actually realize: the manager is 'overoptimistic' in respect to the efficacy of the command-control process. If equality does hold in (5.1) for every closed and controllable language $E_{hi} \subseteq L_{hi}$, the pair $(\mathbf{G_{lo}}, \mathbf{G_{hi}})$ will be said to possess *hierarchical consistency.* In that case, by Theorem 5.4.1, the command and control process defined in Sect. 5.4 for E_{hi} will actually synthesize E_{hi} in $\mathbf{G_{hi}}$. In the terminology of hierarchical control, every high-level 'task' (represented by a choice of E_{hi}) will be successfully 'decomposed' and executed in the hierarchical control loop.

Achieving equality in (5.1) for arbitrary controllable specification languages E_{hi} in general requires a further refinement of the transition structure of $\mathbf{G_{lo}}$, in other words, the possibly costly step of enhancing the information sent up by $\mathbf{C_{lo}}$ to $\mathbf{C_{hi}}$ (or by $\mathbf{G_{lo}}$ to $\mathbf{G_{hi}}$, depending on one's interpretation of the setup); of course such enhancement might or might not be feasible in an application. Referring to the reachability tree for $L(\mathbf{G_{lo,ext}})$ as described in Sect. 5.3, say that red vocal nodes n_1, n_2, with $\hat{\omega}(n_1) \neq \hat{\omega}(n_2)$, are *partners* if their silent paths start either at the root node or at the same vocal node, say $n = \text{node}(s)$; share an initial segment labelled $s'\sigma$ with $\sigma \in \Sigma_c$; and this shared segment is followed in turn by segments labelled by strings $s''s_1$, $s''s_2$ respectively, where $s'' \in \Sigma_u^*$ and at least one of the strings s_1, s_2 belongs to Σ_u^* (see Fig. 5.5.1, where we assume $s_2 \in \Sigma_u^*$). We call node$(ss'\sigma)$ the *antecedent* of the partners n_1, n_2. In this structure the controllable events labelled $\tau_{1c} = \hat{\omega}(n_1)$, $\tau_{2c} = \hat{\omega}(n_2)$ in $\mathbf{G_{hi}}$ cannot be disabled independently by a command to $\mathbf{C_{lo}}$. Thus if E_{hi} requires disabling of τ_{2c} (at some state of its transition structure) then it may be true that $\mathbf{C_{lo}}$ is forced to disable τ_{1c} as well, via direct disablement of σ. So a cure in principle is to break up the occurrence of partners: declare that the hitherto silent antecedent node$(ss'\sigma)$ is now a red vocal node with controllable output any new symbol τ_c'', extend T_c by τ_c'' accordingly, and re-evaluate color(n_i), $\hat{\omega}(n_i)$, $(i = 1, 2)$ as appropriate (in Fig. 5.5.1, n_2 would be recolored green and $\hat{\omega}(n_2)$ redefined to be τ_{2u}).

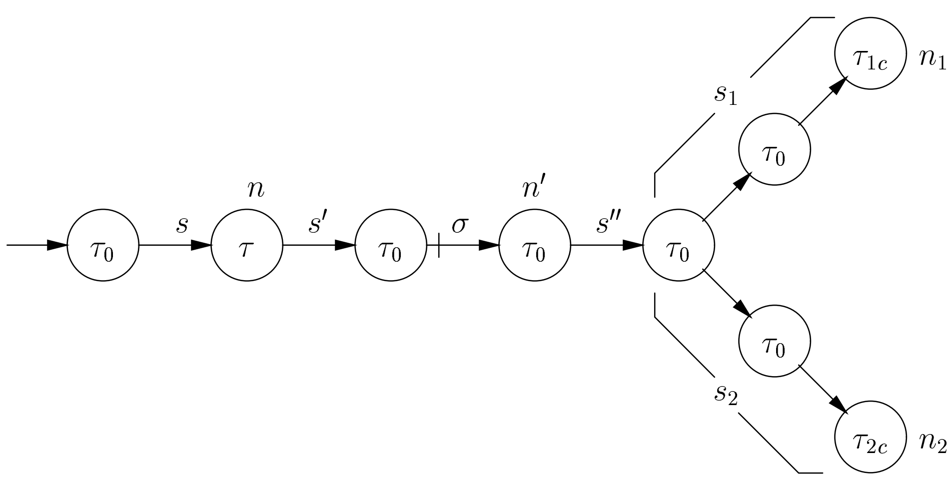
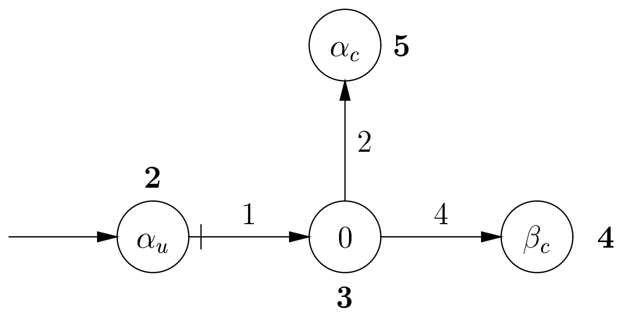


Fig. 5.5.1
Partners n_1, n_2 with antecedent n'.

A formal version of this procedure is provided in Sect. 5.8. The result is embodied in the following definition. Let $\mathbf{G}$ be a Moore transition structure as in (3.1). Then $\mathbf{G}$ is *strictly* *output-control-consistent* (SOCC) if it is output-control-consistent and if in the reachability tree of $L(\mathbf{G})$ no two red vocal nodes are partners. As in Sect. 5.3, this definition could be formally rephrased in terms of the strings of $L(\mathbf{G})$ if desired. In Sect. 5.8 it is shown that the SOCC property can be obtained by no more than a 4-fold increase in state size over that of the original DES $\mathbf{G_{lo}}$ that we started with; in practice, a factor of about 1.5 seems to be much more typical.

Consider again our running example. By inspection of $\mathbf{G_{lo}}$ (i.e. $\mathbf{G_{lo,ext}}$ in Fig. 5.3.2) we note the partner configuration



Its cure is the vocalization of the antecedent 3 of states 4 and 5, say with a new controllable event γ_c. The final results are displayed in Fig. 5.5.2. Notice that in $\mathbf{G_{lo,new}}$ the status of β has changed from β_c to β_u and that $\mathbf{G_{hi,new}}$ is larger than $\mathbf{G_{hi,ext}}$ by one state and transition.

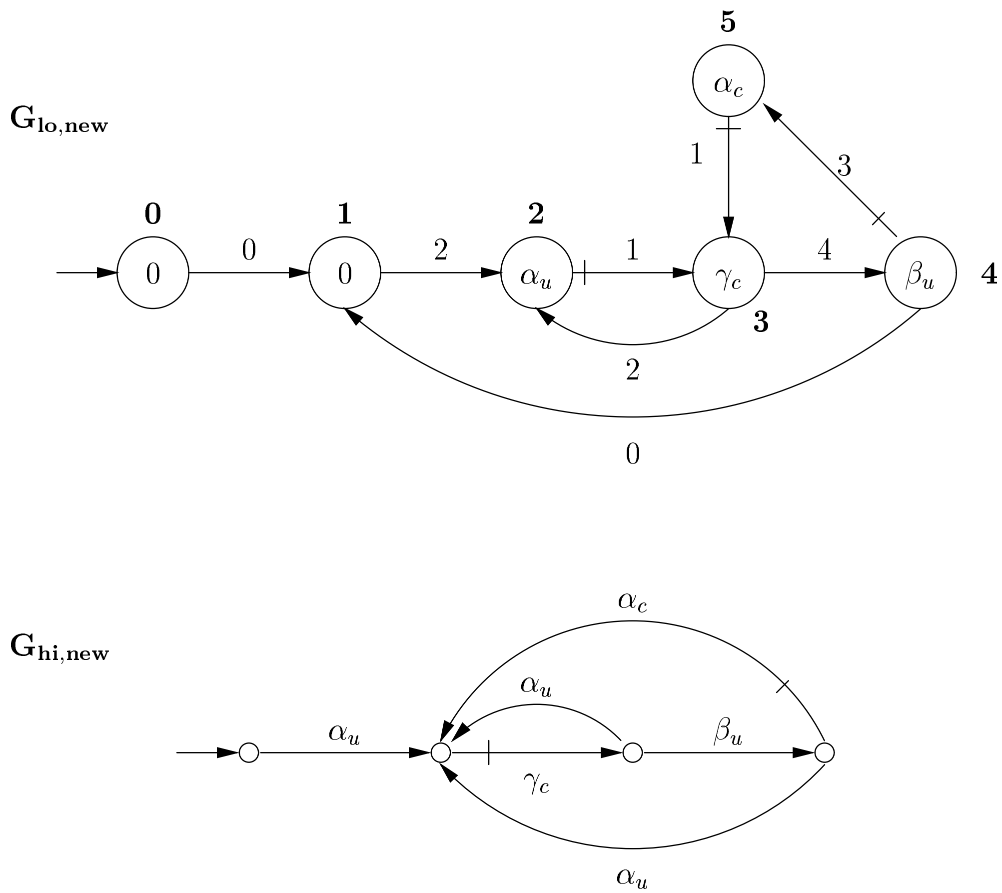


Fig. 5.5.2
Strict output control consistency

Returning to our hierarchical control structure we finally have the desired result.

Theorem 5.5.1

Assume that $\mathbf{G_{lo}}$ is SOCC, and let $E_{hi} \subseteq L(\mathbf{G_{hi}})$ be nonempty, closed and controllable. Then

$$\theta((\theta^{-1}(E_{hi}))^{\uparrow}) = E_{hi}$$

Proof

In the proof write $L(\mathbf{G_{lo}}) =: L_{lo}$, $L(\mathbf{G_{hi}}) =: L_{hi}$, $L(\gamma_{lo}, \mathbf{G_{lo}}) =: K_{lo}$. With $E_{lo} := \theta^{-1}(E_{hi})$, Theorem 5.4.1 can be applied to yield

$$K_{lo} = E_{lo}^{\uparrow} = (\theta^{-1}(E_{hi}))^{\uparrow}$$

which implies

$$\theta(K_{lo}) = \theta((\theta^{-1}(E_{hi}))^{\uparrow}) \subseteq \theta(E_{lo}) = E_{hi}$$

Next observe that $K_{lo} \neq \emptyset$. Otherwise, there is a string $s_o \in L_{lo} \cap \Sigma_u^*$ with $s_o \notin E_{lo} = \theta^{-1}(E_{hi})$, namely $\theta(s_o) \notin E_{hi}$. Thus

$$\theta(s_o) \in L_{hi} \cap (T_u^* - E_{hi})$$

which implies that E_{hi} is either empty or uncontrollable, contrary to hypothesis.

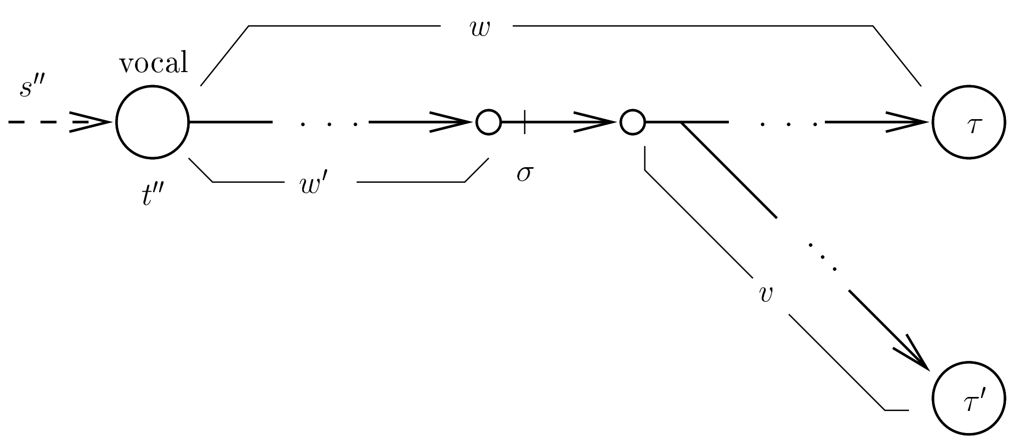
Now suppose that the inclusion $\theta(K_{lo}) \subseteq E_{hi}$ is strict, and let $t \in E_{hi} - \theta(K_{lo})$. Since $\epsilon \in \theta(K_{lo})$ there exists a maximal prefix $t' < t$ with $t' \in \theta(K_{lo})$. Let $s \in E_{lo}$ with $\theta(s) = t$. Since $\epsilon \in K_{lo}$ we can select a prefix $s'' \leq s$ of maximal (possibly zero) length such that $s'' \in K_{lo}$ and node(s'') is vocal (or is the root node). Then $t'' := \theta(s'')$ satisfies $t'' \leq t'$, where the prefix ordering may be strict. Let $w \in \Sigma^*$ with $s''w \leq s$, node($s''w$) vocal, and $\theta(s''w) = \theta(s'')\tau$ for some $\tau \in T$; that is, the path from node(s'') to node($s''w$) is silent. Now $w \in \Sigma^*\Sigma_c\Sigma_u^*$, as otherwise $w \in \Sigma_u^*$, which implies by the controllability of K_{lo} that $s''w \in K_{lo}$, contrary to the maximality of s''. Choose $w' \leq w$ to be of maximal length such that $s''w' \in K_{lo}$, namely $s''w'\sigma \notin K_{lo}$, with $w'\sigma \leq w$ and $\sigma \in \Sigma_c$ (so that σ is disabled by γ_{lo}).

We claim that there must exist a string $v \in \Sigma_u^*$ such that (1) node($s''w'\sigma v$) is vocal, and (2) the path from node(s'') to node($s''w'\sigma v$) is silent, with (3) $\theta(s''w'\sigma v) \notin E_{hi}$. Otherwise all strings v with properties (1)-(3) would belong to $\Sigma^*\Sigma_c\Sigma_u^*$, and K_{lo} (which excludes $s''w'\sigma$) would not be supremal.

Since

$$t''\tau = \theta(s'')\tau = \theta(s''w) \leq \theta(s) = t \in E_{hi}$$

and $\theta(s''w'\sigma v) \notin E_{hi}$, it results finally that $\theta(s''w'\sigma v) = t''\tau'$, say, with $\tau' \neq \tau$, and therefore node($s''w$) and node($s''w'\sigma v$) are partners, in contradiction to the main hypothesis of the theorem. □



Thus when $\mathbf{G_{lo}}$ is SOCC, hierarchical consistency is achieved for the pair $(\mathbf{G_{lo}}, \mathbf{G_{hi}})$.

In *TCT*, hierarchical consistency can be achieved by computing either

$$\mathbf{HCGLO} = \mathbf{hiconsis(OCGLO)},$$

or directly as

$$\mathbf{HCGLO} = \mathbf{hiconsis(GLO)},$$

bypassing the intermediate state of output control consistency. The resulting high-level DES is

$$\mathbf{HCGLOHI} = \mathbf{higen(HCGLO)}$$

More information on **hiconsis** is provided in Appendix 5.9.

Exercise 5.5.1: Use *TCT* to verify the running example of Sects. 5.2-5.5. $\diamond$

Finally it should be noted that our restriction to a hierarchy of two levels was inessential. Once hierarchical consistency has been achieved for the bottom level and first level up, say $(\mathbf{G_o}, \mathbf{G_1})$, the constructions may be repeated on assigning state outputs in $\mathbf{G_1}$ and bringing in a next higher level, $\mathbf{G_2}$. Clearly hierarchical consistency for $(\mathbf{G_1}, \mathbf{G_2})$ can be achieved without disturbing the consistency of $(\mathbf{G_o}, \mathbf{G_1})$. The theory thus possesses the highly desirable attribute of 'vertical modularity'.

To conclude this section we give two results (due to K.C. Wong) which place the property of hierarchical consistency in clear perspective. Let $\mathbf{G_{lo}}$ be OCC. Recall the notation $\mathcal{C}(E)$ for the family of controllable sublanguages of E; thus $\mathcal{C}(L_{lo})$ (resp. $\mathcal{C}(L_{hi})$) is the family of all controllable sublanguages of L_{lo} (resp. L_{hi}).

Let us bring in the

$$\textbf{Main Condition:}\quad \theta\mathcal{C}(L_{lo}) = \mathcal{C}(L_{hi})$$

Main Condition (MC) says, not only that θ preserves controllability, but also that every high-level controllable language is the θ-image of some (usually more than one) low-level controllable language. In other words, equating executable "tasks" with controllable languages, every task that could be specified in the manager's (aggregated) model $\mathbf{G_{hi}}$ is executable in the operator's (detailed) model $\mathbf{G_{lo}}$; high-level policies can always be carried out operationally. (Of course a justification of this interpretation would require that an on-line hierarchical control mechanism be spelled out; but this was done in Sect. 5.4). Now let $E_{hi} \subseteq L_{hi}$ be a high-level legal specification, not necessarily controllable. Suppose that E_{hi} is "proposed" to the operator by specification of its preimage $\theta^{-1}(E_{hi})$. The operator may then synthesize $(\theta^{-1}(E_{hi}))^{\uparrow} \subseteq L_{lo}$, with the result that $\theta((\theta^{-1}(E_{hi}))^{\uparrow})$ is implemented in $\mathbf{G_{hi}}$. One would like this implemented sublanguage of L_{hi} to be precisely the language $E_{hi}^{\uparrow}$ that a manager working at the level of $\mathbf{G_{hi}}$ would synthesize directly (if direct control were feasible): this is the essence of hierarchical consistency. The result to follow states that hierarchical consistency in this strong sense is equivalent to MC.

Theorem 5.5.2

$$\mathrm{MC} \Leftrightarrow [(\forall E_{hi})E_{hi} \subseteq L_{hi} \Rightarrow \theta((\theta^{-1}(E_{hi}))^{\uparrow}) = E_{hi}^{\uparrow}]$$

□

The usefulness of this result resides in the fact that the "complicated" condition of hierarchical consistency (involving the $(\cdot)^{\uparrow}$ operation) is replaced by the formally simpler MC, which involves only the controllability property.

Along the same lines, on weakening MC slightly the following related result can be proved, as a simpler version of the condition of hierarchical consistency defined earlier in this section.

Theorem 5.5.3

$$\theta\mathcal{C}(L_{lo}) \supseteq \mathcal{C}(L_{hi}) \Leftrightarrow [(\forall E_{hi})E_{hi} \in \mathcal{C}(L_{hi}) \Rightarrow \theta((\theta^{-1}(E_{hi}))^{\uparrow}) = E_{hi}]$$

□

It is of interest to note that these results depend on nothing more than the fact that the operations $\theta(\cdot)$ and $(\cdot)^{\uparrow} = \sup\mathcal{C}(\cdot)$ are monotone on sublanguages.

Exercise 5.5.2: Prove Theorems 5.5.2 and 5.5.3.

5.6 Hierarchical Supervision of Transfer Line

The theory will be illustrated by developing a high-level hierarchical supervisor for Transfer Line (cf. Sect. 4.6). We recall that Transfer Line consists of two machines **M1**, **M2** plus a test unit **TU**, linked by buffers **B1**, **B2** in the sequence: **M1,B1,M2,B2,TU** (Fig. 5.6.1). State transition diagrams of **M1**, **M2** and **TU** are displayed in Fig. 5.6.2.

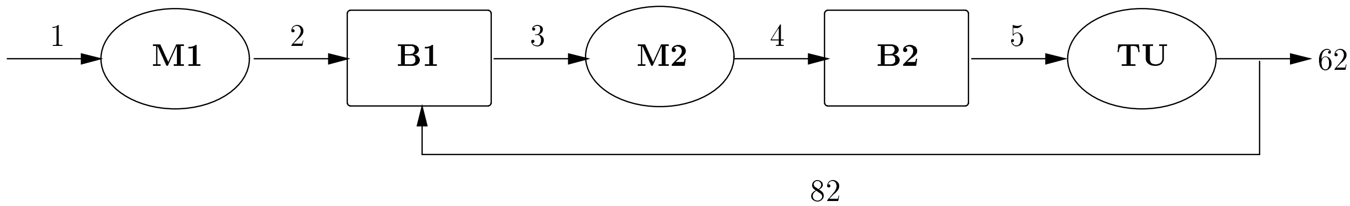


Fig. 5.6.1

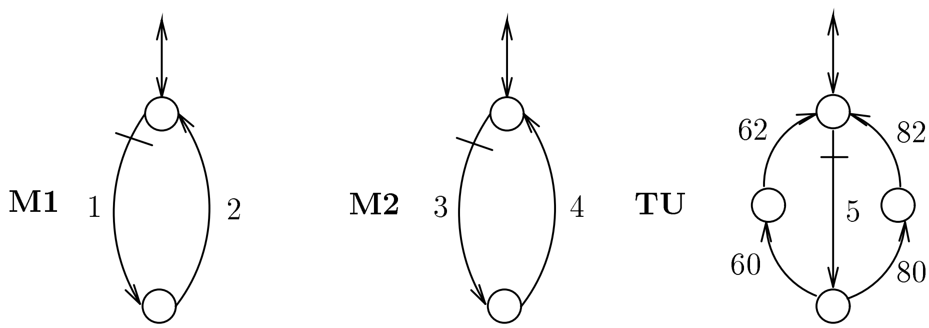


Fig. 5.6.2

TU either "passes" or "fails" each processed workpiece, signaling its decision with events 60, 80 respectively. In case of "pass test", the workpiece is sent to the system output (event 62); in case of "fail test", it is returned to **B1** (event 82) for reprocessing by **M2**. There is no limit on the number of failure/reprocess cycles a given workpiece may undergo.

For ease of display we consider only the simplest case, where **B1** and **B2** each has capacity 1. Initially an optimal low-level supervisor is designed by any of the methods of Chapt. 3 or 4, to ensure that neither of the buffers is subject to overflow or underflow. In detail, let

$$\mathbf{PL} = \mathbf{shuffle}(\mathbf{M1}, \mathbf{M2}, \mathbf{TU});$$

and let **B1SP**, **B2SP** be the buffer specification generators (Fig. 5.6.3).

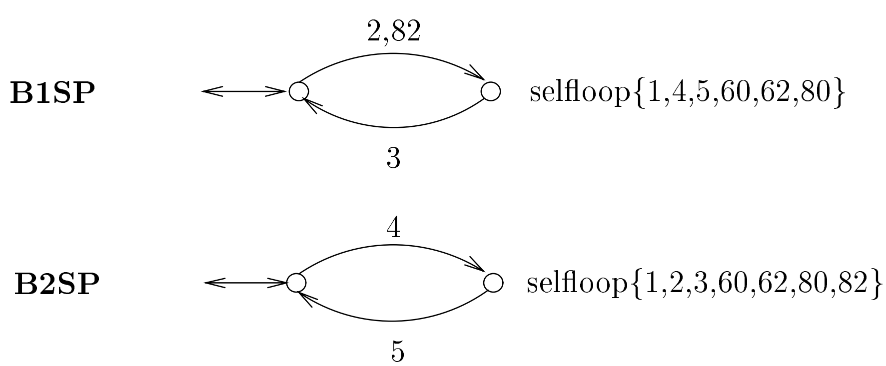


Fig. 5.6.3

Then we set **BSP** = **meet**(**B1SP**,**B2SP**), and

$$\mathbf{PLSUP} = \mathbf{supcon(PL,BSP)}$$

as displayed in Fig. 5.6.4. With **PLSUP** as the starting point for the development of hierarchical structure, we must first assign the "significant" events to be signaled to the "manager". Let us assume that the manager is interested only in the events corresponding to "taking a fresh workpiece" (low-level event 1, signaled as high-level event τ_1, say), and to "pass test" (low-level event 60, signaled as τ_2) or "fail test" (low-level event 80, signaled as τ_3). If too many failures occur the manager intends to take remedial action, which will start by disabling the failure/reprocess cycle. To this end the uncontrollable event 80 is now replaced in the low-level structure by a new controllable event 81. Furthermore, the meaning of the signaled events τ_1, τ_2, τ_3 must be unambiguous, so a transition entering state 1 like [8,62,1] must not be confused with the "significant" transition [0,1,1]; namely a new state (say, 12) must be introduced, transition [8,62,1] replaced by [8,62,12], and a new transition [12,2,2] inserted. The final Moore structure, **GLO**, is displayed in Fig. 5.6.5. Here the vocal [state,output] pairs are [1,τ_1], [8,τ_1], [7,τ_2] and [6,τ_3]. In *TCT*, the foregoing adjustments can be made using **edit** and **vocalize**.

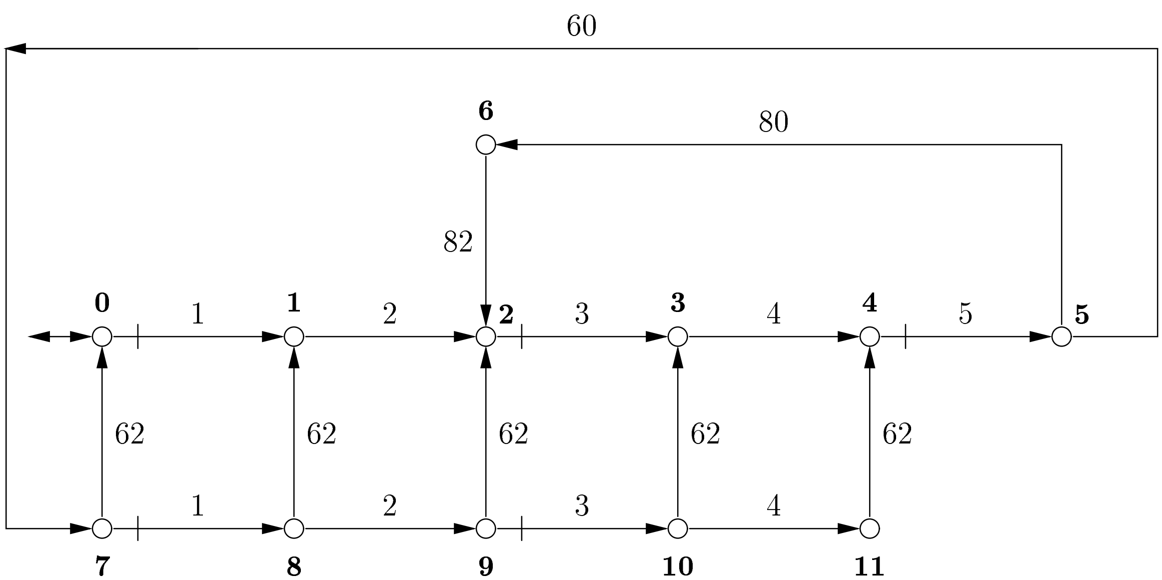


Fig. 5.6.4
PLSUP

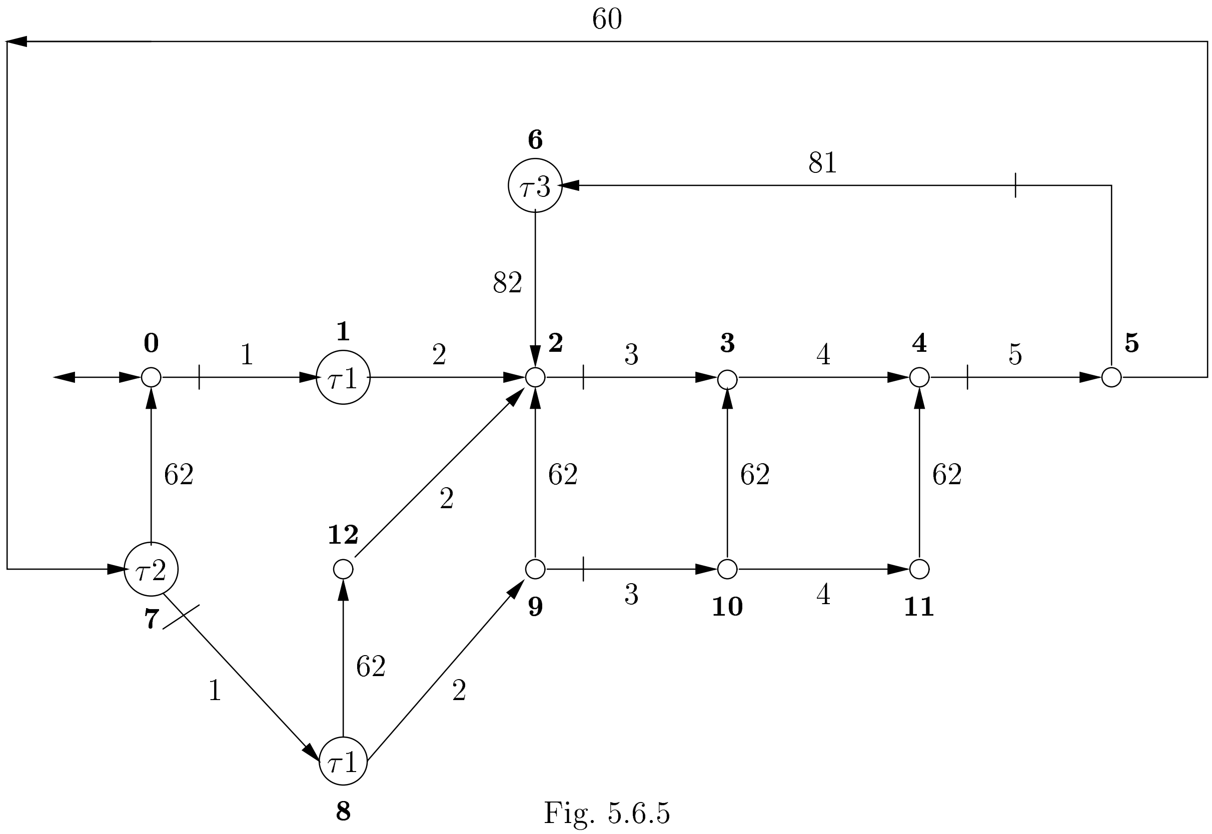


Fig. 5.6.5
GLO

We are now ready to carry out the procedures of the theory. By inspection of Fig. 5.6.5, it is clear that each of τ_1, τ_2, τ_3 is unambiguously controllable, that is, **GLO** is already output-control-consistent. The corresponding high-level model **GHI** is displayed in Fig. 5.6.6. In *TCT*, **GHI** = **higen(GLO)**.

However, for the manager to disable τ_2 will require the operator to disable low-level event 5, which in turn disables the high-level event τ_3 as an undesired side effect; thus **GLO** is not strictly-output-control-consistent (SOCC). To improve matters it is enough to vocalize the low-level state 5 with a new high-level output τ_4, signaling the new "significant" event that "TU takes a workpiece". This step incidentally converts the status of τ_2 from controllable to uncontrollable. With this the construction of a SOCC model, say **CGLO**, from **GLO** is complete (Fig. 5.6.7). The corresponding high-level model **CGHI** is displayed in Fig. 5.6.8, where $\tau_1, \tau_2, \tau_3, \tau_4$ have been coded respectively as 11,20,31,41. In *TCT*, **CGLO** = **hiconsis(GLO)** and **CGHI** = **higen (CGLO)**.

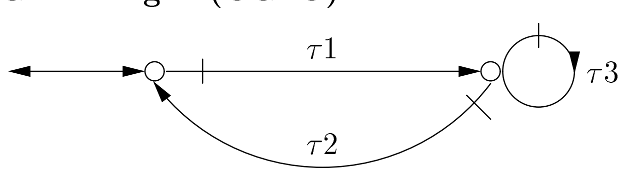


Fig. 5.6.6
GHI

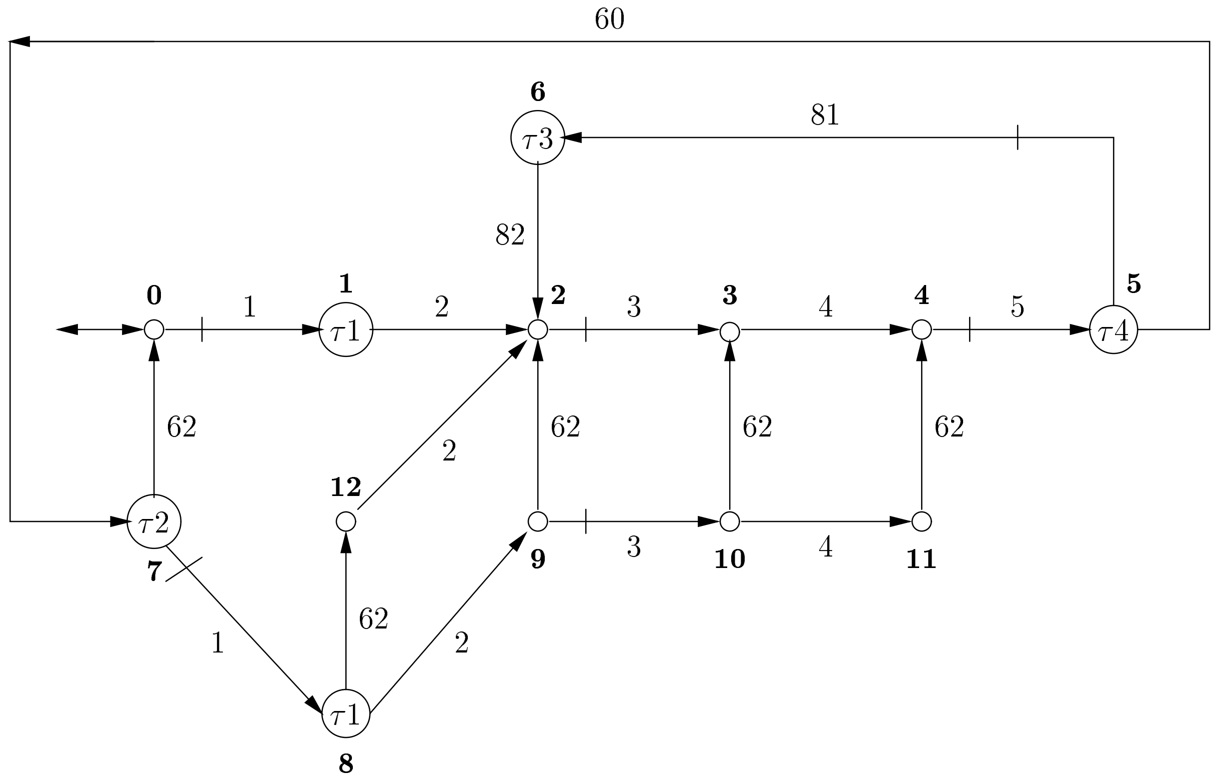


Fig. 5.6.7
CGLO

The simple model **CGHI** can be supervised by the manager to achieve his objective of "quality control". A possible high-level specification might be: "If two consecutive test failures (31) occur, allow TU to operate just once more, then shut down the system"; this is modeled by **HISP** as displayed (Fig. 5.6.9). The resulting supervisor

$$\mathbf{CGHISUP} = \mathbf{supcon}(\mathbf{CGHI},\mathbf{SPECHI})$$

is shown in Fig. 5.6.10. On termination of **CGHISUP** at state 7, and execution by **TU** of event 62, it can be easily verified that **CGLO** will have halted at its marker state 0.

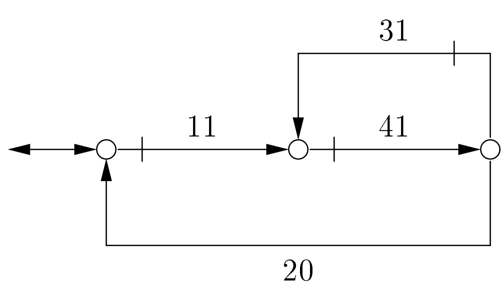
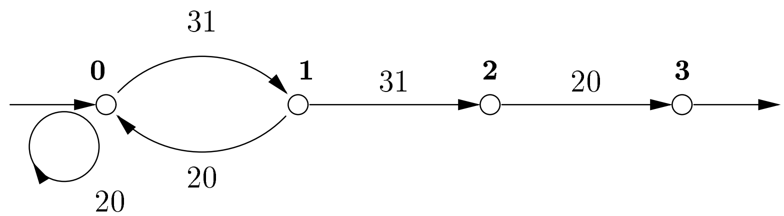


Fig. 5.6.8
CGHI



selfloop $\{11, 41\}$
Fig. 5.6.9
SPECHI

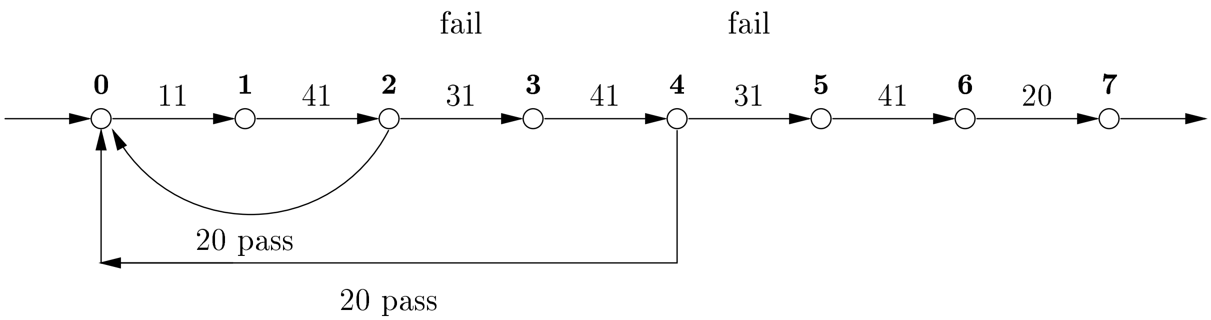


Fig. 5.6.10
CGHISUP

Exercise 5.6.1: To appraise the utility of hierarchical control for Transfer Line, replace **SPECHI** by an equivalent DES **SPECLO** for **GLO** (unvocalized), compute the corresponding low-level supervisor, and compare its state size with that of **CGHISUP**.

Exercise 5.6.2: For a manageable but nontrivial example with a plausible physical interpretation, carry out the design of a SOCC hierarchical control structure, illustrating the successive refinements involved in first achieving (non-strict) output control consistency and then strict consistency. For a particular high-level controllable specification language, define precisely the required command and control γ_{lo}.

5.7 Hierarchical Supervision with Nonblocking

In this section we extend our theory of hierarchical supervision to include marking and nonblocking. Unsurprisingly, nonblocking is not ensured by hierarchical consistency over closed sublanguages, in the sense of Sect. 5.5. For instance, in the example of Fig. 5.7.1, with α, β, γ uncontrollable, evidently

$$\mathcal{C}_{lo}(L_{lo}) = \{\emptyset, L_{lo}\}, \quad \mathcal{C}_{hi}(L_{hi}) = \{\emptyset, L_{hi}\}$$

and $\mathbf{G_{lo}}$ is hierarchically consistent; however, $\mathbf{G}_{lo}$ blocks on executing α.

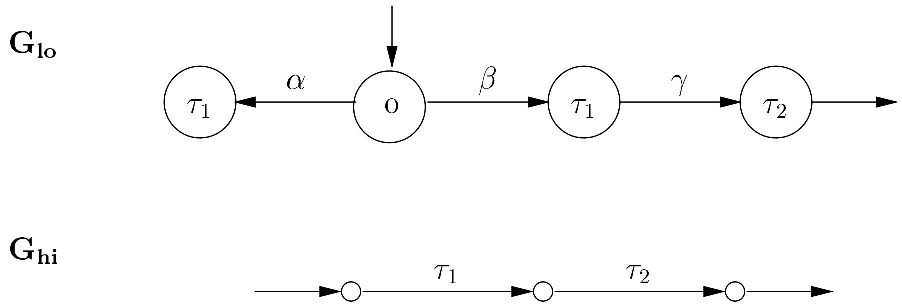


Fig. 5.7.1

Throughout this section it will be convenient to write L for $L_{lo} := L(\mathbf{G_{lo}})$, $L_m \subseteq L$ for the marked behavior of $\mathbf{G_{lo}}$, $M := \theta(L)$ for $L_{hi} := L(\mathbf{G_{hi}})$ and $M_m := \theta(L_m)$ for the marked behavior of $\mathbf{G_{hi}}$.

We begin by generalizing Theorem 5.5.2. Our standing assumption is that $\mathbf{G_{lo}}$ is output-control-consistent (Sect. 5.3). The scenario will be that, with $E_{hi} \subseteq M_m$ a given specification language for $\mathbf{G_{hi}}$, the marked behavior 'virtually' synthesized in $\mathbf{G_{hi}}$ is, as usual (cf. Sect. 3.5)

$$K_{hi} := \sup \mathcal{C}_{hi}(E_{hi})$$

The specification 'announced' to the low-level controller is $\theta^{-1}(E_{hi})$, so the marked behavior synthesized in $\mathbf{G_{lo}}$ is, again as usual,

$$K_{lo} := \sup \mathcal{C}_{lo}(L_m \cap \theta^{-1}(E_{hi}))$$

The desired consistency property is then

$$\theta(K_{lo}) = K_{hi}$$

Theorem 5.7.1.

In the foregoing notation

$$(\forall E_{hi}) E_{hi} \subseteq M_m \Rightarrow \theta(K_{lo}) = K_{hi} \qquad \text{(HCm)}$$

iff

$$\theta \mathcal{C}_{lo}(L_m) = \mathcal{C}_{hi}(M_m) \qquad \text{(MCm)}$$

Proof

(If) Let $E_{hi} \subseteq M_m$ and assume (MCm). Then

$$K_{hi} := \sup \mathcal{C}_{hi}(E_{hi}) \in \mathcal{C}_{hi}(M_m)$$

so by (MCm), $K_{hi} = \theta(K'_{lo})$ for some $K'_{lo} \in \mathcal{C}_{lo}(L_m)$. Therefore

$$\begin{aligned} K'_{lo} &\subseteq \sup \mathcal{C}_{lo}(L_m \cap \theta^{-1}(K_{hi})) \\ &\subseteq \sup \mathcal{C}_{lo}(L_m \cap \theta^{-1}(E_{hi})) = K_{lo} \end{aligned}$$

Thus

$$K_{hi} = \theta(K'_{lo}) \subseteq \theta(K_{lo})$$

But $K_{lo} \subseteq L_m \cap \theta^{-1}(E_{hi})$ implies

$$\theta(K_{lo}) \subseteq M_m \cap E_{hi} = E_{hi}$$

So by (MCm) $\theta(K_{lo}) \subseteq \sup \mathcal{C}_{hi}(E_{hi}) = K_{hi}$. Thus $\theta(K_{lo}) = K_{hi}$ as claimed.

(Only if) Let $K_{hi} \in \mathcal{C}_{hi}(M_m)$ and set $E_{hi} = K_{hi}$ in (HCm). Then

$$K_{hi} = \theta(K_{lo}) \in \theta(\mathcal{C}_{lo}(L_m))$$

namely $\mathcal{C}_{hi}(M_m) \subseteq \theta\mathcal{C}_{lo}(L_m)$. For the reverse inclusion let $K'_{lo} \in \mathcal{C}_{lo}(L_m)$ and set $E'_{hi} := \theta(K'_{lo})$. Then $E'_{hi} \subseteq \theta(L_m) = M_m$, so $K'_{hi} := \sup \mathcal{C}_{hi}(E'_{hi}) \in \mathcal{C}_{hi}(M_m)$. Now $K'_{lo} \subseteq \theta^{-1}(E'_{hi})$ implies

$$K'_{lo} \subseteq \sup \mathcal{C}_{lo}[L_m \cap \theta^{-1}(E'_{hi})]$$

Also (HCm) provides

$$\theta[\sup \mathcal{C}_{lo}(L_m \cap \theta^{-1}(E'_{hi})] = \sup \mathcal{C}_{hi}(E'_{hi})$$

Therefore

$$\theta(K'_{lo}) \subseteq \sup \mathcal{C}_{hi}(E'_{hi}) \subseteq E'_{hi} = \theta(K'_{lo})$$

giving

$$\theta(K'_{lo}) = \sup \mathcal{C}_{hi}(E'_{hi}) = K'_{hi} \in \mathcal{C}_{hi}(M_m)$$

namely $\theta\mathcal{C}_{lo}(L_m) \subseteq \mathcal{C}_{hi}(M_m)$ as required. □

While Theorem 5.7.1 provides an interesting perspective on hierarchical supervision with nonblocking, the "Main Condition with marking" (MCm) is not immediately effective. To satisfy (MCm) we shall endow our causal reporter map θ with a certain 'global observer property'; we shall also require a type of 'local controllability' in $\mathbf{G}_{\mathbf{lo}}$. These properties seem natural from the viewpoint of a designer with some capability of structuring $\mathbf{G}_{\mathbf{lo}}$ in advance.

In the following we use the terminology for reachability tree as in Sect. 5.3. Let

$$L_{\text{voc}} := \{s \in L | s = \epsilon \text{ or node}(s) \text{ is vocal}\}$$

Thus L_{voc} is the subset of strings of L that correspond to the root node or a vocal node of the reachability tree of L. Clearly $\theta(L_{\text{voc}}) = \theta(L) = M$. To avoid fussy details we shall assume, reasonably, that vocalization is 'complete', namely any string of L can be extended (in L) to a string of L_{voc}, i.e. $\bar{L}_{\text{voc}} = L$. Now we say that $\theta : L \to M$ is an L_{voc}–*observer* if

$$(\forall s \in L_{\text{voc}})(\forall t \in M)\theta(s) \leq t \Rightarrow (\exists s' \in \Sigma^*)ss' \in L_{\text{voc}} \ \& \ \theta(ss') = t$$

In other words, whenever $\theta(s)$ can be extended to a string $t \in M$, the underlying L_{voc}-string s can be extended to an L_{voc}-string ss' with the same image under θ: "the manager's expectation can always be executed in $\mathbf{G}_{\mathbf{lo}}$", at least when starting from a string in L_{voc}. In the Example of Fig. 5.7.1, θ is not an L_{voc}–observer, while in that of Figs. 5.6.5, 5.6.6 it is.

Write θ_{voc} for the restriction $\theta | L_{\text{voc}}$. Thus

$$\theta_{\text{voc}} : L_{\text{voc}} \to M, \quad \theta_{\text{voc}}^{-1} : Pwr(M) \to Pwr(L_{\text{voc}})$$

The L_{voc}–observer property can be characterized as commutativity of θ_{voc}^{-1} with prefix-closure. Recall that the operation of prefix-closure maps L_{voc} onto L.

Proposition 5.7.1

θ is an L_{voc}–observer iff, for all $E \subseteq M$,

$$\theta_{\text{voc}}^{-1}(\bar{E}) = \overline{\theta_{\text{voc}}^{-1}(E)} \cap L_{\text{voc}}$$

Proof

(If) Let $s \in L_{\text{voc}}, t \in M, \theta(s) \leq t$. Then $\theta(s) \in \overline{\{t\}} \subseteq M$, so

$$s \in \theta_{\text{voc}}^{-1}(\overline{\{t\}}) = \overline{\theta_{\text{voc}}^{-1}(\{t\})} \cap L_{\text{voc}}$$

Thus for some $s' \in \Sigma^*, ss' \in \theta_{\text{voc}}^{-1}(\{t\})$, namely $\theta_{\text{voc}}(ss') = t$, as required.

(Only if) The direction $\overline{\theta_{\text{voc}}^{-1}(E)} \cap L_{\text{voc}} \subseteq \theta_{\text{voc}}^{-1}(\bar{E})$ is automatic, for if $ss' \in \theta_{\text{voc}}^{-1}(E)$ for $s \in L_{\text{voc}}$ and some s' with $ss' \in L_{\text{voc}}$, then $\theta_{\text{voc}}(s) \leq \theta_{\text{voc}}(ss') \in E$, so $\theta_{\text{voc}}(s) \in \bar{E}$ and $s \in \theta_{\text{voc}}^{-1}(\bar{E})$. For the reverse inclusion, taking $s \in \theta_{\text{voc}}^{-1}(\bar{E})$ we have $t := \theta_{\text{voc}}(s) \in \bar{E}$, so for some $t', tt' \in E$ and $\theta_{\text{voc}}(s) \leq tt'$. Since θ is an L_{voc}–observer, there is some s' with $ss' \in L_{\text{voc}}$ and $\theta_{\text{voc}}(ss') = tt'$, so $s \leq ss' \in \theta_{\text{voc}}^{-1}(E)$, namely $s \in \overline{\theta_{\text{voc}}^{-1}(E)} \cap L_{\text{voc}}$. □

We now bring in a 'local' description of controllability in $\mathbf{G_{lo}}$. Generalizing the control action of Sect. 5.4, one should now think of control decisions in $\mathbf{G_{lo}}$ (made by the 'operator' of Sect. 5.1) being delegated to 'agents', say to Agent(s) for each $s \in L_{\text{voc}}$. The scope of Agent(s) will be the 'local' language $L_{\text{voc}}(s)$ linking node(s) to the adjacent downstream vocal nodes (if any) in the reachability tree of L. Formally, for $s \in L_{\text{voc}}$ let

$$L_{\text{voc}}(s) := \{s' \in \Sigma^* | ss' \in L_{\text{voc}} \ \& \ (\exists \tau \in T)\theta(ss') = \theta(s)\tau\}$$

Along with $L_{\text{voc}}(s)$, define the local reporter map $\theta_s : L_{\text{voc}}(s) \to T$ given by

$$\theta_s(s') := \tau \text{ iff } \theta(ss') = \theta(s)\tau, \ s' \in L_{\text{voc}}(s)$$

For the image of θ_s write

$$T_s := \{\theta_s(s') | s' \in L_{\text{voc}}(s)\} \subseteq T, \ s \in L_{\text{voc}}$$

As seen by Agent(s), the locally controllable sublanguages are

$$\mathcal{C}_{lo}(s) := \{K \subseteq L_{\text{voc}}(s) | \bar{K}\Sigma_u \cap \overline{L_{\text{voc}}(s)} \subseteq \bar{K}\}$$

Apart from $\emptyset$, these are exactly the sublanguages of $L_{\text{voc}}(s)$ that Agent(s) is authorized (and able) to synthesize by supervisory control. The local controllability property of interest is that Agent(s) can synthesize (i.e. select from $\mathcal{C}_{lo}(s)$) a controllable sublanguage for each controllable state output subset which he can see downstream, namely each subset of form

$$T' = (T_s \cap T_u) \cup T'', \quad T'' \subseteq T_s \cap T_c$$

Adjoining $\emptyset$ (if necessary) and denoting the resulting family of T' by $\mathcal{T}_s \subseteq Pwr(T)$, we can write the desired property as

$$\mathcal{T}_s = \theta_s \mathcal{C}_{lo}(s) \tag{1}$$

When (1) holds we shall say that $\mathbf{G_{lo}}$ is *locally output controllable at* s; and if (1) holds at all $s \in L_{\text{voc}}$, that $\mathbf{G_{lo}}$ is *locally output controllable.*

For instance in Fig. 5.3.2, if $s = 02 \in L_{\text{voc}}$, we have

$$L_{\text{voc}}(s) = \{12, 14\}, \ T_s = \{\alpha_c, \beta_c\}$$

$$\mathcal{C}_{lo}(s) = \{\emptyset, \{12, 14\}\}, \ \mathcal{T}_s = \{\emptyset, \{\alpha_c\}, \{\beta_c\}, \{\alpha_c, \beta_c\}\}$$

$$\theta_s \mathcal{C}_{lo}(s) = \{\emptyset, \{\alpha_c, \beta_c\}\} \subsetneq \mathcal{T}_s$$

Thus $\mathbf{G_{lo,ext}}$ is not locally output controllable at s. On the other hand, it is so at $s = 0214$, where

$$\begin{aligned} L_{\text{voc}}(s) &= \{02, 3\}, \ T_s = \{\alpha_u, \alpha_c\} \\ \mathcal{C}_{lo}(s) &= \{\emptyset, \{02\}, \{02, 3\}\} \\ \mathcal{T}_s &= \{\emptyset, \{\alpha_u\}, \{\alpha_u, \alpha_c\}\} \\ &= \theta_s \mathcal{C}_{lo}(s) \end{aligned}$$

Turning now to the manager, let $\mathrm{Elig}_M : M \to Pwr(T)$, given by

$$\mathrm{Elig}_M(t) := \{\tau \in T | t\tau \in M\},\ t \in M$$

As seen by the manager, the local one-step controllable sublanguages at $t \in M$ are, of course, just the subsets of T in the family

$$\mathcal{C}'_{hi}(t) := \{(\mathrm{Elig}_M(t) \cap T_u) \cup T' | T' \subseteq \mathrm{Elig}_M(t) \cap T_c\}$$

and we write $\mathcal{C}_{hi}(t) := \mathcal{C}'_{hi}(t) \cup \{\emptyset\}$. Since $M = \theta(L)$ by definition, it is clear that, for each $s \in L_{\mathrm{voc}}$ with $\theta_{\mathrm{voc}}(s) = t$, we have

$$\mathcal{C}_{hi}(t) \supseteq \mathcal{T}_s \tag{2}$$

In general the inclusion in (2) may be proper at some s, t with $\theta(s) = t$. It is at just such t that the manager enjoys less effective hierarchical control than he might expect from his high-level model $\mathbf{G_{hi}}$ of M.

In Fig. 5.7.1 $\mathbf{G_{lo}}$ is locally output controllable. However, with $s = \alpha, t = \tau_1$, we have

$$\mathcal{T}_s = \{\emptyset\} \subsetneqq \{\emptyset, \{\tau_2\}\} = \mathcal{C}_{hi}(t)$$

The crucial property we require for hierarchical consistency is simply that a control decision can be taken by Agent(s) at node(s), to match any controllable selection by the manager of events $\tau \in T$ at $\theta(s) = t \in M$. In other words, for all $s \in L_{\mathrm{voc}}$ and $t = \theta_{\mathrm{voc}}(s)$,

$$\theta_s \mathcal{C}_{lo}(s) = \mathcal{C}_{hi}(t)$$

If this 'global' condition holds, we say that $\mathbf{G_{lo}}$ is *globally output controllable.*

Lemma 5.7.1

Assume that

(i) $\theta : L \to M$ is an L_{voc}–observer, and
(ii) $\mathbf{G_{lo}}$ is locally output controllable.

Then $\mathbf{G_{lo}}$ is globally output controllable.

Proof

Let $s \in L_{\text{voc}}$ and $t = \theta_{\text{voc}}(s)$. By (i), $t\tau \in M$ implies the existence of $s' \in L_{\text{voc}}(s)$ such that $ss' \in L_{\text{voc}}$ and $\theta_{\text{voc}}(ss') = t\tau$, so $\tau = \theta_s(s') \in \mathcal{T}_s$, namely $\text{Elig}_M(t) \subseteq T_s$. By the definition $M = \theta(L)$,

$$\text{Elig}_M(t) = \cup\{T_v | v \in L_{\text{voc}}, \theta_{\text{voc}}(v) = t\}$$

Therefore $T_s = \text{Elig}_M(t)$, and this implies $\mathcal{T}_s = \mathcal{C}_{hi}(t)$. By (ii), $\mathcal{T}_s = \theta_s\mathcal{C}_{lo}(s)$, and finally

$$\theta_s\mathcal{C}_{lo}(s) = \mathcal{C}_{hi}(t)$$

as claimed. □

On this basis we can establish the condition (MCm) of Theorem 5.7.1. As a technical detail, it is convenient to specialize slightly the marking condition $\theta(L_m) = M_m$ as indicated in (iii) below. In this version only strings in L_{voc} will be marked in L.

Theorem 5.7.2

Assume that

(i) $\theta : L \to M$ is an L_{voc}–observer;
(ii) $\mathbf{G_{lo}}$ is locally output controllable; and
(iii) $L_m = \theta^{-1}(M_m) \cap L_{\text{voc}}$

Then

$$\mathcal{C}_{hi}(M_m) = \theta\mathcal{C}_{lo}(L_m)$$

Proof

For inclusion ($\supseteq$) let $K_{lo} \in \mathcal{C}_{lo}(L_m)$; we show $K_{hi} := \theta(K_{lo}) \in \mathcal{C}_{hi}(M_m)$. First, $K_{lo} \subseteq L_m$ and $\theta(L_m) = M_m$ implies $K_{hi} \subseteq M_m$. For controllability, let $t \in \bar{K}_{hi}, \tau \in T_u$, $t\tau \in M$. Since $\overline{\theta(K_{lo})} = \theta(\bar{K}_{lo})$ (Exercise 5.2.1) there is $s \in \bar{K}_{lo}$ with $\theta(s) = t$, and we can certainly assume $s \in \bar{K}_{lo} \cap L_{\text{voc}}$. As $\theta(s) \le t\tau$, by (i) there exists $s' \in \Sigma^*$ with $ss' \in L_{\text{voc}}$ and $\theta(ss') = t\tau$. So $s' \in L_{\text{voc}}(s)$ and as $\tau \in T_u, s' \in \Sigma_u^*$ (by output-control-consistency, Sect. 5.3); therefore $ss' \in \bar{K}_{lo}$ (by controllability of K_{lo}). Finally

$$t\tau = \theta(ss') \in \theta(\bar{K}_{lo}) = \overline{\theta(K_{lo})} = \bar{K}_{hi}$$

as required.

For the reverse inclusion ($\subseteq$) let $\emptyset \neq K_{hi} \in \mathcal{C}_{hi}(M_m)$. We construct $K_{lo} \in \mathcal{C}_{lo}(L_m)$ inductively, such that $\theta(\bar{K}_{lo}) = \bar{K}_{hi}$. By (i), (ii) and Lemma 5.7.1 we know that $\mathbf{G_{lo}}$ is globally output controllable, namely for all $s \in L_{\text{voc}}, t = \theta_{\text{voc}}(s)$,

$$\mathcal{C}_{hi}(t) = \theta_s\mathcal{C}_{lo}(s) \tag{3}$$

Starting with $\epsilon \in \bar{K}_{hi}, \epsilon = \theta_{\text{voc}}(\epsilon)$, we have

$$\mathcal{C}_{hi}(\epsilon) = \theta_\epsilon \mathcal{C}_{lo}(\epsilon)$$

Since $\bar{K}_{hi}$ is controllable,

$$\text{Elig}(\bar{K}_{hi}; \epsilon) := \{\tau \in T | \tau \in \bar{K}_{hi}\} \in \mathcal{C}_{hi}(\epsilon)$$

and so

$$\text{Elig}(\bar{K}_{hi}; \epsilon) = \theta_\epsilon(H_{lo}(\epsilon))$$

for some locally controllable sublanguage $H_{lo}(\epsilon) \in \mathcal{C}_{lo}(\epsilon)$. Thus for every $\tau_1 \in \text{Elig}(\bar{K}_{hi}; \epsilon)$ there is at least one string $s_1 \in H_{lo}(\epsilon) \subseteq L_{\text{voc}}(\epsilon)$ with $\theta_{\text{voc}}(s_1) = \tau_1$. We continue in the evident way: for such τ_1 we have

$$\text{Elig}(\bar{K}_{hi}; \tau_1) := \{\tau \in T | \tau_1\tau \in \bar{K}_{hi}\} \in \mathcal{C}_{hi}(\tau_1)$$

so again by (3)

$$\text{Elig}(\bar{K}_{hi}; \tau_1) = \theta_{s_1}(H_{lo}(s_1))$$

for some $H_{lo}(s_1) \in \mathcal{C}_{lo}(s_1)$. In general, for $t \in \bar{K}_{hi}$ and $s \in L_{\text{voc}}$ with $\theta_{\text{voc}}(s) = t$, we shall have

$$\text{Elig}(\bar{K}_{hi}; t) := \{\tau \in T | t\tau \in \bar{K}_{hi}\} = \theta_s(H_{lo}(s))$$

for some locally controllable sublanguage $H_{lo}(s) \in \mathcal{C}_{lo}(s)$.

Denote by H_{lo} the prefix-closure of all strings of form

$$s = s_1 ... s_k, \quad k \in \mathbb{N}$$

such that

$$s_1 ... s_j \in L_{\text{voc}}, \quad 1 \leq j \leq k$$

$$\theta_{\text{voc}}(s) \in \bar{K}_{hi}$$

$$\text{Elig}(\bar{K}_{hi}; \theta_{\text{voc}}(s_j)) = \theta_{s_j}(H_{lo}(s_j)), \quad H_{lo}(s_j) \in \mathcal{C}_{lo}(s_j), \quad 1 \leq j \leq k$$

$$s_j \in H_{lo}(s_{j-1}), \quad 1 \leq j \leq k, \;\; s_0 := \epsilon$$

Clearly

$$\theta(H_{lo}) = \theta_{\text{voc}}(H_{lo} \cap L_{\text{voc}}) = \bar{K}_{hi}$$

We claim that $H_{lo} \in \mathcal{C}_{lo}(L)$. Let $s \in H_{lo}, \sigma \in \Sigma_u, s\sigma \in L$, and let s' be the maximal prefix of s such that $s' \in L_{\rm voc}$. If $s' < s$ then $s \in L - L_{\rm voc}$ and $s = s'v$ for some $v \in \bar{H}_{lo}(s')$. Since $H_{lo}(s')$ is locally controllable, $v\sigma \in \bar{H}_{lo}(s')$ so $s\sigma = s'v\sigma \in H_{lo}$. If $s' = s$ and $s\sigma \in L$ then again $\sigma \in \bar{H}_{lo}(s)$ by controllability of $H_{lo}(s)$, so $s\sigma \in H_{lo}$. This proves the claim.

Define

$$K_{lo} := H_{lo} \cap L_{\rm voc} \cap \theta^{-1}(K_{hi})$$

Then $K_{lo} \subseteq L_{\rm voc} \cap \theta^{-1}(M_m) = L_m$. To establish $K_{lo} \in \mathcal{C}_{lo}(L_m)$ it suffices to verify $\bar{K}_{lo} = H_{lo}$, or simply $H_{lo} \subseteq \bar{K}_{lo}$. Let $s \in H_{lo}, t := \theta(s)$. We claim there exists w with $sw \in K_{lo}$, i.e. $sw \in H_{lo} \cap L_{\rm voc} \cap \theta^{-1}(K_{hi})$. If already $s \in K_{lo}$, set $w = \epsilon$. If not, let $u \leq s$ be the maximal prefix of s with $u \in L_{\rm voc}$ and write $s = uv$. By construction of H_{lo}, we know that for some $w_1 \in \Sigma^*$ and locally controllable sublanguage $H_{lo}(u) \in \mathcal{C}_{lo}(u)$,

$$vw_1 \in H_{lo}(u),\ \mathrm{Elig}(\bar{K}_{hi}; t) = \theta_u(H_{lo}(u))$$

By definition of $\mathcal{C}_{lo}(u)$, we know $H_{lo}(u) \subseteq L_{\rm voc}(u)$ and thus $\theta_u(vw_1) = \tau_1(\text{say})$ with $t\tau_1 \in \bar{K}_{hi}$. This means $sw_1 = uvw_1 \in H_{lo} \cap L_{\rm voc} \cap \theta^{-1}(\bar{K}_{hi})$. If already $t\tau_1 \in K_{hi}$ then set $w = w_1$; if not, suppose $t\tau_1\tau_2 ... \tau_k \in K_{hi}$. Clearly

$$\tau_2 \in \mathrm{Elig}(\bar{K}_{hi}; t\tau_1),\ t\tau_1 = \theta_{\rm voc}(sw_1)$$

Repetition of the previous argument produces $w_2 \in \Sigma^*$ with

$$sw_1w_2 \in H_{lo} \cap L_{\rm voc} \cap \theta^{-1}(\bar{K}_{hi})$$

and we are done after at most k steps.

It only remains to verify that $\theta(K_{lo}) = K_{hi}$. Let $t \in K_{hi}(\subseteq M_m)$. Since $\theta(H_{lo}) = \bar{K}_{hi}$ there is $s \in H_{lo} \cap L_{\rm voc}$ such that $\theta_{\rm voc}(s) = t$, namely

$$s \in H_{lo} \cap L_{\rm voc} \cap \theta^{-1}(K_{hi}) = K_{lo}$$

On the other hand $\theta(K_{lo}) = \theta_{\rm voc}(K_{lo}) \subseteq K_{hi}$. □

Combining Theorems 5.7.1 and 5.7.2 we have immediately

Corollary 5.7.1.

Let $E_{hi} \subseteq M_m, K_{hi} := \sup \mathcal{C}_{hi}(E_{hi})$, and

$$K_{lo} := \sup \mathcal{C}_{lo}(L_m \cap \theta^{-1}(E_{hi}))$$

Assume conditions (i) - (iii) of Theorem 5.7.2. Then $(\mathbf{G_{lo}}, \theta)$ is hierarchically consistent, in the sense that $\theta(K_{lo}) = K_{hi}$. □

Exercise 5.7.1: By examining the initial state and vocal states in **CGLO**, Fig. 5.6.7, verify that the assumptions of Theorem 5.7.2 are valid for Transfer Line. ◇

The foregoing results can be regarded as a fundamental basis for hierarchical control with nonblocking. Of course, algorithmic design, and verification of the appropriate conditions, remain as challenging issues. To conclude this section we provide a more specialized result, building directly on the theory for closed languages in Sects. 5.4, 5.5, including the property of strict output-control-consistency (SOCC) and the control action of Sect. 5.4. For this we place a much stronger observer condition on θ, and modify our description of nonblocking slightly to facilitate use of Theorem 5.5.1. The price of greater concreteness, in Theorem 5.7.3 below, is somewhat involved argumentation in the proof.

With $M_m = \theta(L_m)$ as before we say that $\theta : L \to M$ is an L_m–*observer* if

$$(\forall t \in M_m)(\forall s \in L)\theta(s) \leq t \Longrightarrow (\exists u \in \Sigma^*)su \in L_m \ \&\ \theta(su) = t$$

In other words, whenever $\theta(s)$ can be extended in T^* to a string $t \in M_m$, the underlying string $s \in L$ can be extended to a string $su \in L_m$ with the same image under θ : "the manager's expectation is never blocked in $\mathbf{G_{lo}}$". This property fails for the example of Fig. 5.7.1, as one sees by taking $t = \tau_1\tau_2$ and $s = \alpha$.

The following more general definition will also be useful. Let $H_{lo} \subseteq L, H_{hi} \subseteq M$. We say that H_{lo} is *hierarchically nonblocking (HNB)* with respect to H_{hi} if

$$\begin{aligned}(\forall t \in \bar{H}_{hi})(\forall s \in \bar{H}_{lo})(\forall t' \in T^*)\theta(s) = t \ \&\ tt' \in H_{hi} \\ \Longrightarrow (\exists s' \in \Sigma^*)ss' \in H_{lo} \ \&\ \theta(ss') = tt'\end{aligned}$$

In words, whenever $\theta(s)$ can be extended to a string in H_{hi}, s can be extended to a string in H_{lo} with the same θ–image. This is essentially the L_m–observer property for θ, but 'parametrized' by H_{lo}, H_{hi} in place of L_m, M_m. We can now state the main result.

Theorem 5.7.3

Let $M_m = \theta(L_m)$ and let $\theta : L \to M$ be an L_m–observer. Assume $\mathbf{G_{lo}}$ is SOCC (as in Sect. 5.5). Let $\emptyset \neq K_{hi} \in \mathcal{C}_{hi}(M_m)$, and define

$$H_{lo} := \sup \mathcal{C}_{lo}(\theta^{-1}(\bar{K}_{hi}))$$

Then $\theta(H_{lo}) = \bar{K}_{hi}$ and H_{lo} is HNB with respect to K_{hi}.

Proof

Since $\bar{K}_{hi}$ is nonempty, closed and controllable, and $\mathbf{G_{lo}}$ is SOCC, we know by Theorem 5.5.1 that $\theta(H_{lo}) = \bar{K}_{hi}$.

Note that H_{lo} is closed. Suppose H_{lo} is not HNB with respect to K_{hi}. Then for some $t \in \bar{K}_{hi}, s \in H_{lo}$ and $w \in T^*$ we have

$$\theta(s) = t,\ tw \in K_{hi}$$

but for all $x \in \Sigma^*$ with $sx \in L$,

$$\theta(sx) = tw \Rightarrow sx \notin H_{lo} \tag{4}$$

Note that $w \neq \epsilon$; otherwise, by (4) with $x = \epsilon$, $s \notin H_{lo}$, contrary to hypothesis. However, as θ is an L_m–observer, and

$$\theta(s) = t < tw \in K_{hi} \subseteq M_m$$

there is $u \in \Sigma^*, u \neq \epsilon$, with $su \in L_m$ and $\theta(su) = tw$ (refer to Table 5.7.1). By (4), $su \notin H_{lo}$. Let u' ($\epsilon \leq u' < u$) be the maximal prefix of u such that $su' \in H_{lo}$. By controllability of H_{lo}, there is $\sigma \in \Sigma_c$ with $u'\sigma \leq u$ (and $su'\sigma \notin H_{lo}$). Let $u = u'\sigma u''$ (where possibly $u'' = \epsilon$). We have

$$\theta(su'\sigma) \leq \theta(su) = tw \in K_{hi}$$

so $\theta(su'\sigma) \in \bar{K}_{hi}$. Also, since $su' \in H_{lo}$ but $su'\sigma \notin H_{lo}$, and because H_{lo} is supremal controllable with respect to the specification $\theta^{-1}(\bar{K}_{hi})$, there is $v \in \Sigma_u^*$ such that $su'\sigma v \notin \theta^{-1}(\bar{K}_{hi})$, i.e. $\theta(su'\sigma v) \notin \bar{K}_{hi}$. Thus

$$t \leq \theta(su'\sigma) \leq tw,$$

so $\theta(su'\sigma) = tw'$ (say) $\in \bar{K}_{hi}$, while

$$\begin{aligned} \theta(su'\sigma v) &= \theta(su'\sigma)y \qquad \text{(say)} \\ &= tw'y \\ &\notin \bar{K}_{hi} \end{aligned}$$

We consider the two possible cases for node($su'\sigma$), in the reachability tree of L (as in Sect. 5.5).

1. Node($su'\sigma$) is vocal. Since $v \in \Sigma_u^*$, we have that $y \in T_u^*$. Thus $tw' \in \bar{K}_{hi}$ while $tw'y \notin \bar{K}_{hi}$, contradicting the controllability of $\bar{K}_{hi}$.

2. Node($su'\sigma$) is silent. We claim there is some vocal node in the path from node($su'\sigma$) to node($su'\sigma u''$). Otherwise

$$\theta(su') = \theta(su'\sigma) = \theta(su'\sigma u'') = \theta(su) = tw$$

But as $su' \in H_{lo}$ we have a contradiction to (4), and conclude that $\theta(su'\sigma u'') > \theta(su'\sigma)$. Let

$$\theta(su'\sigma u'') = \theta(su'\sigma)\tau_1 ... \tau_k$$

and let $u''' \leq u''$ with

$$\text{node}(su'\sigma u''') \text{ vocal}, \ \theta(su'\sigma u''') = \theta(su'\sigma)\tau_1$$

Recall that $tw' \in \bar{K}_{hi}$ while $tw'y \notin \bar{K}_{hi}$, so $\epsilon \neq y = \tau_1'\tau_2'...\tau_l'$ (say). Let $v' \leq v$ be such that node $(su'\sigma v')$ is vocal with

$$\theta(su'\sigma v') = \theta(su'\sigma)\tau_1' = tw'\tau_1'$$

We claim that $\tau_1' \neq \tau_1$. Otherwise

$$tw'\tau_1' = tw'\tau_1 = \theta(su'\sigma)\tau_1 = \theta(su'\sigma u''') \leq \theta(su'\sigma u'') = \theta(su) = tw \in K_{hi}$$

i.e. $tw'\tau_1' \in \bar{K}_{hi}$. But then $v = v'v''$ (say), and $v'' \in \Sigma_u^*$, imply $\tau_j' \in T_u (2 \leq j \leq l)$, hence $tw'y \in \bar{K}_{hi}$ (by controllability), a contradiction. So $\tau_1' \neq \tau_1$ as claimed, and therefore

$$\text{node}(su'\sigma u'''), \ \text{node}(su'\sigma v')$$

are partners (in the sense of Sect. 5.5). This contradicts our hypothesis that $\mathbf{G_{lo}}$ is SOCC. □

string $\in L$	θ(*string*) $\in M$
s	t
$su = su'\sigma u''$	tw
$su'\sigma$	tw'
$su'\sigma u''$	$tw = tw'\tau_1...\tau_k$
$su'\sigma u'''$	$tw'\tau_1$
$su'\sigma v = su'\sigma v' v''$	$tw'y = tw'\tau_1'...\tau_l'$
$su'\sigma v'$	$tw'\tau_1'$

Table 5.7.1
String factorizations in proof of Theorem 5.7.3.

Corollary 5.7.2

With L_{voc} as defined earlier in this section, let

$$L_m := \theta^{-1}(M_m) \cap L_{voc}$$

and (with H_{lo} as in Theorem 5.7.3) define

$$K_{lo} := H_{lo} \cap L_m$$

Then $H_{lo} = \bar{K}_{lo}$. □

The L_m–observer property of θ is stronger than necessary for the result in Theorem 5.7.3.

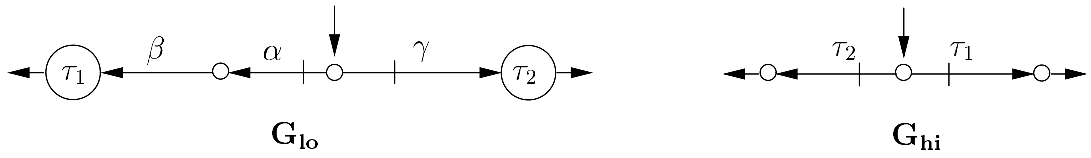


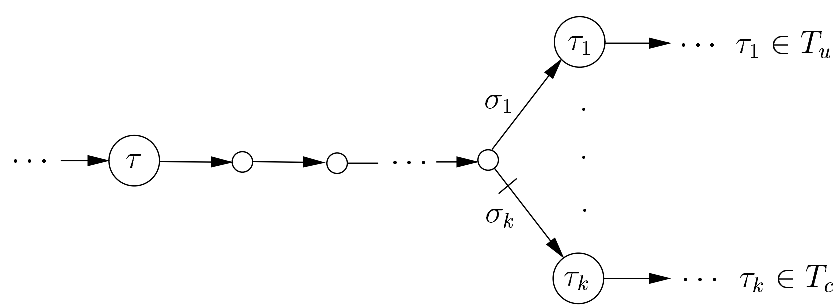
Fig. 5.7.2

In Fig. 5.7.2, $\theta(\alpha) = \theta(\epsilon) = \epsilon, \theta(\epsilon\gamma) = \theta(\gamma) = \tau_2$ but $\theta(\alpha s') \neq \tau_2$ for any s', so θ is not an L_m–observer. However, for any choice of $K_{hi} \in \mathcal{C}_{hi}(M_m), H_{lo} := \sup \mathcal{C}_{lo}(\theta^{-1}(\bar{K}_{hi}))$ is HNB.

Exercise 5.7.2: Prove Corollary 5.7.2 and interpret the result in terms of 'low-level non-blocking'.

Exercise 5.7.3: Verify that the assumptions of Theorem 5.7.3 are valid for the Transfer Line **CGLO**, Fig. 5.6.7.

Exercise 5.7.4: Investigate how Theorem 5.7.3 might be improved by (possibly) weakening the L_m–observer property but strengthening the SOCC assumption. For instance, the L_m–observer property becomes more 'reasonable' if the local control structure is everywhere as sketched below:



Formalize, relate to Theorem 5.7.3, and discuss necessity of the L_m–observer property in this situation.

5.8 Appendix: Computational Algorithm for Output-Control-Consistency

We provide an algorithm for implementing the procedure of Sect. 5.3 for achieving output-control-consistency. It is assumed that $\mathbf{G_{lo}}$ is represented as a finite-state Moore transition structure

$$\mathbf{G_{lo}} = (Q, \Sigma, T_o, \delta, \omega, q_o, Q_m)$$

as defined in Sect. 5.2. The state-transition graph (including state outputs) of $\mathbf{G_{lo}}$ will be denoted simply by $\mathbf{G}$. Recall that $T_o = T \cup \{\tau_o\}$, τ_o being the 'silent' output symbol. Adapting the terminology of Sect. 5.3 to $\mathbf{G}$, we say that a state $q \in Q$ is *silent* if $\omega(q) = \tau_o$ or is *vocal* if $\omega(q) \in T$. The initial state q_o is silent. A path in $\mathbf{G}$, displayed as

$$q' \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \longrightarrow ... \xrightarrow{\sigma_{k-1}} q_{k-1} \xrightarrow{\sigma_k} q$$

is *silent* if (i) q' is either a vocal state or the initial state, and (ii) either $k = 1$, or $k > 1$ and the states $q_1, ..., q_{k-1}$ are silent. A silent path is *red* if at least one of its transition labels $\sigma_1, ..., \sigma_k$ is controllable; otherwise it is *green*. For each $q \in Q$ let $P(q)$ be the set of all silent paths that end at q; because of possible loops in $\mathbf{G}$, $P(q)$ may be infinite. Then $\mathbf{G}$ is *output-control-consistent* (OCC) if and only if, for each vocal state $q \in Q$, either every $p \in P(q)$ is red or every $p \in P(q)$ is green.

In general $\mathbf{G}$ will fail to be OCC. Our objective is to replace $\mathbf{G}$ by a new version $\mathbf{G_{new}}$ that is OCC, has the same marked and closed behavior as $\mathbf{G}$, and incorporates the modified (split) output structure described for the reachability tree in Sect. 5.3. To this end we define, for any graph $\mathbf{G}$ of the type described, the predecessor and successor functions:

$$\text{in\_set} : Q \to Pwr(Q \times \Sigma)$$

$$\text{in\_set}(q) := \{(q', \sigma) \in Q \times \Sigma | \delta(q', \sigma) = q\}$$

$$\text{out\_set} : Q \to Pwr(\Sigma \times Q)$$

$$\text{out\_set}(q) := \{(\sigma, q'') \in \Sigma \times Q | \delta(q, \sigma) = q''\}$$

It can and will be arranged that $\text{in\_set}(q) = \emptyset$ if and only if $q = q_o$. For any pair (q', σ) define

$$\text{out\_color}(q', \sigma) \in \{\text{red,green,amber}\}$$

according to the rule

$$\text{out\_color}(q', \sigma) := \begin{cases} \text{red if: red}(q') \ \& \ \text{silent}(q') \ \& \ \text{uncont}(\sigma) \\ \quad \text{or: cont}(\sigma) \\ \text{green if: green}(q') \ \& \ \text{silent}(q') \ \& \ \text{uncont}(\sigma) \\ \quad \text{or: vocal}(q') \ \& \ \text{uncont}(\sigma) \\ \text{amber if: amber}(q') \ \& \ \text{uncont}(\sigma) \end{cases}$$

(Here cont(σ) means $\sigma \in \Sigma_c$, uncont(σ) means $\sigma \in \Sigma_u$). Then define, for $q \neq q_o$,

$$\text{new\_color}(q) := \begin{cases} \text{red if out\_color}(q', \sigma) = \text{red} \\ \quad \text{for all } (q', \sigma) \in \text{in\_set}(q) \\ \text{green if out\_color}(q', \sigma) = \text{green} \\ \quad \text{for all } (q', \sigma) \in \text{in\_set}(q) \\ \text{amber otherwise} \end{cases}$$

Without essential loss of generality it will be assumed that every state of **G** is coreachable with respect to the vocal states: that is, from every state there is some vocal state that can be reached by a path in **G**. It is then straightforward to show from the foregoing assumptions and definitions that **G** is OCC just in case

(i) for all $q \in Q$, either color(q) = red or color(q) = green, and

(ii) the following stability condition is satisfied:

$$(\forall q \in Q) \text{ new\_color}(q) = \text{color}(q) \qquad (stable)$$

The idea of the OCC algorithm is iteratively to modify the graph **G** by a process of recoloring, and elimination of amber states by state-splitting, until (*stable*) is achieved. The formal procedure is summarized as Procedure OCC in the pseudo-Pascal Unit POCC listed in Section 5.11. By the procedure Initialize, OCC first assigns to all q color(q) := green. By RecolorStates, each state other than the initial state q_o (which stays green) is recolored in arbitrary order according to the assignment

$$\text{color(state)} := \text{new\_color(state)} \tag{$*$}$$

If thereby any state changes color, (*stable*) is falsified.

In case (*stable*) is false, each amber state (if any), say q_a, is processed by FixAmberStates. Initially the subprocedure SplitAmberStates splits q_a into siblings q_r, q_g with the assignments

$$\text{color}(q_r) := \text{red}, \quad \text{color}(q_g) := \text{green}$$

If $q_a \in Q_m$ then the new states q_r, q_g are declared to be marker states as well, while the subprocedure MakeNewOutputs assigns new outputs

$$\omega(q_r) := [\tau, 1], \quad \omega(q_g) := [\tau, 0] \quad \text{if} \quad \omega(q_a) = \tau \in T$$

$$\omega(q_r) = \omega(q_g) = \tau_o \quad \text{if} \quad \omega(q_a) = \tau_o$$

Next the local transition structure at q_a is modified. The subprocedure MakeNewTrans executes the following:

(i) for all $(q', \sigma) \in \text{in\_set}\ (q_a)$
case $q' \neq q_a$

back-connects (q_r, q_g) to create transitions

$[q', \sigma, q_r]$ if out_color $(q', \sigma) = \text{red}$
$[q', \sigma, q_g]$ otherwise

case $q' = q_a$ & q_a silent

creates transitions

$[q_r, \sigma, q_r], [q_g, \sigma, q_r]$ if cont(σ)
$[q_r, \sigma, q_r], [q_g, \sigma, q_g]$ if uncont(σ)

case $q' = q_a$ & q_a vocal

creates transitions

$[q_r, \sigma, q_r], [q_g, \sigma, q_r]$ if cont(σ)
$[q_r, \sigma, q_g], [q_g, \sigma, q_g]$ if uncont(σ)

(The selfloop cases $q' = q_a$ are treated by first copying q_a as $q'_a \neq q_a$, splitting both q'_a and q_a, back-connecting as in first case, then merging q'_r, q_r and q'_g, q_g.)

(ii) for all $(\sigma, q'') \in \text{out\_set}(q_a)$, forward-connects q_r, q_g to create transitions

$$[q_r, \sigma, q''], \quad [q_g, \sigma, q'']$$

(iii) removes q_a with its associated transitions from the database.

A list of split states is maintained to ensure that a state is split at most once. If on a subsequent iteration a state that is a member of a split pair is recolored amber, then instead of SplitAmberStates the subprocedure SwitchOldTrans is invoked and executes the following:

(i) gets the siblings q_r, q_g corresponding to q_a (it will be shown below that necessarily $q_a = q_g$)

(ii) for all $(q', \sigma) \in \text{in\_set}(q_a)$ such that out_color $(q', \sigma) = \text{red}$, creates $[q', \sigma, q_r]$ and deletes $[q', \sigma, q_a]$.

The foregoing process is iterated by the **repeat** loop until (*stable*) (hence condition (*)) becomes true.

It must be shown that OCC terminates. For this we first note

Property 1

Once a state (or a state sibling) has been colored red, it remains red in subsequent iterations.

Proof

Suppose some red state reverts to green or amber on a subsequent iteration. Consider the first instance (state q and iteration $\#N$) of such a change. At iteration $\#(N-1)$, for each element

$$(q', \sigma) \in \text{ in\_set}(q)$$

it was the case that either $\sigma \in \Sigma_c$ or (if $\sigma \in \Sigma_u$) q' was silent and red. Since the controllable status of σ and the vocal status of q' are both invariant, q' must have changed from red to green or amber, contradicting the assumption that q is the first state to have so changed. □

Since a state is split at most once, Property 1 implies

Property 2

Eventually the state set size, and the subsets of red states and of green states on reentry to the **repeat** loop, remain invariant. □

To prove termination it now suffices to show that a state can change from green to amber at most finitely often. By Property 2 and the action of SwitchOldTrans, eventually all transitions

$$[q', \sigma, q] \quad \text{with out\_color}(q', \sigma) = \text{red}$$

will have been switched to states q with $\text{color}(q) = \text{red}$. Under this condition, on reentry to the **repeat** loop it will be true for any transition $[q', \sigma, q]$ that

$$\text{color}(q) = \text{green} \Rightarrow \text{out\_color}(q', \sigma) = \text{green}$$

It follows that for all states q, (*) is satisfied and therefore (*stable*) is true, as claimed.

Because a state is split at most once, when OCC terminates the state set will have at most doubled in size; it is also clear that the closed and marked behaviors of the generator described by the graph remain unchanged.

Computational effort can be estimated as follows. Suppose $\mathbf{G}$ has n states and m transitions. Both RecolorStates and FixAmberStates will require $O(nm)$ steps (comparisons and assignments), while the number of iterations is at worst $O(n+m)$, giving $O(nm(n+m))$ steps for the overall computation.

5.9 Appendix: Conceptual Procedure for Strict-Output-Control-Consistency

To achieve strict-output-control-consistency, a conceptual procedure based on the reachability tree of $\mathbf{G_{lo}}$ (say, *tree*) can be organized as follows.

Starting at the root node, order the nodes of *tree* level-by-level (i.e. breadth first) as 0,1,2,...; and write $n < n'$ if n precedes n' in the ordering. Let $\zeta \notin T_o$ be a new output symbol, and let

$$T_{o,new} = T_o \dot{\cup} \{\zeta\}$$

be the extension of T_o. We write path(nsn'), and refer to the *path* nsn', if there is a path in *tree* starting at n and leading via $s \in \Sigma^*$ to n'. Say the node n is *vocalizable* if

(i) n is silent;

(ii) $(\exists n_o < n, \sigma \in \Sigma_c)$ path$(n_o \sigma n)$;

(iii) $(\exists n_1 > n, s_1 \in \Sigma_u^*)$ path(ns_1n_1), ns_1n_1 is silent, and n_1 is vocal; and

(iv) $(\exists n_2 > n, s_2 \in \Sigma^*)$ path(ns_2n_2), ns_2n_2 is silent, n_2 is vocal, and $\hat{\omega}(n_2) \neq \hat{\omega}(n_1)$.

Conditions (i)-(iv) express the fact that n_1, n_2 are partners with respect to n. Examining each node in order, modify *tree* as follows. If node n is vocalizable, then *vocalize* n by redefining $\hat{\omega}$ at n as $\hat{\omega}_{new}(n) = \zeta$. [If $\hat{\omega}(n_1) = \tau_c \in T_c$ then ultimately we shall redefine $\hat{\omega}$ at n_1 as $\hat{\omega}_{new}(n_1) = \tau_u$, but at this stage no change of state output will be introduced.] Otherwise, if n is not vocalizable, go on to the successor node of n.

Since vocalization of n has no effect on nodes $n' < n$, the procedure is well defined, transforming *tree* to *newtree*, say. Furthermore if n is vocalizable it remains so as the procedure moves to nodes $n'' > n$ (vocalization at a given level is never rendered superfluous by vocalization later on), because the uncontrollable silent path nsn_1 is never modified. Define str:$\mathcal{N} \to \Sigma^*$ by node(str$(n)) = n$. Write $s = \text{str}(n)$, $s' = \text{str}(n')$ and define $n \equiv n'$ (mod *tree*) to mean (cf. Sect. 5.3)

$$s \equiv s' (\text{mod } L_o) \text{ and } s \equiv s' (\text{mod } \theta)$$

Note that $n \equiv n'$ if and only if the subtrees of *tree* rooted at n and n' respectively are identical. Corresponding to *newtree* we shall have the map

$$\theta_{new} : \Sigma^* \to T_{o,new}^*$$

The equivalence $\equiv$ (mod θ_{new}) is then defined in similar fashion (Sect. 5.3), as is equivalence $\equiv$ (mod *newtree*).

Now suppose that n, n' are vocalizable. If $n < n'$ and there is $s \in \Sigma^*\Sigma_c$ such that path(nsn'), it is clear that vocalization of n can have no effect on the subtree rooted at n'; and the same is true *a fortiori* if there is no path from n to n'. Consequently, if nodes n, n' are vocalizable, and $n \equiv n'$ (mod *tree*), then also $n \equiv n'$ (mod *newtree*). Next suppose that neither n nor n' is vocalizable and that $n \equiv n'$ (mod *tree*). Since the subtrees of *tree* rooted respectively at n and at n' are identical, and neither n nor n' gets vocalized, the vocalization procedure applied to these subtrees must yield the same result; that is, the subtrees of *newtree* rooted at n and n' must be identical, and so again $n \equiv n'$ (mod *newtree*).

The foregoing discussion can be summarized by the assertions that the cells of $\equiv$ (mod *newtree*) are formed by splitting the cells of $\equiv$ (mod *tree*) according to the partition of nodes in *tree* into vocalizable and nonvocalizable; and that only cells of the silent nodes of *tree* are so affected. Thus, in the regular case, if the canonical Moore automaton $\mathbf{G_{lo}}$ (corresponding to *tree*) has N_s silent states and N_v vocal states, the canonical Moore automaton $\mathbf{G_{lo,new}}$ (corresponding to *newtree*) will have no more than N_s silent states and $N_v + N_s$ vocal states.

The vocalization procedure in no way depended on a prior assumption that $\mathbf{G_{lo}}$ was output-control-consistent: in (iv) above the condition $\hat{\omega}(n_1) \neq \hat{\omega}(n_2)$ is true after the OCC output assignment procedure (Sect. 5.4) if and only if it was true beforehand. Thus vocalization could be carried out initially, before the OCC procedure itself, with no difference to the result. Suppose this is done. It is clear that *newtree* can then be rendered OCC by the OCC procedure, and that this process will not introduce any vocalizable nodes (by the remark just made about (iv)). The final result is therefore SOCC, as required, and the final state count in terms of the parameters above is bounded by $2(N_v + 2N_s)$, or less than four times the state count of the Moore structure provided at the start.

5.10 Appendix: Computational Algorithm for Hierarchical Consistency

While the property of strict-output-control-consistency is sufficient for hierarchical consistency (Theorem 5.5.1) and, as described in Sect. 5.9, is conceptually straightforward to achieve, it falls short of being necessary. In this section a somewhat weaker (albeit still not necessary) condition that ensures the desired result will be introduced, together with an effective algorithm to achieve hierarchical consistency in the regular case. This algorithm is slightly more efficient than the procedure of Sect. 5.9. in that possibly fewer states need to be newly vocalized. Regrettably, the approach is rather complicated to describe. A high-level pseudo-Pascal version is listed as Program PSHC in Section 5.11. PSHC **uses** Procedure OCC from Unit POCC, as well as Procedure HCC described below.

Consider a (finite) state-transition graph $\mathbf{G}$ for $\mathbf{G_{lo}}$, where we assume that $\mathbf{G_{lo}}$ is already output-control-consistent. Thus, in the terminology for state-transition graphs introduced in Sect. 5.8, each vocal state of $\mathbf{G}$ is unambiguously either red (controllable output) or green (uncontrollable output). The following definitions will be needed. A *silent path suffix (sps)* is a path

$$q \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \longrightarrow \dots \xrightarrow{\sigma_{k-1}} q_{k-1} \xrightarrow{\sigma_k} q'$$

with $q_1, ..., q_{k-1}$ silent and q' vocal (but q unrestricted). An *agent* is a pair

$$(q, \sigma) \in Q \times \Sigma_c$$

such that there is an *sps* as displayed, with $\sigma_1 = \sigma \in \Sigma_c$ and $\sigma_2, ..., \sigma_k \in \Sigma_u$ (thus q' is red). Say q' is *critical* for (q, σ), and denote the subset of critical states by $C(q, \sigma)$. According to the definition of γ_{lo}, we have $\gamma_{lo}(\_, \sigma) = 0$ (i.e. an event $\sigma \in \Sigma_c$ is disabled under command and control) just at states $q \in Q$ such that (q, σ) is an agent and $C(q, \sigma)$ contains a state q' such that $\omega(q') = \tau'$ with $\gamma_{hi}(\_, \tau') = 0$. Next, the *blocking set* $B(q, \sigma)$ of an agent (q, σ) is the subset of (red, vocal) states $q' \in Q$ such that an *sps* exists as displayed above, with $\sigma_1 = \sigma$ but $\sigma_2, ..., \sigma_k$ unrestricted; thus $B(q, \sigma) \supseteq C(q, \sigma)$, and the inclusion may be strict. An agent (q, σ) is *unary* if $|B(q, \sigma)| = 1$, i.e. contains just one state, which is thus critical. Let $p, q \in Q$ with p red and vocal but q arbitrary. An *sps* from q to p is *dedicated* if each of its transitions $[a, \lambda, b]$ is such that either (a, λ) is not an agent, or (a, λ) is a unary agent with $B(a, \lambda) = \{p\}$. Define the set

$$\begin{aligned} D(q) \quad := \quad & \{p \in Q \mid \text{red}(p) \ \& \ \text{vocal}(p) \\ & \ \& \text{ there exists a dedicated } sps \text{ joining } q \text{ to } p\} \end{aligned}$$

An agent (q, σ) will be called *admissible* if

$$B(q, \sigma) \subseteq D(q);$$

that is, each state p (potentially) blocked by (q, σ) is reachable from q along a dedicated *sps*. Otherwise (q, σ) is *inadmissible*. Note that if $q' = \delta(q, \sigma)$ (with q' vocal, $\sigma \in \Sigma_c$) then (q, σ) is always admissible; so if (q, σ) is inadmissible then $\delta(q, \sigma)$ is silent.

The idea of the algorithm (Procedure HCC) is to identify the inadmissible agents (q, σ), then vocalize the states $\delta(q, \sigma)$. The connection with the conceptual procedure of Sect. 5.9 is provided by

Proposition 5.10.1

If $\mathbf{G_{lo}}$ is SOCC then every agent of $\mathbf{G_{lo}}$ is admissible. If $\mathbf{G_{lo}}$ is OCC and every agent in some finite transition graph (FTG) of $\mathbf{G_{lo}}$ is unary, then $\mathbf{G_{lo}}$ is SOCC.

Proof

Suppose an agent (q, σ) is inadmissible. Then there is $p \in B(q, \sigma)$ with $p \notin D(q)$, so there is no dedicated *sps* from q to p. By definition of $B(q, \sigma)$ there is an *sps*

$$q \xrightarrow{\sigma} q_1 \xrightarrow{\sigma_2} q_2 \longrightarrow \dots \xrightarrow{\sigma_{k-1}} q_{k-1} \xrightarrow{\sigma_k} p$$

Since this *sps* is not dedicated, it includes a transition $[a, \lambda, b]$ such that (a, λ) is an agent and it is not the case that (a, λ) is a unary agent with $B(a, \lambda) = \{p\}$. Thus either (a, λ) is a unary agent with $B(a, \lambda) \neq \{p\}$, or (a, λ) is a non-unary agent. The first case is impossible since the *sps* ends at p. In the second case, since (a, λ) is an agent there is an *sps*

$$a \xrightarrow{\lambda} a_1 \xrightarrow{\lambda_2} a_2 \longrightarrow ... \xrightarrow{\lambda_{h-1}} a_{h-1} \xrightarrow{\lambda_h} a'$$

with a' red and vocal and $\lambda_2, ..., \lambda_h \in \Sigma_u$; and since (a, λ) is non-unary there is an *sps*

$$a \xrightarrow{\lambda} a_1 \xrightarrow{\mu_2} b_2 \longrightarrow ... \xrightarrow{\mu_{j-1}} b_{j-1} \xrightarrow{\mu_j} a''$$

with a'' red and vocal, $a'' \neq a'$, and the μ's unrestricted. Let n be a node in $\mathbf{T}$ corresponding to a, and let the strings $\lambda\lambda_2...\lambda_h$ resp. $\lambda\mu_2...\mu_j$ lead from n to nodes n_1 resp. n_2. Clearly n_1, n_2 are partners, namely $\mathbf{G_{lo}}$ is not SOCC.

For the second assertion note that the reachability tree of $\mathbf{G_{lo}}$ can be constructed inductively from an FTG for $\mathbf{G_{lo}}$ by tracing all silent paths in the FTG from the root node to a vocal node or from one vocal node to another. If every agent $a = (q, \sigma)$ in the FTG is unary then all *sps* in the FTG which start with a terminate at the same (red, vocal) state, hence the corresponding silent paths in the tree terminate at equivalent nodes. But this fact rules out the existence of partners. □

We note that $\mathbf{G_{lo}}$ need not be SOCC even though every agent of its FTG is admissible. For instance, consider the structures $\mathbf{G}$ and $\mathbf{T}$ shown in Fig. 5.10.1; $\mathbf{T}$ is the reachability tree corresponding to $\mathbf{G}$. In $\mathbf{G}$ the agent (0,1) is admissible because the *sps* $0 \xrightarrow{3} 2$ and $0 \xrightarrow{5} 3$ are dedicated, and obviously (0,3),(0,5) are admissible. Hence every agent is admissible, but in $\mathbf{T}$ nodes 4 and 5 are partners.

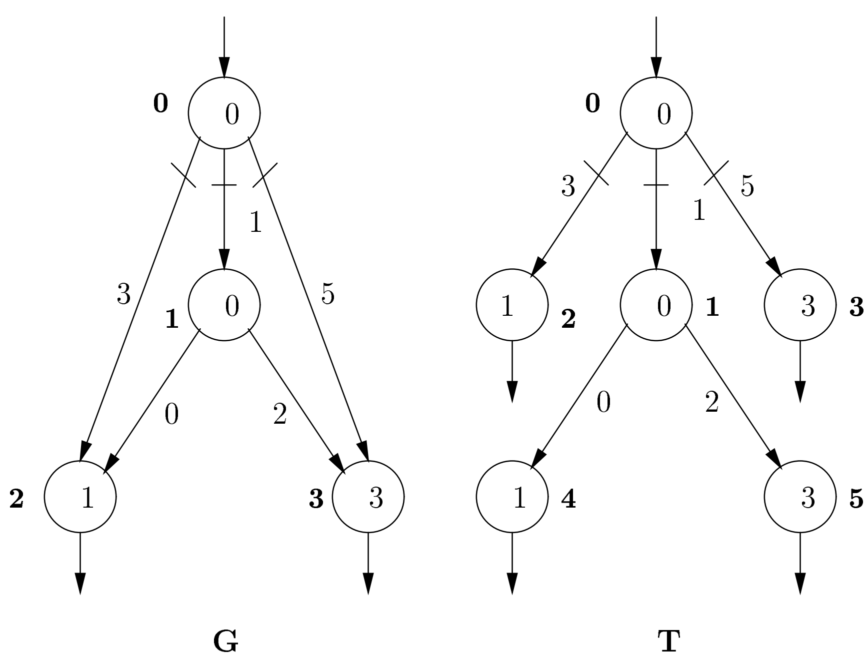


Fig. 5.10.1

The counterpart of Theorem 5.5.1 that we need is the following.

Proposition 5.10.2

Assume that $\mathbf{G_{lo}}$ is OCC and that every agent in the (finite) state transition graph of $\mathbf{G_{lo}}$ is admissible. Let $E_{hi} \subseteq L(\mathbf{G_{hi}})$ be nonempty, closed and controllable. Then

$$\theta((\theta^{-1}(E_{hi}))^{\uparrow}) = E_{hi}$$

Proof

The proof will just be summarized, with details left to the reader. Write $L(\mathbf{G_{lo}}) =: L_{lo}$, $L(\mathbf{G_{hi}}) =: L_{hi}$ and $L(\gamma_{lo}, \mathbf{G_{lo}}) =: K_{lo}$. As in the proof of Theorem 5.5.1 it suffices to show that

$$\theta(K_{lo}) = E_{hi}$$

To the contrary, assume that $E_{hi} - \theta(K_{lo}) \neq \emptyset$. We begin with the same construction as in the proof of Theorem 5.5.1, and select $t, t', t'', s, s'', w, w', \sigma$ and τ as before, so that $t \in E_{hi} - \theta(K_{lo})$, $t' < t$ is maximal for $t' \in \theta(K_{lo})$, $s \in E_{lo}$ with $\theta(s) = t$, $\theta(s'') = t'' \leq t'$, $\theta(s''w) = t''\tau$, $s''w' \in K_{lo}$, $s''w'\sigma \notin K_{lo}$, $w'\sigma \leq w$, $s''w \leq s$, and $\sigma \in \Sigma_c$. This time we work in the finite state transition graph of $\mathbf{G_{lo}}$ and write $q'' := \delta(q_o, s'')$, $q' := \delta(q_o, s''w')$ and $q := \delta(q_o, s''w)$. Like node(s'') and node$(s''w)$ in the former proof, here q'' and q are vocal. Note that (q', σ) is an agent, with $q \in B(q', \sigma)$. By hypothesis, there is a dedicated *sps* joining q' to each state p in the set $B(q', \sigma)$, hence there is an *sps* from q'' to q via q', along which no element $\sigma \in \Sigma_c$ is disabled by γ_{lo}. That is, there is $v \in \Sigma^*$ of the form $v = w'v'$ such that

$$\delta(q_o, s''v) = \delta(q_o, s''w) = q, \quad s''v \in K_{lo} \quad \text{and} \quad \theta(s''v) = t''\tau$$

In case $t'' = t'$, we have that $t'\tau \in \theta(K_{lo})$, contradicting the maximality of t', and we are done. Otherwise, suppose $t'' < t'$. Since $s''w \leq s$, we have $s''wx = s$ for some $x \in \Sigma^*$. Now replace s by $s_{new} := s''vx$, s'' by $s''v$, t'' by $t''_{new} := t''\tau$, q'' by $q''_{new} := q$, and repeat the argument starting from state q''_{new}. At each stage we extend $t'' < t'$ by one element τ at a time, to eventually achieve that $t''_{new} = t'$. After one more, final stage we obtain $t'\tau \leq t$ with $t'\tau \in \theta(K_{lo})$. Since this contradicts the maximality of t', we must have $\theta(K_{lo}) = E_{hi}$ after all. □

Observe that an agent (q, σ) that is inadmissible can be rendered admissible simply by vocalizing the state $q' := \delta(q, \sigma)$ (cf. Sect. 5.9). It is clear that the totality of such vocalizations cannot introduce further agents, i.e. convert a pair $(\hat{q}, \hat{\sigma})$ (with $\hat{\sigma} \in \Sigma_c$ and $\delta(\hat{q}, \hat{\sigma})!$) to an agent if it was not an agent before. In fact, if a new agent $(\hat{q}, \hat{\sigma})$ were thereby created, a new *sps* would appear, of the form

$$\hat{q} \xrightarrow{\hat{\sigma}} q_1 \xrightarrow{\sigma_2} q_2 \longrightarrow \ldots \xrightarrow{\sigma_{k-1}} q_{k-1} \xrightarrow{\sigma_k} q'$$

where $\sigma_2, ..., \sigma_k \in \Sigma_u$. Because (q, σ) was already an agent, there is an *sps*

$$q' \xrightarrow{\sigma_1'} q_1 \xrightarrow{\sigma_2'} q_2 \longrightarrow ... \xrightarrow{\sigma_{k-1}'} q_{k-1} \xrightarrow{\sigma_k'} p$$

with $\sigma_1', ..., \sigma_k' \in \Sigma_u$. Clearly the catenation of these two *sps* was, prior to the vocalization of q', an *sps* joining $\hat{q}$ to p, namely $(\hat{q}, \hat{\sigma})$ was previously an agent after all.

In HCC each agent is inspected in turn, with any agent that is inadmissible immediately vocalized; as shown above, in this process no new agents are created. However, it may result from the process that an admissible agent is converted to one that is inadmissible. But as any agent, once vocalized, remains admissible, by repetition of the process a number of times at most equal to the number of agents (i.e. no more than the number of transitions) all agents will eventually be rendered admissible. HCC therefore loops until this condition is satisfied.

In general, vocalization will destroy the property that $\mathbf{G_{lo}}$ is output-control-consistent. If the algorithm of Sect. 5.8. (Procedure OCC) is executed once more, OCC will be restored. It remains to show that in this final step, involving state-splitting and recoloring, no inadmissible agents are created. Indeed, a new agent is created only if a former agent (q, σ) is split, into (q_r, σ), (q_g, σ) say. Consider a dedicated *sps* formerly joining q to q' (say). If q' is not split, the *sps* will get replaced by a dedicated *sps* to q' from each of q_r, q_g, so the two new agents are again admissible; while if q' is split into q_r', q_g' this conclusion holds with q_r' in place of q'. We infer that the inclusion $B(q, \sigma) \subseteq D(q)$ is true for each agent in the new transition structure provided it was true for each agent in the old.

Computational effort can be estimated as follows. Suppose $\mathbf{G}$ has n states and m transitions. For a pair (q, σ), the subsets $D(q)$ and $B(q, \sigma)$ can each be identified by examining all (state,transition) pairs, namely in $O(nm)$ steps. Checking the inclusion $B(q, \sigma) \subseteq D(q)$ requires at most $O(n^2)$ steps. As there are at most m agents, Procedure HCC therefore requires $O(n^3m + n^2m^2)$ steps. Combining this result with that of Appendix 5.8 for Procedure OCC, we obtain a complexity bound of

$$O(n^3m + n^2m^2 + nm^2) = O(n^3m + n^2m^2)$$

for the overall Program PSHC for achieving high-level hierarchical consistency.

5.11 Listing for Pseudo-Pascal Unit POCC and Program PSHC

```
UNIT POCC;

  {pseudo-Pascal Unit with procedure OCC for achieving output-control-consistency;
  operates on database defining transition structure of G_lo, to create
  G_lo_new}

  INTERFACE

  CONST
    {specify integers MaxStateSize, MaxAlphabetSize}

  TYPE
    States = 0..MaxStateSize;
    Events = 0..MaxAlphabetSize;
    Colors = (red, green, amber);
    In_Events = array[States, Events] of Integer;
    Out_Events = array[Events, States] of Integer;

  PROCEDURE OCC;

  FUNCTION GetSize:  States;

  IMPLEMENTATION

  VAR
    stable:  Boolean;

  FUNCTION GetSize;
    begin
      {reads size of state set from database}
    end;

  FUNCTION GetColor(state:  States):  Colors;
    begin
      {reads state color from database}
    end;

  PROCEDURE SetColor(state:  States; color:  Colors);
    begin
      {assigns state color to database}
```

```
  end;

PROCEDURE SetSibling(state, sibling:  States);
  begin
    {assigns sibling to state in database}
  end;

FUNCTION GetSplit(state:  States):  Boolean;
  begin
    {returns true if state listed as split in database}
  end;

PROCEDURE SetSplit(state:  States; split:  Boolean);
  begin
    {lists state as split in database}
  end;

PROCEDURE InSet(state:  States; var in_set:  In_Events);
  begin
    {collects in_set(state) from database}
  end;

PROCEDURE OutSet(state:  States; var out_set:  Out_Events);
  begin
    {collects out_set(state) from database}
  end;

PROCEDURE Initialize;
  var size, state:  States;
  begin
    size:= GetSize;
    for state:= 0 to size-1 do
      begin
        SetColor(state, green);
        SetSplit(state, false)
      end
  end;

PROCEDURE RecolorStates;

  FUNCTION NewColor(in_set:  In_Events):  Colors;
    begin
      {returns color determined by in_set}
    end;
```

```
  VAR old_color, new_color:  Colors;
      in_set:  In_Events;
      size, state:  States;

  begin {RecolorStates}
    size:= GetSize;
    for state:= 0 to size-1 do
      begin
        old_color:= GetColor(state);
        InSet(state, in_set);
        new_color:= NewColor(in_set);
        if new_color <> old_color then
          begin
            stable:= false;
            SetColor(state, new_color)
          end
      end
  end;

PROCEDURE FixAmberStates;

  var size, state:  States;
      count:  Integer;

  PROCEDURE SplitAmberStates(state:  States; var count:  Integer);

  PROCEDURE MakeSibling(var count:  Integer);
    var sibling:  States;
    begin
      sibling:= size+count;
      count:= count+1;
      SetSibling(state, sibling);
      SetColor(state, green);
      SetSplit(state, true);
      SetColor(sibling, red);
      SetSplit(sibling, true)
    end;

  PROCEDURE MakeNewOutputs(state:  States);
    begin
      {writes to database:
       tags a vocal state and its sibling with appropriate
       controllable or uncontrollable version of state output}
```

```
    end;

  PROCEDURE MakeNewTrans(state:  States;in_set:  In_Events;
                         out_set:  Out_Events);
    begin
      {writes to database:
       redistributes incoming transitions
       between state and its sibling}
    end;

  var in_set:  In_Events;
      out_set:  Out_Events;

  begin {SplitAmberStates}
    MakeSibling(count);
    MakeNewOutputs(state);
    InSet(state, in_set);
    OutSet(state, out_set);
    MakeNewTrans(state, in_set, out_set)
  end;

PROCEDURE SwitchOldTrans(state:  States);
  begin
    {writes to database:
     redistributes outgoing transitions between
     state and its sibling}
  end;

  begin {FixAmberStates}
    size:= GetSize;
    count:= 0;
    for state:= 0 to size-1 do
      if GetColor(state) = amber then
        begin
          if not GetSplit(state) then SplitAmberStates(state, count)
          else SwitchOldTrans(state)
        end
  end;

PROCEDURE OCC;

  BEGIN {PROCEDURE OCC}
    Initialize;
    repeat
```

```
      stable:= true;
      RecolorStates; {returns stable = false unless
                      OCC achieved}
      if stable = false then FixAmberStates
    until stable
  END;

END.

PROGRAM PSHC;

  {pseudo-Pascal program for achieving strict-output-control-consistency;
  operates on database defining transition structure of G_lo, to
  create G_lo_new}

  USES POCC;

  PROCEDURE HCC;

  TYPE
    Array_States = array[States] of Integer;

  FUNCTION Agent(state:  States; event:  Events):  Boolean;
    begin
      {returns true if (state,event) is an agent}
    end;

  PROCEDURE GetDedicatedSet(state:  States; var dedicated_set:  Array_States);
    begin
      {collects dedicated set from database}
    end;

  PROCEDURE GetBlockingSet(state:  States; event:  Events;
                          var blocking_set:  Array_States);
    begin
      {collects blocking set from database}
    end;

  FUNCTION Subset(blocking_set, dedicated_set:  Array_States):  Boolean;
    begin
      {returns true if blocking_set is a subset of dedicated_set}
    end;

  PROCEDURE Vocalize(state:  States; event:  Events);
```

```
  begin
    {finds transition (state,event,new_state) in database;
    vocalizes new_state}
  end;

var flag:  Boolean;
    size, state:  States;
    event:  Events;
    dedicated_set, blocking_set:  Array_States;

begin {PROCEDURE HCC}
    size:= GetSize;
    repeat
    flag:= true;
    for state:= 0 to size-1 do
      begin
        for event:= 0 to MaxAlphabetSize do
          ifAgent(state,event) then
            begin
              GetDedicatedSet(state,dedicated_set);
              GetBlockingSet(state,event,blocking_set);
              ifnot Subset(blocking_set,dedicated_set) then
                begin
                  flag:= false; {agent is inadmissible}
                  Vocalize(state,event)
                end
            end
      end
    until flag = true {all agents are admissible}
end;

BEGIN {PROGRAM PSHC}
    OCC;
    HCC;
    OCC
END.
```

5.12 Notes and References

The material of this chapter originates with the theses of H. Zhong [T09, T20] and related publications [J14, C34, C38]. Theorems 5.5.2 and 5.5.3 are due to K.C. Wong [T12], who has also addressed the hierarchical nonblocking problem [T28, J30, J31]. The specific results of Sect. 5.7 are new, but adapted from [T20, J30]; cf. also Pu[T41]. Dual approaches to hierarchical supervisory control, based on state aggregation, have been reported by Schwartz [T25] and Hubbard & Caines [2002]; or on state space decomposition, by Wang [T29] and Leduc [T46].

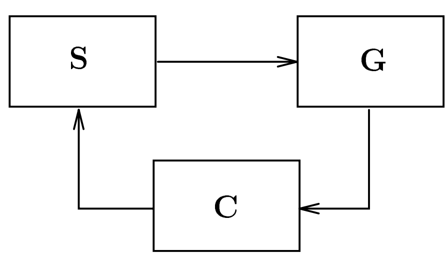
Hierarchy is a long-standing topic in control theory and has been discussed by many authors, notably Mesarovic et al. [1970]. For a perceptive (and classic) essay on the benefits of hierarchical organization the reader is referred to Simon [1967].

Chapter 6

Supervisory Control With Partial Observations

6.1 Natural Projections and Normal Languages

In this chapter we consider the problem of supervisory control under the assumption that only a subset of the event labels generated by the plant can actually be observed by the supervisor. This subset in general need have no particular relation to the subset of controllable events. Our model will lead to a natural definition of observable language, in terms of which the existence of a supervisor for the standard type of control problem considered in previous chapters can be usefully discussed. It will turn out that, while observability is a somewhat difficult property to work with, a stronger property that we call normality provides an effective alternative and often leads to a satisfactory solution of the problem of supervision.



The general setup is shown in the figure. The generator **G** modelling the system to be controlled is of the form considered in previous chapters. The new feature here is the communication channel **C** linking **G** to the supervisor **S**. We consider only the simplest case, where the events visible to **S** form a subset Σ_o of the alphabet Σ associated with **G**. Apart from this feature, **S** operates in the usual way, enabling or disabling events in the controllable subset Σ_c of Σ. No particular relation is postulated to hold between Σ_c and Σ_o:

in particular, $\mathbf{S}$ can potentially disable controllable events that are not observable, namely $\Sigma_c - \Sigma_o$ need not be empty. To model the channel $\mathbf{C}$ we bring in the *natural projection*

$$P : \Sigma^* \rightarrow \Sigma_o^*$$

defined inductively according to

$$\begin{aligned}
P(\epsilon) &= \epsilon \\
P(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \Sigma_o \\ \epsilon & \text{otherwise} \end{cases} \\
P(s\sigma) &= P(s)P(\sigma) \quad \text{for } s \in \Sigma^*, \sigma \in \Sigma
\end{aligned}$$

Thus the effect of P on a string s is just to erase from s the events that do not belong to Σ_o, leaving the order of Σ_o-events in s unchanged. If $s_1, s_2 \in \Sigma^*$ then $P(s_1 s_2) = P(s_1)P(s_2)$, namely P is *catenative.* For $s, s' \in \Sigma^*$ define

$$s \equiv s' \pmod{\ker P} \quad \text{if} \quad Ps = Ps'$$

Because P is catenative, ker P is a right congruence on Σ^* (in general with infinite index).

In *TCT*, P is implemented by **project**. Let $\mathbf{E}$ be a DES over Σ, and **NULL** be a list of the events $\sigma \in \Sigma$ such that $P\sigma = \epsilon$ (i.e., $\sigma \in \Sigma - \Sigma_o$). Then

$$\mathbf{PE} := \mathbf{project}(\mathbf{E}, \mathbf{NULL})$$

is a (minimal-state) DES with

$$L_m(\mathbf{PE}) = PL_m(\mathbf{E}), \qquad L(\mathbf{PE}) = PL(\mathbf{E})$$

Because **project** uses the subset construction (Sect. 2.5) to ensure that **PE** is deterministic, this procedure requires, in the worst case, computational effort (both time and computer memory) that is exponential in the state size of $\mathbf{E}$.

Denote by

$$P^{-1} : Pwr(\Sigma_o^*) \rightarrow Pwr(\Sigma^*)$$

the usual inverse image function of P. In *TCT* P^{-1} is implemented by **selfloop**. Thus let **GO** be a DES over Σ_o, and let

$$\mathbf{PINVGO} := \mathbf{selfloop}(\mathbf{GO}, \mathbf{NULL})$$

Then

$$\begin{aligned}
L_m(\mathbf{PINVGO}) &= P^{-1}L_m(\mathbf{GO}) \\
L(\mathbf{PINVGO}) &= P^{-1}L(\mathbf{GO})
\end{aligned}$$

Associated with any projection P is a natural property of languages defined as follows. Let

$$N \subseteq M \subseteq \Sigma^*$$

Define N to be (M, P)-*normal* if

$$N = M \cap P^{-1}(PN)$$

Note that in this equality the inclusion $\subseteq$ is automatic, while the reverse inclusion is not. Thus N is (M, P)-normal if and only if it can be recovered from its projection along with a knowledge of the structure of M. Equivalently, from $s \in N$, $s' \in M$ and $P(s') = P(s)$ one may infer that $s' \in N$. An (M, P)-normal language N is the largest sublanguage $\hat{N}$ of M with the property that $P\hat{N} = PN$. It is easily seen that both M and the empty language $\emptyset$ are (M, P)-normal. In fact, if $K \subseteq \Sigma_o^*$ is arbitrary, then the language

$$N := M \cap P^{-1}K$$

is always (M, P)-normal.

Exercise 6.1.1: Let $[s]$ denote the cell of ker P containing $s \in \Sigma^*$. If $N \subseteq \Sigma^*$ show that

$$P^{-1}(PN) = \cup\{[s] | s \in N\}$$

Then show that N is (M, P)-normal iff

$$N = \cup\{[s] \cap M | s \in N\}$$

Illustrate with a sketch of Σ^* partitioned by ker P. ◇

With P fixed in the discussion, and the language $E \subseteq \Sigma^*$ arbitrary, bring in the family of languages

$$\mathcal{N}(E; M) = \{N \subseteq E | N \text{ is } (M, P)\text{-normal}\},$$

the class of (M, P)-normal sublanguages of E. Then $\mathcal{N}(E; M)$ is nonempty ($\emptyset$ belongs), and enjoys the following algebraic closure property.

Proposition 6.1.1

The class of languages $\mathcal{N}(E; M)$ is a complete sublattice (with respect to sublanguage inclusion) of the lattice of sublanguages of E. In particular, the intersection and union of (M, P)-normal sublanguages are normal. □

From Proposition 6.1.1 it follows by the usual argument that the language

$$\sup \mathcal{N}(E; M)$$

exists in $\mathcal{N}(E;M)$: that is, any language E contains a unique supremal (M,P)-normal sublanguage, the 'optimal' (M,P)-normal approximation to E from below.

Exercise 6.1.2: With $[s]$ as in Exercise 6.1.1, show that

$$\sup \mathcal{N}(E;M) = \cup\{[s]\cap M \,|\, [s]\cap M \subseteq E\}$$

Illustrate with a sketch. $\diamond$

In supervisory control the two most important classes of (M,P)-normal languages result from setting $M = L_m(\mathbf{G})$ or $M = L(\mathbf{G})$ respectively. A simple relation between them is the following.

Proposition 6.1.2

Assume that $\mathbf{G}$ is trim (in particular $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$), $N \subseteq L_m(\mathbf{G})$ is $(L_m(\mathbf{G}),P)$-normal, and the languages $L_m(\mathbf{G})$, $P^{-1}PN$ are nonconflicting. Then $\bar{N}$ is $(L(\mathbf{G}),P)$-normal. $\square$

Again in connection with supervisory control the following result will find application later.

Proposition 6.1.3

Let $E \subseteq L_m(\mathbf{G})$. The class of languages

$$\bar{\mathcal{N}}(E;L(\mathbf{G})) = \{N \subseteq E \,|\, \bar{N} \quad \text{is } (L(\mathbf{G}),P)\text{-normal}\}$$

is nonempty and closed under arbitrary unions. $\square$

As before, from Proposition 6.1.3 we infer that the language

$$\sup \bar{\mathcal{N}}(E;L(\mathbf{G}))$$

exists in $\bar{\mathcal{N}}(E;L(\mathbf{G}))$.

Examples show that in general the supremal $(L(\mathbf{G}),P)$-normal sublanguage of a closed language need not be closed. However, we have the following important result, which states that if $C \subseteq L(\mathbf{G})$ is closed, N is the supremal $(L(\mathbf{G}),P)$-normal sublanguage of C, and B is the supremal closed sublanguage of N, then in fact B is the supremal sublanguage of C whose closure is $(L(\mathbf{G}),P)$-normal.

Proposition 6.1.4 (Lin)

Let $C \subseteq L(\mathbf{G})$ be closed. Then

$$\sup \bar{\mathcal{N}}(C; L(\mathbf{G})) = \sup \mathcal{F}(\sup \mathcal{N}(C; L(\mathbf{G})))$$

Proof

In the proof write 'normal' for '$(L(\mathbf{G}), P)$-normal', $N := \sup \mathcal{N}(C; L(\mathbf{G}))$, $B := \sup \mathcal{F}(N)$, $\hat{B} := L(\mathbf{G}) \cap P^{-1}(PB)$. Clearly $\hat{B}$ is closed and normal. Since $B \subseteq N$ and N is normal,

$$\hat{B} \subseteq L(\mathbf{G}) \cap P^{-1}(PN) = N$$

Therefore $\hat{B} \subseteq B$. But automatically $\hat{B} \supseteq B$ (since $B \subseteq L(\mathbf{G})$), so $\hat{B} = B$, i.e. B is normal. Let $D := \sup \bar{\mathcal{N}}(C; L(\mathbf{G}))$. Now $B \subseteq C$ and $\bar{B} = B$ is normal, so $B \subseteq D$. Also $\bar{D} \subseteq \bar{C} = C$ and $\bar{D}$ normal imply $\bar{D} \subseteq N$, and then $\bar{D}$ closed implies that $\bar{D} \subseteq B$, so $D \subseteq B$. That is, $D = B$, as claimed. □

For $\sup \mathcal{N}(E; M)$ with $E \subseteq M$ but E, M otherwise arbitrary, we have the following explicit description.

Proposition 6.1.5 (Lin-Brandt formula)

Let $E \subseteq M$. Then

$$\sup \mathcal{N}(E; M) = E - P^{-1}P(M - E)$$

Proof

In the proof write 'normal' for '(M, P)-normal', $S := P^{-1}P(M - E)$ and $N := E - S$. To see that N is normal, suppose

$$u \in M, \quad Pu \in PN$$

For some $v \in N$, $Pu = Pv$. We claim $u \notin S$: otherwise, there exists $t \in M - E$ with $Pu = Pt$, so $Pv = Pt$, i.e. $v \in S$, a contradiction. We also claim that $u \in E$: otherwise $u \in M - E$ and therefore $u \in S$, a contradiction. Thus $u \in E - S = N$, namely

$$M \cap P^{-1}PN \subseteq N$$

hence N is normal. Now let $K \subseteq E$ be normal. We claim that $K \subseteq N$: otherwise there is $s \in K$ (so $s \in E$) with $s \in S$, and so there is $t \in M - E$ with $Ps = Pt$, i.e. $Pt \in PK$. But $t \in M$ and $t \in P^{-1}PK$, so by normality $t \in K$, a contradiction to $K \subseteq E$. □

An implementation of the Lin-Brandt formula is available in *TCT*:

$$\mathbf{N} = \mathbf{supnorm}(\mathbf{E}, \mathbf{M}, \mathbf{NULL})$$

Here **E**, **M** are representative DES for E and M, with E arbitrary, and **NULL** is the list of (unobservable) events nulled by P. Then **N** represents $\sup \mathcal{N}(E \cap M; M)$; thus the user need not arrange in advance that $E \subseteq M$. Like **project**, **supnorm** is computationally expensive.

Exercise 6.1.3: Illustrate the Lin-Brandt formula with a sketch showing Σ^* partitioned by $\ker P$, along with sublanguages $E \subseteq M \subseteq \Sigma^*$. In light of Exercises 6.1.1, 6.1.2, the formula should now be 'obvious'; clearly it is valid for sets and functions in general. ◇

As the last topic of this section we introduce the following related property that is sometimes useful. Let $R \subseteq \Sigma^*$. Say that R is $(L(\mathbf{G}), P)$-*paranormal* if

$$\bar{R}(\Sigma - \Sigma_o) \cap L(\mathbf{G}) \subseteq \bar{R}$$

Thus R is $(L(\mathbf{G}), P)$-paranormal if the occurrence of unobservable events never results in exit from the closure of R. By analogy with controllability it is clear, for instance, that the class of $(L(\mathbf{G}), P)$-paranormal sublanguages of an arbitrary sublanguage of Σ^* is nonempty, closed under union (but not necessarily intersection), and contains a (unique) supremal element.

Proposition 6.1.6

If the closure of R is $(L(\mathbf{G}), P)$-normal, then R is $(L(\mathbf{G}), P)$-paranormal. □

The converse of Proposition 6.1.6 is false: an $(L(\mathbf{G}), P)$-paranormal sublanguage of $L(\mathbf{G})$, closed or not, need not be $(L(\mathbf{G}), P)$-normal. However, the result can be useful in showing that a given closed language R is not $(L(\mathbf{G}), P)$-normal, by showing that an unobservable event may cause escape from R.

To illustrate these ideas we consider three examples.

Example 6.1.1

Let $\Sigma = \{\alpha, \beta\}$, $\Sigma_o = \{\alpha\}$, $L(\mathbf{G}) = \{\epsilon, \alpha, \beta\}$, $C = \{\epsilon, \alpha\}$. Then C is closed and $PC = C$. However

$$L(\mathbf{G}) \cap P^{-1}(PC) = \{\epsilon, \alpha, \beta\} = L(\mathbf{G}) \subsetneqq C$$

and C is not $(L(\mathbf{G}), P)$-normal. C is not $(L(\mathbf{G}), P)$-paranormal either:

$$\bar{C}(\Sigma - \Sigma_o) \cap L(\mathbf{G}) = C\beta \cap L(\mathbf{G}) = \{\beta, \alpha\beta\} \cap \{\epsilon, \alpha, \beta\} = \{\beta\} \not\subseteq C;$$

namely the unobservable event β catenated with the string $\epsilon \in C$ results in escape from C. On the other hand, for the sublanguage $A := \{\alpha\}$,

$$L(\mathbf{G}) \cap P^{-1}P\alpha = L(\mathbf{G}) \cap \beta^*\alpha\beta^* = \{\alpha\} = A$$

so that A is $(L(\mathbf{G}), P)$-normal, and therefore $A = \sup \mathcal{N}(C; L(\mathbf{G}))$. It can be checked that $\sup \mathcal{N}(C; L(\mathbf{G}))$ is correctly calculated by the Lin-Brandt formula (Proposition 6.1.5). Note also that $\bar{A} = C$, showing that the supremal $(P, L(\mathbf{G}))$-normal sublanguage of a closed language need not be closed; here, in fact, $\sup \bar{\mathcal{N}}(C; L(\mathbf{G})) = \emptyset$, in agreement with Proposition 6.1.4.

Now let $B := \{\alpha, \beta\}$. Whereas B is $(L(\mathbf{G}), P)$-paranormal, we have

$$L(\mathbf{G}) \cap P^{-1}(PB) = L(\mathbf{G}) \cap P^{-1}\{\alpha, \epsilon\} = L(\mathbf{G}) \supsetneqq B$$

so B is not $(P, L(\mathbf{G}))$-normal.

Example 6.1.2

As another example let $\Sigma = \{\alpha, \beta, \gamma\}$, $\Sigma_o = \{\gamma\}$,

$$\begin{aligned} L(\mathbf{G}) &= \{\epsilon, \alpha, \alpha\gamma, \alpha\gamma\gamma, \beta, \beta\gamma, \beta\gamma\gamma\} = \overline{(\alpha + \beta)\gamma^2} \\ C &= \{\epsilon, \alpha, \alpha\gamma, \beta, \beta\gamma, \beta\gamma\gamma\} = \overline{(\alpha + \beta\gamma)\gamma} \end{aligned}$$

Then

$$L(\mathbf{G}) \cap P^{-1}P(\mathbf{C}) = L(\mathbf{G}) \supsetneqq C$$

so C is not $(L(\mathbf{G}), P)$-normal; in fact Lin-Brandt yields

$$\sup \mathcal{N}(C; L(\mathbf{G})) = \{\epsilon, \alpha, \beta, \alpha\gamma, \beta\gamma\} = \overline{(\alpha + \beta)\gamma}$$

so in this case $\sup \bar{\mathcal{N}}(C; L(\mathbf{G}))$ and $\sup \mathcal{N}(C; L(\mathbf{G}))$ coincide. On the other hand C is $(L(\mathbf{G}), P)$-paranormal, since the occurrence of unobservable events α or β does preserve membership in C.

Example 6.1.3

Let $\Sigma = \{\alpha, \beta\}$, $\Sigma_o = \{\alpha\}$, $L(\mathbf{G}) = \{\epsilon, \alpha, \beta, \beta\alpha\}$, $A = \{\beta\alpha\}$. Then A is $(L(\mathbf{G}), P)$-paranormal:

$$\bar{A}(\Sigma - \Sigma_0) \cap L(\mathbf{G}) = \{\epsilon, \beta, \beta\alpha\}\beta \cap L(\mathbf{G}) = \{\beta\} \subseteq \bar{A}$$

However, the closure $\bar{A}$ is not $(L(\mathbf{G}), P)$-normal, because

$$L(\mathbf{G}) \cap P^{-1}(P\bar{A}) = \{\epsilon, \alpha, \beta, \beta\alpha\} \supsetneqq \bar{A}$$

$\diamond$

Exercise 6.1.4: With $\Sigma_o \subseteq \Sigma$, let $P : \Sigma^* \to \Sigma_o^*$ be the natural projection, and let $A \subseteq \Sigma^*$, $B \subseteq \Sigma_o^*$. Show that

$$\overline{PA} = P\bar{A}, \quad \overline{P^{-1}B} = P^{-1}\bar{B}$$

Exercise 6.1.5: Supply proofs of Propositions 6.1.1, 6.1.2, 6.1.3 and 6.1.6.

Exercise 6.1.6: Show by example that $\bar{\mathcal{N}}(E; L(\mathbf{G}))$ is not in general closed under intersection.

Exercise 6.1.7: Does there exist a language A that is normal but not paranormal? If so, what can be said about $\bar{A}$?

Exercise 6.1.8: Show that, with $E \subseteq M$,

$$P \sup \mathcal{N}(E; M) = PE - P(M - E)$$

Exercise 6.1.9: With $L \subseteq \Sigma^*$ let

$$\mathcal{F}(L) := \{H \subseteq L | H = \bar{H}\}$$

Show that

$$\sup \mathcal{F}(L) = L - (\Sigma^* - L)\Sigma^*$$

In particular, L is closed iff

$$L \cap (\Sigma^* - L)\Sigma^* = \varnothing$$

Exercise 6.1.10: For $i = 1, 2$ let $L_i \subseteq \Sigma_i^*$, $\Sigma_0 := \Sigma_1 \cap \Sigma_2$, and $P_0 : (\Sigma_1 \cup \Sigma_2)^* \longrightarrow \Sigma_0^*$ be the natural projection. Call P_0 an L_i – *observer* if

$$(\forall t \in P_0 L_i)(\forall s \in \bar{L}_i) P_0 s \le t \;\Rightarrow\; (\exists u \in \Sigma_i^*) su \in L_i \;\&\; P_0(su) = t$$

In other words, whenever $P_0 s$ can be extended to a string $t \in P_0 L_i$, the underlying string $s \in \bar{L}_i$ can be extended to a string $su \in L_i$ with the same projection. Assume that P_0 is an L_i-observer ($i = 1, 2$) and that $P_0 L_1, P_0 L_2$ are nonconflicting. Show that

$$\overline{L_1 \parallel L_2} = \bar{L}_1 \parallel \bar{L}_2$$

Specialize to each of the cases $\Sigma_1 = \Sigma_2$ and $\Sigma_0 = \emptyset$. **Hint**: Make use of Exercises 3.3.5 and 6.1.4.

6.2 Observable Languages

In order to define observability, it is convenient to associate with each string s two distinguished subsets of events, as follows. Let $K \subseteq \Sigma^*$ be arbitrary. For $s \in \Sigma^*$ define the *active* event set

$$A_K(s) = \begin{cases} \{\sigma \in \Sigma | s\sigma \in \bar{K}\}, & s \in \bar{K} \\ \emptyset & \text{otherwise} \end{cases}$$

and the *inactive* event set

$$IA_K(s) = \begin{cases} \{\sigma \in \Sigma | s\sigma \in L(\mathbf{G}) - \bar{K}\}, & s \in \bar{K} \\ \emptyset & \text{otherwise} \end{cases}$$

Thus $A_K(s)$ consists of just those events whose occurrence following a prefix s of K preserves the prefix property; while events in $IA_K(s)$ could occur in $\mathbf{G}$, but destroy the prefix property. Next we define the binary relation K-*active* on Σ^*, denoted by act_K, according to: $(s, s') \in \mathrm{act}_K$ iff

(i) $A_K(s) \cap IA_K(s') = \emptyset = A_K(s') \cap IA_K(s)$, and

(ii) $s \in \bar{K} \cap L_m(\mathbf{G})$ & $s' \in \bar{K} \cap L_m(\mathbf{G}) \Rightarrow (s \in K \Leftrightarrow s' \in K)$

Equivalently, for all $s, s' \in \Sigma^*$, $(s, s') \in \mathrm{act}_K$ if and only if

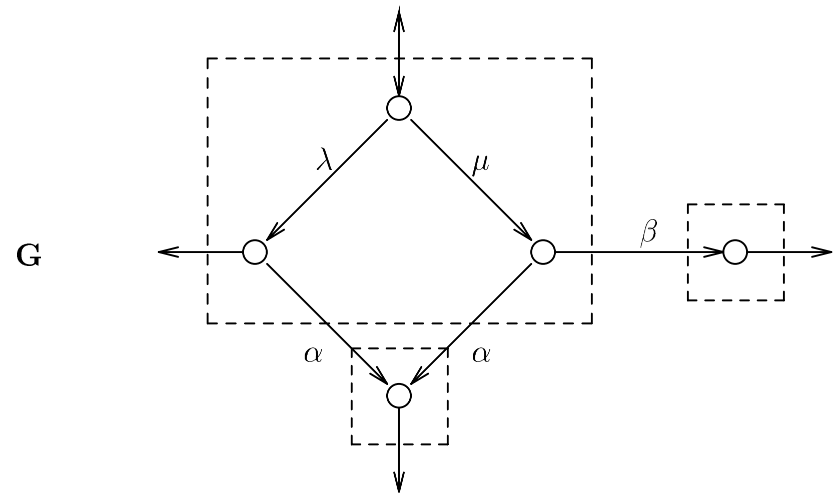
(i′) $(\forall \sigma) s\sigma \in \bar{K}$ & $s' \in \bar{K}$ & $s'\sigma \in L(\mathbf{G}) \Rightarrow s'\sigma \in \bar{K}$, and

(ii′) $s \in K \cap L_m(\mathbf{G})$ & $s' \in \bar{K} \cap L_m(\mathbf{G}) \Rightarrow s' \in K$, and

(iii′) conditions (i′) and (ii′) hold with s and s' interchanged.

Note that a pair $(s, s') \in \mathrm{act}_K$ if either of s or s' does not belong to $\bar{K}$, because if $s \notin \bar{K}$ then $A_K(s) = IA_K(s) = \emptyset$. Otherwise (the nontrivial case) membership of a pair of strings (s, s') in act_K means, roughly, that prefixes s and s' of K have identical one-step continuations with respect to membership in $\bar{K}$; and, if each is in $L_m(\mathbf{G})$ and one actually belongs to K, then so does the other. It should be noted that act_K is a tolerance relation on Σ^*, namely it is reflexive and symmetric but need not be transitive. Notice finally that if K is closed, or $L_m(\mathbf{G})$-closed, then conditions (ii) and (ii′) are satisfied automatically and may be dropped.

Example 6.2.1



Let $\Sigma = \{\alpha, \beta, \lambda, \mu\}$, $\Sigma_o = \{\alpha, \beta\}$,

$$L(\mathbf{G}) = \overline{\lambda\alpha + \mu(\alpha + \beta)}, \quad K = \overline{\lambda\alpha + \mu\beta}$$

Then, for instance

$$\begin{aligned} A_K(\epsilon) &= \{\lambda, \mu\}, & IA_K(\epsilon) &= \emptyset \\ A_K(\lambda) &= \{\alpha\}, & IA_K(\lambda) &= \emptyset \\ A_K(\mu) &= \{\beta\}, & IA_K(\mu) &= \{\alpha\} \end{aligned}$$

So the string pairs $(\epsilon, \lambda), (\epsilon, \mu) \in \mathrm{act}_K$, but $(\lambda, \mu) \notin \mathrm{act}_K$. Suppose

$$L_m(\mathbf{G}) = \epsilon + \lambda(\epsilon + \alpha) + \mu(\alpha + \beta), \quad J = \epsilon + \lambda\alpha + \mu\beta$$

Then $\bar{J} = K$ but J is not even $L_m(\mathbf{G})$-closed, since

$$\lambda \in \bar{J} \cap L_m(\mathbf{G}), \quad \lambda \notin J.$$

Now $\epsilon \in J \cap L_m(\mathbf{G})$ and $\lambda \in \bar{J} \cap L_m(\mathbf{G})$, but $\lambda \notin J$, so in this case $(\epsilon, \lambda) \notin \mathrm{act}_J$. $\diamond$

We can now frame the definition desired. With $P : \Sigma^* \to \Sigma_o^*$ as before, say that a language $K \subseteq \Sigma^*$ is $(\mathbf{G}, P)$-*observable*, or simply *observable*, if

$$\ker P \leq \mathrm{act}_K$$

The definition states that the equivalence relation ker P refines act_K, namely that P preserves at least the information required to decide consistently the question of continuing membership in $\bar{K}$ after the hypothetical occurrence of an event σ, as well as to decide membership in K when membership in $\bar{K} \cap L_m(\mathbf{G})$ happens to be known. If two strings 'look the same' (have the same projections), then a decision rule that applies to one can be used for the other. By contrast, if K is not $(\mathbf{G}, P)$-observable, then an event (observable or not) may have different consequences for look-alike strings. For example, in case $K \subseteq L(\mathbf{G})$ is closed, there would exist $s,\ s' \in K$ with $Ps = Ps'$, and $\sigma \in \Sigma$, such that $s\sigma \in K$ but $s'\sigma \in L(\mathbf{G}) - K$. Nevertheless, observability does not preclude the existence of $s \in K$ and $\sigma \in \Sigma - \Sigma_0$ (hence $Ps = P(s\sigma)$) such that $s\sigma \in L(\mathbf{G}) - K$: see the remark below Example 6.2.2.

In the transition graph for Example 6.2.1, the nodes are grouped to display ker P. Since

$$P\epsilon = P\lambda = P\mu$$

neither J nor K is observable.

Our next result states an important relationship between observability and normality.

Proposition 6.2.1

Assume that either $K \subseteq L(\mathbf{G})$ and K is closed, or $K \subseteq L_m(\mathbf{G})$ and K is $L_m(\mathbf{G})$-closed. If $\bar{K}$ is $(L(\mathbf{G}), P)$-normal then K is $(\mathbf{G}, P)$-observable. $\square$

The converse statement is false.

Example 6.2.2

Let $\Sigma = \{\alpha, \beta\}$, $\Sigma_o = \{\beta\}$ and

$$L(\mathbf{G}) = \{\epsilon, \alpha, \beta, \alpha\beta\} = \overline{(\epsilon + \alpha)\beta}, \quad K = \{\epsilon, \beta\}$$

Thus K is closed. Also

$$\begin{aligned} A_K(\epsilon) &= \{\beta\}, \qquad IA_K(\epsilon) = \{\alpha\} \\ A_K(\beta) &= \emptyset, \qquad IA_K(\beta) = \emptyset \end{aligned}$$

and so K is $(\mathbf{G}, P)$-observable. However

$$L(\mathbf{G}) \cap P^{-1}(PK) = \{\epsilon, \alpha, \beta, \alpha\beta\} = L(\mathbf{G}) \supsetneqq K$$

and therefore K is not $(L(\mathbf{G}), P)$-normal. $\diamond$

From the viewpoint of an observer, the essential difference between a $(\mathbf{G}, P)$-observable language and a closed $(L(\mathbf{G}), P)$-normal language is that with a normal language one can always tell, by watching the projection Ps of an evolving string s, if and when the string exits from the language; but with an observable language in general this is not the case. For instance in the foregoing example, the occurrence of α would represent an unobservable exit from K. As we shall see in Sect. 6.5 (Propositions 6.5.1, 6.5.2) this difference between observability and normality has the following implication for supervisory control. Suppose the (closed, controllable) language to be synthesized is normal. Then no unobservable event will cause a string to exit, in particular no controllable unobservable event. Thus no such event will ever be disabled by the supervisor.

Exercise 6.2.1: Show that the main condition that K be $(\mathbf{G}, P)$-observable (from (i$'$) above) can be written

$$(\forall s', \sigma) s' \in \bar{K} \quad \& \quad s'\sigma \in L(\mathbf{G}) \quad \& \quad [(\exists s) s\sigma \in \bar{K} \quad \& \quad Ps = Ps'] \Rightarrow s'\sigma \in \bar{K}$$

Roughly, the test for $s'\sigma \in \bar{K}$ is the existence of a look-alike s such that $s\sigma \in \bar{K}$. **Hint:** Use the predicate logic identity

$$(\forall x, y) P(y) \ \& \ Q(x, y) \Rightarrow R(y) \ \equiv \ (\forall y) P(y) \ \& \ [(\exists x) Q(x, y)] \Rightarrow R(y)$$

Exercise 6.2.2: Prove Proposition 6.2.1.

Exercise 6.2.3: Formalize (i.e. provide a rigorous version of) the statement following Example 6.2.2, and provide a proof. $\diamond$

To conclude this section, we provide a partial converse to Proposition 6.2.1.

Proposition 6.2.2

Let $K \subseteq L_m(\mathbf{G})$ be $\mathbf{G}$-controllable and $(\mathbf{G}, P)$-observable. Assume that $P\sigma = \sigma$ for all $\sigma \in \Sigma_c$. Then K is $(L_m(\mathbf{G}), P)$-normal and $\bar{K}$ is $(L(\mathbf{G}), P)$-normal. $\square$

Exercise 6.2.4: Prove Proposition 6.2.2, under the weaker assumption that $P\sigma = \sigma$ for all $\sigma \in \Sigma_c$ such that σ is actually disabled by a supervisor synthesizing K.

6.3 Feasible Supervisory Control

We now introduce the concept of supervisory control, proceeding just as in Sect. 3.4, except for taking into account the constraint that control must be based purely on the result of observing the strings generated by $\mathbf{G}$ through the channel $\mathbf{C}$, namely on information transmitted by $P : \Sigma^* \to \Sigma_o^*$. With

$$\mathbf{G} = (\_, \Sigma, \_, \_, \_), \quad \Sigma = \Sigma_c \cup \Sigma_u$$

as usual, define as before the set of all control patterns

$$\Gamma = \{\gamma \in Pwr(\Sigma) | \gamma \supseteq \Sigma_u\}$$

A *feasible supervisory control* for $\mathbf{G}$ is any map $V : L(\mathbf{G}) \to \Gamma$ such that

$$\ker(P|L(\mathbf{G})) \leq \ker V$$

Here $P|L(\mathbf{G})$ denotes the restriction of P to $L(\mathbf{G})$. As before we write $V/\mathbf{G}$ to suggest '$\mathbf{G}$ under the supervision of V'. The *closed behavior* $L(V/\mathbf{G})$ and *marked behavior* $L_m(V/\mathbf{G})$ are defined exactly as in Sect. 3.4, as is the property that V is nonblocking for $\mathbf{G}$. Our first main result is the expected generalization of Theorem 3.4.1.

Theorem 6.3.1

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. There exists a nonblocking feasible supervisory control V for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$ if and only if

(i) K is controllable with respect to $\mathbf{G}$, and

(ii) K is observable with respect to $(\mathbf{G}, P)$, and

(iii) K is $L_m(\mathbf{G})$-closed.

Proof

(If) The proof follows the same lines as that of Theorem 3.4.1, but extended to ensure the feasibility property. First bring in the function

$$Q : \bar{K} \to Pwr(\Sigma)$$

according to

$$Q(s) := \{\sigma \in \Sigma | (\forall s' \in \bar{K}) Ps' = Ps \; \& \; s'\sigma \in L(\mathbf{G}) \Rightarrow s'\sigma \in \bar{K}\}$$

Now define $V : L(\mathbf{G}) \to \Gamma$ as follows. If $s \in \bar{K}$ then

$$V(s) := \Sigma_u \cup (\Sigma_c \cap Q(s))$$

while if $s \in L(\mathbf{G}) - \bar{K}$ and $Ps = Pv$ for some $v \in \bar{K}$, let

$$V(s) := V(v)$$

$V(s)$ is well-defined in the latter case, for if also $Ps = Pw$ with $w \in \bar{K}$ then by $\ker P \leq \mathrm{act}_K$ we conclude that $(v, w) \in \mathrm{act}_K$, from which it easily follows that $Q(v) = Q(w)$, so $V(v) = V(w)$. Finally if $s \in L(\mathbf{G}) - \bar{K}$ and there is no $v \in \bar{K}$ such that $Ps = Pv$ then let

$$V(s) := \Sigma_u$$

Next we show that V is feasible, namely $\ker(P|L(\mathbf{G})) \le \ker V$. Let $s_1, s_2 \in L(\mathbf{G})$, $Ps_1 = Ps_2$. We consider the three cases (i) $s_1, s_2 \in \bar{K}$, (ii) $s_1 \in \bar{K}$, $s_2 \in L(\mathbf{G}) - \bar{K}$, and (iii) $s_1, s_2 \in L(\mathbf{G}) - \bar{K}$. As to (i) it is easily checked that $Q(s_1) = Q(s_2)$, so $V(s_1) = V(s_2)$, namely $(s_1, s_2) \in \ker V$ as claimed. For (ii), by definition $V(s_2) = V(s_1)$, so $(s_1, s_2) \in \ker V$. In case (iii), if $Ps_1 = Pv$ for some $v \in \bar{K}$, then by definition $V(s_1) = V(v)$, and $Ps_2 = Ps_1$ implies similarly $V(s_2) = V(v)$; while if $Ps_1 = Pv$ for no $v \in \bar{K}$ then

$$V(s_1) = V(s_2) = \Sigma_u \ ;$$

so in either subcase $(s_1, s_2) \in \ker V$, as required.

To complete the proof it may be shown by induction on length of strings that

$$L(V/\mathbf{G}) = \bar{K}$$

and then directly that $L_m(V/\mathbf{G}) = K$. As the argument is similar to the proof of Theorem 3.4.1, we just provide the inductive step. Thus suppose $s \in L(V/\mathbf{G})$, $s \in \bar{K}$ and $s\sigma \in L(V/\mathbf{G})$, i.e. $\sigma \in V(s)$ and $s\sigma \in L(\mathbf{G})$. If $\sigma \in \Sigma_u$ then $s\sigma \in \bar{K}$ by controllability; while if $\sigma \in \Sigma_c \cap Q(s)$ then $s\sigma \in \bar{K}$ by definition of Q. Conversely suppose $s\sigma \in \bar{K}$. If $\sigma \in \Sigma_u$ then clearly $\sigma \in V(s)$ so $s\sigma \in L(V/\mathbf{G})$. Suppose $\sigma \in \Sigma_c$. We claim $\sigma \in Q(s)$: for if $s' \in \bar{K}$ with $Ps' = Ps$ then by observability $(s, s') \in \mathrm{act}_K$, and then $s'\sigma \in L(\mathbf{G})$ implies $s'\sigma \in \bar{K}$, the required result. Thus it follows that $\sigma \in V(s)$, and as $s\sigma \in L(\mathbf{G})$ we again conclude that $s\sigma \in L(V/\mathbf{G})$. This shows that $L(V/\mathbf{G}) = \bar{K}$, as claimed.

(Only if) Let V be a nonblocking feasible supervisory control for $\mathbf{G}$ with $L_m(V/\mathbf{G}) = K$. As the proof that K is controllable and $L_m(\mathbf{G})$-closed is unchanged from the proof of Theorem 3.4.1, it suffices to show that K is observable. So let $(s, s') \in \ker P$, $s\sigma \in \bar{K}$, $s' \in \bar{K}$ and $s'\sigma \in L(\mathbf{G})$. Since $s, s' \in L(\mathbf{G})$ and $\ker(P|L(\mathbf{G})) \le \ker V$, there follows $V(s) = V(s')$. Therefore $s\sigma \in \bar{K}$ implies in turn $\sigma \in V(s)$, $\sigma \in V(s')$, and $s'\sigma \in \bar{K}$. This verifies the observability condition (i′) of Sect. 6.2; condition (ii′) is automatic since K is $L_m(\mathbf{G})$-closed; while condition (iii′) is true by symmetry of the argument. □

Corollary 6.3.1

Let $K \subseteq L(\mathbf{G})$ be nonempty and closed. There exists a feasible supervisory control V for $\mathbf{G}$ such that $L(V/\mathbf{G}) = K$ if and only if K is controllable with respect to $\mathbf{G}$ and observable with respect to $(\mathbf{G}, P)$. □

For brevity we refer to a nonblocking feasible supervisory control (for $\mathbf{G}, P$) as an NFSC. As before we may generalize this idea to incorporate marking as well as control in the supervisory action. Thus if $M \subseteq L_m(\mathbf{G})$ we define a *marking nonblocking feasible supervisory control* for the triple $(M, \mathbf{G}, P)$, or MNFSC, as a map $V : L(\mathbf{G}) \to \Gamma$ as defined above, but now with marked behavior given by

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M$$

However, for this definition to satisfy the intended interpretation of 'marking' we must place a further restriction on M. For instance, in a manufacturing system a string $s \in L_m(\mathbf{G})$ might correspond to 'completion of a finished workpiece', while $s \in M$ might mean 'completion of a batch of finished workpieces'. If a batch consists of 10 workpieces, then we would not want the supervisor to confuse a string s corresponding to 6 batches with a string s' corresponding to 61 workpieces. It is natural, then, to require that s, s' be distinguishable, namely look different when viewed through P. In general terms we require

$$(\forall s, s' \in \Sigma^*) s \in M \ \& \ s' \in \bar{M} \cap L_m(\mathbf{G}) \ \& \ s' \notin M \Rightarrow Ps \neq Ps'$$

or more directly

$$(\forall s, s' \in \Sigma^*) s \in M \ \& \ s' \in \bar{M} \cap L_m(\mathbf{G}) \ \& \ Ps = Ps' \Rightarrow s' \in M$$

or succinctly

$$\ker(P|(\bar{M} \cap L_m(\mathbf{G}))) \leq \{M, \bar{M} \cap L_m(\mathbf{G}) - M\}$$

If this condition is satisfied then we shall say that M is $(\mathbf{G}, P)$-*admissible*. As the counterpart to Theorem 3.4.2 we now have

Theorem 6.3.2

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$, and let K be $(\mathbf{G}, P)$-admissible. There exists an MNFSC V for $(K, \mathbf{G}, P)$ such that

$$L_m(V/\mathbf{G}) = K$$

if and only if K is controllable with respect to $\mathbf{G}$ and observable with respect to $(\mathbf{G}, P)$.

Proof

The proof of sufficiency may be left to the reader (cf. the proof of Theorem 3.4.2). As to necessity, the proof that K is controllable is unchanged from the proof of Theorem 6.3.1. For observability let $s, s' \in L(\mathbf{G})$ with $Ps = Ps'$. The proof of condition (i$'$) of Sect. 6.2 is unchanged, while condition (ii$'$) is just the property that K is $(\mathbf{G}, P)$-admissible.

$\square$

Example 6.3.1: Construction of a feasible supervisor

While not fully implemented in TCT, the following procedure constructs a feasible supervisor, although it cannot guarantee nonblocking. Initially assume that all events are observable, and construct a proper supervisor, say $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ for $\mathbf{G}$ (cf. Sect. 3.6). For instance, $\mathbf{S}$ might be obtained as **supcon(G,E)**. With the natural projection $P : \Sigma^* \to \Sigma_o^*$ given, replace all transitions in $\mathbf{S}$ whose event labels σ are nulled by P ($P\sigma = \varepsilon$) by a 'silent transition', say ν, where $\nu \notin \Sigma$. In effect, this step converts **S** into a nondeterministic transition structure, to which we apply the subset construction (Sect. 2.5) as follows. The state set of our new supervisor will be denoted by Y, with elements y that label subsets of X. Define the initial state subset

$$y_o := \{\xi(x_o, s) \mid s = \nu^k \quad \text{for some } k \geq 0 \quad \& \quad \xi(x_o, s)!\}$$

Choose $\sigma_1 \in \Sigma_o$ and define

$$y_1 := \cup\{\xi(x, \sigma_1 s) \mid x \in y_o, \quad s = \nu^k \ \text{ for some } k \geq 0 \quad \& \quad \xi(x, \sigma_1 s)!\}$$

Define y_2 similarly, from y_o and $\sigma_2 \in \Sigma_o - \{\sigma_1\}$, and repeat until Σ_o is exhausted. The subset obtained at any step is discarded if it is empty or if it appeared previously. This process yields a list of distinct nonempty subsets $y_o, y_1, ..., y_r$, and one-step 'subset' transitions of form (y_o, σ, y_i), $\sigma \in \Sigma_o$, $i \in \{0, 1, ..., r\}$. The procedure is repeated with each of the subsets $y_1, y_2, ...$ and each $\sigma \in \Sigma_o$, until no new subset transitions are obtained (in the worst case this will take on the order of $2^{|X|}$ subset determinations). The result is the projected DES

$$\mathbf{PS} = (Y, \Sigma_o, \eta, y_o, Y_m)$$

where Y is the final subset listing $\{y_o, y_1, ..., \}$, Y_m is the 'marked' sublist such that $y \in Y_m$ iff $x \in y$ for some $x \in X_m$, and $\eta(y, \sigma) = y'$ iff $\xi(x, \sigma) = x'$ for some $x \in y$, $x' \in y'$ $(\sigma \in \Sigma_o)$.

To define the supervisory action of **PS** (over the total alphabet Σ), first introduce the disabling predicate $D(x, \sigma)$ to mean that $\sigma \in \Sigma_c$ and **S** actively disables σ at x (in TCT, σ is listed at x in the **condat** table for **S**). Next introduce a partial function $F : Y \times \Sigma \to \{0, 1\}$ according to:

$$F(y, \sigma) = 0 \quad \text{if} \quad (\exists x \in y)\ D(x, \sigma)$$

i.e. σ is controllable and is disabled at some $x \in y$;

$$F(y, \sigma) = 1 \quad \text{if} \quad \sigma \in \Sigma - \Sigma_o \quad \& \quad (\exists x \in y) \ \ \xi(x, \sigma)! \quad \& \quad [\sigma \in \Sigma_u$$

$$\text{or} \quad [\sigma \in \Sigma_c \quad \& \quad (\forall x' \in y)\neg D(x', \sigma)]]$$

i.e. σ is unobservable and is enabled at some $x \in y$, and is either (1) uncontrollable, or (2) controllable and nowhere disabled in y. Otherwise, $F(y, \sigma)$ is undefined. Finally, modify the transition structure of **PS**, to create **FPS** as follows:

(i) If $F(y, \sigma) = 0$, delete any transition in **PS** of form (y, σ, y'), i.e. declare $\eta(y, \sigma)$ undefined;

(ii) if $F(y, \sigma) = 1$, add the selfloop $\eta(y, \sigma) = y$.

The resulting structure **FPS** will be feasible[1] and controllable. It is not guaranteed to be coreachable, or nonblocking for the plant **G**. But if these properties happen to hold, then **FPS** provides a solution to the problem of feasible supervisory control. ◇

Exercise 6.3.1: Verify that the supervisor constructed above provides the same control action as the supervisory control V used in the proof ('If' part) of Theorem 6.3.1. **Hint:** Assume that **S** represents K.

Exercise 6.3.2: Create an example to show that the feasible supervisor constructed according to Example 6.3.1 may turn out to be blocking.

Example 6.3.2: Application of feasible supervision

The previous construction is illustrated by the following problem of mutual exclusion under partial observation. Consider agents **A1**, **A2** as shown in Fig. 6.3.1.

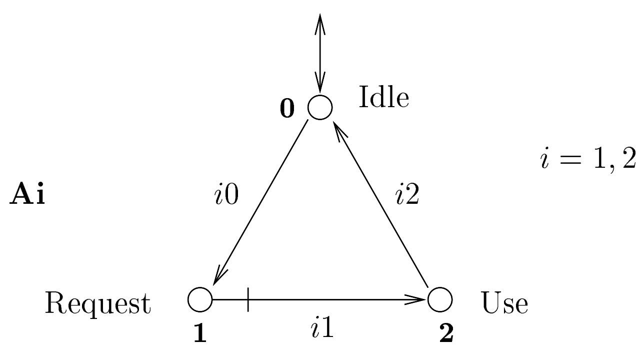


Fig. 6.3.1
Agents subject to mutual exclusion

The state names refer to a single shared resource, so simultaneous occupancy of the state pair (2,2) is prohibited. An additional specification is that resource usage be 'fair' in the sense of 'first-request-first-use', implemented by means of a queue. It is assumed that events 11, 21 (transitions from *Request* to *Use*) are unobservable. To attempt a solution, start by constructing **A = sync(A1,A2)**, then **ASPEC** (left to the reader), and finally the supervisor **ASUPER = supcon (A,ASPEC)**, with the result displayed in Fig. 6.3.2.

[1] In the sense that no state change occurs under an unobservable event.

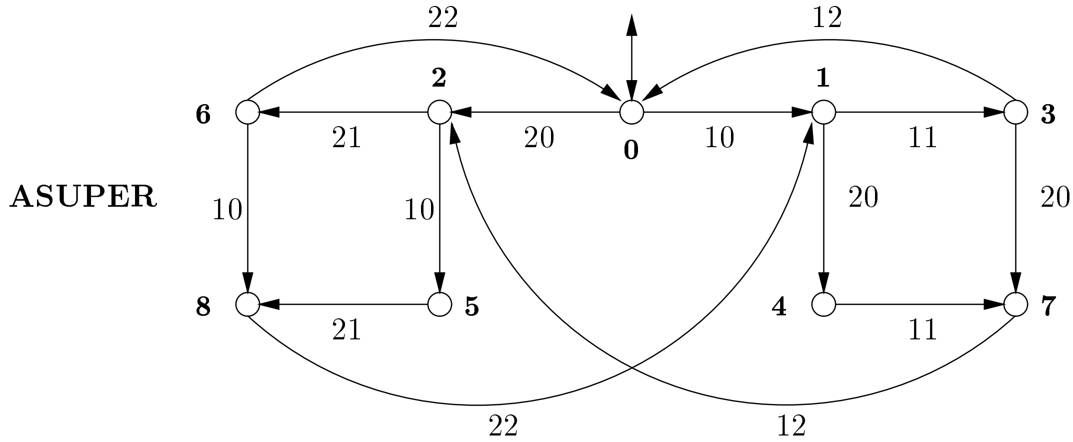


Fig. 6.3.2
Supervisor with full observation

With events 11, 21 unobservable, application of the subset construction to **ASUPER** yields **PASUPER**, with state set

$$y_o = \{0\}, \quad y_1 = \{1,3\}, \quad y_2 = \{2,6\}, \quad y_3 = \{4,7\}, \quad y_4 = \{5,8\}$$

and displayed transition structure (Fig. 6.3.3). Now the table **condat(A,ASUPER)** shows that event 11 is disabled at $x = 5, 8$ while 21 is disabled at $x = 4, 7$. From this we assert

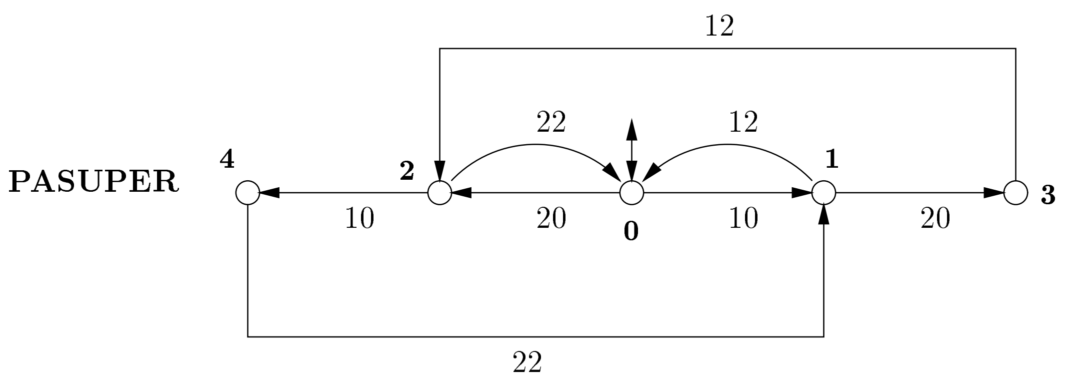


Fig. 6.3.3
Projection of **ASUPER**

$$D(5,11), \quad D(8,11), \quad D(4,21), \quad D(7,21)$$

and obtain

$$F(y_3, 21) = F(y_4, 11) = 0, \quad F(y_1, 11) = F(y_2, 21) = F(y_3, 11) = F(y_4, 21) = 1$$

Since events 11,21 (being unobservable) do not occur in **PASUPER**, the desired feasible supervisor **FPASUPER** is obtained from **PASUPER** by selflooping 11 at y_1, y_3 and 21 at y_2, y_4, with the result displayed in Fig. 6.3.4.

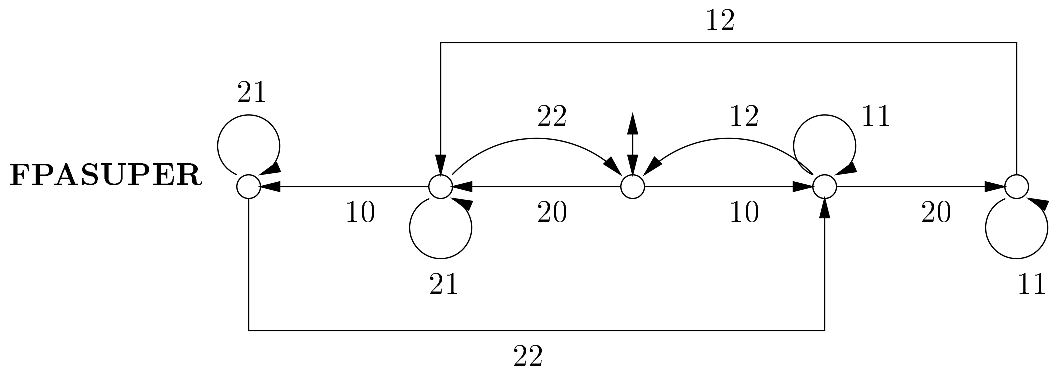


Fig. 6.3.4
Feasible supervisor

It is easily checked that **FPASUPER** and **A** are nonconflicting, so the result is nonblocking for **A**, and in fact the controlled behavior of **A** is identical with the behavior under supervision by **ASUPER**. Intuitively it is clear that observation of events 11 and 21 is irrelevant to control.

6.4 Infimal Closed Observable Sublanguages

Let $\mathbf{G}, P$ be as before, and let $A \subseteq L(\mathbf{G})$. Consider the class of languages

$$\bar{\mathcal{O}}(A) = \{K \supseteq A \mid K \text{ is closed and } (\mathbf{G}, P)\text{-observable}\}$$

We have the following (dual) analog of Theorem 3.5.1.

Theorem 6.4.1

$\bar{\mathcal{O}}(A)$ is nonempty and closed under arbitrary intersections. In particular, $\bar{\mathcal{O}}(A)$ contains a (unique) infimal element [which we denote by $\inf \bar{\mathcal{O}}(A)$].

Proof

Clearly $L(\mathbf{G}) \in \bar{\mathcal{O}}(A)$. Let $K_\beta \in \bar{\mathcal{O}}(A)$ for all β in some index set B, and let

$$K = \cap\{K_\beta | \beta \in B\}$$

Then K is closed. Suppose $(s, s') \in \ker P$, $s\sigma \in K$, $s' \in K$ and $s'\sigma \in L(\mathbf{G})$. We have for each β that $s\sigma \in K_\beta$ and $s' \in K_\beta$, so $s'\sigma \in K_\beta$. Hence $s'\sigma \in K$, and K is observable. In particular

$$\inf \bar{\mathcal{O}}(A) = \cap\{K | K \in \bar{\mathcal{O}}(A)\}$$

□

In general the conclusion of Theorem 6.4.1 fails if the observable languages are not closed, nor does it help to require them to be $L_m(\mathbf{G})$-closed.

Example 6.4.1

Let $\Sigma = \{\alpha, \beta, \gamma, \delta\}$, $\Sigma_o = \{\beta, \gamma, \delta\}$ and

$$L(\mathbf{G}) = \overline{\delta + \alpha\delta(\beta + \gamma)}, \quad L_m(\mathbf{G}) = \{\alpha, \delta, \alpha\delta(\beta + \gamma)\}$$

$$K_1 = \alpha + \delta + \alpha\delta\beta, \quad K_2 = \alpha + \delta + \alpha\delta\gamma$$

The reader may verify that K_1 and K_2 are both $L_m(\mathbf{G})$-closed and observable. Now

$$\bar{K}_1 = \{\epsilon, \alpha, \delta, \alpha\delta, \alpha\delta\beta), \quad \bar{K}_2 = \{\epsilon, \alpha, \delta, \alpha\delta, \alpha\delta\gamma\}$$

$$K_1 \cap K_2 = \{\alpha, \delta\}, \quad \overline{K_1 \cap K_2} = \{\epsilon, \alpha, \delta\}$$

$$\bar{K}_1 \cap \bar{K}_2 = \{\epsilon, \alpha, \delta, \alpha\delta\}$$

$$\bar{K}_1 \cap \bar{K}_2 - \overline{K_1 \cap K_2} = \{\alpha\delta\}$$

Taking $s = \epsilon$, $s' = \alpha$ gives $s, s' \in \overline{K_1 \cap K_2}$, while

$$s\delta = \delta \in \overline{K_1 \cap K_2}, \quad s'\delta = \alpha\delta \in \bar{K}_1 \cap \bar{K}_2 - \overline{K_1 \cap K_2}$$

Thus $K_1 \cap K_2$ is not observable. ◇

Furthermore, in general it is not true that the union of observable languages (closed or not) is observable.

Example 6.4.2

Let $\Sigma = \{\alpha, \beta\}$, $\Sigma_o = \{\beta\}$, with

$$L(\mathbf{G}) = \overline{(\epsilon + \alpha)\beta}, \quad K_1 = \{\alpha\}, \quad K_2 = \{\beta\}$$

Then K_1 and K_2 are both observable, but for $K = K_1 \cup K_2$ we have

$$\epsilon, \alpha \in \bar{K}, \quad P(\epsilon) = P(\alpha), \quad \epsilon\beta = \beta \in \bar{K}, \quad \alpha\beta \notin \bar{K}$$

and thus K is not observable. $\diamond$

We conclude from these results that the class of observable languages containing a given language (and with no closure requirement), despite its seemingly natural definition from the viewpoint of system theory, is algebraically rather badly behaved. A more satisfactory approach will be described in the section to follow. In the meantime we can, however, solve a problem of optimal supervision that addresses only the closed behavior of the resulting system. The result will be applicable provided nonblocking is not an issue.

Without essential loss of generality, we assume for the remainder of this section that

$$\Sigma_o \cup \Sigma_c = \Sigma$$

namely every event is either observable or controllable. As a consequence, every uncontrollable event is observable. Let A and E be closed sublanguages with

$$A \subseteq E \subseteq L(\mathbf{G})$$

We interpret E as 'legal behavior' and A as 'minimally adequate behavior'. Our objective is:

Obtain a feasible supervisory control V such that

$$A \subseteq L(V/\mathbf{G}) \subseteq E \tag{$*$}$$

First suppose that $A = \emptyset$. The supervisory control V defined by permanently disabling all controllable events is feasible and it is enough to check that $L(V/\mathbf{G}) \subseteq E$. If $A \neq \emptyset$, bring in the language class $\bar{\mathcal{O}}(A)$ as before, and the class $\bar{\mathcal{C}}(E)$ defined by

$$\bar{\mathcal{C}}(E) = \{K \subseteq E \mid K \text{ is closed and controllable}\}$$

Recall from Sect. 3.5 that $\bar{\mathcal{C}}(E)$ is closed under arbitrary intersections. We now have the following abstract solvability condition.

Theorem 6.4.2 (Lin)

Assume $A \neq \emptyset$. The problem (*) is solvable if and only if

$$\inf \bar{\mathcal{O}}(A) \subseteq \sup \bar{\mathcal{C}}(E)$$

Proof

(Only if) Let $K = L(V/\mathbf{G})$. Then K is closed. Taking $L_m(\mathbf{G}) = L(\mathbf{G})$ in Corollary 6.3.1 we obtain that K is controllable and observable, so

$$\inf \bar{\mathcal{O}}(A) \subseteq K \subseteq \sup \bar{\mathcal{C}}(E)$$

from which the condition follows.

(If) The family of sublanguages

$$\bar{\mathcal{K}} = \{K' | K' \supseteq \inf \bar{\mathcal{O}}(A) \ \& \ K' \in \bar{\mathcal{C}}(E)\}$$

is nonempty ($\sup \bar{\mathcal{C}}(E)$ belongs). Since $\bar{\mathcal{C}}(E)$ is closed under intersections, the language

$$\hat{K} := \inf \bar{\mathcal{K}} = \cap\{K' | K' \in \bar{\mathcal{K}}\}$$

belongs to $\bar{\mathcal{K}}$ and is thus closed and controllable.

Write $\hat{A} = \inf \bar{\mathcal{O}}(A)$. It will be shown that $\hat{K}$ is given explicitly by

$$\hat{K} = \hat{A}\Sigma_u^* \cap L(\mathbf{G})$$

Denote the right side of the proposed equality by $K^\#$. Clearly $K^\#$ is closed and contains $\hat{A}$. Also, $K^\#$ is controllable: for if $s = s_1 s_2$ with $s_1 \in \hat{A}$, $s_2 \in \Sigma_u^*$, $\sigma \in \Sigma_u$ and $s\sigma \in L(\mathbf{G})$, then

$$s\sigma \in \hat{A}\Sigma^* \cap L(\mathbf{G}) = K^\#;$$

and it is easy to see (by induction on strings) that any closed controllable language containing $\hat{A}$ must contain $K^\#$.

We claim that $\hat{K}$ is even observable. With this established, it only remains to invoke Corollary 6.3.1 for the desired result.

Because $\hat{K}$ is closed, to prove the claim it suffices to show

$$(\forall s, s', \sigma) s\sigma \in \hat{K} \ \& \ s' \in \hat{K} \ \& \ s'\sigma \in L(\mathbf{G}) \ \& \ Ps = Ps' \Rightarrow s'\sigma \in \hat{K}$$

Taking $\sigma \in \Sigma_u$ in the antecedent yields the result by controllability of $\hat{K}$. Suppose $\sigma \in \Sigma_c$ and assume, for a proof by contradiction, that $s'\sigma \notin \hat{K}$. We must have $s\sigma \in \hat{A}$. Clearly $s'\sigma \notin \hat{A}$, for if $s'\sigma \in \hat{A}$ and $s'\sigma \in L(\mathbf{G})$ then $s'\sigma \in L(\mathbf{G}) \cap \hat{A}\Sigma_u^* = \hat{K}$, contrary to our assumption.

It will be shown that $s' \in \hat{A}$. Otherwise, if $s' \notin \hat{A}$, let $w'\sigma'$ be the longest prefix of s' such that $w' \in \hat{A}$ and $w'\sigma' \notin \hat{A}$: because $\hat{A}$ is nonempty and closed, we have at least $\epsilon \in \hat{A}$, so if $s' \notin \hat{A}$ then $|s'| > 0$, and a prefix of the form described must surely exist. Furthermore $\sigma' \in \Sigma_u$: in fact, the assumption $s' \in \hat{K}$ implies $w'\sigma' \in \hat{K} = \hat{A}\Sigma_u^* \cap L(\mathbf{G})$, and then $w'\sigma' \notin \hat{A}$ requires $\sigma' \in \Sigma_u$ as stated. Now

$$\Sigma = \Sigma_c \cup \Sigma_o$$

implies $\Sigma_u \subseteq \Sigma_o$, so that $\sigma' \in \Sigma_o$. Since $Ps = Ps'$ by hypothesis, there is a prefix $w\sigma'$ of s such that $Pw = Pw'$. Since $s \in \hat{A}$ so is $w\sigma'$. Therefore

$$w\sigma' \in \hat{A}, \quad w' \in \hat{A}, \quad w'\sigma' \in L(\mathbf{G})$$

and by observability of $\hat{A}$ there follows $w'\sigma' \in \hat{A}$. This contradicts the supposition above that $w'\sigma' \notin \hat{A}$. Therefore $s' \in \hat{A}$ after all. Finally we have

$$Ps = Ps', \quad s\sigma \in \hat{A}, \quad s' \in \hat{A}, \quad s'\sigma \in L(\mathbf{G}) - \hat{A}$$

in contradiction to the fact that $\hat{A}$ is observable. The claim is proved, and with it the desired result. □

Example 6.4.3

The requirement in Theorem 6.4.2 that the relevant languages be closed cannot be dropped. Suppose, for instance, we replace $\bar{\mathcal{C}}(E)$ by $\mathcal{C}(E)$, the family of all controllable sublanguages of E, and replace $\bar{\mathcal{K}}$ by

$$\mathcal{K} = \{K | K \supseteq \inf \bar{\mathcal{O}}(A) \ \& \ K \in \mathcal{C}(E)\}$$

Then $\inf \mathcal{K}$ need not exist. As an example, let

$$\Sigma = \Sigma_o = \{\alpha, \beta, \gamma\}, \quad \Sigma_c = \{\beta, \gamma\}, \quad \Sigma_u = \{\alpha\}$$

$$L_m(\mathbf{G}) = \{\epsilon, \alpha\beta, \alpha\gamma\}, \quad A = \{\epsilon\}, \quad E = L_m(\mathbf{G})$$

Since all events are observable, $\inf \bar{\mathcal{O}}(A) = A$. Since $\alpha \in A\Sigma_u \cap L(\mathbf{G})$ and $\alpha \notin A$, A is not controllable. Because $L_m(\mathbf{G})$ is controllable, if $\inf \mathcal{K}$ exists then $A \subseteq \inf \mathcal{K} \subseteq L_m(\mathbf{G})$. Therefore the possible candidates for $\inf \mathcal{K}$ are

$$\{\epsilon, \alpha\beta\}, \quad \{\epsilon, \alpha\gamma\}, \quad \text{or} \quad \{\epsilon, \alpha\beta, \alpha\gamma\}$$

but none of these is infimal.

Example 6.4.4

If A and E are not closed, a solution to our problem (*) need not exist, even if A is observable and E is controllable. Let

$$\Sigma = \{\alpha, \beta\}, \quad \Sigma_o = \{\alpha\}, \quad \Sigma_c = \{\beta\}$$

$$L(\mathbf{G}) = \{\epsilon, \alpha, \alpha\beta, \beta, \beta\alpha, \beta\alpha\beta\}, \quad L_m(\mathbf{G}) = L(\mathbf{G}) - \{\epsilon\}$$

We take

$$A = \{\beta\}, \quad E = \{\alpha, \beta, \beta\alpha\beta\}$$

Now $\bar{A} = \{\epsilon, \beta\}$ and $\beta \in L_m(\mathbf{G})$ so $A = \bar{A} \cap L_m(\mathbf{G})$, i.e. A is $L_m(\mathbf{G})$-closed. Also (in the active/inactive set notation of Sect. 6.2)

$$\begin{aligned} A_A(\epsilon) = \{\beta\}, &\quad IA_A(\epsilon) = \{\alpha\}, \\ A_A(\beta) = \emptyset, &\quad IA_A(\beta) = \{\alpha\} \end{aligned}$$

hence A is observable. However, as

$$\beta \in \bar{A}, \quad \alpha \in \Sigma_u, \quad \beta\alpha \in L(\mathbf{G}) - \bar{A}$$

A is not controllable. Next, it can be verified that E is controllable; however, as

$$\begin{aligned} \alpha, \beta\alpha \in \bar{E}, &\quad P\alpha = \alpha = P(\beta\alpha), \\ A_E(\alpha) = \emptyset, &\quad IA_E(\alpha) = \{\beta\}, \\ A_E(\beta\alpha) = \{\beta\}, &\quad IA_E(\beta\alpha) = \emptyset \end{aligned}$$

it follows that E is not observable. Thus neither A nor E is a solution of the problem (*). Finally, if

$$A \subsetneqq K \subsetneqq E$$

then

$$K = K_1 := \{\alpha, \beta\} \quad \text{or} \quad K = K_2 := \{\beta, \beta\alpha\beta\},$$

but neither K_1 nor K_2 is controllable, and we conclude that (*) is not solvable. On the other hand, if E is replaced by $\bar{E}$ then the problem (*) is solved by

$$K = \{\epsilon, \alpha, \beta, \beta\alpha\}$$

$\diamond$

In general, if E is not closed then (*) may fail to be solvable simply because E has too few sublanguages.

6.5 Supervisory Control and Normality

As we saw in the previous section, the observability property can be conveniently exploited in supervisory control only when the relevant languages are all closed. Even then, because observability is not preserved under union, in general an optimal (minimally restrictive) supervisory control will not exist. We obtain a better structured problem if we replace observability by the stronger requirement of normality. To this end we set up our problem anew, in such a way that this section is independent of Sects. 6.2 - 6.4.

Let the controlled DES $\mathbf{G}$ over $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ be given, along with the observing agent's projection $P : \Sigma^* \to \Sigma_o^*$. As in Sect. 3.4, define the set of control patterns

$$\Gamma = \{\gamma \in Pwr(\Sigma) | \gamma \supseteq \Sigma_u\}$$

Just as before, we bring in the concept of a supervisory control $V : L(\mathbf{G}) \to \Gamma$. However, V must now 'respect' the observational constraint that control be based purely on the result of observing the strings generated by $\mathbf{G}$ through the channel $\mathbf{C}$, namely on the information transmitted by P. We say V is *feasible* if

$$(\forall s, s' \in L(\mathbf{G}))Ps = Ps' \Rightarrow V(s) = V(s'),$$

namely "look-alike strings result in the same control decision." Succinctly,

$$\ker(P|L(\mathbf{G})) \leq \ker V$$

As to marking, we require as usual that

(i) $L_m(V/\mathbf{G}) \subseteq L_m(\mathbf{G})$

It's natural to require as well that marking 'respect' the observational constraint, namely "look-alike strings in $L_m(\mathbf{G}) \cap L(V/\mathbf{G})$ are either both marked or both unmarked":

(ii) $(\forall s, s')s \in L_m(V/\mathbf{G})$ & $s' \in L_m(\mathbf{G}) \cap L(V/\mathbf{G})$ & $Ps' = Ps \Rightarrow s' \in L_m(V/\mathbf{G})$

If both (i) and (ii) hold we shall say that V is *admissible*. Admissibility is related to normality as follows.

Lemma 6.5.1

(i) If $L_m(V/\mathbf{G})$ is $(L_m(\mathbf{G}), P)$-normal then V is admissible.

(ii) If $L(V/\mathbf{G})$ is $(L(\mathbf{G}), P)$-normal and V is admissible then $L_m(V/\mathbf{G})$ is $(L_m(\mathbf{G}), P)$-normal. □

Thus if $L(V/\mathbf{G})$ is $(L(\mathbf{G}), P)$-normal then V admissible means that $L_m(V/\mathbf{G})$ is a union of sublanguages of the form $[s] \cap L_m(\mathbf{G})$, with $[s]$ a cell of ker P.

Exercise 6.5.1: Prove Lemma 6.5.1. ◇

Now let $E \subseteq L_m(\mathbf{G})$ be a specification language. We introduce

SCOP (Supervisory control and observation problem)

Find nonblocking, feasible, admissible V such that

$$\emptyset \neq L_m(V/\mathbf{G}) \subseteq E$$

To investigate SCOP we bring in the following three families of languages.

$$\mathcal{C}(E) := \{K \subseteq E | K \text{ is controllable wrt } \mathbf{G}\}$$

$$\mathcal{N}(E; L_m(\mathbf{G})) := \{K \subseteq E | K \text{ is } (L_m(\mathbf{G}), P) - \text{normal}\}$$

$$\bar{\mathcal{N}}(E; L(\mathbf{G})) := \{K \subseteq E | \bar{K} \text{ is } (L(\mathbf{G}), P) - \text{normal}\}$$

Each family is nonempty ($\emptyset$ belongs), and is closed under arbitrary unions. Let

$$\mathcal{S}(E) := \mathcal{C}(E) \cap \mathcal{N}(E; L_m(\mathbf{G})) \cap \bar{\mathcal{N}}(E; L(\mathbf{G}))$$

Then $\mathcal{S}(E)$ is nonempty and closed under arbitrary unions, so that $\sup \mathcal{S}(E)$ exists in $\mathcal{S}(E)$. Now we can provide a sufficient condition for the solution of SCOP.

Theorem 6.5.1

Let $K \neq \emptyset$ and $K \in \mathcal{S}(E)$. Define $V : L(\mathbf{G}) \to \Gamma$ according to:

$$V(s) := \begin{cases} \Sigma_u \cup \{\sigma \in \Sigma_c | P(s\sigma) \in P\bar{K}\} & Ps \in P\bar{K} \\ \Sigma_u, & Ps \in PL(\mathbf{G}) - P\bar{K} \end{cases}$$

and define

$$L_m(V/\mathbf{G}) := L(V/\mathbf{G}) \cap K$$

Then V solves SCOP, with

$$L_m(V/\mathbf{G}) = K \ .$$

Proof

Clearly V is feasible. We first claim

$$L(V/\mathbf{G}) = \bar{K}$$

Notice that

$$K \neq \emptyset \Rightarrow \bar{K} \neq \emptyset \Rightarrow \epsilon \in \bar{K}$$

To show $L(V/\mathbf{G}) \subseteq \bar{K}$, let $s \in L(V/\mathbf{G})$, $s \in \bar{K}$, and $s\sigma \in L(V/\mathbf{G})$. By definition of $L(V/\mathbf{G})$, we have $s\sigma \in L(\mathbf{G})$ and $\sigma \in V(s)$; and $s \in \bar{K}$ implies $Ps \in P\bar{K}$. If $\sigma \in \Sigma_u$ then, since $K \in \mathcal{C}(E)$,

$$s\sigma \in \bar{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{K}$$

If $\sigma \in \Sigma_c$, then $P(s\sigma) \in P\bar{K}$, which implies

$$s\sigma \in L(\mathbf{G}) \cap P^{-1}(P\bar{K}) = \bar{K} \ ,$$

since $K \in \bar{\mathcal{N}}(E; L(\mathbf{G}))$.

Next we show $\bar{K} \subseteq L(V/\mathbf{G})$. Let $s \in \bar{K}$, $s \in L(V/\mathbf{G})$, and $s\sigma \in \bar{K}$. Then $Ps \in P\bar{K}$; also $s\sigma \in \bar{L}_m(\mathbf{G}) \subseteq L(\mathbf{G})$. If $\sigma \in \Sigma_u$ then $\sigma \in V(s)$. If $\sigma \in \Sigma_c$ then, since $P(s\sigma) \in P\bar{K}$, again $\sigma \in V(s)$. Thus $s \in L(V/\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, and $\sigma \in V(s)$, so $s\sigma \in L(V/\mathbf{G})$, and our claim is proved.

To see that V is nonblocking, note that

$$\begin{aligned} L_m(V/\mathbf{G}) &:= L(V/\mathbf{G}) \cap K \\ &= \bar{K} \cap K \\ &= K \quad , \end{aligned}$$

namely $\bar{L}_m(V/\mathbf{G}) = L(V/\mathbf{G})$.

Finally, V is admissible, by the fact that

$$L_m(V/\mathbf{G}) = K \in \mathcal{N}(E; L_m(\mathbf{G}))$$

and Lemma 6.5.1. □

It is well to note that the replacement of observability by normality will restrict the resulting supervisory control by prohibiting the disablement of any controllable event that happens not to be observable: i.e. only observable events will be candidates for disablement. We may state this fact precisely as follows.

Proposition 6.5.1

Let $K \subseteq L(\mathbf{G})$ be controllable. If $\bar{K}$ is $(L(\mathbf{G}), P)$-normal, then

$$(\forall s \in \Sigma^*, \sigma \in \Sigma) s \in \bar{K} \quad \& \quad s\sigma \in L(\mathbf{G}) - \bar{K} \Rightarrow \sigma \in \Sigma_o \cap \Sigma_c$$ □

Exercise 6.5.2: Prove Proposition 6.5.1. ◇

On a more positive note, observability is tantamount to normality in the pleasant circumstance that all controllable events are observable under P. For convenience we restate Proposition 6.2.2.

Proposition 6.5.2

Let $K \subseteq L_m(\mathbf{G})$ be controllable and observable. Assume $P\sigma = \sigma$ for all $\sigma \in \Sigma_c$. Then K is $(L_m(\mathbf{G}), P)$-normal and $\bar{K}$ is $(L(\mathbf{G}), P)$-normal. □

Exercise 6.5.3: Prove Proposition 6.5.2, under the weaker assumption (suitably formalized) that $P\sigma = \sigma$ for all $\sigma \in \Sigma_c$ except possibly for those σ that are never disabled in the synthesis of K. ◇

Now let $\mathbf{G}_o$ be defined over the alphabet Σ_o, with

$$\begin{aligned} L_m(\mathbf{G}_o) &= PL_m(\mathbf{G}) \\ L(\mathbf{G}_o) &= PL(\mathbf{G}) \end{aligned}$$

Thus $\mathbf{G}_o$ is the 'observer's local model of $\mathbf{G}$'. Let

$$\Sigma_{c,o} := \{\sigma \in \Sigma_c | P\sigma = \sigma\} = \Sigma_c \cap \Sigma_o$$

and

$$\Sigma_{u,o} := \Sigma_u \cap \Sigma_o$$

be respectively the controllable and uncontrollable event subsets in $\mathbf{G}_o$. For $E_o \subseteq \Sigma_o^*$, let

$$\mathcal{C}_o(E_o) := \{K_o \subseteq E_o | K_o \text{ is controllable wrt } \mathbf{G}_o\}$$

Fix a specification language $E \subseteq \Sigma^*$, and bring in languages

$$\begin{aligned} N_o &:= P \sup \mathcal{N}(E; L_m(\mathbf{G})), \\ K_o &:= \sup \mathcal{C}_o(N_o) \\ J &:= P^{-1} K_o \\ K &:= L_m(\mathbf{G}) \cap J \ . \end{aligned}$$

Theorem 6.5.2

Assume $\mathbf{G}$ is nonblocking, i.e. $\bar{L}_m(\mathbf{G}) = L(\mathbf{G})$. If $(L_m(\mathbf{G}), J)$ are nonconflicting and $K \neq \emptyset$, then SCOP is solvable with

$$L_m(V/\mathbf{G}) = K \ .$$

Proof

It will be shown that $K \in \mathcal{S}(E)$. For this, let

$$N := \sup \mathcal{N}(E; L_m(\mathbf{G}))$$

Then

$$\begin{aligned} K &\subseteq L_m(\mathbf{G}) \cap P^{-1}(N_o) \\ &= L_m(\mathbf{G}) \cap P^{-1}(PN) \\ &= N \quad \text{(by normality)} \\ &\subseteq E \end{aligned}$$

Also (cf. remark preceding Exercise 6.1.1),

$$K = L_m(\mathbf{G}) \cap P^{-1} K_o$$

implies that K is $(L_m(\mathbf{G}), P)$-normal, i.e.

$$K \in \mathcal{N}(E; L_m(\mathbf{G})) \tag{1}$$

Since $L_m(\mathbf{G}), J$ are nonconflicting, we have

$$\begin{aligned} \bar{K} &= \overline{L_m(\mathbf{G}) \cap J} \\ &= \bar{L}_m(\mathbf{G}) \cap \bar{J} \\ &= L(\mathbf{G}) \cap P^{-1} \bar{K}_o \qquad \text{(by Exercise 6.1.4)} \end{aligned}$$

i.e. $\bar{K}$ is $(L(\mathbf{G}), P)$-normal, namely

$$K \in \bar{\mathcal{N}}(E; L(\mathbf{G})). \tag{2}$$

To see that K is controllable, let $s \in \bar{K}$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G})$. Then $Ps \in \bar{K}_o$. If $P\sigma = \sigma$, then

$$(Ps)\sigma = P(s\sigma) \in P(L(\mathbf{G})) = L(\mathbf{G}_o)$$

By $\mathbf{G}_o$-controllability of $\bar{K}_o$, we have $P(s\sigma) \in \bar{K}_o$, i.e.

$$s\sigma \in L(\mathbf{G}) \cap P^{-1}(\bar{K}_o) = \bar{K}$$

If $P\sigma = \epsilon$, then

$$P(s\sigma) = Ps \in \bar{K}_o$$

so again

$$s\sigma \in L(\mathbf{G}) \cap P^{-1}(\bar{K}_o) = \bar{K}$$

as required. Thus

$$K \in \mathcal{C}(E) \tag{3}$$

and by (1)-(3), $K \in \mathcal{S}(E)$. The result now follows by Theorem 6.5.1. $\square$

Exercise 6.5.4: Write $L_m = L_m(\mathbf{G})$, $L = L(\mathbf{G})$ and $P_L := P|L$; thus $P_L : L \to \Sigma_0^*$. Call P_L an L_m-*observer* if

$$(\forall t \in PL_m)(\forall s \in L)Ps \leq t \Rightarrow (\exists u \in \Sigma^*)su \in L_m \ \ \& \ \ P(su) = t$$

In other words, whenever Ps can be extended in Σ_0^* to a string $t \in PL_m$, the underlying string $s \in L$ can be extended to a string $su \in L_m$ with the same projection; the "local observer's expectation is never blocked in $\mathbf{G}$". Let $\bar{L}_m = L$, $K \subseteq L_m$ and P_L be an L_m-observer. Show that

$$\overline{L_m \cap P^{-1}(PK)} = L \cap P^{-1}(P\bar{K})$$

If K is (L_m, P)-normal conclude that $\bar{K}$ is (L, P)-normal, in which case

$$\mathcal{S}(E) = \mathcal{C}(E) \cap \mathcal{N}(E; L_m(\mathbf{G}))$$

With J as defined prior to Theorem 6.5.2, show that $(L_m(\mathbf{G}), J)$ are necessarily nonconflicting, and so it is enough to assume in Theorem 6.5.2 that $K \neq \emptyset$. $\diamond$

The foregoing results are brought together in the following step-by-step design method.

TCT Procedure for SCOP

0. Given **G**, **E**, and the list **NULL** of P-unobservable events
1. **N** := **supnorm(E,G,NULL)**
2. **NO** := **project(N,NULL)**[2]
3. **GO** := **project(G,NULL)**
4. **KO** := **supcon(GO,NO)** {proposed 'observer's supervisor'}
5. **KODAT** := **condat(GO,KO)**
6. **PINVKO** := **selfloop(KO,NULL)**
7. **nonconflict(G,PINVKO)** = *true*?
8. **K** = **meet(G,PINVKO)**

[2] Note that steps 1 and 2 could be combined by use of Exercise 6.1.8.

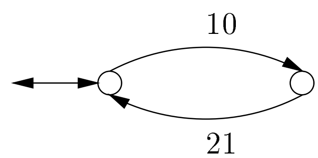
9. **K** nonempty?

If this procedure terminates successfully (with 'yes' at steps 7 and 9), then **PINVKO** provides a solution to SCOP, and **K** is the corresponding controlled behavior.

Example 6.5.1 (SCOP for Small Factory)

Take **MACH1**, **MACH2** as in Small Factory, with specification

BUF



$SL\{11, 12, 13, 20, 22, 23\}$

$$\begin{aligned} \mathbf{FACT} &= \mathbf{sync}(\mathbf{MACH1}, \mathbf{MACH2}) \\ \Sigma_o &= \{10, 11, 20, 21\} \end{aligned}$$

Thus the unobservable events pertain to breakdown and repair. By the design procedure we obtain the following.

0. **FACT, BUF, NULL** := [12,13,22,23]
1. **N** = **supnorm(BUF,FACT,NULL)** (18,42)
2. **NO** = **project(N,NULL)** (8,18)
3. **FACTO** = **project(FACT,NULL)** (4,12)
4. **KO** = **supcon(FACTO,NO)** (6,11)
5. **KODAT** = **condat(FACTO,KO)**
6. **PINVKO** = **selfloop(KO,NULL)** (6,35)
7. **nonconflict(FACT,PINVKO)** = *true*
8. **K** = **meet(FACT,PINVKO)** (20,37)
9. **K** is nonempty

Thus termination is successful. In the following additional steps we compare our result with the ideal case where all events are observable.

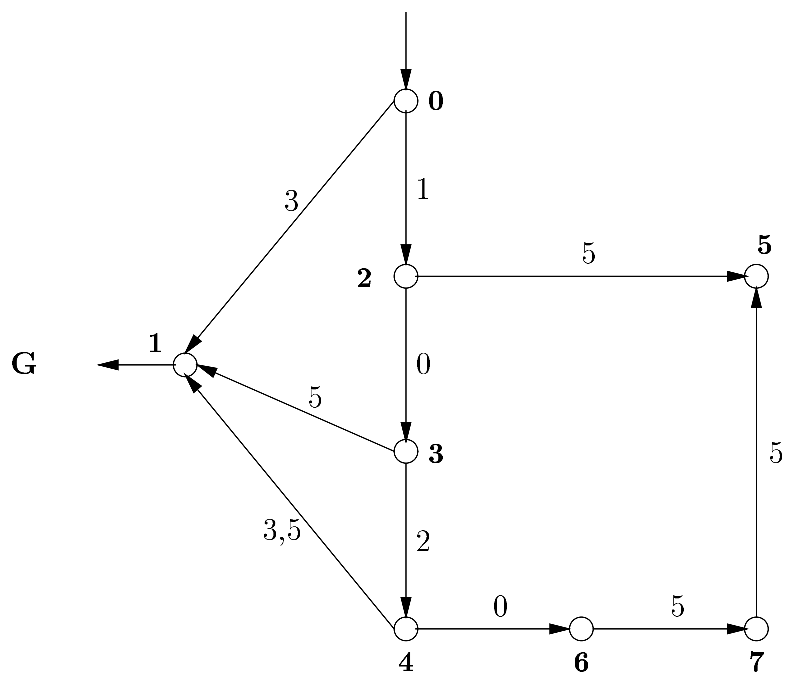
10. **MK = minstate(K)** (12,25)

11. **BUFSUP = supcon(FACT,BUF)** (12,25)

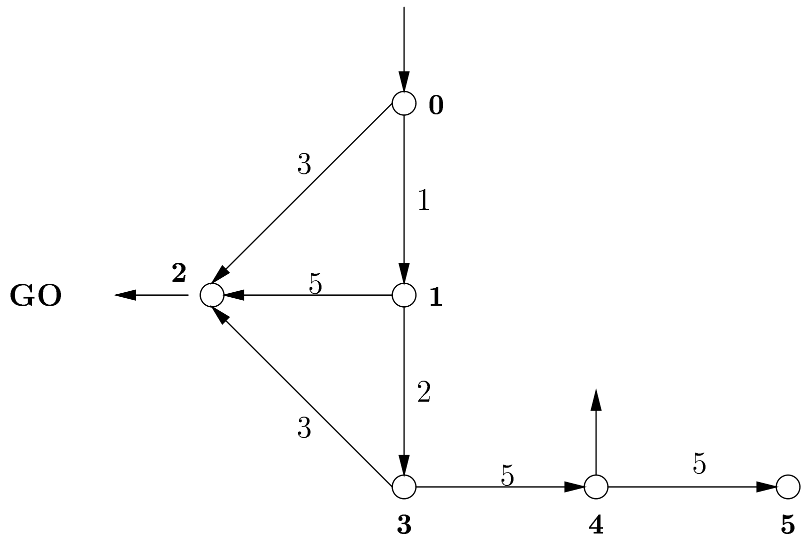
12. **isomorph(MK,BUFSUP)** = *true*

From this we conclude (rigorously if unsurprisingly) that optimal control of the buffer does not require observation of breakdowns and repairs. ◇

Exercise 6.5.5: A possible drawback of the *TCT* design procedure for SCOP is that the observer's supervisor, being based on the projected plant model (**GO** in Step 4), may fail to account for possible blocking (cf. the discussion in Sect. 2.5). As a result, conflict may result at Step 7, whereupon the design procedure fails. For instance, consider the plant model **G** displayed below with alphabet $\Sigma = \{0, 1, 2, 3, 5\}$, in which event 0 is unobservable.



The observer's plant model $\mathbf{GO} = \mathbf{project}(\mathbf{G}, [0])$ is the following:



Note that the closed and marked behaviors of **GO** are exactly the projections of the corresponding behaviors of **G** as required. However, **GO** has obliterated the information that the event sequence (1,0,2,0) in **G** leads to a non-coreachable state in **G**. With reference to Exercise 6.5.4, verify that here P_L is *not* an L_m-observer.

Attempt the *TCT* design procedure using the specification language Σ^* (i.e. requiring only nonblocking), verify that it fails at Step 7, and explain how the proposed supervisor could cause blocking. Start again, enhancing the plant model by self-looping its non-coreachable states (5,6,7) with an auxiliary uncontrollable but (hypothetically) observable event (4, say). Repeating the design procedure (with Σ unchanged, so that non-coreachable states are prohibited), verify that it now succeeds, and describe the control action. Finally, generalize this approach for arbitrary specifications, and provide conditions for its validity.

Exercise 6.5.6: It is almost obvious *a priori* that the approach of this section will fail for the two-agents problem in Example 6.3.1. Why?

Exercise 6.5.7: Suppose that the approach of this section and that of Example 6.3.1 both succeed in a particular instance; that is, both approaches yield feasible nonblocking supervisors, **S1** and **S2** respectively, for the given plant and specification DES. Show that the (marked) controlled behavior using **S2** is at least as large as that using **S1**. **Hint:** Using the procedure for **SCOP** compute K_1 (say) as in Theorem 6.5.2 and let **S1** synthesize K_1. Denote the corresponding supervisory control by V_1 as in Theorem 6.5.1. For **S2** compute $K_2 = \sup \mathcal{C}(E)$, define V_2 as in the proof of Theorem 6.3.1, and use the result of Exercise

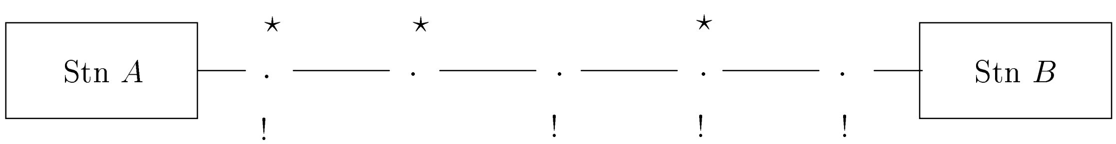
6.3.1. Noting $K_1 \subseteq K_2$, show that $L(V_1/\mathbf{G}) \subseteq L(V_2/\mathbf{G})$. For marking, assume for some $M \subseteq L_m(\mathbf{G})$ that $L_m(V_i/\mathbf{G}) = M \cap L(V/\mathbf{G})$ $(i = 1, 2)$.

Exercise 6.5.8: For a mild generalization of the setup in this chapter, suppose T is an alphabet disjoint from Σ, but that $P : \Sigma^* \to T^*$ remains catenative, and that for each $\sigma \in \Sigma$, either $P\sigma \in T$ or $P\sigma = \varepsilon$. Revise the exposition accordingly. [Note that disjointness is assumed merely for formal convenience.]

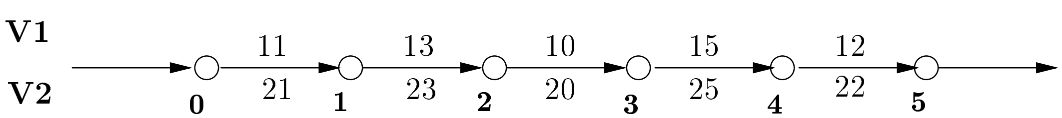
6.6 Control of a Guideway

The following example illustrates the ideas of this chapter in an intuitively simple setting, as well as the computation using *TCT* of the various DES required. Of course, this 'universal' approach to the example problem is far from being the most efficient; furthermore the piecemeal *TCT* computations could certainly be combined into higher-level procedures if desired.

Stations A and B on a guideway are connected by a single one-way track from A to B. The track consists of 4 sections, with stoplights ($\star$) and detectors (!) installed at various section junctions.



Two vehicles, **V1**, **V2** use the guideway simultaneously. **V_** is in state 0 (at A), state i (while travelling in section i, $i = 1, ..., 4$), or state 5 (at B).



To avoid collision, control of the stoplights must ensure that **V1** and **V2** never travel on the same section of track simultaneously: i.e. the **V**'s are subject to mutual exclusion of the state pairs (i, i), $i = 1, ..., 4$. Controllable events are odd-numbered; the unobservable events are $\{13, 23\}$.

By *TCT* the solution can be carried out as follows. Bracketed numbers (m, n) report the state size m and number of transitions n of the corresponding DES.

Following the procedure and notation in Sect. 6.5 (cf. Theorem 6.5.1), steps 0 to 9 compute the plant (generator) **G** = **V**, the legal specification language (generator) **E**, then the proposed feasible supervisor **K**.

0. **create(V1)** (6,5)
   create(V2) (6,5)
   V = sync(V1,V2) (36,60)
   E = mutex(V1,V2,[(1,1),(2,2),(3,3),(4,4)]) (30,40)
   NULL = [13,23]
1. **N = supnorm(E,V,NULL)** (26,32)
2. **NO = project(N,NULL)** (20,24)
3. **VO = project(V,NULL)** (25,40)
4. **KO = supcon(VO,KO)** (20,24)
5. **KODAT = condat(VO,KO)**
   {**KODAT** could instead be named **KO**, as in step 4 **KO** is filed as **KO.DES**, but in step 5 the result of **condat** is filed with suffix **.DAT**}
6. **PIKO = selfloop(KO,NULL)** (20,64)
7. **nonconflict(V,PIKO)** = *true*
8. **K = meet(V,PIKO)** (26,32)
9. **K** is nonempty by step 8.

It can be verified that in this example **K** turns out to be isomorphic to **N**.

The supervisory action of **K** can be read from the tabulated transition structure or from the transition graph and is the following (where tsi stands for 'track section i'): If **V2** starts first (event 21), it must enter $ts4$ before **V1** may start (event 11: disabled by light #1). **V1** may then continue into $ts3$ (event 10), but may not enter $ts4$ (event 15: disabled by light #3) until **V2** enters Stn B (event 22). Light #2 is not used. In fact, switching light #2 to red would mean disabling event 13 or 23; but these events are unobservable, while **K** is normal. If all events were observable, supervision could be based on **E**, allowing **V1** to start when **V2** has entered $ts2$. But then **V1** must halt at light #2 until **V2** has entered $ts4$.

The transition graph for **K** when **V2** starts first is displayed in Fig. 6.6.1 (for **E** adjoin the events shown dashed).

This example illustrates that the replacement of observability by normality as the property to be sought in control synthesis results in general in some 'loss of performance' in the sense of a restriction on control behavior that is not strictly necessary. For brevity write E etc. for $L_m(\mathbf{E})$. We claim first that E is not observable. For let $s = (21)$, $s' = (21)(23)$. The projection P nulls the event set {13,23}, so $Ps = Ps' = (21)$, i.e. $(s, s') \in \ker P$. By inspection of the transition structure of **V1,V2** we see that

$$\begin{aligned}
A_E(s) &= \{\sigma \mid s\sigma \in \bar{E}\} = \{23\}, \\
IA_E(s) &= \{\sigma \mid s\sigma \in L(\mathbf{V}) - \bar{E}\} = \{11\} \\
A_E(s') &= \{11, 20\} \\
IA_E(s') &= \emptyset
\end{aligned}$$

The fact that $A_E(s') \cap IA_E(s) = \{11\} \neq \emptyset$ proves the claim.

To obtain a controllable and observable sublanguage of E, delete from E the transitions [4,11,7] and [7,20,11], along with their mirror-image counterparts [3,21,6], [6,10,10]; call the resulting language COB. It is clear that COB is controllable, since the first transition in each of these pairs is controllable. Now the conditions

$$s, s' \in \overline{\text{COB}}, \quad (s, s') \in \ker P$$

plus the assumption that $s \neq s'$, hold (in the displayed graph for E) for

$$s = 21, \quad s' = (21)(23)$$

or vice-versa, and this time we have

$$\begin{aligned}
A_{\text{COB}}(s) &= \{23\}, \quad IA_{\text{COB}}(s) &= \{11\} \\
A_{\text{COB}}(s') &= \{20\}, IA_{\text{COB}}(s') &= \{11\}
\end{aligned}$$

for which the null intersection requirement is satisfied; and similarly for the remaining pairs $(s, s') \in \ker P$. So COB is observable. Therefore COB can be synthesized by a feasible supervisor; by inspection the supervisory control requires the supervisor to: (i) disable event 11 after (21), keeping 11 disabled until after the next observable event, and (ii) enable 11 but disable 13 after (21)(23)(20), and so on, as in the synthesis of E. Note that control calls for the disablement of the unobservable event 13, whereas in the synthesis of a closed normal language (cf. K, above) only observable events ever need to be disabled.

We check directly that $\bar{E}$ is not normal. Let

$$\begin{aligned}
t &= (21)(23)(11) \in \bar{E} \\
s &= Pt = (21)(11) \in L(\mathbf{V})
\end{aligned}$$

so $s = Ps = Pt \in P\bar{E}$ and $s \in L(\mathbf{V}) \cap P^{-1}(P(\bar{E}))$. But $s \notin \bar{E}$, i.e.

$$\bar{E} \subsetneqq L(\mathbf{V}) \cap P^{-1}(P(\bar{E}))$$

We can also check directly that $\overline{\mathrm{COB}}$ is not normal, by exhibiting a string terminating with an unobservable event that must be disabled. Let

$$\begin{aligned} s &= (21)(23)(20)(11)(13) \\ t &= (21)(23)(20)(11) \end{aligned}$$

Then

$$\begin{aligned} Ps &= (21)(20)(11) \\ Pt &= (21)(20)(11) \end{aligned}$$

Now $t \in \overline{\mathrm{COB}}$ and $Ps = Pt$, so $Ps \in P(\overline{\mathrm{COB}})$, and so

$$s \in P^{-1}P(\overline{\mathrm{COB}}) \cap L(\mathbf{V})$$

but $s \notin \overline{\mathrm{COB}}$. Summarizing, we have

$$K \subsetneqq \mathrm{COB} \subsetneqq E$$

Exercise 6.6.1: Complete the detailed verification that COB is observable.

Exercise 6.6.2: Apply the method of Example 6.3.1 to the Guideway example of this section, starting from the optimal supervisor under full observation.

Exercise 6.6.3: Develop an original example of your own along the lines of this section.

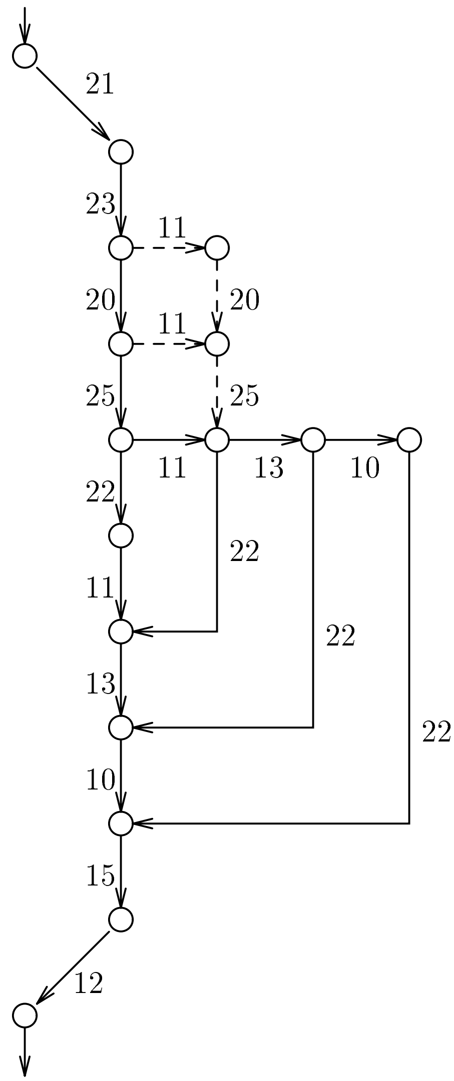


Fig. 6.6.1
Transition graph for **K** when **V2** starts first
(For **E** adjoin events shown dashed)

```
E    # states:  30    state set:  0 ...  29    initial state:  0

marker states:           29

vocal states:  none

# transitions:  40

transitions:

[  0, 11,  1] [  0, 21,  2] [  1, 13,  3] [  2, 23,  4]
[  3, 10,  5] [  3, 21,  6] [  4, 11,  7] [  4, 20,  8]
[  5, 15,  9] [  5, 21, 10] [  6, 10, 10] [  7, 20, 11]
[  8, 11, 11] [  8, 25, 12] [  9, 12, 13] [  9, 21, 14]
[ 10, 15, 14] [ 11, 25, 15] [ 12, 11, 15] [ 12, 22, 16]
[ 13, 21, 17] [ 14, 12, 17] [ 14, 23, 18] [ 15, 13, 19]
[ 15, 22, 20] [ 16, 11, 20] [ 17, 23, 21] [ 18, 12, 21]
[ 18, 20, 22] [ 19, 10, 23] [ 19, 22, 24] [ 20, 13, 24]
[ 21, 20, 25] [ 22, 12, 25] [ 23, 22, 26] [ 24, 10, 26]
[ 25, 25, 27] [ 26, 15, 28] [ 27, 22, 29] [ 28, 12, 29]

E printed.


EDAT


Control data are displayed as a list of supervisor states where disabling occurs,
together with the events that must be disabled there.

control data:

            1:  21                 2:  11
            6:  23                 7:  13
           10:  23                11:  13
           22:  25                23:  15

EDAT printed.
```

```
K    # states:  26    state set:  0 ...  25    initial state:  0

marker states:           25

vocal states:  none

# transitions:  32

transitions:

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

K printed.


KDAT


Control data are displayed as a list of supervisor states where disabling occurs,
together with the events that must be disabled there.

control data:

         1:  21                2:  11
         3:  21                4:  11
         5:  21                6:  11
        18:  25               19:  15

KDAT printed.
```

6.7 Notes and References

This chapter is based largely on the doctoral thesis of F. Lin [T08] and related publications [J09, C16].

Chapter 7

State-Based Control of Discrete-Event Systems

7.1 Introduction

In previous chapters our approach to modelling the DES control problem has started from two underlying languages, respectively generated by the plant and accepted by the specification. The system state descriptions were brought in as vehicles for representation and computation rather than as essential to the problem description. In the present chapter we adopt a dual and more conventional viewpoint, in which the underlying state transition structure is assigned a more basic role. Two illustrative examples are provided in the appendix, Sect. 7.7. In addition, the state viewpoint will facilitate the treatment of systems – notably the vector discrete-event systems of Chapt. 8 – where the underlying state set has regular algebraic structure that can be exploited for modelling compactness and computational efficiency.

7.2 Predicates and State Subsets

Let $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ be a DES, as defined in previous chapters. In order to place 'conditions' on the state q of $\mathbf{G}$, it will be convenient to use a logic formalism. While not strictly necessary (in fact we could make do with state subsets just as well), the logic terminology is sometimes closer to natural language and intuition. Thus, a *predicate* P defined on Q is a function $P : Q \to \{0, 1\}$. P can always be identified with the corresponding state subset

$$Q_P := \{q \in Q \mid P(q) = 1\} \subseteq Q \ ;$$

thus the complement $Q - Q_P = \{q \in Q \mid P(q) = 0\}$. We say that P *holds*, or *is satisfied*, precisely on the subset Q_P, and that $q \in Q_P$ *satisfies* P. The satisfaction relation $P(q) = 1$

will often be written $q \models P$ ('q satisfies P'). Write $Pred(Q)$ for the set of all predicates on Q, often identified with the power set $Pwr(Q)$. Boolean expressions will be formed as usual by logical negation, conjunction and disjunction; in standard notation:

$$(\neg P)(q) = 1 \text{ iff } P(q) = 0$$

$$\begin{array}{llll} (P_1 \wedge P_2)(q) = 1 & \text{iff} \quad P_1(q) = 1 & \text{and} & P_2(q) = 1 \\ (P_1 \vee P_2)(q) = 1 & \text{iff} \quad P_1(q) = 1 & \text{or} & P_2(q) = 1 \end{array}$$

Recall the De Morgan rules

$$\neg(P_1 \wedge P_2) = (\neg P_1) \vee (\neg P_2), \qquad \neg(P_1 \vee P_2) = (\neg P_1) \wedge (\neg P_2)$$

Given $P \in Pred(Q)$, we say that $P_1, P_2 \in Pred(Q)$ are *equivalent rel* P if

$$P_1 \wedge P = P_2 \wedge P \ ,$$

namely P_1 and P_2 coincide when restricted, or 'relativised', to the subset Q_P. As the logic counterpart to subset containment we introduce on $Pred(Q)$ the partial order $\preceq$ defined by

$$P_1 \preceq P_2 \text{ iff } P_1 \wedge P_2 = P_1 \quad (\text{iff } \neg P_1 \vee P_2)$$

Thus $P_1 \preceq P_2$ (P_1 'precedes' P_2) just when P_1 is stronger than P_2 in the sense that P_1 'implies' P_2; equivalently if $q \models P_1$ then $q \models P_2$; and we also say that P_1 is a *subpredicate* of P_2. Under the identification of $Pred(Q)$ with $Pwr(Q)$ and $\preceq$ with $\subseteq$ it is clear that $(Pred(Q), \preceq)$ is a complete lattice. The top element $\top$ of this lattice, defined by $Q_\top = Q$, will be denoted by *true* (and is the weakest possible predicate), while the bottom element $\bot$, defined by $Q_\bot = \emptyset$, will be written $\bot = false$ (and is the strongest possible predicate).

Exercise 7.2.1: Justify the terms "strong" and "weak" in this usage.

7.3 Predicate Transformers

First we consider reachability issues for $\mathbf{G}$. Let $P \in Pred(Q)$. The *reachability predicate* $R(\mathbf{G}, P)$ is defined to hold precisely on those states that can be reached in $\mathbf{G}$ from q_o via states satisfying P, according to the inductive definition:

1. $q_o \models P \Rightarrow q_o \models R(\mathbf{G}, P)$
2. $q \models R(\mathbf{G}, P)$ & $\sigma \in \Sigma$ & $\delta(q, \sigma)!$ & $\delta(q, \sigma) \models P \Rightarrow \delta(q, \sigma) \models R(\mathbf{G}, P)$
3. No other states q satisfy $R(\mathbf{G}, P)$.

Explicitly, $q \models R(\mathbf{G}, P)$ if and only if there exist an integer $k \geq 0$, states $q_1, ..., q_k \in Q$, and events $\sigma_0, \sigma_1, ..., \sigma_{k-1} \in \Sigma$ such that

1. $\delta(q_i, \sigma_i)! \ \& \ \delta(q_i, \sigma_i) = q_{i+1}, \ \ i = 0, 1, ..., k-1$
2. $q_i \models P, \ \ i = 0, 1, ..., k$
3. $q_k = q$

For fixed $\mathbf{G}$, $R(\mathbf{G}, \cdot) : Pred(Q) \to Pred(Q)$ is an example of a *predicate transformer*, i.e. a map transforming predicates to predicates. Clearly $R(\mathbf{G}, P) \preceq P$; also $R(\mathbf{G}, \cdot)$ is monotone with respect to $\preceq$, in the sense that $P \preceq P'$ implies $R(\mathbf{G}, P) \preceq R(\mathbf{G}, P')$. Note that $R(\mathbf{G}, true)$ is the reachable set for $\mathbf{G}$, namely the set of states reachable from q_o without constraint.

Fix $\sigma \in \Sigma$. The *weakest liberal precondition* corresponding to $\sigma \in \Sigma$ is the predicate transformer $M_\sigma : Pred(Q) \to Pred(Q)$ defined as follows:

$$M_\sigma(P)(q) := \begin{cases} 1 \text{ if either } \delta(q, \sigma)! \ \& \ \delta(q, \sigma) \models P, \text{ or } \ \textit{not} \ \delta(q, \sigma)! \\ 0 \text{ otherwise (i.e. } \delta(q, \sigma)! \text{ and } \ \textit{not} \ \delta(q, \sigma) \models P) \end{cases}$$

Equivalently and more concisely,

$$q \models M_\sigma(P) \text{ iff } \delta(q, \sigma) \models P \text{ whenever } \delta(q, \sigma)!$$

It is clear that $M_\sigma(\cdot)$ is monotone. The action of M_σ can be visualized in terms of the 'dynamic flow' on Q induced by one-step state transitions under σ, wherever such transitions are defined. Thus $M_\sigma(P)$ is just the condition on (states in) Q that ensures that from a state, say q, there is a one-step transition into Q_P under the occurrence of the event σ (so $M_\sigma(P)$ is indeed a 'precondition' for P), or else that σ cannot occur at q at all, i.e. $\delta(q, \sigma)$ is not defined (in this sense the precondition is 'liberal'). Note that 'weakest' means 'the largest state subset with the asserted property', namely 'the weakest assumption required to establish either that σ was not enabled or that σ occurred and led to P'.

The weakest liberal precondition is retrospective, drawing attention from a present condition (i.e. P) to its one-step antecedent. A dual concept is the predicate transformer *strongest postcondition* N_σ: $Pred(Q) \to Pred(Q)$ defined according to

$$N_\sigma(P)(q) := \begin{cases} 1 \text{ if } (\exists q' \models P) \ \delta(q', \sigma)! \text{ and } \delta(q', \sigma) = q \\ 0 \text{ otherwise} \\ \quad \text{(i.e. } (\forall q' \models P) \text{ (either } \textit{not} \ \delta(q', \sigma)! \text{ or } \delta(q', \sigma) \neq q) \end{cases}$$

The strongest postcondition is prospective, drawing attention from a present condition to its one-step consequent; and 'strongest' means 'the smallest state subset with the asserted property', namely 'the strongest inference that can be made solely on the assumption that P held and σ occurred'.

We shall see that weakest liberal precondition plays a role in controllability properties, while it can be shown that strongest postcondition relates to observability.

Later we shall use the connection between state reachability and language behavior, summarized as follows. The *closed behavior* $L(\mathbf{G}, q)$ corresponding to initialization of $\mathbf{G}$ at an arbitrary state $q \in Q$ is defined to be

$$L(\mathbf{G}, q) := \{w \in \Sigma^* \mid \delta(q, w)!\}$$

while the corresponding *marked behavior* is, of course,

$$L_m(\mathbf{G}, q) := \{w \in L(\mathbf{G}, q) \mid \delta(q, w) \in Q_m\}$$

Similarly we define the closed [resp. marked] language *induced* by a predicate $P \in Pred(Q)$ to be

$$L(\mathbf{G}, P) := \{w \in L(\mathbf{G}, q_o) \mid (\forall v \leq w)\ \delta(q_o, v) \models P\}$$

[resp.

$$L_m(\mathbf{G}, P) := \{w \in L(\mathbf{G}, P) \mid \delta(q_o, w) \in Q_m\}]$$

The *reachable set* $R(\mathbf{G}, q)$ of states reachable from arbitrary $q \in Q$ is then

$$R(\mathbf{G}, q) := \{\delta(q, w) \mid w \in L(\mathbf{G}, q)\}$$

Exercise 7.3.1: Consider an agent observing the behavior of $\mathbf{G}$, who knows only that $\mathbf{G}$ was initialized in some (unspecified) state $q \in Q_P \subseteq Q$ with $P \in Pred(Q)$, and that $\mathbf{G}$ subsequently generated the string $s = \sigma_1\sigma_2...\sigma_k$. Show that the agent's best estimate of the state at s is the predicate

$$N_{\sigma_k}(...N_{\sigma_2}(N_{\sigma_1}(P))...)$$

7.4 State Feedback and Controllability

We define a *state feedback control (SFBC)* for $\mathbf{G}$ to be a total function

$$f : Q \to \Gamma$$

where

$$\Gamma := \{\Sigma' \subseteq \Sigma \mid \Sigma' \supseteq \Sigma_u\}$$

Thus f attaches to each state q of $\mathbf{G}$ a subset of events that always contains the uncontrollable events. The event $\sigma \in \Sigma$ is *enabled* at q if $\sigma \in f(q)$, and is *disabled* otherwise; σ is always enabled if $\sigma \in \Sigma_u$. For $\sigma \in \Sigma$ introduce the predicate $f_\sigma : Q \to \{0, 1\}$ defined by

$$f_\sigma(q) := 1 \text{ iff } \sigma \in f(q)$$

Thus a SFBC f is specified by the family of predicates $\{f_\sigma \mid \sigma \in \Sigma\}$. The closed-loop transition function induced by f will be written δ^f, given by

$$\delta^f(q, \sigma) := \begin{cases} \delta(q, \sigma) \text{ if } \delta(q, \sigma)! \text{ and } f_\sigma(q) = 1 \\ \text{undefined, otherwise} \end{cases}$$

We write $\mathbf{G}^f := (Q, \Sigma, \delta^f, q_o, Q_m)$ for the closed-loop DES formed from $\mathbf{G}$ and f.

If f is a SFBC for $\mathbf{G}$ then, of course, $R(\mathbf{G}^f, P)$ denotes the reachability predicate for $\mathbf{G}^f$ (initialized at q_o). Since for any q and σ, $\delta^f(q, \sigma)!$ only if $\delta(q, \sigma)!$ it is evident that $R(\mathbf{G}^f, P) \preceq R(\mathbf{G}, P)$.

The following definition is fundamental. We say that $P \in Pred(Q)$ is *controllable* (with respect to $\mathbf{G}$) if

$$P \preceq R(\mathbf{G}, P) \quad \& \quad (\forall \sigma \in \Sigma_u) P \preceq M_\sigma(P)$$

Thus controllability asserts that if q satisfies P then (i) q is reachable from q_o via a sequence of states satisfying P, and (ii) if $\sigma \in \Sigma_u$, $\delta(q, \sigma)!$ and $q' = \delta(q, \sigma)$ then q' satisfies P, namely 'P is invariant under the flow induced by uncontrollable events'. Note that the predicate *false* is trivially controllable.

Theorem 7.4.1

Let $P \in Pred(Q)$, $P \neq false$. Then P is controllable if and only if there exists a SFBC f for $\mathbf{G}$ such that $R(\mathbf{G}^f, true) = P$.

Thus a nontrivial predicate P is controllable precisely when it can be 'synthesized' by state feedback.

Proof

(If) Assume $R(\mathbf{G}^f, true) = P$. Since $\delta^f(q, \sigma)!$ implies $\delta(q, \sigma)!$ it is clear that any q such that $q \models P$ can be reached from q_o along a sequence of states with the same property, so $P \preceq R(\mathbf{G}, P)$. Let $\sigma \in \Sigma_u$ and $q \models P$. If $\delta(q, \sigma)!$ then $\delta^f(q, \sigma)!$ (since f is a SFBC) and then

$$\delta(q, \sigma) = \delta^f(q, \sigma) \models R(\mathbf{G}^f, true) = P$$

This implies that $q \models M_\sigma(P)$, and therefore $P \preceq M_\sigma(P)$.

(Only if) Assume P controllable and define the SFBC $f : Q \to \Gamma$ by

$$(\forall \sigma \in \Sigma_c) f_\sigma := M_\sigma(P)$$

First it will be shown that $R(\mathbf{G}^f, true) \preceq P$. Let $q \models R(\mathbf{G}^f, true)$. Then for some $k \geq 0$, $q_1, ..., q_k (= q) \in Q$ and $\sigma_0, \sigma_1, ..., \sigma_{k-1} \in \Sigma$, we have

$$\delta(q_i, \sigma_i)!, \quad \delta(q_i, \sigma_i) = q_{i+1}, \quad f_{\sigma_i}(q_i) = 1, \quad i = 0, 1, ..., k-1$$

By hypothesis, $\hat{q} \models P$ for some $\hat{q}$, and by controllability $\hat{q} \models R(\mathbf{G}, P)$, so in particular $q_o \models P$. We claim that $q_1 \models P$. For if $\sigma_o \in \Sigma_u$ then controllability implies that $q_o \models M_{\sigma_o}(P)$, so $\delta(q_o, \sigma_o) = q_1 \models P$; while if $\sigma_o \in \Sigma_c$ then $f_{\sigma_o}(q_o) = 1$, namely $q_o \models M_{\sigma_o}(P)$ and again $\delta(q_o, \sigma_o) = q_1 \models P$. By repetition of this argument for $q_2, ..., q_k$ we conclude that $q_k = q \models P$. It remains to show that $P \preceq R(\mathbf{G}^f, true)$. Let $q \models P$. By controllability, $q \models R(\mathbf{G}, P)$. For some $k \geq 0$, $q_1, ..., q_k (= q) \in Q$ and $\sigma_o, \sigma_1, ..., \sigma_{k-1} \in \Sigma$ we have

$$q_i \models P, \quad i = 0, ..., k$$

$$\delta(q_i, \sigma_i)!, \quad \delta(q_i, \sigma_i) = q_{i+1}, \quad i = 0, ..., k-1$$

and therefore

$$q_i \models M_{\sigma_i}(P), \quad i = 0, 1, ..., k-1$$

If $\sigma_i \in \Sigma_u$ then $f_{\sigma_i}(q_i) = 1$ because f is a SFBC; while if $\sigma_i \in \Sigma_c$ then $f_{\sigma_i}(q_i) = M_{\sigma_i}(P)(q_i) = 1$ (as just shown). Therefore $\delta^f(q_i, \sigma_i)!$ and $\delta^f(q_i, \sigma_i) = q_{i+1}$ $(i = 0, 1, ..., k-1)$, namely

$$q = q_k \models R(\mathbf{G}^f, true)$$

as claimed. □

Now suppose $P \in Pred(Q)$ is not controllable. As usual, we seek a controllable predicate that best approximates P from below. Following standard procedure, bring in the family of controllable predicates that are stronger than P, namely

$$\mathcal{CP}(P) := \{K \in Pred(Q) \mid K \preceq P \ \& \ K \text{ controllable}\}$$

Proposition 7.4.1

$\mathcal{CP}(P)$ is nonempty and is closed under arbitrary disjunctions; in particular the supremal element $\sup \mathcal{CP}(P)$ exists in $\mathcal{CP}(P)$.

Proof

We have already seen that $false \in \mathcal{CP}(P)$. Now let $K_\lambda \in \mathcal{CP}(P)$ for $\lambda \in \Lambda$, some index set. It will be shown that

$$K := \bigvee_{\lambda \in \Lambda} K_\lambda \in \mathcal{CP}(P)$$

It is clear that $K \preceq P$, so it remains to test controllability. Let $q \models K$, so $q \models K_\lambda$ for some λ. By controllability of K_λ, $q \models R(\mathbf{G}, K_\lambda)$, and as $K_\lambda \preceq K$ there follows $q \models R(\mathbf{G}, K)$. If $\sigma \in \Sigma_u$ then similarly $q \models M_\sigma(K_\lambda)$, hence $q \models M_\sigma(K)$ as required. Finally, the supremal element of $\mathcal{CP}(P)$ is

$$\sup \mathcal{CP}(P) = \bigvee \{K \mid K \in \mathcal{CP}(P)\}$$

□

The following characterization will be useful later. Define the predicate transformer $\langle \cdot \rangle$ according to

$$q \models \langle P \rangle \text{ if } (\forall w \in \Sigma_u^*)\delta(q, w)! \Rightarrow \delta(q, w) \models P$$

Note that $\langle P \rangle \preceq P$ (since $\delta(q, \epsilon) = q$) and in fact $\langle P \rangle$ is the weakest subpredicate of P that is invariant under the flow induced by uncontrollable events.

Proposition 7.4.2

$$\sup \mathcal{CP}(P) = R(\mathbf{G}, \langle P \rangle)$$

Proof

Claim 1. $R(\mathbf{G}, \langle P \rangle) \in \mathcal{CP}(P)$

We show that $R(\mathbf{G}, \langle P \rangle)$ is controllable. Let $q \models R(\mathbf{G}, \langle P \rangle)$. Then $q_o \models \langle P \rangle$ and there are $k \geq 0$, $q_1, ..., q_k (= q) \in Q$ and $\sigma_o, \sigma_1, ..., \sigma_{k-1} \in \Sigma$ such that

$$q_i \models \langle P \rangle, \quad i = 1, ..., k$$

$$\delta(q_i, \sigma_i)!, \quad \delta(q_i, \sigma_i) = q_{i+1}, \quad i = 0, ..., k-1$$

We note that $q_i \models R(\mathbf{G}, \langle P \rangle)$ since $q_j \models \langle P \rangle$ for $j = 0, 1, ..., i$. In particular $q \models R(\mathbf{G}, R(\mathbf{G}, \langle P \rangle))$, so $R(\mathbf{G}, \langle P \rangle) \preceq R(\mathbf{G}, R(\mathbf{G}, \langle P \rangle))$. Next we choose $\sigma \in \Sigma_u$ with $\delta(q, \sigma)!$ and establish $q \in M_\sigma(R(\mathbf{G}, \langle P \rangle))$, namely $q' := \delta(q, \sigma) \models R(\mathbf{G}, \langle P \rangle)$. For this let $w \in \Sigma_u^*$ with $\delta(q', w)!$. Then $\sigma w \in \Sigma_u^*$, and

$$\delta(q', w) = \delta(\delta(q, \sigma), w) = \delta(q, \sigma w) \models P \quad \text{(since } q \models \langle P \rangle\text{)}$$

Thus $q' \models \langle P \rangle$. Then $q \models R(\mathbf{G}, \langle P \rangle)$ together with $q' = \delta(q, \sigma)$ implies $q' \models R(\mathbf{G}, \langle P \rangle)$, so $q \models M_\sigma(R(\mathbf{G},\ \langle P \rangle))$, completing the proof of Claim 1.

Claim 2. Let $P' \in \mathcal{CP}(P)$. Then $P' \preceq R(\mathbf{G}, \langle P \rangle)$.

Let $q \models P'$. Then $q_o \models P'$ and there exist $k \geq 0$, $q_1, ..., q_k (= q) \in Q$ and $\sigma_o, \sigma_1, ..., \sigma_{k-1} \in \Sigma$ such that

$$q_i \models P', \quad i = 1, ..., k$$

$$\delta(q_i, \sigma_i)!, \quad \delta(q_i, \sigma_i) = q_{i+1} \quad i = 0, ..., k-1$$

Fix i and $\sigma \in \Sigma_u$. Then $q_i \models P' \preceq M_\sigma(P')$ and $\delta(q_i, \sigma)!$ imply $\delta(q_i, \sigma) \models P'$. If $w \in \Sigma_u^*$ then by induction on $|w|$ we infer that if $\delta(q_i, w)!$ then $\delta(q_i, w) \models P' \preceq P$. There follows in turn

$$q_o \models \langle P \rangle, \quad q_1 \models \langle P \rangle, ..., q_k \models \langle P \rangle, \quad \text{i.e. } q \models \langle P \rangle,$$

namely $q \models R(\mathbf{G}, \langle P \rangle)$, and Claim 2 is proved.

The result follows from Claims 1 and 2. □

Corollary 7.4.1

$$\sup \mathcal{CP}(P) \neq false \quad \text{iff} \quad R(\mathbf{G}, \langle P \rangle) \neq false \quad \text{iff} \quad q_o \models \langle P \rangle$$

□

Under the assumption that $\sup \mathcal{CP}(P) \neq false$, we may define a corresponding 'optimal', i.e. behaviorally least restrictive, SFBC f^* to synthesize $R(\mathbf{G}, \langle P \rangle)$. Imitating the proof ('Only if') of Theorem 7.4.1 we may set

$$(\forall \sigma \in \Sigma_c) f_\sigma^*(q) := \begin{cases} 1 & \text{if } \delta(q, \sigma)! \ \&\ \delta(q, \sigma) \models \langle P \rangle \\ 0 & \text{otherwise} \end{cases}$$

Note that $f_\sigma^*(q)$ may be evaluated arbitrarily (in particular $= 0$) if *not* $\delta(q, \sigma)!$. This formula suggests that in practice optimal control can be implemented 'on-line', namely by testing, at the current state q, the satisfaction relation $\delta(q, \sigma) \models \langle P \rangle$ for each controllable event σ such that $\delta(q, \sigma)!$. Efficient implementation of f_σ^* in an application will thus depend on devising an economical algorithmic representation of $\langle P \rangle$, namely of 'reachability on the uncontrollable subsystem'. While in general this reachability property may be intractable or even undecidable, we shall see in the next chapter that an efficient algorithm is often available for vector discrete-event systems and linear predicates.

Exercise 7.4.1: Show that the predicate $P \in Pred(Q)$ is controllable if and only if : $P = R(\mathbf{G}, P)$ and the language $L(\mathbf{G}, P)$ is controllable.

Exercise 7.4.2: Show that, for arbitrary $P \in Pred(Q)$,

$$L(\mathbf{G}, \sup \mathcal{CP}(P)) = \sup \mathcal{C}(L(\mathbf{G}, P))$$

7.5 Balanced State Feedback Controls and Modularity

A SFBC $f : Q \to \Gamma$ is *balanced* if

$$(\forall q, q' \in Q)(\forall \sigma \in \Sigma)\ q, q' \models R(\mathbf{G}^f, true) \quad \& \quad \delta(q,\sigma)! \quad \& \quad \delta(q,\sigma) = q'$$

$$\Rightarrow f_\sigma(q) = 1$$

A balanced SFBC is one which, among all SFBC synthesizing the same (controllable) predicate, enables at every reachable state q the largest possible set of (controllable) events σ for which $\delta(q,\sigma)!$.

Exercise 7.5.1: Show that an arbitrary SFBC can always be replaced by a balanced SFBC without changing the reachable set. ◇

Let $P \in Pred(Q)$, $P \neq false$, be expressible as a conjunction of predicates $P_1, ..., P_k$:

$$P = \bigwedge_{i=1}^{k} P_i$$

We shall think of P as a specification for the controlled behavior of the DES $\mathbf{G}$, and the $P_i \in Pred(Q)$ as partial specifications. Our objective will be to implement an optimal SFBC for P by means of modular SFBC for the P_i.

Write $R(f/\mathbf{G})$ for $R(\mathbf{G}^f, true)$. For $i = 1, ..., k$ let $f_i : Q \to \Gamma$ be an optimal SFBC for P_i, i.e.

$$R(f_i/\mathbf{G}) = \sup \mathcal{CP}(P_i)$$

The *modular SFBC* f formed from the f_i is given by

$$f_\sigma := \bigwedge_{i=1}^{k} f_{i,\sigma}, \qquad \sigma \in \Sigma$$

i.e. $f_\sigma(q) = 1$ iff $f_{i,\sigma}(q) = 1$ for $i = 1, ..., k$: an event is enabled by f if and only if it is enabled by each f_i. In this case write symbolically

$$f := \bigwedge_{i=1}^{k} f_i$$

Theorem 7.5.1

Assume that f_i is balanced $(i = 1, ..., k)$. Then f is balanced, and

$$R(f/\mathbf{G}) = \sup \mathcal{CP}(P)$$

Proof

Clearly $R(f/\mathbf{G}) \preceq \wedge_{i=1}^{k} R(f_i/\mathbf{G})$, from which it easily follows that f is balanced. Next, we have

$$R(f_i/\mathbf{G}) = R(\mathbf{G}, \langle P_i \rangle) \preceq P_i, \qquad i = 1, ..., k$$

so that $R(f/\mathbf{G}) \preceq P$, hence $R(f/\mathbf{G}) \in \mathcal{CP}(P)$, and therefore $R(f/\mathbf{G}) \preceq \sup \mathcal{CP}(P)$. It remains to check that

$$\sup \mathcal{CP}(P) \preceq R(f/\mathbf{G})$$

Now $\sup \mathcal{CP}(P) = R(\mathbf{G}, \langle P \rangle)$, and

$$\langle P \rangle = \langle \bigwedge_{i=1}^{k} P_i \rangle = \bigwedge_{i=1}^{k} \langle P_i \rangle$$

so it must be shown that

$$R(\mathbf{G}, \bigwedge_{i=1}^{k} \langle P_i \rangle) \preceq R(f/\mathbf{G})$$

Let

$$q \models R(\mathbf{G}, \bigwedge_{i=1}^{k} \langle P_i \rangle)$$

Then there are $m \geq 0$, $q_1, ..., q_m (= q)$ and $\sigma_o, \sigma_1, ..., \sigma_{m-1}$ such that

$$\delta(q_j, \sigma_j)!, \quad \delta(q_j, \sigma_j) = q_{j+1}, \quad j = 0, 1, ..., m-1$$

$$q_j \models \bigwedge_{i=1}^{k} \langle P_i \rangle, \qquad j = 0, 1, ..., m$$

Thus $q_j \models R(\mathbf{G}, \langle P_i \rangle) = R(f_i/\mathbf{G})$ $(i = 1, ..., k;\ j = 0, 1, ..., m)$. Since each f_i is balanced,

$$f_{i,\sigma_j}(q_j) = 1, \qquad i = 1, ..., k; \quad j = 0, 1, ..., m-1$$

which implies in turn

$$\bigwedge_{i=1}^{k}(f_i)_{\sigma_j}(q_j) = 1, \qquad j = 0, 1, ..., m-1$$

$$q_j \models R(f/\mathbf{G})$$

$$q = q_m \models R(f/\mathbf{G})$$

as required. □

Exercise 7.5.2: Provide an example to show that the conclusion of Theorem 7.5.1 may fail if the assumption that the f_i are balanced is dropped.

7.6 Dynamic State Feedback Control

The foregoing methods are readily extended to the situation where additional memory elements are included in the state description, permitting the control action to depend not only on the current state of the plant $\mathbf{G}$, but also on various properties of past behavior.

Let $\mathbf{G}_i = (Q_i, \Sigma, \delta_i, q_{i,o}, Q_{i,m})$ $(i = 1, 2)$ be DES defined over the same alphabet Σ. We recall that the *product DES* $\mathbf{G} = \mathbf{G}_1 \times \mathbf{G}_2$, $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$, is defined according to

$$\begin{aligned} Q &= Q_1 \times Q_2 \\ \delta((q_1, q_2), \sigma) &= (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) \end{aligned}$$

(just in case $\delta(q_1, \sigma)!$ and $\delta(q_2, \sigma)!$),

$$q_o = (q_{1,o}, q_{2,o}), \qquad Q_m = Q_{1,m} \times Q_{2,m}$$

Let $\mathbf{H} = (Y, \Sigma, \eta, y_o, Y)$ be a DES with marker subset the entire state set (so marked and closed behaviors coincide). We say that $\mathbf{H}$ is a *memory* for $\mathbf{G}$ if $L(\mathbf{H}) \supseteq L(\mathbf{G})$. Such $\mathbf{H}$ does not constrain $\mathbf{G}$ under product of DES, namely

$$L(\mathbf{G} \times \mathbf{H}) = L(\mathbf{G}), \qquad L_m(\mathbf{G} \times \mathbf{H}) = L_m(\mathbf{G})$$

Memory can provide a medium for the 'recording' of specification languages restricting the behavior of $\mathbf{G}$, in the following sense. Let $E \subseteq \Sigma^*$ be a closed language, and let $P \in Pred(Q \times Y)$. We say that the pair (E, P) is *compatible* with $\mathbf{G} \times \mathbf{H}$ if

$$L(\mathbf{G} \times \mathbf{H}, P) = E \cap L(\mathbf{G})$$

Thus enforcement of the predicate P on the state set of the product DES is equivalent to restricting the closed behavior of $\mathbf{G}$ to the language E. The precise connection is the following, where we write $\mathcal{CP}_{\mathbf{G}\times\mathbf{H}}(\cdot)$ for the family of controllable (with respect to $\mathbf{G}\times\mathbf{H}$) subpredicates (on $Q\times Y$) of its argument, and $\mathcal{C}_{\mathbf{G}\times\mathbf{H}}(\cdot)$ (resp. $\mathcal{C}_{\mathbf{G}}$) for the family of controllable (with respect to $\mathbf{G}\times\mathbf{H}$ (resp. $\mathbf{G}$)) sublanguages of its argument.

Theorem 7.6.1

Let $\mathbf{H}$ be a memory for $\mathbf{G}$. Then

$$\mathcal{C}_{\mathbf{G}\times\mathbf{H}}(E) = \mathcal{C}_{\mathbf{G}}(E)$$

Also, if $E \subseteq \Sigma^*$ is closed and (E, P) is compatible with $\mathbf{G}\times\mathbf{H}$, then

$$L(\mathbf{G}\times\mathbf{H}, \sup\mathcal{CP}_{\mathbf{G}\times\mathbf{H}}(P)) = \sup\mathcal{C}_{\mathbf{G}}(E\cap L(\mathbf{G}))$$

□

To implement behavior of the type just described, we bring in *dynamic state feedback control (DSFBC)*, defined as SFBC in the sense already employed, but now on the state set of $\mathbf{G}\times\mathbf{H}$. Thus a DSFBC is a map $f : Q\times Y \to \Gamma$, with component predicates $f_\sigma : Q\times Y \to \{0,1\}$ for $\sigma\in\Sigma$, just as before. For emphasis we may refer to the pair $F := (f, \mathbf{H})$ as a *DSFBC for* $\mathbf{G}$ and write $L(F/\mathbf{G})$, $L_m(F/\mathbf{G})$ for the corresponding controlled sublanguages of $L(\mathbf{G})$, $L_m(\mathbf{G})$. Finally we say that the DSFBC F is *balanced* if f is balanced as a SFBC for $\mathbf{G}\times\mathbf{H}$ defined on $Q\times Y$. In this terminology we have the following consequence of Theorems 7.4.1 and 7.6.1, and Exercise 7.5.1.

Corollary 7.6.1

Let $E \subseteq \Sigma^*$, E closed, and assume $\sup\mathcal{C}_{\mathbf{G}}(E\cap L(\mathbf{G})) \neq \emptyset$. There exists a balanced DSFBC $F = (f, \mathbf{H})$ for $\mathbf{G}$ such that

$$L(F/\mathbf{G}) = \sup\mathcal{C}_{\mathbf{G}}(E\cap L(\mathbf{G}))$$

□

Exercise 7.6.1: Let $E \subseteq L(\mathbf{G})$ be nonempty and closed. Show that there exist a memory $\mathbf{H}$ for $\mathbf{G}$ and a predicate $P \in Pred(Q\times Y)$ such that (E, P) is compatible with $\mathbf{G}\times\mathbf{H}$. **Hint:** Consider the reachability tree for $L(\mathbf{G})$.

Exercise 7.6.2: For a 'distinguished' event $\alpha\in\Sigma$, let $\mathbf{H}$ model a counter that records the number of occurrences of α, mod 10 say. Suggest some pairs (E, P) that could be compatible with $\mathbf{G}\times\mathbf{H}$.

Exercise 7.6.3: Prove Theorem 7.6.1 and Corollary 7.6.1. **Hint:** Use the results of Exercises 7.4.2 and 7.6.1. $\diamond$

Finally we indicate how the principle of DSFBC can be adapted to modularity. With a view to later applications we assign to languages the primary role of specifications. Let $E_i \subseteq \Sigma^*$ $(i = 1, ..., k)$ be closed languages which we take to be specification languages for the closed behavior of $\mathbf{G}$. Let $E = E_1 \cap ... \cap E_k$. For each i let $\mathbf{H}_i = (Y_i, \Sigma, \eta_i, y_{i,o}, Y_i)$ be a memory for $\mathbf{G}$, and $P_i \in Pred(Q \times Y_i)$ such that (P_i, E_i) is compatible with $\mathbf{G} \times \mathbf{H}_i$. Define $P \in Pred(Q \times Y_1 \times ... \times Y_k)$ according to

$$(q, y_1, ..., y_k) \models P \quad \text{iff} \quad (q, y_i) \models P_i, \qquad i = 1, ..., k$$

Let $F_i = (f_i, \mathbf{H}_i)$ be a balanced DSFBC for $\mathbf{G}$ such that $L(F_i/\mathbf{G}) = \sup \mathcal{C}_{\mathbf{G}}(E_i \cap L(\mathbf{G}))$. Now define $F = (f, \mathbf{H}_1 \times ... \times \mathbf{H}_k)$ according to

$$f_\sigma(q, y_1, ..., y_k) = 1 \quad \text{iff} \quad f_{i,\sigma}(q, y_i) = 1, \qquad i = 1, ..., k$$

With the foregoing conditions in place, we have

Theorem 7.6.2

The DES $\mathbf{H} := \mathbf{H}_1 \times ... \times \mathbf{H}_k$ is a memory for $\mathbf{G}$, the pair (E, P) is compatible with $\mathbf{G} \times \mathbf{H}$, $F := (f, \mathbf{H})$ is a balanced DSFBC for $\mathbf{G}$, and

$$L(F/\mathbf{G}) = L(\mathbf{G} \times \mathbf{H}, \sup \mathcal{CP}_{\mathbf{G}\times\mathbf{H}}(P)) = \sup \mathcal{C}_{\mathbf{G}}(E \cap L(\mathbf{G}))$$

$\square$

Exercise 7.6.4: Prove Theorem 7.6.2, and provide a concrete illustration.

Exercise 7.6.5: Marking and Nonblocking. In this chapter so far, the marker states of $\mathbf{G}$ have played no role, and nonblocking has not been explicitly treated. To complete the story in this respect, define a predicate $P \in Pred(Q)$ to be *nonblocking for* $\mathbf{G}$ if

$$(\forall q \models P)(\exists s \in \Sigma^*)\delta(q, s)! \quad \& \quad \delta(q, s) \in Q_m \quad \& \quad (\forall w \leq s)\delta(q, w) \models P$$

Notice that $P = false$ is trivially nonblocking; in any case, if $q \models P$, there is a path in Q leading from q to some state in Q_m along which P is satisfied. Define a SFBC f for $\mathbf{G}$ to be *nonblocking for* $\mathbf{G}$ if $R(\mathbf{G}^f, true)$ is a nonblocking predicate. Assuming $P \neq false$, show that there is a nonblocking SFBC f for $\mathbf{G}$ such that $R(\mathbf{G}^f, true) = P$, if and only if P is controllable and nonblocking. Next show that the family of nonblocking predicates stronger than a given predicate, say SPEC $\in Pred(Q)$, is closed under arbitrary disjunctions. Then show that the family of predicates that are stronger than SPEC and are both nonblocking and controllable, is closed under arbitrary disjunctions, hence possesses a weakest element, the supremal element of the family. If not *false*, this element will thus determine an optimal nonblocking SFBC that enforces SPEC. Go on to investigate the requirement of nonblocking when SFBC is modular. As might be expected, the conjunction of nonblocking predicates generally fails to be nonblocking: prove this by example. Say two nonblocking predicates are *nonconflicting* if their conjunction is nonblocking. On this basis, develop corresponding refinements of the results of Sects. 7.5 and 7.6. Finally, link the above state-based approach to nonblocking with the linguistic approach of Chapter 3, identifying and proving the relations between dual concepts.

Exercise 7.6.6: Partial state observations

Extend the state feedback control theory of this chapter to the case where the state q is known to the supervisor only modulo some equivalence relation $\rho \in \mathcal{E}(Q)$. Find a necessary and sufficient condition that $P \in Pred(Q)$ can be synthesized by SFBC. Hint: Consider (among other things) the 'observability' condition

$$(\forall q, q' \models P)(\forall \sigma \in \Sigma_c) q \equiv q' (\mathrm{mod} \rho) \ \& \ \delta(q, \sigma)! \ \& \ \delta(q', \sigma)!$$

$$\& \ \delta(q, \sigma) \models P \Rightarrow \delta(q', \sigma) \models P$$

7.7 Notes and References

Predicate transformers were introduced by E.W. Dijkstra [1976], and first applied to DES control theory in [J06, J07]. This chapter is based on the latter references together with work of Y. Li [C24, C27, C29, T17, J21]. The two industrial examples to follow are adapted from Sørensen et al. [1993] and Falcione & Krogh [1993] respectively.

7.8 Appendix: Two Industrial Examples

In this appendix we sketch two exercises on DES modelling by predicates, specifically in a boolean formalism where the state structure is determined by a list of independent propositional variables. For complementary details the reader may consult the cited references.

Exercise 7.8.1: Gas Burner (Sørensen et al. [1993]): The process state is given by the simultaneous truth values of 5 boolean variables; state transitions are triggered by any of 10 distinct events. Some of these are:

predicate HR	=	"heat is required"
X	=	"an explosion has occurred (system is in an exploded condition)"
event 'on'	=	"system on-button is pushed"
'ex'	=	"system explodes"

Transitions are determined by rules of two types: enablement rules ENR and next-state rules NSR. An example of ENR is: Event 'on' can occur iff predicate HR = *false*; while NSR might be: 'on' takes any state with HR = *false* into the corresponding state with HR = *true*. To each event there may be associated several rules, corresponding to different subsets of states. Multiple rules should be checked for consistency, to ensure transitions are deterministic.

The rule base is easily encoded into a case-statement associating with each event an if-then clause in terms of boolean state components. From this the one-step state transition matrix (32×10) is computed once for all; then the reduced transition structure over the reachability set can be generated by depth-first search. [This naive monolithic approach is exponentially complex in the number of predicates and is not recommended for large systems!]

1. Referring to the cited paper for ENR and NSR, carry out this procedure for the gas burner example, and verify the authors' state transition model, writing your own program to calculate the reachability set. You should get a final state size of just 9. In the notation of the paper, label events as follows:

on	off	df1	df2	df3	df4	sr	ff	cr	ex
0	1	2	3	4	5	6	7	8	9

   and encode state components according to

HR	D	F	B	X
0	1	2	3	4

   Each state component will take values 1 or 0, corresponding to *true* or *false*. States are numbered from 0 to 31 corresponding to all possible boolean combinations of the vector [HR D F B X], evaluated as

   $$HR \star 1 + D \star 2 + F \star 4 + B \star 8 + X \star 16$$

   For instance, the initial state is $2 = [0\ 1\ 0\ 0\ 0]$.

2. Extend the foregoing modelling procedure to include control, e.g. RW disablement, assuming for instance that $\Sigma_c = \{\text{on,off,sr,cr}\}$. Comment on whether or not the system could then be safely turned on and, if not, suitably modify the design, for instance by using forcible events in the sense of Sect.3.8. Generalize the approach to take into account that sometimes additional memory (an extended state space) is required to represent the control specifications, namely when the latter involve conditions on the past behavior of the process. For instance, "Stop the process after 10 sensor failures" could be enforced with the aid of an auxiliary counter.

Exercise 7.8.2: Neutralization System (Falcione & Krogh [1993]): A neutralization system in an industrial chemical process could consist of a reaction tank, heater, mixer, and valves for filling and emptying the tank. The plant seen by the programmable logic controller (PLC) consists of boolean variables to represent the operator setting (process off or on), fluid level, temperature and pressure. The controller variables are boolean variables for the valve states (closed or open), temperature and pH indicator lights (off or on), and heater and mixer states (off or on). Recall that pH is a measure of alkalinity; i.e. if pH is too low, more neutralizer is added. In detail the plant state variables are the following:

$$\begin{array}{lll} x0 & = \text{start} = 1 \text{ when} & \text{process ON} \\ x1 & = ls1 & \text{level} \geq \text{level1} \\ x2 & = ls2 & \text{level} \geq \text{level2} \\ x3 & = ls3 & \text{level} \geq \text{level3} \\ x4 & = ts & \text{temp} \geq \text{OK} \\ x5 & = as & \text{pH} \geq \text{OK} \end{array}$$

Here level1 $\leq$ level2 $\leq$ level3, i.e. the possible level combinations are $(x1, x2, x3) = (0, 0, 0)$, $(1, 0, 0), (1, 1, 0), (1, 1, 1)$.

The controller state variables are:

$$\begin{array}{rll} u1 = v1 & = 1 \text{ iff} & \text{valve1 is OPEN (tank fluid feed from reservoir)} \\ u2 = m & & \text{mixer is ON} \\ u3 = h & & \text{heater is ON} \\ u4 = tl & & \text{temp indicator light is ON} \\ u5 = v4 & & \text{valve4 is OPEN (tank fluid drain back to reservoir)} \\ u6 = v2 & & \text{valve2 is OPEN (neutralizer feed)} \\ u7 = al & & \text{pH indicator light is ON} \\ u8 = v3 & & \text{valve3 is OPEN (tank fluid drain to next tank)} \end{array}$$

The controller state transition structure is given by assignment (:=) statements:

$$\begin{array}{lll} u1 & := & (u1 + x0).\overline{u8}.\overline{x2} \\ u2 & := & (u2 + x2).x1 \\ u3 & := & \overline{u8}.x2.\overline{x4} \\ u4 & := & x2.x4 \\ u5 & := & (u5 + x3).x2 \\ u6 & := & \overline{u5}.\overline{u8}.x2.\overline{x5} \\ u7 & := & x2.x5 \\ u8 & := & (u8 + x2.x4.x5).\overline{u5}.x1 \end{array}$$

Note that + is boolean disjunction (thus 1+1 = 1), $-$ boolean negation and . boolean conjunction.

Initially the process is OFF ($x0 = 0$), the tank is empty ($x1 = x2 = x3 = 0$), temperature is low ($x4 = 0$), pH is unspecified ($x5 = 0$ or 1). In the controller, all valves are CLOSED ($u1 = u5 = u6 = u8 = 0$) and the mixer and heater are OFF ($u2 = u3 = 0$). From the control logic this implies that the indicator lights are both OFF ($u4 = u7 = 0$).

The rules of operation are the following. To start the process, set $x0 := 1$, opening valve1 ($u1 = 1$) which stays open until the tank has filled to level2 ($x2 = 1$). When level2 is reached start the mixer ($u2 := 1$); if the level drops below level1 ($x1 = 0$) stop the mixer ($u2 := 0$). Energize the heater ($u3 := 1$) if the temperature is too low ($x4 = 0$) and the tank level is at least level2 ($x2 = 1$). If pH is too low ($x5 = 0$) open the neutralizer feed valve2 ($u6 := 1$). If the tank becomes full (level $\geq$ level3, i.e. $x3 = 1$) then open valve4 (drain tank fluid back to reservoir, i.e. $u5 := 1$); this will close valve2 ($u6 := 0$) to stop the flow of neutralizer. When the fluid level drops just below level2 ($x2 = 0$), close valve4 ($u5 := 0$). When both temperature and pH are OK ($x4 = x5 = 1$), de-energize the heater ($u3 := 0$) and open valve3 ($u8 := 1$) to drain the tank. When the tank is empty ($x1 = 0$), close valve3 ($u8 := 0$) and restart the process.

While the description specifies how the controller reacts to the plant, no model has been specified as to how the plant responds to the controller, so the control loop is open, and no analysis of controlled behavior is possible. One could experiment with various DES models for the plant. For instance, take the plant to be the shuffle of 3 processes, LEVEL, TEMP and ALK. LEVEL has the 4 states listed above, with events "level_rise" and "level_drop", where "level_rise" is enabled when valve1 or valve2 is OPEN and valve3 and valve4 are CLOSED (i.e. $(u1 + u6).\overline{u5}.\overline{u8}$), while "level_drop" is enabled when valve1 and valve2 are CLOSED and valve3 or valve4 is OPEN (i.e. $\overline{u1}.\overline{u6}.(u5+u8)$). TEMP has 2 states ($x4 = 0$ or 1) with events "temp_rise", enabled when heater is ON ($u3 = 1$) and "temp_drop", enabled when $u3 = 0$. Similarly ALK has states ($x5 = 0$ or 1) with events "alk_rise" enabled when valve2 is OPEN ($u6 = 1$). Notice that what we have here is a simple qualitative model of the plant physics. The example illustrates why there is current interest in qualitative modelling: this is simply high-level discrete aggregation, and (if the model is valid) is all you need for logic control design.

The plant transition triples are:

(level0,level_drop,level0)
(level0,level_rise,level1)
(level1,level_drop,level0)
(level1,level_rise,level2)
(level2,level_drop,level1)
(level2,level_rise,level3)
(level3,level_drop,level2)
(level3,level_rise,level3)

Note that level_drop (resp. level_rise) leaves level0 (resp.level3) unchanged.

In this formalism one can regard the occurrence of an event in a given time window or sampling interval as a boolean variable. Recalling that level0 = (0,0,0), level1 = (1,0,0), level2 = (1,1,0), level3 = (1,1,1), we can bring in new boolean level variables

$$\begin{aligned} l0 &= \overline{x1}.\overline{x2}.\overline{x3} \\ l1 &= x1.\overline{x2}.\overline{x3} \\ l2 &= x1.x2.\overline{x3} \\ l3 &= x1.x2.x3 \end{aligned}$$

with transition assignments

$$\begin{aligned} l0 &:= l0.\overline{\text{level\_rise}} + l1.\text{level\_drop} \\ l1 &:= l1.\overline{\text{level\_rise}}.\overline{\text{level\_drop}} + l0.\text{level\_rise} + l2.\text{level\_drop} \end{aligned}$$

and so on. These rules could be expressed in terms of the xi by the equations

$$\begin{aligned} x1 &= l1 + l2 + l3 \\ x2 &= l2 + l3 \\ x3 &= l3 \end{aligned}$$

The events are enabled or disabled according to the controller and plant states, i.e. events can occur only in control and plant states where they are enabled in accordance with the physical interpretation. Thus the enablement preconditions for rise or drop in level are:

$$\begin{aligned} \text{enable(level\_rise)} &= (u1 + u6).\overline{u5}.\overline{u8} \\ \text{enable(level\_drop)} &= \overline{u1}.\overline{u6}.(u5 + u8) \end{aligned}$$

The transition rules are

$$\begin{aligned} \text{level\_rise} &:= \text{enable(level\_rise).random} \\ \text{level\_drop} &:= \text{enable(level\_drop).}\overline{\text{random}} \end{aligned}$$

where random (= boolean 0 or 1) represents the mechanism of event selection.

Similarly

$$x4 := \overline{x4}.\text{temp\_rise} + x4.\overline{\text{temp\_drop}}$$
$$x5 := \overline{x5}.\text{alk\_rise} + x5$$

with preconditions

$$\begin{aligned} \text{enable(temp\_rise)} &= u3 \\ \text{enable(temp\_drop)} &= \overline{u3} \\ \text{enable(alk\_rise)} &= u6 \end{aligned}$$

Note that we have embodied a physical assumption that if $u3 = 0$ (heater is OFF) then temp_drop is enabled, so temp could drop but need not; whereas if temp is OK and heater is ON then temp_drop is disabled, and the condition $x4 = 1$ is stable. A more refined model could include a temperature controller with ON/OFF thresholds bracketing the setpoint.

The equations can be used for simulation or analysis, with an enabled event chosen randomly at each simulation step and the state and control variables updated. The process can now be considered a standard RW model with the feedback already incorporated: for each state (x, u) certain events will be enabled and exactly one of them will be selected as the next event. Then (x, u) is updated and the process repeats.

In general the model is of the form

x_new := qualitative_physics(x,u,events), u_new := control_logic(x, u).

As always, the total process with state (x, u) is driven by the events constituting the changes in physical state under the action of the plant dynamics and control logic.

The archtypal system of this general kind is a level controller (e.g. water pump or furnace control), where the plant state variables are $x1$, $x2$ with

$$\begin{array}{ll} x1 = 1 \text{ iff} & \text{level} \geq \text{level1} \\ x2 = 1 & \text{level} \geq \text{level2} \end{array}$$

and level2 > level1 as above. The controller variable is u, with $u = 1$ representing that the pump or furnace is ON. The process has a main switch TOGGLE.

Then

$$u := \text{TOGGLE}.(u + \overline{x1}).\overline{x2}$$

namely "keep pumping if [(already pumping or level < level1) and level < level2]". So the pump starts low, stops high, and tries to keep level between level1 and level2. Once the level reaches level2 the pump shuts off, only turning on again if the level drops below level1. Ignoring TOGGLE we have

$$u := f(u, x1, x2), say,$$

with u_new $= 1$ iff $x2 = 0$ and either $u = 1$ or $x1 = 0$.

The "correctness" of such a controller means that the plant state subset $\{(x1, x2)|x1 = 1\}$ is globally attractive, like the regulation condition "tracking error = 0" of standard control theory. Namely one should eventually reach and remain in this subset from any initialization and in the absence of further disturbances. In the absence of disturbances (e.g. leaks or cooling) a trivial plant model shows the attractive state is (1,1) because, when once $u = 1$, the condition $u = 1$ is maintained until $x2 = 1$. If $x2 = 1$ then $x1 = 1$ (by the plant model) and, again by the plant model, in the absence of disturbances $x2$ just remains at 1. So one has to think of the plant model as autonomous except for occasional disturbances, which are then fully corrected, just as in regulation theory. The effect of a disturbance is just to move the plant to some arbitrary initial state. Thus the (undisturbed) plant model has to say that if $x1 = 0$ and u is kept at 1 then eventually $x1 = 1$; if $x1 = 1$ and u is kept at 1 then eventually $x2 = 1$; and if once $x2 = 1$ then (regardless of u) $x2 = 1$ always. One could get u to turn off when $x2 = 1$ by considering that u is controllable (can be disabled) and introducing an overflow condition that is to be prohibited.

In the neutralization process, the assumptions are that temperature (resp. pH) always rises as long as the heater is ON (resp. the neutralizer valve is OPEN); and that level always rises (resp. falls) under appropriate valve conditions of OPEN or CLOSED. One then needs to prove that from the empty tank condition (or possibly other, disturbed, initial conditions) the target temperature and pH are reached with level $\geq$ level1; and that subsequently the tank is emptied in an appropriate way. This could be done either with brute force calculation or by analysis using logic and/or stability (e.g. Liapunov) arguments. Considerations of this sort are explored in the currently popular area of hybrid (mixed discrete/continuous) systems.

Write a simulation program for the neutralization process, exploring various plant models. Develop standard DES (generator/automaton) representations, and investigate supervisory control and stability.

Chapter 8

Supervision of Vector Discrete-Event Systems

8.1 Introduction

In this chapter we specialize the control theory of discrete-event systems developed previously, to a setting of vector addition systems. We adopt the state-based approach of Chapter 7. It is natural to enhance the abstract automaton model by exploiting algebraic regularity of internal system structure when it exists. An obvious instance of such structure is arithmetic additivity over the integers. For instance, the state of a manufacturing workcell might be the current contents of its buffers and the numbers of machines in various modes of operation: thus, when a machine completes a work cycle, the status vector of machines and the vector of buffer contents would be suitably incremented. Similar examples are furnished by various kinds of traffic systems.

System modelling by vector addition systems is a long-standing technique, especially in the setting of Petri nets. For us, however, Petri nets will serve only as a graphical representation tool, and we make no use of net theory as such. Our treatment will be self-contained, using only elementary linear algebra and integer programming.

We first develop the base model, then the feedback synthesis of control-invariant state subsets, along with the appropriate versions of controllability, observability and modularity. These results are illustrated by examples from the literature on manufacturing systems.

8.2 Vector Discrete-Event Systems

Let $\mathbb{Z}$ denote the integers $(..., -1, 0, 1, ...)$ and $\mathbb{N}$ the natural numbers (0,1,2,...). Let $\mathbb{Z}^n$ (resp. $\mathbb{N}^n$) denote the space of n-vectors (i.e. ordered n-tuples) with components in $\mathbb{Z}$ (resp. $\mathbb{N}$), along with vector addition, and scalar multiplication by integers (resp. natural numbers). In algebra, $\mathbb{Z}^n$ so equipped is a 'module over the ring of integers', not a vector space; nevertheless we shall loosely speak of its elements as 'vectors' and use vector space terminology on grounds of familiarity. We shall employ the 'direct sum' operation $\oplus$ to form structures like $\mathbb{Z}^n \oplus \mathbb{Z}^m$ ($\simeq \mathbb{Z}^{n+m}$ under the obvious isomorphism) or $\mathbb{N}^n \oplus \mathbb{Z}^m$. In such cases we may write $x = x' \oplus x''$ to denote the decomposition of x into its natural projections onto the direct summands. If v is an arbitrary vector in some $\mathbb{Z}^k$, we write $v \geq 0$ to mean that each component of v is nonnegative, i.e. that in fact $v \in \mathbb{N}^k$, thought of as embedded in $\mathbb{Z}^k$.

A *vector discrete-event system (VDES)* is a DES $\mathbf{G} = (X, \Sigma, \xi, x_o, X_m)$ defined as in previous chapters (although we make a notational change to X from Q and to ξ from δ), but with vector structure, in the foregoing sense, attached to the state set X and transition function ξ. Thus in general $X = \mathbb{N}^n \oplus \mathbb{Z}^m$, while $\xi : X \times \Sigma \to X$ will always have the additive form:

$$\xi(x, \sigma) = x + e_\sigma$$

for some $e_\sigma \in \mathbb{Z}^{n+m}$, the *displacement vector* corresponding to σ. Writing $x = x' \oplus x''$ as above, we note that in general ξ will be a partial function, defined for just those (x, σ) pairs such that $x' \in \mathbb{N}^n$ and $(x+e_\sigma)' = x'+e'_\sigma \in \mathbb{N}^n$, or briefly $x' \geq 0$, $x'+e'_\sigma \geq 0$; however, no such restriction will apply to the second components x'' and $(x + e_\sigma)''$, in $\mathbb{Z}^m$. In particular if the $\mathbb{N}^n$ summand is absent, so $X = \mathbb{Z}^m$, then ξ will be a total function. By the usual inductive definition, ξ is extended to strings $s \in \Sigma^*$, so from now on we consider $\xi : X \times \Sigma^* \to X$ (pfn).

Let Σ be indexed as $\Sigma = \{\sigma_1, ..., \sigma_k\}$ and write e_i for e_{σ_i}. With $X = \mathbb{N}^n \oplus \mathbb{Z}^m$ as above, write $p := n + m$ and regard x and the e_i as (column) arrays of size $p \times 1$. Bring in the matrix

$$E := [e_1...e_k] \in \mathbb{Z}^{p\times k},$$

the *displacement matrix* for $\mathbf{G}$. Now consider strings $s \in \Sigma^*$. It will be useful to count the occurrences of the various event symbols in s. For this define

$$V : \Sigma^* \to \mathbb{N}^k : s \mapsto [v_1(s)...v_k(s)] \in \mathbb{N}^{k\times 1}$$

where $v_j(s)$ is the number of occurrences of σ_j in s. [Note that we may display a column array as a row, to save space: the definition should resolve any ambiguity, and our vector-matrix operations will always be consistent with the array definitions.] $V(s)$ is the *occurrence vector* of s. $V(\cdot)$ can be regarded as a morphism of monoids ($\mathbb{N}^k$ is an *additive* or *commutative*, monoid), with

$$V(\varepsilon) = 0 \in \mathbb{N}^k, \qquad V(s_1 s_2) = V(s_1) + V(s_2)$$

In this notation we have

$$\xi(x,s)! \Rightarrow \xi(x,s) = x + EV(s)$$

The evaluation of $\xi(x,s)$ depends just on x and $V(s)$, but it makes sense only when $\xi(x,s)$ is defined. With $X = \mathbb{N}^n \oplus \mathbb{Z}^m$ and $x = x' \oplus x''$, $\xi(x,s)!$ if and only if $(x + EV(w))' \geq 0$ for all $w \leq s$, namely nonnegativity of the $\mathbb{N}^n$-projection is preserved for all prefixes of s. Thus it might well be that for certain x and s one has $x' \geq 0$ and $(x + EV(s))' \geq 0$, but for some w, $\varepsilon < w < s$, the nonnegativity condition fails; in that case, $\xi(x,s)$ is not defined.

Remark 8.2.1: A more general definition of VDES which might be preferred results on strengthening the enablement conditions as follows. Suppose $X = \mathbb{N}^n$. Given $e_\sigma \in \mathbb{Z}^n$ as before, let $f_\sigma \in \mathbb{N}^n$ be any vector $\geq \max(0, -e_\sigma)$ (computed componentwise), and declare that $\xi(x,\sigma)!$ iff $x \geq f_\sigma$. This will guarantee that $x + e_\sigma \geq 0$. Alternatively, one can pick vectors $e_\sigma^+, e_\sigma^- \in \mathbb{N}^n$ independently and define $e_\sigma := e_\sigma^+ - e_\sigma^-$, $f_\sigma := e_\sigma^-$; this is equivalent to the usual transition enablement definition for a Petri net allowing selfloops. In this chapter we use only the restricted definition corresponding to the choice $f_\sigma = -e_\sigma$. See, however, Exercise 8.8.2 for how a selfloop can be simulated when it is needed.

8.3 VDES Modelling

Example 8.3.1: FACT#1

Consider a 'factory' consisting of 10 machines, each a DES modelled over the alphabet $\Sigma = \{\alpha, \beta, \lambda, \mu\}$ in the usual way, and displayed as a Petri net in Fig. 8.3.1. We do not distinguish the machines individually, being interested only in the numbers resident in the three states indexed $\{1, 2, 3\}$ corresponding respectively to {Idle, Working, Down}. The state of the factory is then $x \in \mathbb{N}^3$, with $x_i \geq 0$ and $x_1 + x_2 + x_3 = 10$. If it is assumed that two or more machines never make a transition simultaneously, then the possible transitions are $x \mapsto x + e_\sigma$ $(\sigma \in \Sigma)$, with

$$E = [e_\alpha \ e_\beta \ e_\lambda \ e_\mu] = \begin{bmatrix} -1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Thus the effect, if any, of each event on a state component is just to increment or decrement it by 1. The initial state could be taken as $x_o = [10 \ 0 \ 0] \in \mathbb{N}^{3\times 1}$ and a transition is defined if and only if the condition $x \geq 0$ is preserved.

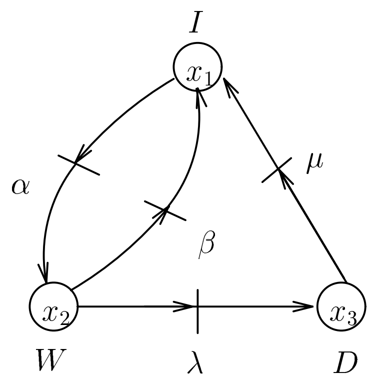


Fig. 8.3.1

Note that, at the expense of additional notation, it would be quite straightforward to model the occurrence of compound events, defined as the simultaneous execution of individual events by distinct machines. Just as before, a compound event would be represented by a suitable displacement vector and restricted by the appropriate nonnegativity condition. For example, if either two or three events α could occur together, label these simultaneous events $\alpha 2$, $\alpha 3$ and bring in the corresponding displacements

$$e_{\alpha 2} = [-2\ 2\ 0], \qquad e_{\alpha 3} = [-3\ 3\ 0]$$

The corresponding transitions are defined just in case, respectively, $x_1 \geq 2$ or $x_1 \geq 3$. $\diamond$

Example 8.3.2: FACT#2

By augmenting the state of **FACT#1** one can bring in 'memory' to record features of past behavior. For instance, let

$$\begin{aligned} x_4 &:= \#\text{ workpieces produced} \\ &= \#\text{ occurrences of event } \beta \text{ since initialization} \end{aligned}$$

Thus x_4 models a counter for β, with initially $x_4 = 0$. Making this extension to **FACT#1** yields the new state space $X = \mathbb{N}^4$ and displacement matrix for **FACT#2**:

$$E = \begin{bmatrix} -1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Note that x_4 may grow without bound as the process evolves, although in an application we might impose a control specification such as $x_4 \leq 100$, perhaps to respect the capacity of a storage buffer. $\diamond$

Example 8.3.3: FACT#3

We extend **FACT#2** to allow a count of the excess of successful work cycles over machine breakdowns, defining $x_5 := \#\beta - \#\lambda$ (where $\#\sigma$ denotes the number of occurrences of σ since initialization). Now we must allow $x_5 \in \mathbb{Z}$ and take for the new state space $X = \mathbb{N}^4 \oplus \mathbb{Z}$. Then, for instance,

$$e_\beta = [1\ \ -1\ 0\ 1\ 1], \quad e_\lambda = [0\ \ -1\ 1\ 0\ \ -1] \in \mathbb{Z}^{5\times 1}$$

and the reader will easily construct E.

Remark 8.3.1: Simultaneous events

We generalize the remark at the end of Example 8.3.1 as follows. Regard the event set $\Sigma = \{\sigma_1, ..., \sigma_m\}$ as a basis set of simple events for the generation of compound events, defined as arbitrary formal linear combinations $\sigma(r_1, ..., r_m) := r_1\sigma_1 + ... + r_m\sigma_m$, where the $r_j \in \mathbb{N}$ (thus any simple event is compound). The full event set, including the null event, is now the set of all compound events. The interpretation of a compound event is just that $r_j \geq 0$ instances of σ_j occur simultaneously, for all $j = 1, ..., m$. In Example 8.3.1, an event $2\alpha + 3\beta + \lambda + 3\mu$ would mean that simultaneously 2 machines start work, 3 end work successfully, 1 breaks down and 3 complete repair. Intuition suggests that this only makes sense if we imagine a small time delay between the 'beginning' of a compound event and its 'termination', so the foregoing event can occur only if the state vector satisfies $x \geq [2\ \ 3+1\ \ 3] = [2\ \ 4\ \ 3]$. In general for $\sigma \in \Sigma$ write $e_j := e_{\sigma_j}$, $e_j^- := \max\{-e_j, 0\}$, and declare that $\xi(x, \sigma(r_1, ..., r_m))!$ iff $x \geq r_1 e_1^- + ... + r_m e_m^-$, in which case

$$\xi(x, \sigma(r_1, ..., r_m)) := x + r_1 e_1 + ... + r_m e_m$$

In this chapter we prohibit (nonsimple) compound events, albeit they are sometimes of interest in applications, especially in problems of mutual exclusion.

Exercise 8.3.1: Show that in Example 8.3.1, $\sigma(r_1, r_2, r_3, r_4)$ is enabled only if $r_1 + r_2 + r_3 + r_4 \leq 10$, so the set of compound events that can actually occur is finite. Exactly how many are there? Find a general formula for the case $r_1 + ... + r_k \leq N$.

Exercise 8.3.2: With $\xi(x, s) = x + EV(s)$ whenever $\xi(x, s)!$, and $E \in \mathbb{Z}^{n\times k}$, interpret solutions $t \in \mathbb{Z}^{k\times 1}$ of $Et = 0$, and solutions $p \in \mathbb{Z}^{1\times n}$ of $pE = 0$. In the terminology of Petri nets, such t are 'transition invariants' and p are 'place invariants'. For an application of the latter see Exercise 8.13.11.

8.4 Linear Predicates

It will be appropriate to require of predicates on X that they be compatible with the algebraic setting, in particular that they be generated by basic predicates that are 'linear' in the state. Such predicates occur commonly in the applications: e.g. in conditions like "either $x_1 \leq 5$ or $x_2 > 10$". For simplicity of notation consider that $X = \mathbb{Z}^n$ (if X has a component $\mathbb{N}^m$ then embed $\mathbb{N}^m \subseteq \mathbb{Z}^m$). Our basic predicates P will be those that distinguish state subsets $X_P \subseteq X$ of the form

$$X_P := \{x = [x_1 ... x_n] \in \mathbb{Z}^n \mid a_1 x_1 + ... + a_n x_n \leq b\}$$

where $a_1, ..., a_n, b \in \mathbb{Z}$. Representing $a \in \mathbb{Z}^{1\times n}$ (a row array) and $x \in \mathbb{Z}^{n\times 1}$ (a column array), we have succinctly,

$$X_P = \{x \in X \mid ax \leq b\}$$

or

$$x \models P \quad \text{iff} \quad ax \leq b$$

Call such P a *linear predicate* on X and denote by $Pred_{\mathbf{lin}}(X)$ the corresponding subset of $Pred(X)$. Finally, let $\overline{Pred_{\mathbf{lin}}(X)}$ be the Boolean closure of $Pred_{\mathbf{lin}}(X)$, namely the smallest subset of $Pred(X)$ that contains $Pred_{\mathbf{lin}}(X)$ and is closed under the Boolean operations $\neg$, $\wedge$ and $\vee$. We have $P \in \overline{Pred_{\mathbf{lin}}(X)}$ if and only if P can be built up by applying the Boolean operations a finite number of times to a finite number of predicates in $Pred_{\mathbf{lin}}(X)$. Thus $P = (x_1 \leq 5) \vee (\neg(x_2 \leq 10))$ for the example above.

Exercise 8.4.1: If $X = \mathbb{N}^n \oplus \mathbb{Z}^m$ and $S \subset X$ is a finite subset, show that the predicate $P = (x \in S)$ belongs to $\overline{Pred_{\mathbf{lin}}(X)}$.

Exercise 8.4.2: Show that $\overline{Pred_{\mathbf{lin}}(X)}$ is a proper Boolean subalgebra of $Pred(X)$.

8.5 State Feedback and Controllability of VDES

In this section we exemplify for VDES the definitions and results of Sect. 7.4. Recalling **FACT#2** in Sect. 8.3 let us assume that items produced (on occurrence of event β) are placed in a buffer of capacity 100. To prevent possible buffer overflow we must maintain the control specification

$$\#\text{ free slots in buffer } \geq \#\text{ machines working}$$

or $100 - x_4 \geq x_2$, i.e. the linear predicate

$$\text{SPEC} := (x_2 + x_4 \leq 100)$$

Assume $\Sigma_c = \{\alpha\}$. Now SPEC is true under the initial condition $x = [10\ 0\ 0\ 0]$. It is easily seen that SPEC is maintained true provided α is enabled only if SPEC holds with positive slack, i.e. when $x_2 + x_4 < 100$. For this define SFBC $f : X \times \Sigma \to \{0, 1\}$ according to

$$f_\alpha(x) = \begin{cases} 1 & \text{if } x_2 + x_4 < 100 \\ 0 & \text{otherwise} \end{cases}$$

Exercise 8.5.1: Draw a graph with vertical axis $x_2 + x_4$, and horizontal axis 'real' time marked with the quasi-random instants of successive events, corresponding to the string $\alpha\ \alpha\ \beta\ \alpha\ \lambda\ \alpha\ \mu\ \beta\ \lambda$.... Next, assuming the process has run long enough to approach the 'control boundary' $x_2 + x_4 = 100$, generate a string to display (on a similar graph) the action of the SFBC f.

Exercise 8.5.2: Reasoning *ad hoc*, show that f enforces SPEC with minimal restriction placed on the behavior of **FACT#2**. ◇

We shall be using the theory of Sect. 7.4 to solve problems of the foregoing type systematically. We begin by illustrating reachability and controllability for VDES.

Example 8.5.1: FACT#4

We simplify **FACT#2** by eliminating the breakdown/repair feature, taking $\Sigma = \{\alpha, \beta\}$, $\Sigma_c = \{\alpha\}$ as shown in Fig. 8.5.1. In **FACT#4** $X = \mathbb{N}^3$, with $x = [x_1, x_2, x_3] \in \mathbb{N}^{3\times 1}$; x_1, x_2 are the number of machines idle (I) and working (W) respectively; x_3 is the number of occurrences of β since initialization; and the initial state is $x_o = [10\ 0\ 0]$. Thus

$$E = [e_\alpha\ e_\beta] = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}$$

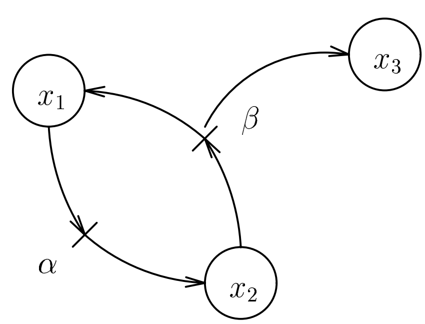


Fig. 8.5.1

We first consider the predicate $P \in Pred(X)$ given by

$$x \models P \quad \text{iff} \quad (x_1 + x_2 = 10) \quad \& \quad (x_3 \leq 100)$$

P amounts to our previous buffer constraint, conjoined with the 'obvious' invariant on the total number of machines in the system, included here for technical convenience.

To investigate whether P is controllable, we start by checking the condition

$$P \preceq R(\mathbf{FACT\#4}, P) \qquad (?)$$

Let $x \models P$, i.e. $x = [10 - j \ j \ k]$ for some $j, k : 0 \leq j \leq 10$, $0 \leq k \leq 100$. We attempt to construct a string that will lead **FACT#4** from x_o to x while preserving P. For instance, let $s := (\alpha\beta)^k\alpha^j$, corresponding to k successful work cycles followed by j machine transitions from I to W. It is clear that $x = \xi(x_o, s)!$. We claim:

$$(\forall w \leq s) y := \xi(x_0, w)! \quad \& \quad y \models P$$

In fact it is easy to verify $\xi(x_0, w)!$ and $y \models P$ for strings w of each of the prefix types

$$\begin{array}{ll} w = (\alpha\beta)^l, & 0 \leq l \leq k-1 \\ w = (\alpha\beta)^l\alpha, & 0 \leq l \leq k-1 \\ w = (\alpha\beta)^k\alpha^l, & 0 \leq l \leq j \end{array}$$

This proves the claim, and thus (?) above. Next we check

$$(\forall \sigma \in \Sigma_u) P \preceq M_\sigma(P) \qquad (??)$$

Let $\sigma = \beta$ and $x = [9 \ 1 \ 100]$. We have $x \models P$, $\xi(x, \beta)!$ and

$$y := \xi(x, \beta) = x + e_\beta = [10 \ 0 \ 101]$$

Since *not* $y \models P$, (??) fails, so P is not controllable.

Exercise 8.5.3: Explain intuitively why P is not controllable, i.e. how P might fail under the occurrence of an uncontrollable event. $\diamond$

Now consider the alternative predicate $Q \in Pred(X)$ given by

$$x \models Q \quad \text{iff} \quad (x_1 + x_2 = 10) \quad \& \quad (x_3 \leq 100 - x_2)$$

In this case $x \models Q$ if and only if $x = [10 - j \ j \ k]$ where $0 \leq j \leq 10$, $0 \leq k \leq 100 - j$. Taking $s = (\alpha\beta)^k\alpha^j$ we verify $x \models R(\mathbf{G}, Q)$ as shown above for P. To check $Q \preceq M_\beta(Q)$ observe that

$$\begin{array}{lll} \xi(x, \beta)! & \text{iff} & x + e_\beta \geq 0 \\ & \text{iff} & [10 - j + 1 \quad j - 1 \quad k + 1] \geq 0 \\ & \text{iff} & j \geq 1 \end{array}$$

i.e. x must satisfy

$$0 \le k \le 100 - j, \qquad 1 \le j \le 10$$

and then

$$y := \xi(x, \beta) = [10 - l \;\; l \;\; m]$$

where $l = j - 1$, $m = k + 1$. Since $0 \le l \le 9$ and

$$0 \le m \le 1 + (100 - j) = 1 + (100 - (l + 1)) = 100 - l$$

we get that $y \models Q$. It follows that Q is controllable.

According to Theorem 7.4.1 the controllability of Q guarantees the existence of a SFBC f such that $Q = R(f/\mathbf{FACT\#4})$, and in fact the proof showed that f can be defined by

$$f_\sigma := M_\sigma(Q), \qquad \sigma \in \Sigma_c$$

With $\Sigma_c = \{\alpha\}$ we get

$$f_\alpha(x) = 1 \quad \text{iff} \quad \begin{cases} \text{either } \textit{not}\ \xi(x, \alpha)! \\ \text{or } \xi(x, \alpha)!\ \&\ \xi(x, \alpha) \models Q \end{cases}$$

Now

$$\begin{aligned} \xi(x, \alpha)! \;\; &\text{iff} \;\; x + e_\alpha \ge 0 \\ &\text{iff} \;\; [10 - j \; j \; k] + [-1 \; 1 \; 0] \ge 0 \\ &\text{iff} \;\; [10 - j - 1 \;\; j + 1 \;\; k] \ge 0 \\ &\text{iff} \;\; (-1 \le j \le 9)\ \&\ (k \ge 0) \end{aligned}$$

For $\xi(x, \alpha) \models Q$ we then require

$$0 \le k \le 100 - (j + 1)$$

or

$$f_\alpha(x) = 1 \quad \text{iff} \quad \begin{cases} \text{either } j = 10 \\ \text{or } (j \le 9)\ \&\ (0 \le k \le 100 - (j + 1)) \end{cases}$$

For instance, this control would enable α if $j = 0$ and $k \le 99$, but disable α if $j = 1$ and $k = 99$. Notice that the SFBC synthesizing Q is far from unique: for instance, there is no need to enable α if $j = 10$, as there are then no machines idle. It is clear that the way $f_\sigma(x)$ is defined when *not* $\xi(x, \sigma)!$ may in principle be arbitrary. ◇

From these examples it appears that direct calculation of reachable sets can become rather involved, even for the simplest VDES models. On the other hand, as far as P-invariance is concerned, the calculation need be done only along sequences of uncontrollable events. In the next section we explore this issue in more detail.

8.6 Reachability and Loop-Freeness

Let $\mathbf{G} = (X, \Sigma, \xi, x_o, X_m)$ be a VDES. In later application of the results of this section, $\mathbf{G}$ will be taken as 'the uncontrollable subsystem of the plant', to be defined in due course. For now, we focus generally on how the components of the vector x are successively incremented and decremented under the occurrence of events. Let $X = \mathbb{Z}^n$ and $x = [x_1 ... x_n]$. This coordinatization of X will be fixed and the subsequent definitions will depend on it. The corresponding displacement vectors will be written $e_\sigma = [e_\sigma(1) ... e_\sigma(n)]$, with the $e_\sigma(i) \in \mathbb{Z}$. Write $I = \{1, ..., n\}$ and for $\sigma \in \Sigma$ define

$$\begin{aligned} \sigma^\uparrow &:= \{i \in I \mid e_\sigma(i) < 0\} \\ \sigma^\downarrow &:= \{i \in I \mid e_\sigma(i) > 0\} \end{aligned}$$

Thus for $i \in \sigma^\uparrow$ the components x_i of x are negatively incremented (i.e. positively decremented) by the occurrence of σ, while for $i \in \sigma^\downarrow$ the x_i are positively incremented. The index subsets $\sigma^\uparrow$ (resp. $\sigma^\downarrow$) can be visualized as (labelling) the components of x that are 'upstream' (resp. 'downstream') from σ; the $\sigma^\uparrow$-labelled x-components act as 'source' state variables for σ, while the $\sigma^\downarrow$ act as 'sinks'. Dually, for $i \in I$ define

$$\begin{aligned} i^\uparrow &:= \{\sigma \in \Sigma \mid e_\sigma(i) > 0\} \\ i^\downarrow &:= \{\sigma \in \Sigma \mid e_\sigma(i) < 0\} \end{aligned}$$

The occurrence of an event $\sigma \in i^\uparrow$ positively increments x_i, i.e. σ acts as a source for x_i; while $\sigma \in i^\downarrow$ negatively increments x_i, i.e. σ acts as a sink for x_i. Dually again, $i^\uparrow$ (resp. $i^\downarrow$) represents the subset of events that are upstream (resp. downstream) from the state component x_i. Notice that our notation is necessarily unsymmetrical. With state variables, we need to distinguish between (a) the component index (i), standing for the component 'x_i' as a fixed symbol for a state variable in the ordered n-tuple of state variables, and (b) the (current) value assumed by the state variable x_i, an integer subject to incremental change as the process evolves. For events (labelled) σ no such distinction is applicable.

Observe that $i \in \sigma^\uparrow$ (resp. $\sigma^\downarrow$) if and only if $\sigma \in i^\downarrow$ (resp. $i^\uparrow$).

Let $[\tau_1, ..., \tau_k]$ be a list of elements from Σ, possibly with repetitions, and let $[i_1, ..., i_k]$ be a similar list from I. The interleaved list $L := [\tau_1, i_1, ..., \tau_k, i_k]$ will be called a *loop* in $\mathbf{G}$ if

$$\tau_1 \in i_1^\downarrow, ..., \tau_k \in i_k^\downarrow$$

and

$$i_1 \in \tau_2^\downarrow, \quad i_2 \in \tau_3^\downarrow, ..., i_{k-1} \in \tau_k^\downarrow, \quad i_k \in \tau_1^\downarrow$$

Equivalently the loop relations can be displayed as

$$i_1 \in \tau_2^\downarrow, \;\; \tau_2 \in i_2^\downarrow, ..., i_{k-1} \in \tau_k^\downarrow, \;\; \tau_k \in i_k^\downarrow, \;\; i_k \in \tau_1^\downarrow, \;\; \tau_1 \in i_1^\downarrow$$

If no loop in $\mathbf{G}$ exists, $\mathbf{G}$ is *loop-free*.

Bring in the *state-variable source subset*

$$I^{\uparrow} := I - \cup\{\sigma^{\downarrow} \mid \sigma \in \Sigma\}$$

Thus for $i \in I^{\uparrow}$, x_i is never positively incremented by the occurrence of an event, namely x_i can only stay constant or decrease in value as the DES $\mathbf{G}$ evolves.

Lemma 8.6.1

Assume $\Sigma \neq \emptyset$. If $\mathbf{G}$ is loop-free then for some $\sigma \in \Sigma$, $\sigma^{\uparrow} \subseteq I^{\uparrow}$.

Proof

Suppose the contrary, namely

$$(\forall \sigma \in \Sigma)\sigma^{\uparrow} - I^{\uparrow} \neq \emptyset$$

Pick $\tau_1 \in \Sigma$ arbitrarily and let $i_1 \in \tau_1^{\uparrow} - I^{\uparrow}$. Then $i_1 \notin I^{\uparrow}$ implies that $i_1 \in \tau_2^{\downarrow}$ for some $\tau_2 \in \Sigma$. Since $\tau_2^{\uparrow} - I^{\uparrow} \neq \emptyset$, we pick $i_2 \in \tau_2^{\uparrow} - I^{\uparrow}$, and then $i_2 \in \tau_3^{\downarrow}$ for some $\tau_3 \in \Sigma$. Continuing this process we obtain a sequence $\tau_1, i_1, \tau_2, i_2, ..., \tau_j, i_j, ...$ such that

$$i_j \in \tau_j^{\uparrow} \cap \tau_{j+1}^{\downarrow}, \qquad j = 1, 2, ...$$

or equivalently

$$\tau_j \in i_j^{\downarrow}, \quad i_j \in \tau_{j+1}^{\downarrow}, \qquad j = 1, 2, ...$$

Since the index set I is finite ($|I| = n$) we may select the least $j > 1$, say $j = k + 1$ ($k \geq 1$), such that $i_j = i_l$ for some $l < j$, and without loss of generality assume $l = 1$. Then we have

$$i_1 \in \tau_2^{\downarrow}, \quad \tau_2 \in i_2^{\downarrow}, ..., i_{k-1} \in \tau_k^{\downarrow}, \quad \tau_k \in i_k^{\downarrow}, \quad i_k \in \tau_{k+1}^{\downarrow}, \quad \tau_{k+1} \in i_1^{\downarrow}$$

This states that

$$L := [\tau_{k+1}, i_1, \tau_2, i_2, ..., \tau_k, i_k]$$

is a loop in $\mathbf{G}$, contrary to hypothesis. □

We shall need the idea of a 'subsystem' of $\mathbf{G}$ obtained by picking out a subset of the components of the state vector and a subset of events. With I as before, let $\hat{I} \subseteq I$, and let $\hat{\Sigma} \subseteq \Sigma$. The corresponding *subsystem* $\hat{\mathbf{G}}$ of $\mathbf{G}$ is

$$\hat{\mathbf{G}} := (\hat{X}, \hat{\Sigma}, \hat{\xi}, \hat{x}_o, \hat{X}_m)$$

where $\hat{X}$, $\hat{x}_o$, $\hat{X}_m$ are the natural projections of X, x_o, X_m on the components with indices in $\hat{I}$, and $\hat{\xi}$ is the restriction of ξ:

$$\hat{\xi}: \; \hat{X} \times \hat{\Sigma} \to \hat{X}: \; \hat{x} \mapsto \hat{x} + \hat{e}_\sigma \qquad \text{(pfn)}$$

With $\sigma \in \hat{\Sigma}$ we declare $\hat{\xi}(\hat{x}, \sigma)!$ whenever $\xi(x, \sigma)!$ for some x with projection $\hat{x}$. For instance, if $\hat{X} = \mathbb{N}^m$, $\hat{\xi}(\hat{x}, \sigma)!$ provided $\hat{x} \geq 0$ and $\hat{x} + \hat{e}_\sigma \geq 0$. Thus $\hat{\mathbf{G}}$ is indeed a VDES.

Next recall the definition of the closed behavior generated by $\mathbf{G}$ corresponding to initialization at an arbitrary state:

$$L(\mathbf{G}, x) := \{s \in \Sigma^* \mid \xi(x, s)\, !\}$$

Lemma 8.6.2

Let $\hat{\mathbf{G}}$ be a subsystem of $\mathbf{G}$ obtained by removing one or more elements of Σ, but keeping $\hat{X} = X$. Let $s \in L(\mathbf{G}, x)$, $x' := \xi(x, s)$, and $\hat{s} \in \hat{\Sigma}^*$. Then

$$\hat{s} \in L(\hat{\mathbf{G}}, x') \quad \text{iff} \quad s\hat{s} \in L(\mathbf{G}, x)$$

□

Lemma 8.6.3

Let $X = \mathbb{N}^n$, $\sigma \in \Sigma$, $k \in \mathbb{N}$. Then

$$x + ke_\sigma \geq 0 \Rightarrow \sigma^k \in L(\mathbf{G}, x)$$

Proof

The statement is true for $k = 0$. Assume inductively that it is true for $k \leq l$, and let $x + (l+1)e_\sigma \geq 0$. Clearly

$$x' := x + le_\sigma \geq 0$$

so $\sigma^l \in L(\mathbf{G}, x)$ and $x' = \xi(x, \sigma^l)!$. Also $x' + e_\sigma \geq 0$ implies $\xi(x', \sigma)!$, so $\sigma^{l+1} \in L(\mathbf{G}, x)$, as required. □

Lemma 8.6.4

Let $X = \mathbb{N}^n$, $x \in X$ and $x + \Sigma\{k_\sigma e_\sigma \mid \sigma \in \Sigma\} \in X$ for some $k_\sigma \in \mathbb{N}$. For some $\tau \in \Sigma$ assume $\tau^\uparrow \subseteq I^\uparrow$. Then $x + k_\tau e_\tau \in X$.

Proof

If $i \in I^{\uparrow}$ then $\Sigma\{k_\sigma e_\sigma(i) \mid \sigma \neq \tau\} \leq 0$, so that

$$x_i + k_\tau e_\tau(i) \geq x_i + \Sigma\{k_\sigma e_\sigma(i) \mid \sigma \in \Sigma\} \geq 0$$

If $i \notin I^{\uparrow}$ then $i \notin \tau^{\uparrow}$, so $k_\tau e_\tau(i) \geq 0$, and again $x_i + k_\tau e_\tau(i) \geq 0$. □

For the remainder of this section we assume $\Sigma = \{\sigma_1, ..., \sigma_m\}$ and that $X = \mathbb{N}^n$, namely all states x satisfy $x \geq 0$. Recalling from Sect. 8.6.1 the definition of the occurrence vector $V(\cdot)$, we know that for $s \in \Sigma^*$, $x \in X$,

$$s \in L(\mathbf{G}, x) \Rightarrow \xi(x, s) = x + EV(s) \geq 0$$

Our main result states that, under the condition of loop freeness, this implication can be reversed.

Theorem 8.6.1

Assume $\mathbf{G}$ is loop-free. Then for every $x \in \mathbb{N}^{n\times 1}$ and $v \in \mathbb{N}^{m\times 1}$,

$$(\exists s \in L(\mathbf{G}, x))V(s) = v \quad \text{ iff } \quad x + Ev \geq 0$$

Proof

(Only if) The result is immediate by $\xi(x, s)!$ and $\xi(x, s) = x + EV(s)$.

(If) Let $x \geq 0$, $v \geq 0$, $x + Ev \geq 0$. We may write $v(i)$ for v_i if convenient. By Lemma 8.6.1, $\sigma^{\uparrow} \subseteq I^{\uparrow}$ for some $\sigma \in \Sigma$, say σ_1. Lemma 8.6.4 (with $k_\sigma = v_\sigma$) yields $x + v_1 e_1 \geq 0$. With $s_1 := \sigma_1^{v(1)}$ Lemma 8.6.3 gives $s_1 \in L(\mathbf{G}, x)$, so $\xi(x, s_1)!$ and $x_1 := \xi(x, s_1) \geq 0$. Let $\hat{\mathbf{G}}$ be the subsystem of $\mathbf{G}$ obtained by removing σ_1 (but keeping $\hat{X} = X$), so $\hat{\Sigma} := \Sigma - \{\sigma_1\}$. Let $\hat{I}^{\uparrow}$ be the state-variable source subset for $\hat{\mathbf{G}}$. It is clear that $\mathbf{G}$ loop-free implies $\hat{\mathbf{G}}$ loop-free, so (if $\hat{\Sigma} \neq \emptyset$) we pick $\sigma \in \hat{\Sigma}$, say $\sigma = \sigma_2$, with $\sigma^{\uparrow} \subseteq \hat{I}^{\uparrow}$. Now we have

$$x_1 + \sum_{i=2}^{m} v_i e_{\sigma_i} = x + \sum_{i=1}^{m} v_i e_{\sigma_i} = x + Ev \geq 0$$

By Lemma 8.6.4, $x_1 + v_2 e_{\sigma_2} \geq 0$; Lemma 8.6.3 gives $s_2 := \sigma_2^{v(2)} \in L(\hat{\mathbf{G}}, x_1)$, and by Lemma 8.6.2, $s_1 s_2 \in L(\mathbf{G}, x)$. So $x_2 := \xi(x, s_1 s_2)!$ and $x_2 \geq 0$. Continuing in this way we get finally

$$s := s_1 s_2 ... s_m \in L(\mathbf{G}, x)$$

with

$$s_i = \sigma_i^{v(i)}, \qquad i = 1, ..., m$$

and $V(s) = v$. □

8.7 Loop-Freeness and Optimal Control

Let $\mathbf{G}$ be a VDES as before, with $X = \mathbb{N}^n$. In this section we apply Theorem 8.6.1 to obtain an (often) efficient way to compute SFBC 'on-line', whenever the specification predicate is linear, and $\mathbf{G}$ satisfies a condition of loop-freeness. Formally, let $\hat{\Sigma} = \Sigma_u$, $\hat{X} = X$ and define $\mathbf{G}_u$ $(:= \hat{\mathbf{G}})$ to be the *uncontrollable subsystem* of $\mathbf{G}$. Let $P \in Pred_{\mathbf{lin}}(X)$, with

$$x \models P \quad \text{iff} \quad ax \leq b$$

for some $a \in \mathbb{Z}^{1\times n}$, $b \in \mathbb{Z}$. We arrange the indexing so that $\Sigma_u = \{\sigma_1, ..., \sigma_m\}$, with

$$E_u = [e_1 ... e_m] \in \mathbb{Z}^{n\times m}$$

We write $|s|_i$ to denote the number of occurrences of σ_i in s, and for $s \in \Sigma_u^*$ bring in the occurrence vector

$$V_u(s) := [|s|_1 ... |s|_m] \in \mathbb{N}^{m\times 1}$$

Recalling the characterization $\sup \mathcal{CP}(P) = R(\mathbf{G}, \langle P\rangle)$ of Proposition 7.4.2, our first task is to calculate $\langle P\rangle$. Using the fact that $L(\mathbf{G}_u, x)$ is prefix-closed, we have that $x \models \langle P\rangle$ if and only if

$$(\forall s \in L(\mathbf{G}_u, x))\xi(x, s) \models P$$

$$\begin{aligned}
\text{iff} \quad (\forall s \in L(\mathbf{G}_u, x))x + E_u V_u(s) &\models P \\
\text{iff} \quad (\forall s \in L(\mathbf{G}_u, x))ax + aE_u V_u(s) &\leq b \\
\text{iff} \quad ax + \max\{aE_u V_u(s) \mid s \in L(\mathbf{G}_u, x)\} &\leq b
\end{aligned}$$

In general the indicated maximization problem may be intractable, a fact which makes the following result especially interesting.

Proposition 8.7.1

If $\mathbf{G}_u$ is loop-free, then $x \models \langle P\rangle$ if and only if

$$ax + cv^*(x) \leq b$$

Here $c := aE_u \in \mathbb{Z}^{1\times m}$, and $v^*(x)$ is a solution of the integer linear programming problem:

$$cv = \text{maximum}$$

with respect to $v \in \mathbb{Z}^{m\times 1}$ such that $v \geq 0$ and $E_u v \geq -x$.

Proof

By Theorem 8.6.1 applied to $\mathbf{G}_u$,

$$\{V_u(s) \mid s \in L(\mathbf{G}_u, x)\} = \{v \in \mathbb{N}^m \mid x + E_u v \geq 0\}$$

Therefore

$$\max\{aE_uV_u(s) \mid s \in L(\mathbf{G}_u, x)\} = cv^*(x),$$

and the result for $\langle P \rangle$ follows as claimed. □

We can now provide our final result, on computation of the optimal control.

Theorem 8.7.1

Assume $\mathbf{G}_u$ is loop-free. An optimal SFBC f^*, if one exists, enforcing the linear predicate $P := (ax \leq b)$, is given for $\sigma \in \Sigma_c$ by the formula:

$$f^*_\sigma(x) = \begin{cases} 1 \text{ if } \xi(x,\sigma)! \text{ and } ax_{\text{new}} + cv^*(x_{\text{new}}) \leq b \text{ (where } x_{\text{new}} := x + e_\sigma) \\ 0 \text{ otherwise} \end{cases}$$

Furthermore, f^* so defined is balanced.

Proof

The fact that f^* optimally enforces P follows immediately from Sect. 7.4 and Proposition 8.7.1. The property that f^* is balanced results by construction: $f^*_\sigma(x) = 1$ whenever both $x \in R(\mathbf{G}, \langle P \rangle)$ and $\xi(x,\sigma)!$ with $\xi(x,\sigma) \in R(\mathbf{G}, \langle P \rangle)$. □

We remark that $\xi(x,\sigma)!$ just when $x_{\text{new}} \geq 0$, by VDES dynamics. If *not* $\xi(x,\sigma)!$ then $f^*_\sigma(x)$ can in principle be defined arbitrarily.

Corollary 8.7.1

Assume $\mathbf{G}_u$ is loop-free. An optimal SFBC f^* exists iff

$$ax_o + cv^*(x_o) \leq b$$

If this condition fails, then no SFBC exists for $\mathbf{G}$ that enforces P.

Proof

By the results of Sect. 7.4 an optimal control f^* exists iff $\sup \mathcal{CP}(P) \neq false$, and this is true iff $x_o \models \sup \mathcal{CP}(P)$. Since $\sup \mathcal{CP}(P) = R(\mathbf{G}, \langle P \rangle)$, f^* exists iff $x_o \models \langle P \rangle$, and the assertion follows by Proposition 8.7.1. □

Exercise 8.7.1: For a given VDES $\mathbf{G}$, suppose $\mathbf{G}_u$ is not loop-free, but you decide to use the linear integer programming method of Theorem 8.7.1 anyway, because it is computationally convenient. Could such a control design violate the specification? Either prove it could not or find an example to show that it may. If it does not, its only fault might be that it is overly conservative. In that case, create an example to illustrate.

8.8 Example: FACT#5

We consider the small factory with Petri net shown below, where a group of 10 input machines feeds a buffer, which in turn supplies a group of 5 machines at the output. Let $I1$, $W1$, $D1$ denote the numbers of input machines in state I, W, D, with a similar notation for output machines, and let B denote the number of items in the buffer. We define the state vector x as

$$x := [I1 \;\; W1 \;\; D1 \;\; I2 \;\; W2 \;\; D2 \;\; B] \in \mathbb{N}^{7\times 1}$$

with

$$x_o = [10 \;\; 0 \;\; 0 \;\; 5 \;\; 0 \;\; 0 \;\; 0]$$

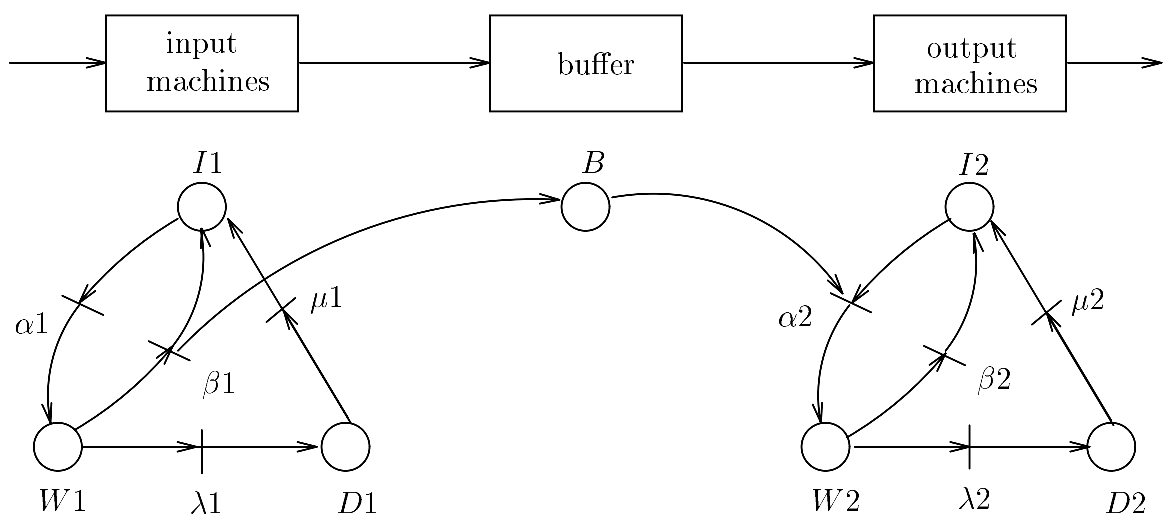


Fig. 8.8.1
Petri net for **FACT #5**

Listing the events in order $(\alpha_1, \beta_1, \lambda_1, \mu_1, \alpha_2, \beta_2, \lambda_2, \mu_2)$ we get the displacement matrix $E \in \mathbb{Z}^{7\times 8}$ displayed below.

$$
\begin{array}{c|cccccccc|}
 & \alpha_1 & \beta_1 & \lambda_1 & \mu_1 & \alpha_2 & \beta_2 & \lambda_2 & \mu_2 \\
I1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
W1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
D1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
I2 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 \\
W2 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 \\
D2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\
B & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0
\end{array}
$$

Taking $\Sigma_u = \{\beta_1, \lambda_1, \beta_2, \lambda_2\}$ and extracting the corresponding submatrix of E results in

$$
E_u = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{Z}^{7\times 4}
$$

It is easy to check – for instance by inspection of its Petri net – that $\mathbf{G}_u$ is loop-free.

Assume that the buffer capacity is 100, and that we undertake to prevent overflow, namely to enforce the predicate $P_{\text{over}} := (ax \leq b)$, where

$$
a := [0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 1] \in \mathbb{Z}^{1\times 7}, \qquad b := 100
$$

This gives

$$
c := aE_u = [1 \;\; 0 \;\; 0 \;\; 0] \in \mathbb{Z}^{1\times 4}
$$

Writing $v := [v_1 \; v_2 \; v_3 \; v_4] \in \mathbb{Z}^{4\times 1}$, we attempt to maximize $cv = v_1$, subject to $v \geq 0$ and $E_u v \geq -x$. With $x = [x_1 ... x_7] \in \mathbb{N}^{7\times 1}$ the constraints become

$$
\begin{aligned}
v_1 &\geq -x_1 \\
-v_1 - v_2 &\geq -x_2 \\
v_2 &\geq -x_3 \\
v_3 &\geq -x_4 \\
-v_3 - v_4 &\geq -x_5 \\
v_4 &\geq -x_6 \\
v_1 &\geq -x_7
\end{aligned}
$$

together with $v \geq 0$. All but the second and fifth of these conditions are enforced by VDES dynamics, which maintain $x \geq 0$. Thus the effective constraints reduce to

$$
v_1 \geq 0, \quad v_2 \geq 0, \quad v_1 + v_2 \leq x_2, \quad v_3 + v_4 \leq x_5
$$

Clearly v_1 is maximized at $v_1 = x_2$, $v_2 = 0$, $v_3 = \omega$, $v_4 = \omega$, where ω denotes 'don't care'. For α_1 the optimal control defined in Sect. 8.7 is therefore

$$\begin{aligned} f^*_{\alpha_1}(x) = 1 \quad &\text{iff} \quad a x_{\text{new}} + c v^*(x_{\text{new}}) \leq b \;\; (x_{\text{new}} := x + e_{\alpha_1}) \\ &\text{iff} \quad (x_7)_{\text{new}} + (x_2)_{\text{new}} \leq 100 \\ &\text{iff} \quad x_7 + (x_2 + 1) \leq 100 \\ &\text{iff} \quad x_2 + x_7 \leq 99 \end{aligned}$$

In words, α_1 is enabled if and only if the number of input machines at work plus the current buffer content is at most one less than the buffer capacity, and this is obviously intuitively correct.

Exercise 8.8.1: Under the same assumptions, investigate how to prevent buffer underflow, namely enforce

$$P_{\text{under}} := (x_7 \geq 0) = (-x_7 \leq 0)$$

Following a similar procedure for α_2, verify that optimal control enables α_2 in all states, namely the only enablement condition for the occurrence of α_2 is $\xi(x, \alpha_2)!$, or $(x_4 \geq 1)$ & $(x_7 \geq 1)$, and this is enforced automatically by VDES dynamics.

Exercise 8.8.2: Selfloop simulation

Consider the additional specification that no input machine should be repaired (i.e. μ_1 is disabled) as long as some output machine is broken down. Writing $\#\sigma$ for the number of occurrences of event σ since initialization, we have for the number of output machines broken down,

$$D2 = x_6 = \#\lambda_2 - \#\mu_2$$

Since μ_1 must be disabled if $D2 > 0$, and up to $I2(0) = x_4(0) = 5$ output machines can be down at one time, $D2 > 0$ means $C := 5 - D2 < 5$. This specification can be modelled using a 'VDES with selfloop' as displayed by the Petri net in Fig. 8.8.2(a).

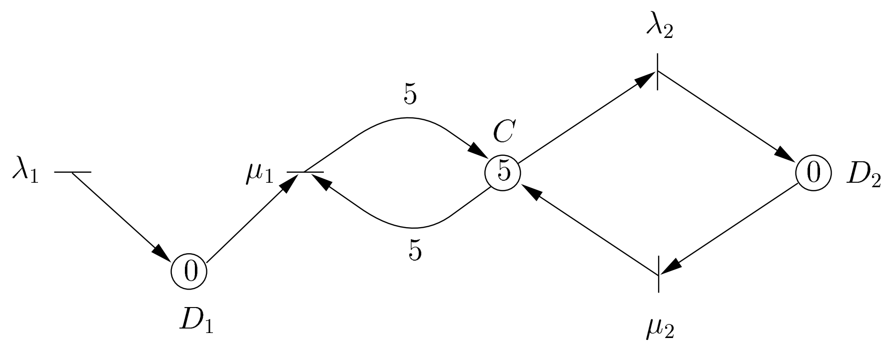


Fig. 8.8.2(a)

To incorporate this specification in our standard model of VDES (from which selfloops are excluded) one may interpose a new coordinate V and uncontrollable transition ν as in Fig. 8.8.2(b).

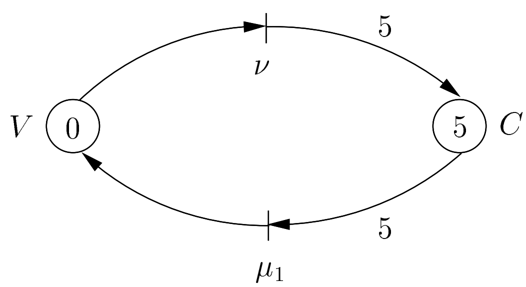


Fig. 8.8.2(b)

At first glance this device may seem unrealistic, since when $C = 0$ and $V = 1$, following an occurrence of μ_1, the (uncontrollable!) event λ_2 will be disabled pending the occurrence of ν. Recall, however, that no timing constraints are imposed, so ν can be assumed to occur arbitrarily soon after its enablement.

(a) More formally, consider the configuration of Fig. 8.8.2(a), defined as a 'generalized' VDES, say **G**, in the sense of Remark 8.2.1. Take for the state vector and initial condition

$$x = [D1 \quad D2 \quad C], \quad x_0 = [0 \quad 0 \quad 5]$$

and for the alphabet $\{\lambda_1, \mu_1, \lambda_2, \mu_2\}$. The enablement conditions for $\lambda_1, \lambda_2, \mu_2$ are just as in a standard VDES, while for μ_1 we define vectors

$$e^+_{\mu_1} = [0 \quad 0 \quad 5], \; e^-_{\mu_1} = [1 \quad 0 \quad 5], \; e_{\mu_1} = [-1 \quad 0 \quad 0]$$

and the enablement condition $x \geq e^-_{\mu_1}$. With the semantics of $\mathbf{G}$ now well-defined, let $L(\mathbf{G})$ be the corresponding closed behavior, of course over $\{\lambda_1, \mu_1, \lambda_2, \mu_2\}$. Next define a VDES $\mathbf{H}$ by incorporating Fig. 8.8.2(b) into Fig. 8.8.2(a). The state vector and initial condition of $\mathbf{H}$ can be taken as

$$x = [D1 \;\; D2 \;\; C \;\; V] \in \mathbb{Z}^{4\times 1}, \quad x_0 = [0 \;\; 0 \;\; 5 \;\; 0]$$

and the alphabet as $\{\lambda_1, \mu_1, \lambda_2, \mu_2, \nu\}$. The displacement matrix E is then

$$E = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & -5 & -1 & 1 & 5 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

With the semantics that of a standard VDES, let the closed behavior be $L(\mathbf{H})$. Finally, if $P : \{\lambda_1, \mu_1, \lambda_2, \mu_2, \nu\}^* \to \{\lambda_1, \mu_1, \lambda_2, \mu_2\}^*$ is the natural projection (erasing ν), show that $L(\mathbf{G}) = PL(\mathbf{H})$.

(b) With **FACT#5** redefined as a new VDES by incorporating the additional structure of the VDES $\mathbf{H}$, in part (a), re-solve for the optimal control to prevent buffer overflow, and compare the result with the control law found in the earlier part of this section.

8.9 Memory and Dynamic State Feedback Control for VDES

We now apply to VDES the constructions of Sect. 7.6. As in Sect. 8.8 assume that the plant VDES $\mathbf{G}$ has state set $\mathbb{N}^n$. To $\mathbf{G}$ we shall adjoin a VDES $\mathbf{H} = (Y, \Sigma, \eta, y_o, Y)$, and in view of the vector structures write $\mathbf{G} \oplus \mathbf{H}$ for the 'direct sum' VDES:

$$\mathbf{G} \oplus \mathbf{H} = (X \oplus Y, \Sigma, \xi \oplus \eta, x_o \oplus y_o, X_m \oplus Y)$$

Typically the state vector $y \in Y$ will play the role of a memory variable (as distinct from a material variable like 'numbers of machines'), and so we shall take $Y = \mathbb{Z}^p$ for some $p > 0$. The displacement vector for $\mathbf{H}$ corresponding to σ will be denoted by $h_\sigma \in \mathbb{Z}^{p\times 1}$; the corresponding displacement vector in $\mathbf{G} \oplus \mathbf{H}$ is $e_\sigma \oplus h_\sigma \in \mathbb{Z}^{(n+p)\times 1}$, with

$$(\xi \oplus \eta)(x \oplus y, \sigma)! \quad \text{iff} \quad \xi(x, \sigma)! \quad \text{iff} \quad x + e_\sigma \geq 0$$

In other words, the memory $\mathbf{H}$ places no additional 'physical' constraint on the transitions of $\mathbf{G} \oplus \mathbf{H}$.

As before let $\Sigma = \{\sigma_1, ..., \sigma_m\}$. We define a *linear dynamic predicate* to be an element $P_{\text{dyn}} \in Pred_{\mathbf{lin}}(\mathbb{N}^m)$:

$$v = [v_1 ... v_m] \models P_{\text{dyn}} \quad \text{iff} \quad \sum_{i=1}^{m} c_i v_i \leq d$$

where the c_i and $d \in \mathbb{Z}$, i.e.

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

where $c := [c_1, ..., c_m] \in \mathbb{Z}^{1\times m}$. For the behavior of $\mathbf{G}$ subject to P_{dyn}, bring in

$$L(\mathbf{G}, P_{\text{dyn}}) := \{s \in L(\mathbf{G}) \mid (\forall w \leq s)V(w) \models P_{\text{dyn}}\}$$

With P_{dyn} we associate the memory $\mathbf{H}$ as above, where

$$Y := \mathbb{Z}, \quad y_o := d, \quad \eta(y, \sigma_i) := y - c_i, \quad (i = 1, ..., m)$$

It should now be clear that enforcing $s \in L(\mathbf{G}, P_{\text{dyn}})$ is tantamount to enforcing the predicate $y \geq 0$ in $\mathbf{G} \oplus \mathbf{H}$. Formally, define $P_{\text{sta}} \in Pred_{\mathbf{lin}}(X \oplus Y)$ according to

$$x \oplus y \models P_{\text{sta}} \quad \text{iff} \quad -y \leq 0$$

Then we have

Lemma 8.9.1

$$L(\mathbf{G} \oplus \mathbf{H}, P_{\text{sta}}) = L(\mathbf{G}, P_{\text{dyn}})$$

$\square$

By Theorem 7.6.1 and Lemma 8.9.1, we have the main result, in the notation of Sect. 7.6, as follows.

Theorem 8.9.1

Let f^* be an optimal SFBC enforcing P_{sta} in $\mathbf{G} \oplus \mathbf{H}$, and write $F^* = (f^*, \mathbf{H})$. Then F^* is an optimal DSFBC for $\mathbf{G}$ relative to P_{dyn}, namely

$$L(F^*/\mathbf{G}) = \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}, P_{\text{dyn}}))$$

$\square$

With a slight modification of the definition of P_{dyn}, the computational result of Section 8.8 will apply in the present situation as well; for an example see Sect. 8.11 below.

8.10 Modular Dynamic State Feedback Control for VDES

By following the scheme outlined at the end of Sect. 7.6 it is quite straightforward to describe modular control of VDES, by adjoining a direct sum of memory VDES of the type $\mathbf{H}$ in the previous section. Again taking $\Sigma = \{\sigma_1, ..., \sigma_m\}$, suppose we wish to enforce on the occurrence vector $V(\cdot)$ of $\mathbf{G}$ a predicate $P_{\text{dyn}} \in \overline{Pred_{\mathbf{lin}}(\mathbb{N}^m)}$ of conjunctive form:

$$v \models P_{\text{dyn}} \quad \text{iff} \quad \bigwedge_{i=1}^{k} (c_i v \leq d_i)$$

where $v \in \mathbb{Z}^{m\times 1}$, $c_i \in \mathbb{Z}^{1\times m}$, $d_i \in \mathbb{Z}$ $(i = 1, ..., k)$. For each conjunct $P_{\text{dyn},i} := (c_i v \leq d_i)$, construct memory $\mathbf{H}_i$ as in Sect. 8.9, and let $F_i^* := (f_i^*, \mathbf{H}_i)$ be a corresponding optimal balanced DSFBC for $\mathbf{G}$. It follows by Theorem 7.6.2 that

$$F^* := \{f_1^* \wedge ... \wedge f_k^*, \mathbf{H}_1 \oplus ... \oplus \mathbf{H}_k\}$$

is then an optimal balanced DSFBC enforcing P_{dyn} on the behavior of $\mathbf{G}$.

To conclude this section we restate for future reference the correspondence between VDES and conjunctive predicates.

Proposition 8.10.1

For any predicate of form P_{dyn} there is a VDES $\mathbf{H}$ with state set $\mathbb{N}^k$ and event set Σ such that

$$L(\mathbf{G}) \cap L(\mathbf{H}) = L(\mathbf{G}, P_{\text{dyn}})$$

Dually for any VDES $\mathbf{H}$ with state set $\mathbb{N}^k$ and event set Σ $(|\Sigma| = m)$ there exists a predicate P_{dyn} on $\mathbb{N}^m$ such that the above equality is true.

Proof

For the first statement, apply the construction of Sect. 8.9 to each conjunct of P_{dyn}. For the second statement, reverse the procedure: given $\mathbf{H} = (Y, \Sigma, \eta, y_o, Y)$ with $Y = \mathbb{N}^k$, $y_o = [y_{o1} ... y_{ok}]$, $\Sigma = \{\sigma_1, ..., \sigma_m\}$ and $\eta(y, \sigma) = y + h_\sigma$ (defined when $y \geq 0$, $y + h_\sigma \geq 0$), write $h_{\sigma_j} =: h_j = [h_{j1} ... h_{jk}] \in \mathbb{Z}^{k\times 1}$ $(j = 1, ..., m)$ and let

$$d_i := y_{oi}, \qquad c_{ij} := -h_{ji}, \qquad (i = 1, ..., k; \;\; j = 1, ..., m)$$

Then define $c_i := [c_{i1} ... c_{im}] \in \mathbb{Z}^{1\times m}$ $(i = 1, ..., k)$, and finally

$$P_{\text{dyn}} := \bigwedge_{i=1}^{k} \; (d_i - c_i v \geq 0)$$

□

Thus each state variable of a VDES can be regarded as a 'memory' variable that records a weighted sum of event occurrence numbers. The initial and occurrence conditions of a VDES with state space $\mathbb{N}^k$ impose the requirement that all k memory state variables be maintained nonnegative. A VDES on $\mathbb{N}^k$ thus expresses the same language as a conjunction of linear dynamic specifications. Thus such a VDES can be used to provide a control specification in the first place.

8.11 Example: FACT#2

Returning to **FACT#2** (Sects. 8.3, 8.5), we attempt to enforce predicates

$$P_1 := (x_4 \leq 100), \qquad P_2 := (10 + \#\beta \geq 3\#\lambda)$$

where $\#\sigma$ means the number of occurrences of σ in the string generated by $\mathbf{G}$ since initialization. P_2 is supposed to limit the number of breakdowns, relative to the number of workpieces successfully processed; since breakdown (λ) is uncontrollable, this may require that eventually the process be shut down.

Following the procedure of Sect. 8.9, to represent P_2 bring in the memory

$$\mathbf{H} = (Y, \Sigma, \eta, y_o, Y)$$

with

$$Y := \mathbb{Z}, \qquad \Sigma = \{\alpha, \beta, \lambda, \mu\}, \qquad y_o := 10$$

$$h_\alpha = 0, \qquad h_\beta = 1, \qquad h_\lambda = -3, \qquad h_\mu = 0$$

where $\eta(y, \sigma) = y + h_\sigma$. It is clear that in $\mathbf{G} \oplus \mathbf{H}$, the state variable y will record the quantity

$$y = 10 + \#\beta - 3\#\lambda$$

and P_2 is tantamount to $(y \geq 0)$. The state vector of $\mathbf{G} \oplus \mathbf{H}$ is

$$x \oplus y = [x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; y] \in \mathbb{N}^4 \oplus \mathbb{Z}$$

initialized at $[10 \;\; 0 \;\; 0 \;\; 0 \;\; 10]$. Note that the VDES dynamics for $\mathbf{G} \oplus \mathbf{H}$ automatically enforce only $x \geq 0$, and not $y \geq 0$. The full displacement matrix is

$$E := \begin{bmatrix} -1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -3 & 0 \end{bmatrix}$$

For the control design we wish to use integer programming as in Sect. 8.7. To do so we must respect the assumption (cf. Theorem 8.6.1) that our VDES state variables remain nonnegative under the defined (and uncontrolled) dynamic transition action. Since the memory variable $y \in \mathbb{Z}$ is not thus constrained we first write it as the difference of two nonnegative variables, say

$$y = y_1 - y_2$$

with $y_1 = 10 + \#\beta$, $y_2 = 3\#\lambda$. We now redefine

$$\begin{aligned} Y &:= \mathbb{N}^2, \quad y := \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \Sigma = \{\alpha, \beta, \lambda, \mu\}, \quad y_0 := \begin{bmatrix} 10 \\ 0 \end{bmatrix} \\ h_\alpha &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad h_\beta = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad h_\lambda = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \quad h_\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

with $\eta(y, \sigma) = y + h_\sigma \in \mathbb{N}^2$ and $P_2 = (y_1 - y_2 \geq 0)$. The new state vector of $\mathbf{G} \oplus \mathbf{H}$ is

$$x \oplus y = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad y_1 \quad y_2] \in \mathbb{N}^6$$

initialized at $[10 \quad 0 \quad 0 \quad 0 \quad 10 \quad 0]$. Note that the VDES dynamics for $\mathbf{G}$ itself automatically enforce $x \oplus y \geq 0$. The full displacement matrix is

$$E := \begin{bmatrix} -1 & 1 & 0 & 1 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{bmatrix} \in \mathbb{Z}^{6\times 4}$$

To start the design we note that $(\mathbf{G} \oplus \mathbf{H})_u$ is loop-free. Referring to Sect. 8.7, write E_u for the 6×2 submatrix of E corresponding to events β and λ, i.e. columns 2 and 3. For P_1, let

$$a = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0] \in \mathbb{Z}^{1\times 6}, \qquad b = 100$$

Then

$$c := aE_u = [1 \quad 0] \in \mathbb{Z}^{1\times 2}$$

and we take $v = [v_1 \quad v_2] \in \mathbb{N}^{2\times 1}$. We are to maximize $cv = v_1$ subject to $v \geq 0$ and $E_u v \geq -(x \oplus y)$, i.e.

$$v_1 \geq 0, \qquad v_2 \geq 0,$$

$$v_1 \geq -x_1, \qquad -v_1 - v_2 \geq -x_2, \qquad v_2 \geq -x_3,$$

$$v_1 \geq -x_4, \qquad v_1 \geq -y_1, \qquad 3v_2 \geq -y_2$$

In view of $x_i \geq 0$, $y_j \geq 0$ by the dynamics of $\mathbf{G} \oplus \mathbf{H}$, the effective constraints are

$$v_1 \geq 0, \qquad v_2 \geq 0, \qquad v_1 + v_2 \leq x_2$$

from which we obtain the solution

$$v_1^* = x_2, \qquad v_2^* = 0$$

As expected, the solution for P_1 is independent of the memory element $\mathbf{H}$. From Theorem 8.7.1 we get for the optimal control

$$\begin{aligned} f_\alpha^{(1)*}(x \oplus y) = 1 \quad & \text{iff} \quad a(x \oplus y)_{\text{new}} + cv^*((x \oplus y)_{\text{new}}) \le b \\ & \text{iff} \quad x_{4,\text{new}} + x_{2,\text{new}} \le 100 \end{aligned}$$

using $(x \oplus y)_{\text{new}} := (x \oplus y) + (e_\alpha \oplus h_\alpha)$.

For P_2 we have $-y_1 + y_2 \le 0$, or $a(x \oplus y) \le b$ with

$$a = [0 \;\; 0 \;\; 0 \;\; 0 \;\; -1 \;\; 1], \qquad b = 0$$

Thus $c = aE_u = [-1 \;\; 3]$, and our problem is to maximize

$$cv = -v_1 + 3v_2$$

under the same effective conditions as before, namely $v_i \ge 0$, $v_1 + v_2 \le x_2$. This gives $v_1^* = 0$, $v_2^* = x_2$ and $cv^*(x) = 3x_2$. Thus for $\sigma \in \Sigma_c$ we may take

$$f_\sigma^{(2)*}(x \oplus y) = 1 \quad \text{iff} \quad -y_{1,\text{new}} + y_{2,\text{new}} + 3x_{2,\text{new}} \le 0$$

where $(x \oplus y)_{\text{new}} := (x \oplus y) + (e_\sigma \oplus h_\sigma)$. Combining the SFBC for P_1 with the DSFBC for P_2 we obtain for the conjunction

$$f_\sigma^*(x \oplus y) = 1 \quad \text{iff} \quad (x_{2,\text{new}} + x_{4,\text{new}} \le 100) \wedge (3x_{2,\text{new}} - y_{1,\text{new}} + y_{2,\text{new}} \le 0)$$

whenever $\sigma \in \Sigma_c$, in particular for $\sigma = \alpha$.

This example provides an opportunity to implement optimal control by means of a 'control' VDES, say $\mathbf{G_{con}}$, coupled (via $\oplus$) to $\mathbf{G}$. To see how this is done, rearrange the conditions for f_σ^* in the form

$$100 - x_{2,\text{new}} - x_{4,\text{new}} \ge 0, \qquad -3x_{2,\text{new}} + y_{1,\text{new}} - y_{2,\text{new}} \ge 0$$

Introduce control coordinates z_1, z_2 which we shall try to arrange so that

$$z_1 = 100 - x_2 - x_4, \qquad z_2 = -3x_2 + y_1 - y_2$$

as the process evolves; initially $z_1 = 100$, $z_2 = 10$. Heuristically note that

$$\begin{aligned} z_1 &= 100 - (\#\alpha - \#\beta - \#\lambda) - \#\beta = 100 - \#\alpha + \#\lambda \\ z_2 &= -3(\#\alpha - \#\beta - \#\lambda) + (10 + \#\beta) - 3\#\lambda = 10 - 3\#\alpha + 4\#\beta \end{aligned}$$

With the ordering $(\alpha, \beta, \lambda, \mu)$ we therefore take the displacements in z_1, z_2 to be

$$k(z_1) = [-1 \;\; 0 \;\; 1 \;\; 0], \qquad k(z_2) = [-3 \;\; 4 \;\; 0 \;\; 0]$$

Thus if we let $\mathbf{G_{con}} := (Z, \Sigma, \zeta, z_o, Z)$ with $Z = \mathbb{N}^2$, and let the foregoing displacements define ζ, then the behavior of $\mathbf{G} \oplus \mathbf{G_{con}}$ (as a VDES with state space $\mathbb{N}^6$) will be exactly the behavior of $\mathbf{G}$ under f^*, inasmuch as σ will be enabled only if $z_{1,\text{new}} \ge 0$, $z_{2,\text{new}} \ge 0$. Notice that the only negative entries in the k_1, k_2 vectors correspond to controllable events (specifically, α). Thus the requirement of coordinate nonnegativity enforced, by assumption 'physically', by VDES over $\mathbb{N}^6$, captures the control action in a plausible way: no further 'control technology' is needed. This approach will be pursued systematically in Sect. 8.13.

8.12 Modelling and Control of a Production Network

We consider the modelling and control of a production network, adapted from work by Al-Jaar and Desrochers. A Petri net for the system is shown in Fig. 8.12.1.

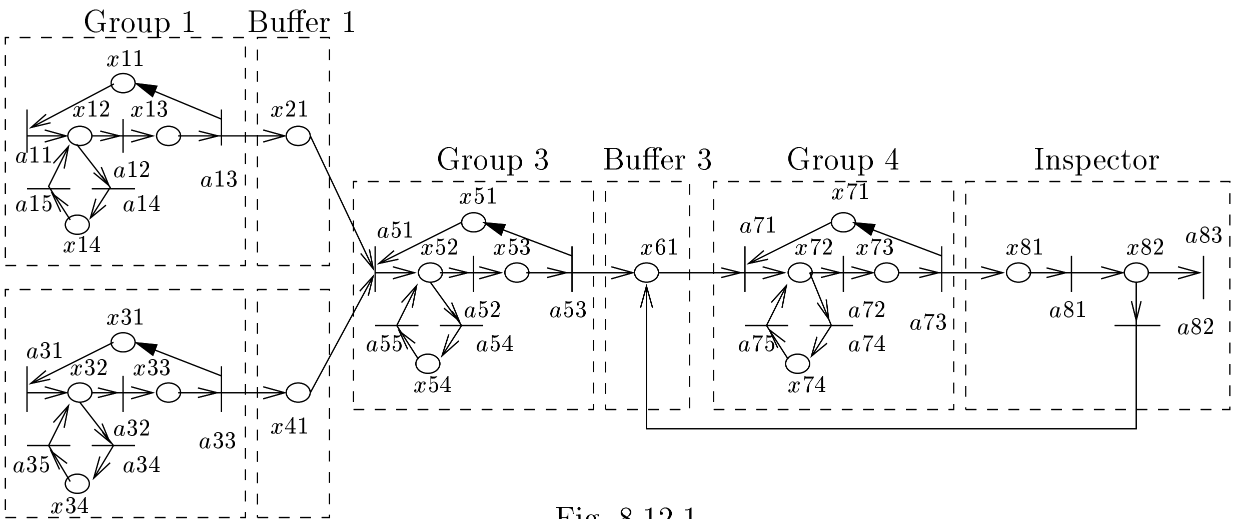


Fig. 8.12.1
Production Network

The system operates as follows. Machines in Groups 1 and 2 receive parts from a non-depleting inventory and deposit the finished parts in Buffers 1 and 2 respectively. Machines in Group 3 fetch parts from Buffers 1 and 2 for assembling. The assembled workpiece is deposited in Buffer 3 to be further processed by Group 4. The processed workpiece is sent to an inspection unit which can either output the workpiece as a finished product or return it for reworking by Group 4.

We use a modular approach to modeling this production network. First we model modules of the network individually and then compose them to form the model of the complete system.

(1) Modelling of the Machines

The state vector of machine group 1,2,3,4 is indexed respectively $i = 1, 3, 5, 7$ and is

$$x_i = [x_i^1, x_i^2, x_i^3, x_i^4] \in \mathbb{N}^{4\times 1}, \qquad i = 1, 3, 5, 7\ .$$

The state components denote, respectively, ‘idle’, ‘processing’, ‘holding (part prior to sending on)’, ‘broken down’. With reference to Fig. 8.12.1, transitions $\alpha_i^1, \alpha_i^2, \alpha_i^3, \alpha_i^4, \alpha_i^5$ $(i = 1, 3, 5, 7)$ represent ‘start processing’, ‘finish processing’, ‘return to idle’, ‘break down’, ‘return to processing after repair’.

The corresponding state transition function $\xi_i : X_i \times \Sigma_i \to X_i$ is given by

$$\xi_i(x_i, \alpha_i^j) = x_i + e_{i,\alpha_i^j} \qquad (i = 1, 3, 5, 7; \quad j = 1, 2, 3, 4, 5)$$

where

$$e_{i,\alpha_i^1} = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad e_{i,\alpha_i^2} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$$

$$e_{i,\alpha_i^3} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix} \quad e_{i,\alpha_i^4} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \quad e_{i,\alpha_i^5} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}$$

Initially, all machines are in the 'idle' state:

$$x_{i,0} = \begin{bmatrix} g_i \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The vector models of these machines are then

$$\mathbf{G}_i = (X_i, \Sigma_i, \xi_i, x_{i,o})$$

where

$$\Sigma_i = \{\alpha_i^1, \alpha_i^2, \alpha_i^3, \alpha_i^4, \alpha_i^5\}$$

with $\Sigma_{i,c} = \{\alpha_i^3\}$, i.e. we assume that only the event 'return to idle' is controllable.

(2) Modelling of the Inspection Unit

The model for the inspector is

$$\mathbf{G}_8 = (X_8, \Sigma_8, \xi_8, x_{8,o})$$

where

$$\Sigma_8 = \{\alpha_7^3, \alpha_8^1, \alpha_8^2, \alpha_8^3\}$$

$$\Sigma_{8,c} = \{\alpha_8^1\}$$ [1]

and

$$x_8 = [x_8^1, x_8^2] \in \mathbb{N}^{2\times 1}$$

$$x_{8,o} = [0, 0]$$

$$\xi_8(x_8, \alpha_8^i) = x_8 + e_{8,\alpha_8^i}$$

with

$$e_{8,\alpha_7^3} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad e_{8,\alpha_8^1} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad e_{8,\alpha_8^2} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad e_{8,\alpha_8^3} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

(3) Modelling of Buffers

The three buffers can be modelled as scalar systems. For Buffer 1, we have

$$\mathbf{G}_2 = (X_2, \Sigma_2, \xi_2, x_{2,o})$$

with

$$\Sigma_2 = \{\alpha_1^3, \alpha_5^1\}, \qquad \Sigma_{2,c} = \{\alpha_1^3\}$$

$$X_2 = \mathbb{N} \qquad x_{2,o}^1 = 0$$

$$\begin{aligned} \xi_2(x_2^1, \alpha_1^3) &= x_2^1 + 1 \\ \xi_2(x_2^1, \alpha_5^1) &= x_2^1 - 1 \end{aligned}$$

Buffer 2 is modelled similarly. For Buffer 3, we have

$$\mathbf{G}_6 = (X_6, \Sigma_6, \xi_6, x_{6,o})$$

with

$$\Sigma_6 = \{\alpha_5^3, \alpha_7^1, \alpha_8^2\}, \qquad \Sigma_{6,c} = \{\alpha_5^3\}$$

$$X_6 = \mathbb{N}, \quad x_{6,o}^1 = 0$$

$$\xi_6(x_6^1, \alpha_5^3) = \xi_6(x_6^1, \alpha_8^2) = x_6^1 + 1$$

$$\xi(x_6^1, \alpha_7^1) = x_6^1 - 1$$

(4) Composition

Finally we compose the above components to obtain the VDES model of the production network:

$$\mathbf{G} = (X, \Sigma, \xi, x_0) = \bigoplus_{i=1}^{8} \mathbf{G}_i$$

where

$$\Sigma = \bigcup_{i=1}^{8} \Sigma_i, \quad \Sigma_c = \bigcup_{i=1}^{8} \Sigma_{i,c}$$

$$X = \bigoplus_{i=1}^{8} X_i, \quad x_0 = \bigoplus_{i=1}^{8} x_{i,0}$$

$$\xi(x, \alpha_i^j) = x + e_{\alpha_i^j}$$

with

$$e_{\alpha_i^j} = \bigoplus_{k=1}^{8} e_{k,\alpha_i^j}$$

where we define $e_{k,\alpha_i^j} = 0$ if α_i^j is not in Σ_k. The connections of the system modules are displayed in Fig. 8.12.2.

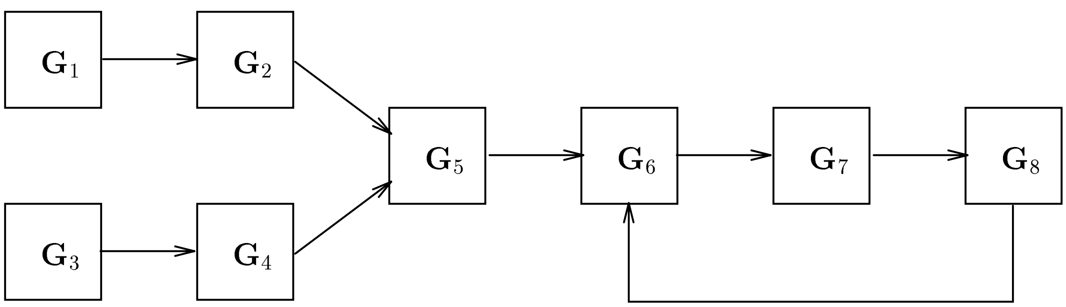


Fig. 8.12.2
Production Network Connection

Note that the connection of $\mathbf{G}_1$ and $\mathbf{G}_2$ is serial, that of $\mathbf{G}_2$ and $\mathbf{G}_4$ parallel, and that of $\mathbf{G}_6$ and $\mathbf{G}_8$ feedback.

(5) Control of the Production Network

We now discuss how to synthesize a modular controller to satisfy the performance specification of the system. The specification is that no buffer overflow and that at most one part be inspected at a given time. We assume that Buffer i has capacity k_i and the buffer inside the inspection unit (x_8^1) has capacity 1. This specification can be formalized as a predicate on the state space X:

$$P = \bigwedge_{i=1}^{4} P_i$$

with

$$\begin{aligned} P_1 &= (x_2^1 \leq k_1) \\ P_2 &= (x_4^1 \leq k_2) \\ P_3 &= (x_6^1 \leq k_3) \\ P_4 &= (x_8^1 \leq 1) \end{aligned}$$

We list the optimal[2] subcontrollers for the above linear specifications.

$$\begin{array}{lcl} f_{1,\alpha}(x) = 0 & \Leftrightarrow & \alpha = \alpha_1^3 \ \text{ and } \ x_2^1 \geq k_1 \\ f_{2,\alpha}(x) = 0 & \Leftrightarrow & \alpha = \alpha_3^3 \ \text{ and } \ x_4^1 \geq k_2 \\ f_{3,\alpha}(x) = 0 & \Leftrightarrow & (\alpha = \alpha_5^3 \ \text{ or } \ \alpha = \alpha_8^1) \ \text{ and } \ x_6^1 + x_8^2 \geq k_3 \\ f_{4,\alpha}(x) = 0 & \Leftrightarrow & \alpha = \alpha_7^3 \ \text{ and } \ x_8^1 \geq 1 \end{array}$$

The conjunction of these subcontrollers is

$$f := \bigwedge_{i=1}^{5} f_i$$

It is easy to check that all subcontrollers in f are balanced. Therefore, this modular controller is optimal in the sense that it synthesizes a largest reachable state set among all controllers which enforce the specification

$$P = \bigwedge_{i=1}^{4} P_i$$

as asserted by Theorem 7.5.1.

The above modular controller can lead to deadlock of the controlled system. To see this, consider the state at which $x_2^1 = k_1$, $x_4^1 = k_2$, $x_6^1 = k_3$, $x_8^1 = 1$ and $x_i^3 = g_i$ $(i = 1, 3, 5, 7)$,

[2]While $\mathbf{G_u}$ is not loopfree, and therefore Theorem 8.7.1 not strictly applicable, the asserted optimality is obvious by inspection.

with all other state variables being 0. At this state all controllable events are disabled and no uncontrollable event can occur. One way to remove the deadlock in the system is to add another subspecification which ensures that the deadlock state cannot be reached.

For this it is sufficient to ensure that the number of empty slots in Buffer 3 ($k_3 - x_6^1$) is maintained at least as great as the number of workpieces that could potentially be returned to Buffer 3 on being tested defective. In the worst case this is the number of workpieces being processed by the machines in Group 4 together with the Inspector, namely

$$x_7^2 + x_7^3 + x_7^4 + x_8^1 + x_8^2$$

So our new subspecification can be taken to be

$$P_5 = (x_6^1 + x_7^2 + x_7^3 + x_7^4 + x_8^1 + x_8^2 \leq k_3)$$

Notice that P_5 implies P_3, so the latter may now be discarded and the controls redesigned on the basis of P_1, P_2, P_4 and P_5.

Exercise 8.12.1: Redesign the controls as just specified. By detailed reasoning from your control design, prove that the controlled system is maximally permissive and nonblocking with respect to the prescribed initial state as marker state. $\diamond$

To illustrate dynamic control, let us consider the following linear dynamic specification:

$$|\alpha_5^3| - |\alpha_8^3| \leq k$$

which specifies that the number of parts in the inspection loop never exceeds an integer k. Here $|\alpha_i^3|$ $(i = 5, 8)$ denotes the number of occurrences of α_i^3. A one-dimensional memory $\mathbf{H}$ can be easily constructed from this specification and is shown in Fig. 8.12.3.

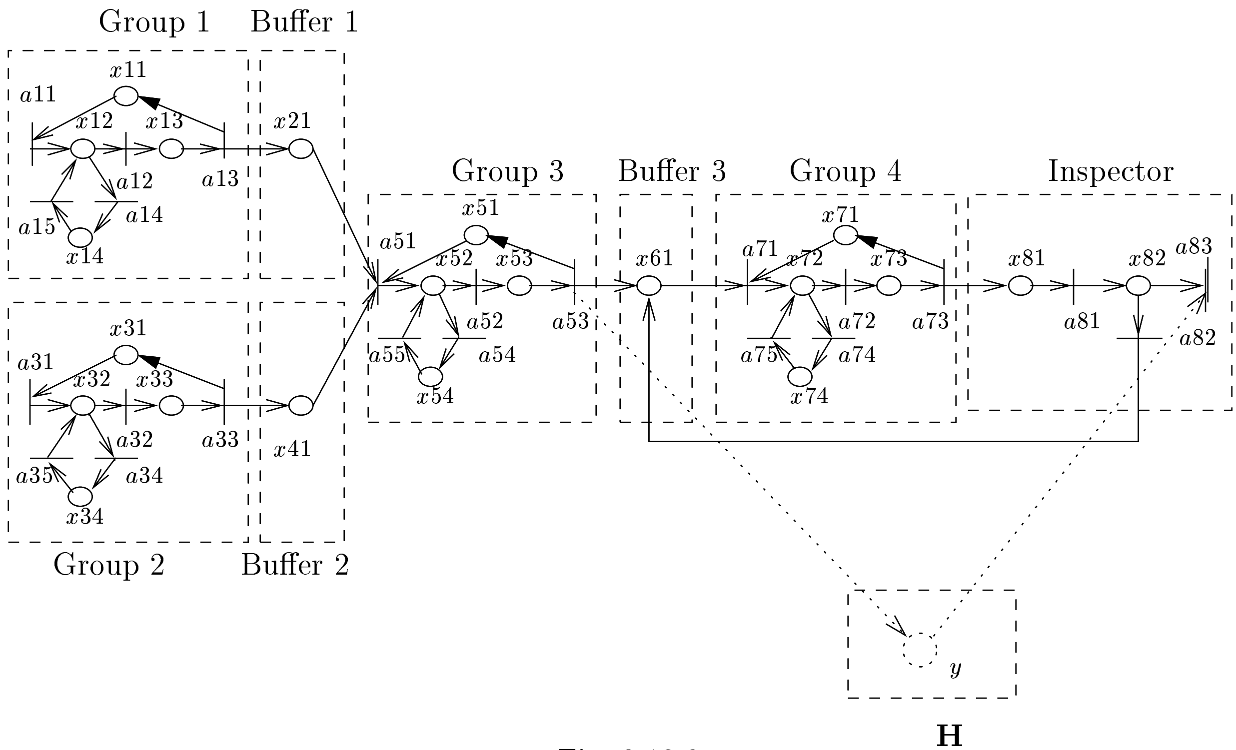


Fig. 8.12.3
Production Network with Dynamic Control

The dynamic specification is then equivalent to a static specification

$$y \leq k$$

on the extended state space $X \oplus Y$ with Y being the one-dimensional state space of **H**. By Theorem 8.9.1, the optimal controller f enforcing this static specification can be defined as

$$f_\alpha = \begin{cases} 0 & \text{if } \alpha = \alpha_5^3 \text{ and } y \geq k \\ 1 & \text{otherwise} \end{cases}$$

Exercise 8.12.2: Verify this result by calculating f^* as in Theorem 8.9.1.

8.13 Representation of Optimal Control by a Control VDES

In this section we return to the problem, illustrated in Sect. 8.11, of representing the optimal control by a VDES. This kind of result has the appeal that control does not require departure from the basic model class, a feature that offers convenience in control implementation, and ease of analysis and simulation of controlled behavior. On the other hand, insisting on a VDES implementation of the control does impose further restrictions on the structure of the plant $\mathbf{G}$ (although not on the control specification).

As usual let $\mathbf{G} = (X, \Sigma, \xi, x_o, X_m)$, with $X = \mathbb{N}^n$ and where ξ is defined by displacement vectors $e_\sigma \in \mathbb{Z}^{n\times 1}$. Assume that the control specification is provided in the form of a VDES $\mathbf{S} = (Y, \Sigma, \eta, y_o, Y)$, where $Y = \mathbb{N}^p$, and with displacement vectors $h_\sigma \in \mathbb{Z}^{p\times 1}$. One may think of $\mathbf{S}$ as tracking the behavior of $\mathbf{G}$, with the specification expressed as the predicate $(y \geq 0)$ on $X \oplus Y$. Write $S := L(\mathbf{S})$, the closed behavior of $\mathbf{S}$. We shall assume that $\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) \neq \emptyset$, so an optimal DSFBC F^* for $\mathbf{G}$ exists (as a SFBC on $X \oplus Y$), such that

$$L(F^*/\mathbf{G}) = \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$$

Let $\mathbf{G_{con}} := (Z, \Sigma, \zeta, z_o, Z)$ be a VDES with $Z = \mathbb{N}^r$. We say that $\mathbf{G_{con}}$ is a *VDES implementation (VDESI)* of F^* provided

$$L(F^*/\mathbf{G}) = L(\mathbf{G} \oplus \mathbf{G_{con}})$$

We shall provide a constructive sufficient condition (due to Shu-Lin Chen, 1992) under which $\mathbf{G_{con}}$ exists; it will then turn out that $r = p$.

Let $A \subseteq \Sigma$ and $\alpha \in A$. Define the event subset $\Sigma(\alpha, A)$ and coordinate (index) set $I(\alpha, A)$ inductively by the rules:

1. $\alpha \in \Sigma(\alpha, A)$
2. $\sigma \in \Sigma(\alpha, A)$ & $i \in \sigma^\uparrow \Rightarrow i \in I(\alpha, A)$
3. $i \in I(\alpha, A)$ & $\sigma \in i^\uparrow \cap A \Rightarrow \sigma \in \Sigma(\alpha, A)$
4. No other elements belong to $\Sigma(\alpha, A)$ or $I(\alpha, A)$.

Note that Rule 2 says that i is placed in $I(\alpha, A)$ if $e_\sigma(i) < 0$. The restriction of $\mathbf{G}$ to $I(\alpha, A)$ and $\Sigma(\alpha, A)$ is the subsystem of $\mathbf{G}$ that is upstream from α, taking into account only the flow due to transitions in A.

Next take $A := \Sigma_u$ and consider (one-dimensional) $\mathbf{S}$ with $p = 1$, $Y = \mathbb{N}$. Define

$$\begin{aligned}
\Sigma_u^- &:= \{\sigma \in \Sigma_u | h_\sigma < 0\} \\
\Sigma^\nabla &:= \cup\{\Sigma(\sigma, \Sigma_u) | \sigma \in \Sigma_u^-\} \\
I^\nabla &:= \cup\{I(\sigma, \Sigma_u) | \sigma \in \Sigma_u^-\}
\end{aligned}$$

Finally, denote by $\mathbf{G}^\nabla$ the restriction of $\mathbf{G}$ to I^∇, Σ^∇. Thus $\mathbf{G}^\nabla$ is just the subsystem of $\mathbf{G}$ of which the flow is uncontrollable and effect is to decrement the (scalar) specification coordinate $y \in Y$ via events in Σ_u^-. Since the specification is precisely that y be maintained nonnegative, it is the structure of $\mathbf{G}^\nabla$ that is crucial for that of the optimal control.

Example 8.13.1: For **FACT#2** we had $P_2 = (10 + \#\beta \geq 3\#\lambda)$, which may be converted to a VDES $\mathbf{S}$ with $Y = \mathbb{N}$, $y_o = 10$, and $[h_\alpha \ h_\beta \ h_\lambda \ h_\mu] = [0 \ 1 \ -3 \ 0]$. We have

$$\Sigma_u^- = \{\lambda\}, \quad \Sigma^\nabla = \{\lambda\}, \quad I^\nabla = \lambda^\uparrow = \{2\}$$

$\diamond$

Now we can state

Theorem 8.13.1 (Shu-Lin Chen)

Given a VDES $\mathbf{G}$ and a specification language S represented by a 1-dimensional VDES $\mathbf{S}$ as above. Assume that an optimal DSFBC F^* for $\mathbf{G}$ exists (i.e. $\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) \neq \emptyset$). In addition assume the conditions

1. $\mathbf{G}^\nabla$ is loop-free.
2. For all $\sigma \in \Sigma^\nabla$, the displacement vector e_σ in $\mathbf{G}$ has at most one negative component, i.e. $|\sigma^\uparrow| \leq 1$; furthermore if for some i, $e_\sigma(i) < 0$, then $e_\sigma(i) = -1$ (by Rule 2 above, $\sigma^\uparrow = \{i\}$ and $i \in I^\nabla$).

Then a VDESI $\mathbf{G_{con}}$ for F^* exists. $\square$

Example 8.13.2: For **FACT#2** the conditions of Theorem 8.13.1 are clearly satisfied; a VDESI was constructed *ad hoc* in Sect. 8.11. $\diamond$

Our proof will be constructive, and include a test for the existence of F^*. Roughly, the procedure is to successively transform the specification VDES $\mathbf{S}$, moving upstream in $\mathbf{G}^\nabla$ against the flow of (uncontrollable) events, until the controllability condition on the transformed version of $\mathbf{S}$ is satisfied. The loop-freeness Condition 1 guarantees termination, while Condition 2 serves to rule out 'disjunctive' control logic (cf. Exercise 8.13.7).

We begin by defining a family of transformations T_α ($\alpha \in \Sigma_u$) on 1-dimensional VDES. As above let $\mathbf{S} = (Y, \Sigma, \eta, y_o, Y)$ with $y_o \geq 0$ and $\eta(y, \sigma) = y + h_\sigma$. **NULL** will stand for the 'empty' VDES with $L(\mathbf{NULL}) := \emptyset$. Under Condition 2 of Theorem 8.13.1, define

$$\mathbf{S}_{\text{new}} := T_\alpha \mathbf{S}$$

as follows.

1. If $h_\alpha \geq 0$ then $T_\alpha = id$ (identity operator), i.e. $\mathbf{S}_{\text{new}} := \mathbf{S}$.

2. If $h_\alpha < 0$ and $\alpha^\uparrow = \{i\}$ (thus $e_\alpha(i) < 0$) then

$$y_{\text{new},o} := y_o + x_o(i)h_\alpha$$

$$h_{\text{new},\sigma} := \begin{cases} h_\sigma + e_\sigma(i)h_\alpha & \text{if} \quad \sigma \in i^\uparrow \cup i^\downarrow \\ h_\sigma & \text{otherwise} \end{cases}$$

If $y_{\text{new},o} \geq 0$ then accept the VDES $\mathbf{S}_{\text{new}}$ with $y_{\text{new},o}$, $h_{\text{new},\sigma}$ ($\sigma \in \Sigma$): otherwise $\mathbf{S}_{\text{new}} :=$ **NULL**.

3. If $h_\alpha < 0$ and $\alpha^\uparrow = \emptyset$ then $\mathbf{S}_{\text{new}} := \mathbf{NULL}$.

4. For all $\sigma \in \Sigma$, $T_\sigma(\mathbf{NULL}) := \mathbf{NULL}$.

In part 2 of the definition of T_α, clearly $y_{\text{new},o} \leq y_o$. Also, Condition 2 of Theorem 8.13.1 implies $e_\alpha(i) = -1$, so

$$h_{\text{new},\alpha} = h_\alpha + e_\alpha(i)h_\alpha = 0$$

In general $h_{\text{new},\sigma}$ is made up of the direct contribution h_σ to y on the occurrence of σ, plus a contribution to y of $e_\sigma(i)h_\alpha$ due to occurrences of α (see the proof of Lemma 8.13.1 below).

Note that activation of part 3 of the definition of T_α is not ruled out by Condition 2 of Theorem 8.13.1.

Example 8.13.3: In **FACT#2** recall that $y = 10 + \#\beta - 3\#\lambda$, $h = [0\ 1\ \ -3\ 0]$, $\lambda \in \Sigma_u$, $h_\lambda = -3 < 0$, $\lambda^\uparrow = \{2\}$. Thus, $\mathbf{S}_{\text{new}} = T_\lambda \mathbf{S}$ is calculated according to

$$y_{\text{new},o} = y_o + x_o(2)h_\lambda = y_o = 10$$

$$2^\uparrow \cup 2^\downarrow = \{\alpha, \beta, \lambda\}$$

$$h_{\text{new},\alpha} = h_\alpha + e_\alpha(2)h_\lambda = 0 + (+1)(-3) = -3$$

$$h_{\text{new},\beta} = h_\beta + e_\beta(2)h_\lambda = (+1) + (-1)(-3) = 4$$

$$h_{\text{new},\lambda} = h_\lambda + e_\lambda(2)h_\lambda = 0$$

$$h_{\text{new},\mu} = h_\mu$$

Thus

$$h_{\text{new}} = [-3\ 4\ 0\ 0]$$

For $\mathbf{S}_{\text{new}}$,

$$\{\sigma \in \Sigma_u | h_\sigma < 0\} = \emptyset,$$

so that now all $T_\sigma = id$. ◇

Let $\hat{I} \subseteq I := \{1, ..., n\}$, $\hat{\Sigma} \subseteq \Sigma$ and let $\hat{\mathbf{G}}$ be the restriction of $\mathbf{G}$ to $\hat{I}, \hat{\Sigma}$. An event $\sigma \in \hat{\Sigma}$ is a *leaf event of* $\hat{\mathbf{G}}$ if, for all $i \in \sigma^{\downarrow} \cap \hat{I}$, $i^{\downarrow} \cap \hat{\Sigma} = \emptyset$, or briefly $(\sigma^{\downarrow} \cap \hat{I})^{\downarrow} \cap \hat{\Sigma} = \emptyset$ (in particular this is true if $\sigma^{\downarrow} \cap \hat{I} = \emptyset$). Evidently a leaf event of $\hat{\mathbf{G}}$ cannot contribute to the occurrence in $\hat{\mathbf{G}}$ of an immediately following event in $\hat{\Sigma}$.

Let LeafEvent$(\hat{\mathbf{G}}, \alpha)$ denote a procedure that selects an arbitrary leaf event α of $\hat{\mathbf{G}}$ (or returns *error* if no leaf event exists). To compute $\mathbf{G_{con}}$ we start with

Procedure 1 (index Σ^{∇} by leaf property);

```
G_var := G^∇;
Σ_var := Σ^∇;
k := |Σ^∇|;
index := 1;
if k = 0 then LEList := [ ] else
    while index ≤ k do
        begin
            LeafEvent(G_var, α);
            LEList[index] := α;
            Σ_var := Σ_var − {α};
            G_var := restriction of G_var to I^∇, Σ_var;
            index := index + 1
        end.
```

Proposition 8.13.1:

Under the conditions of Theorem 8.13.1, Procedure 1 is well-defined.

Proof:

It suffices to check that $\mathbf{G_{var}}$ always has a leaf event if index $\leq k$. But this follows by Condition 1 of Theorem 8.13.1 that $\mathbf{G}^{\nabla}$ is loop-free. □

Exercise 8.13.1: Supply the details. ◇

Procedure 1 returns a listing of Σ^{∇}, LEList $=[\ ]$ or $[\alpha_1, ..., \alpha_k]$. Procedure 2 will then compute the final result as follows.

Procedure 2:

If LEList $=[\ ]$ then $\mathbf{S_{fin}} := \mathbf{S}$ else

$$\mathbf{S_{fin}} := T_{\alpha_k} T_{\alpha_{k-1}} ... T_{\alpha_1}(\mathbf{S}).$$

It will be shown that, under the Conditions 1 and 2 of Theorem 8.13.1, either $\mathbf{S_{fin}} = \mathbf{NULL}$, in which case $\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) = \emptyset$ and F^* does not exist, or else $\mathbf{S_{fin}} = \mathbf{G_{con}}$ is a VDESI for F^*.

Exercise 8.13.2: Let $\mathbf{G}$ be a 1-dimensional VDES with $\Sigma_u = \Sigma = \{\alpha, \beta\}$ and displacement matrix $E = [-1\ \ -1]$. Define $\mathbf{S}$ by $h = [-5\ \ -7]$. Check that α, β are both leaf events and that

$$\mathbf{S_{fin}} = T_\alpha T_\beta \mathbf{S} = T_\beta T_\alpha \mathbf{S}$$

with $y_{\mathrm{fin},o} = y_o - 7x_o$ and $h_{\mathrm{fin}} = [2\ 0]$. Thus $\mathbf{S_{fin}} \neq \mathbf{NULL}$ if and only if $y_o \geq 7x_o$. Explain intuitively.

Exercise 8.13.3: Consider the 6-dimensional VDES $\mathbf{G}$ with $\Sigma_u = \Sigma = \{\sigma_i | i = 1, ..., 7\}$, and

$$E := \begin{bmatrix} 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & -1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 3 & 0 & 2 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

For $\mathbf{S}$ let

$$h := [0\ 0\ 0\ 0\ 0\ \ -2\ \ -3]$$

Thus $\Sigma_u^- = \{\sigma_6, \sigma_7\}$. Show that a possible LEList is $[\sigma_6\ \sigma_2\ \sigma_7\ \sigma_1\ \sigma_5\ \sigma_3\ \sigma_4]$. Calculate h_{fin} and

$$y_{\mathrm{fin},o} = y_o - cx_o$$

for suitable $c \in \mathbb{N}^{1\times 6}$. Interpret the result in terms of a 'worst case' event sequence that maximally decrements y. $\diamond$

Write $S = L(\mathbf{S})$, $S_{\mathrm{new}} = L(\mathbf{S_{new}})$.

Lemma 8.13.1:

Let $\mathbf{S_{new}} = T_\alpha \mathbf{S}$ (with $\alpha \in \Sigma_u$). Then

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{new}}) = \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) \qquad (*)$$

Proof:

If $h_\alpha \geq 0$ then $T_\alpha = id$, $\mathbf{S_{new}} = \mathbf{S}$, and there is nothing to prove.

If $h_\alpha < 0$ and $\alpha^\uparrow = \emptyset$ then $\mathbf{S_{new}} = \text{NULL}$. Also, as $\alpha^j \in L(\mathbf{G})$ for all $j \geq 0$, and $y_o + jh_\alpha < 0$ for j sufficiently large, we have $\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) = \emptyset$, establishing (*) for this case.

It remains to assume $h_\alpha < 0$ and $\alpha^\uparrow \neq \emptyset$, namely $\alpha^\uparrow = \{i\}$, with $e_\alpha(i) = -1$. Let $s \in L(\mathbf{G})$, $n_\sigma = |s|_\sigma (\sigma \in \Sigma)$. Then

$$0 \leq k := \xi(x_o, s)(i) = x_o(i) + \sum_\sigma n_\sigma e_\sigma(i)$$

Note that it suffices to sum over $\sigma \in i^\uparrow \cup i^\downarrow$. First suppose $s \in \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{new}})$. Then

$$\begin{aligned}
0 &\leq \eta_{\text{new}}(y_{o,\text{new}}, s) \\
&= y_{o,\text{new}} + \sum_\sigma n_\sigma h_{\sigma,\text{new}} \\
&= y_o + x_o(i) h_\alpha + \sum_{\sigma \in i^\uparrow \cup i^\downarrow} n_\sigma [h_\sigma + e_\sigma(i) h_\alpha] + \sum_{\sigma \notin i^\uparrow \cup i^\downarrow} n_\sigma h_\sigma \\
&= y_o + \sum_\sigma n_\sigma h_\sigma + \left[x_o(i) + \sum_{\sigma \in i^\uparrow \cup i^\downarrow} n_\sigma e_\sigma(i) \right] h_\alpha \\
&= y_o + \sum_\sigma n_\sigma h_\sigma + k h_\alpha
\end{aligned}$$

and so, as $kh_\alpha \leq 0$,

$$y_o + \sum_\sigma n_\sigma h_\sigma \geq 0$$

The same argument applies to each prefix $s' \leq s$, showing that $\eta(y_o, s)!$, namely $s \in L(\mathbf{G}) \cap S$. Therefore

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{new}}) \subseteq L(\mathbf{G}) \cap S$$

so that

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{new}}) \subseteq \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$$

For the reverse inclusion, take $s \in \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$. With k as before we have, as $e_\alpha(i) = -1$,

$$x_o(i) + \sum_{\sigma \neq \alpha} n_\sigma e_\sigma(i) + (n_\alpha + k) e_\alpha(i) = 0 \tag{$\dagger$}$$

so that $\xi(x_o, s\alpha^j)!$ for $0 \leq j \leq k$, with $\xi(x_o, s\alpha^k)(i) = 0$. By controllability

$$s\alpha^k \in \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$$

In particular $\eta(y_o, s\alpha^k)!$, namely

$$y_o + \sum_{\sigma \neq \alpha} n_\sigma h_\sigma + (n_\alpha + k) h_\alpha \geq 0 \tag{$\dagger\dagger$}$$

Calculating as before,

$$\begin{aligned}
y_{o,\text{new}} + \sum_\sigma n_\sigma h_{\sigma,\text{new}} &= y_o + \sum_\sigma n_\sigma h_\sigma + \left[x_o(i) + \sum_{\sigma \in i^\uparrow \cup i^\downarrow} n_\sigma e_\sigma(i) \right] h_\alpha \\
&= y_o + \sum_{\sigma \neq \alpha} n_\sigma h_\sigma + n_\alpha h_\alpha - k e_\alpha(i) h_\alpha \\
&\qquad + \left[x_o(i) + \sum_{\sigma \neq \alpha} n_\sigma e_\sigma(i) + (n_\alpha + k) e_\alpha(i) \right] h_\alpha \\
&= y_o + \sum_{\sigma \neq \alpha} n_\sigma h_\sigma + (n_\alpha + k) h_\alpha \\
&\geq 0
\end{aligned}$$

using ($\dagger$) and ($\dagger\dagger$). By the same argument applied to each prefix of s, we conclude that $\eta_{\text{new}}(y_{o,\text{new}}, s)!$, namely

$$s \in L(\mathbf{G}) \cap S_{\text{new}}$$

and therefore

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) \subseteq \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{new}})$$

as required. $\square$

Lemma 8.13.2:

Let $\mathbf{G}, \mathbf{S}$ satisfy Condition 2 of Theorem 8.13.1, and assume $\alpha \in \Sigma^\nabla$ with $\alpha^\uparrow = \{i\}$. Let $\mathbf{S}_{\mathbf{new}} = T_\alpha(\mathbf{S})$. Then

$$(\forall \sigma \notin i^\uparrow) h_{\text{new},\sigma} \geq h_\sigma$$

$\square$

Corollary 8.13.1:

Under the conditions of Theorem 8.13.1, let the result of Procedure 1 be

$$\text{LEList} = [\alpha_1, ..., \alpha_k]$$

Then for $j = 2, ..., k$, T_{α_j} does not decrease the components of h_{α_i} for $i = 1, ..., j-1$.

Proof:

Let $\alpha_j^{\uparrow} = \{l\}$ and in turn set $i = 1, ..., j-1$. Note that $\alpha_i \notin l^{\uparrow}$, since otherwise α_i cannot be a leaf event in the restriction of $\mathbf{G}^{\nabla}$ to $I^{\nabla}, \{\alpha_i, ..., \alpha_j, ..., \alpha_k\}$, contrary to Procedure 1. The result follows by Lemma 8.13.2 with $\alpha = \alpha_j$ and putting $\sigma = \alpha_1, ..., \alpha_{j-1}$ in turn. $\square$

Exercise 8.13.4: In the example of Exercise 8.13.3, check that T_{σ_2} does not decrease h_6; T_{σ_7} does not decrease $h_6, h_2; ...; T_{\sigma_4}$ does not decrease h_i for $i = 6, 2, 7, 1, 5, 3$. $\diamond$

Proof of Theorem 8.13.1:

Assume first that Procedure 2 yields $\mathbf{S_{fin}} \neq \mathbf{NULL}$. It will be shown that $\mathbf{S_{fin}}$ is a VDESI for F^*. By construction, LEList contains all $\alpha \in \Sigma_u$ such that $h_\alpha < 0$. Also, if at some stage in Procedure 2 we have $\mathbf{S}'' = T_\alpha \mathbf{S}'$, say, then $h_\alpha(\mathbf{S}'') \geq 0$. By Corollary 8.13.1 it follows that for all $\sigma \in \Sigma_u, h_{\text{fin},\sigma} \geq 0$.

Write $S_{\text{fin}} = L(\mathbf{S_{fin}})$. We claim that S_{fin} is controllable with respect to $\mathbf{G}$. Indeed if $s \in S_{\text{fin}} \cap L(\mathbf{G})$ and $\sigma \in \Sigma_u$ with $s\sigma \in L(\mathbf{G})$, let $\eta_{\text{fin}}(y_{\text{fin},o}, s) = y$, so

$$\eta_{\text{fin}}(y_{\text{fin},o}, s\sigma) = y + h_{\text{fin},\sigma} \geq y,$$

namely $s\sigma \in S_{\text{fin}}$.

It follows that

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{fin}}) = L(\mathbf{G}) \cap S_{\text{fin}}$$

By Lemma 8.13.1,

$$\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S_{\text{fin}}) = \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$$

and thus

$$L(\mathbf{G}) \cap S_{\text{fin}} = \sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S)$$

namely $\mathbf{S_{fin}}$ is a VDESI for F^*.

Finally we note that $\mathbf{S_{fin}} \neq \mathbf{NULL}$ if and only if, at each stage of Procedure 2, we have both $\alpha^{\uparrow} \neq \emptyset$ and $y_{\text{new},o} \geq 0$. But if $\alpha^{\uparrow} = \emptyset$ (i.e. $\alpha \in \Sigma^{\nabla}$ is permanently enabled) there must exist a string $s \in \Sigma_u^* \cap L(\mathbf{G}^\nabla)$ such that $\eta(y_o, s) < 0$, i.e. $s \notin L(\mathbf{S})$, hence $\sup \mathcal{C}_{\mathbf{G}}(L(\mathbf{G}) \cap S) = \emptyset$. The same conclusion follows if $y_{\text{new},o} < 0$ at some stage. Thus (under the conditions of the theorem) the requirement $\mathbf{S}_{\mathbf{fin}} \neq \mathbf{NULL}$ is necessary and sufficient for the existence of F^*.

□

Remark 8.13.2:

Procedure 2 could be modified by dropping the condition that $y_{\text{new},o} \geq 0$ at each stage, and simply reporting whatever value of $y_{\text{fin},o}$ is calculated at termination. A result $y_{\text{fin},o} < 0$ would then represent the least amount by which the original value of y_o should be raised to yield an acceptable (nonnegative) result. ◇

It is straightforward to extend Theorem 8.13.1 to the case of a p-dimensional specification, merely by treating each component as an independent scalar in modular fashion.

Exercise 8.13.5: Justify the last statement in detail, and illustrate with a modular VDESI of dimension 2.

Exercise 8.13.6: Let $\mathbf{G}$ be a 5-dimensional VDES over $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ with $\Sigma_c = \{\sigma_4\}$ and

$$
\begin{aligned}
E &= \begin{bmatrix} -1 & -1 & 2 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \\
h &= [-1 \;\; 0 \;\; 0 \;\; 0 \;\; 0]
\end{aligned}
$$

Let $x_o = [0 \; 0 \; 0 \; 4 \; 0]$, $y_o = 2$. Apply Theorem 8.13.1 with Procedures 1 and 2 to obtain a VDESI for F^*.

Exercise 8.13.7: Show that Condition 2 of Theorem 8.13.1 cannot be dropped altogether. **Hint:** In the following example, verify that Condition 2 is violated, and F^* exists but cannot be implemented as a VDES. Let $\mathbf{G}$ be 2-dimensional with $\Sigma = \{\alpha, \beta, \gamma\}$, $\Sigma_u = \{\alpha\}$ and

$$
\begin{aligned}
E &= \begin{bmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \\
h &= [-1 \;\; 0 \;\; 0]
\end{aligned}
$$

Let $x_o = [0 \; 0]$, $y_o = 2$. Show that optimal control is given by

$$
\begin{aligned}
F_\beta^*(x_1, x_2, y) &= 1 \;\text{ iff }\; \min(x_1, x_2 + 1) \leq y \\
F_\gamma^*(x_1, x_2, y) &= 1 \;\text{ iff }\; \min(x_1 + 1, x_2) \leq y
\end{aligned}
$$

Show that neither of these enablement conditions can be written as an inequality (or conjunction of inequalities) linear in the occurrence vector, so there can be no VDESI for either F^*_β or F^*_γ.

Exercise 8.13.8: Show that Condition 2 of Theorem 8.13.1 is not necessary. **Hint:** In the following example, verify that Condition 2 is violated, but F^* exists and does have a VDES implementation. Let $\mathbf{G}$ be 2-dimensional with $\Sigma = \{\alpha, \beta, \gamma\}$, $\Sigma_c = \{\gamma\}$ and

$$\begin{aligned} E &= \begin{bmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix} \\ h &= [-1 \;\; 0 \;\; 0] \end{aligned}$$

Let $x_o = [0\ 0]$, $y_o = 2$. Check that F^*_γ exists and has the VDESI $\mathbf{S}$ with $h(\mathbf{S}) = [0\ 0\ -1]$ and initial value 2.

Exercise 8.13.9: Let $\mathbf{G}$ be 1-dimensional with $\Sigma = \{\alpha, \beta, \gamma\}$, $\Sigma_c = \{\gamma\}$,

$$\begin{aligned} E &= [-2 \;\; -1 \;\; 1] \\ h &= [-3 \;\; 1 \;\; 0] \end{aligned}$$

Investigate the existence of F^* and a VDES implementation. What conclusion can be drawn about Theorem 8.13.1?

Exercise 8.13.10: Repeat Exercise 8.13.9 for the following. Let $\mathbf{G}$ be 4-dimensional with $\Sigma = \{\alpha, \beta, \lambda, \mu\}$, $\Sigma_u = \{\lambda\}$,

$$\begin{aligned} E &= \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 2 \\ 2 & 1 & -2 & 0 \\ 0 & 0 & 1 & -2 \end{bmatrix} \\ x(0) &= [1 \;\; 2 \;\; 0 \;\; 2] \in \mathbb{N}^{4\times 1} \end{aligned}$$

and specification $x_4 \le 2$.

Exercise 8.13.11: Continuing Exercise 8.13.10, find a 'place invariant' (cf. Exercise 8.3.2) $[c_1 \;\; c_2 \;\; c_3 \;\; c_4]$ with the $c_i > 0$. From this derive *a priori* bounds $x_1, x_4 \le 4$; $x_2, x_3 \le 8$. Using these bounds construct state models $\mathbf{X_i}$ for the VDES components x_i, and form the plant model $\mathbf{X}$ as their synchronous product. Use *TCT* to obtain the optimal supervisor enforcing $(x_4 \le 2)$. Verify consistency with your result in Exercise 8.13.10.

8.14 Appendix: Three Examples from Petri Nets

We provide three examples to show how supervisory control problems described in terms of Petri nets can be treated in the automaton framework of these Notes, Chapters 3, 4 and 6. Commented output from the *TCT* MAKEIT.TXT files is provided for the reader's convenience. Details of problem formulation can be found in the cited literature.

Example 8.14.1: Manufacturing Workcell

Ref: K. Barkaoui, I. Ben Abdallah. A deadlock prevention method for a class of FMS. Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics. Vancouver, Canada, October 1995, pp.4119-4124.

The system is a manufacturing workcell consisting of two input bins I1,I2, four machines M1,...,M4, two robots R1,R2, and two output bins O1,O2. Two production sequences, for RED and GRN workpieces, run concurrently; these are:

GRN: I1 → R1 → (M1 or M2) → R1 → M3 → R2 → O1

RED: I2 → R2 → M4 → R1 → M2 → R1 → O2

In the simplest case, treated here, at most one workpiece of each type (red, green) is allowed in the system at any one time.

Since the machines and production sequences share the robots as resources, there is the *a priori* possibility of deadlock. In fact, without control there is exactly one deadlock state (20) in TEST = **meet**(CELL,SPEC). At this state the components are in states:

R1	R2	M1	M2	M3	M4	RED	GRN
1	0	0	1	0	0	2	3

One deadlock sequence is:

R1 takes in green part (ev 11 in CELL)
R2 takes in red part (ev 91)
R1 loads M2 with green part (ev 21)
R2 loads M4 with red part (ev 101)
R1 unloads red part from M4 (ev 111)

At this point, R1 holds a red, having just unloaded it from M4 (event 111), while M2 holds a green, having finished processing it. R1 must load M2 with the red it's holding (ev 121) but cannot do so because M2 holds the green, which only R1 can unload (ev 41). The deadlock occurs because both R1 and M2 are "full" (with a red, green respectively), and there's no mechanism for making the required swap. The cure is easy: simply eliminate state 20 from TEST; the result is then exactly the optimal controlled behavior SUP. So instead of the old blocking sequence [11,91,21,101,111] we now have [11,91,21, 101,41]; in other words GRN is allowed to progress to its state 4 (ev 41) before RED is allowed to progress to its state 3 (ev 111). The problem arises because a single robot (R1) is shared by 2 machines M2,M4; and M2 is shared by 2 processes (RED,GRN). At the deadlock state RED and GRN have conflicting requirements for M2 and therefore on R1, which is deadlocked because of its previous action on M4. Conclusion: careful sequencing of the interleaved processes RED, GRN is needed to avoid deadlock due to conflicting demands on the shared resources R1 and M2. Of course, this is achieved 'automatically' by **supcon**.

R1	=	Create(R1,[mark 0],[tran [0,11,1],[0,41,1],[0,51,1],[0,111,1], [0,131,1],[1,21,0], [1,31,0],[1,61,0],[1,121,0],[1,141,0]]) (2,10)
R2	=	Create(R2,[mark 0],[tran [0,71,1],[0,91,1],[1,81,0],[1,101,0]]) (2,4)
M1	=	Create(M1,[mark 0],[tran [0,31,1],[1,51,0]]) (2,2)
M2	=	Create(M2,[mark 0],[tran [0,21,1],[0,121,1],[1,41,0],[1,131,0]]) (2, 4)
M3	=	Create(M3,[mark 0],[tran [0,61,1],[1,71,0]]) (2,2)
M4	=	Create(M4,[mark 0],[tran [0,101,1],[1,111,0]]) (2,2)
CELL	=	Sync(R1,R2) (4,28) Blocked_events = None Computing time = 00:00:00.00
CELL	=	Sync(CELL,M1) (8,52) Blocked_events = None Computing time = 00:00:00.00
CELL	=	Sync(CELL,M2) (16,88) Blocked_events = None Computing time = 00:00:00.00
CELL	=	Sync(CELL,M3) (32,160) Blocked_events = None Computing time = 00:00:00.00
CELL	=	Sync(CELL,M4) (64,288) Blocked_events = None Computing time = 00:00:00.00
ALL	=	Allevents(CELL) (1,14) Computing time = 00:00:00.00
GRN	=	Create(GRN,[mark 0],[tran [0,11,1],[1,21,2],[1,31,3],[2,41,4],[3,51,4],[4,61,5], [5,71,6],[6,81,0]]) (7,8)
RED	=	Create(RED,[mark 0],[tran [0,91,1],[1,101,2],[2,111,3],[3,121,4],[4,131,5], [5,141,0]]) (6,6)
SPEC	=	Sync(RED,GRN) (42,90) Blocked_events = None Computing time = 00:00:00.00
false	=	Nonconflict(CELL,SPEC) Computing time = 00:00:00.00

TEST	=	Meet(CELL,SPEC) (35,61) Computing time = 00:00:00.00
SUP	=	Supcon(CELL,SPEC) (34,60) Computing time = 00:00:00.00
SUP	=	Condat(CELL,SUP) Controllable. Computing time = 00:00:00.00

[Only events 51,71 do not appear in the Condat table; therefore they could be replaced by uncontrollable counterparts (say 50,70) without changing the controlled behavior.]

SIMSUP	=	SupReduce(CELL,SUP,SUP) (11,38; lo_bnd 8)[3]
STEST	=	Meet(CELL,SIMSUP) (34,60) Computing time = 00:00:00.00
true	=	Isomorph(STEST,SUP;identity) Computing time = 00:00:00.00
SIMSUP	=	Condat(CELL,SIMSUP) Controllable. Computing time = 00:00:00.00

The following shows that removing the single blocking state 20 is enough to obtain the optimal control from the naive behavior TEST.

ETEST	=	Edit(TEST,[states -[20]],rch) (34,60)
true	=	Isomorph(ETEST,SUP;identity) Computing time = 00:00:00.00

Example 8.14.2: Piston Rod Robotic Assembly Cell

Ref: J.O. Moody, P.J. Antsaklis. Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer, 1998; Sect. 8.4.

With reference to Fig. 8.11 of the cited text, the system consists of an M-1 robot performing various tasks (Petri net places p4,p5,p6,p7), and similarly an S-380 robot (p2,p3); p1 is used for initialization.

---

[3] See Sect. 3.10

M-1 ROBOT

To model this we replace p4 by a generator capable of holding up to two piston pulling tools in a two-slot buffer MR1; the tools are generated by event 40 and selected for use by event 41. The event sequence [41,51,60,70,80] tracks the installation of a cap on a piston rod and the conveyance of its engine block out of the work space. In our model, up to four operations in this sequence could be progressing simultaneously, although a specification (below) will limit this number to one.

MR1	=	Create(MR1,[mark 0],[tran [0,40,1],[1,40,2],[1,41,0],[2,41,1]]) (3, 4)
MR2	=	Create(MR2,[mark 0],[tran [0,41,1],[1,51,0]]) (2,2)
MR3	=	Create(MR3,[mark 0],[tran [0,51,1],[1,60,0]]) (2,2)
MR4	=	Create(MR4,[mark 0],[tran [0,60,1],[1,70,0]]) (2,2)
MR5	=	Create(MR5,[mark 0],[tran [0,70,1],[1,80,0]]) (2,2)
MROB	=	Sync(MR1,MR2) (6,9) Computing time = 00:00:00.00
MROB	=	Sync(MROB,MR3) (12,21) Computing time = 00:00:00.00
MROB	=	Sync(MROB,MR4) (24,48) Computing time = 00:00:00.00
MROB	=	Sync(MROB,MR5) (48,108) Computing time = 00:00:00.00

S-380 ROBOT

Starting from the ready-to-work condition, this robot performs the event sequence [10,20,30] corresponding to readying parts for assembly; its work cycle is closed by event 80.

SROB = Create(SROB,[mark 0],[tran [0,10,1],[1,20,2],[2,30,3], [3,80,0]]) (4,4)

PLANT	=	Sync(MROB,SROB) (192,504) Computing time = 00:00:00.00

Note that the only controllable events are 41,51, which more than satisfies the authors' requirement that events 60,70,80 be uncontrollable.

There are 3 specifications, as detailed in the authors' equations 8.11-8.13. These are linear inequalities on markings, which are easily converted (by inspection of the PN) into counting constraints on suitable event pairs. For instance, 8.12 requires that m4+m5+m6+m7 $\leq$ 1, where mi is the marking of place pi; by inspection, this is equivalent to

$$(|41| - |51|) + (|51| - |60|) + (|60| - |70|) + (|70| - |80|) \leq 1,$$

or simply $|41| - |80| \leq 1$; here $|k|$ is the number of firings of transition k since the start of the process. By inspection of the PN it is clear that the inequality forces events 41,80 to alternate, with 41 occurring first; hence SPEC2, below.

SPEC1	=	Create(SPEC1,[mark 0],[tran [0,10,1],[1,30,0]]) (2,2)
SPEC2	=	Create(SPEC2,[mark 0],[tran [0,41,1],[1,80,0]]) (2,2)
SPEC3	=	Create(SPEC3,[mark 0],[tran [0,30,1],[1,51,0]]) (2,2)
SPEC	=	Sync(SPEC1,SPEC2) (4,8) Computing time = 00:00:00.00
SPEC	=	Sync(SPEC,SPEC3) (8,18) Computing time = 00:00:00.00
PLANTALL	=	Allevents(PLANT) (1,9) Computing time = 00:00:00.00
SPEC	=	Sync(SPEC,PLANTALL) (8,50) Computing time = 00:00:00.00

The supremal supervisor can now be computed, then simplified by the control-congruence reduction procedure.

SUPER = Supcon(PLANT,SPEC) (33,60)
Computing time = 00:00:00.00

SUPER = Condat(PLANT,SUPER) Controllable.
Computing time = 00:00:00.00

SIMSUP = SupReduce(PLANT,SUPER,SUPER) (3,12; lo_bnd = 3)[4]

Thus SIMSUP is strictly minimal.

X = Meet(PLANT,SIMSUP) (33,60)
Computing time = 00:00:00.06

true = Isomorph(X,SUPER;identity)
Computing time = 00:00:00.00

SIMSUP = Condat(PLANT,SIMSUP) Controllable.
Computing time = 00:00:00.00

TEST = Meet(PLANT,SIMSUP) (33,60)
Computing time = 00:00:00.00

true = Isomorph(TEST,SUPER;identity)
Computing time = 00:00:00.00

The authors specify four auxiliary constraints 8.14-8.17, of the form already discussed; we model these as follows, and create the auxiliary specification ASPEC. We test these constraints against the existing controlled behavior SUPER, and confirm that they are already satisfied.

ASP1 = Create(ASP1,[mark 0],[tran [0,20,1],[1,30,0]]) (2,2)

ASP2 = Create(ASP2,[mark 0],[tran [0,41,1],[1,60,0]]) (2,2)

ASP3 = Create(ASP3,[mark 0],[tran [0,60,1],[1,70,0]]) (2,2)

ASP4 = Create(ASP4,[mark 0],[tran [0,70,1],[1,80,0]]) (2,2)

ASPEC = Sync(ASP1,ASP2) (4,8)
Computing time = 00:00:00.00

[4]See footnote 3

ASPEC	=	Sync(ASPEC,ASP3) (8,18) Computing time = 00:00:00.00
ASPEC	=	Sync(ASPEC,ASP4) (16,40) Computing time = 00:00:00.00
ASPEC	=	Sync(ASPEC,PLANTALL) (16,88) Computing time = 00:00:00.00
COASPEC	=	Complement(ASPEC,[]) (17,153) Computing time = 00:00:00.00
X	=	Meet(SUPER,COASPEC) (33,60) Computing time = 00:00:00.00
TX	=	Trim(X) (0,0) Computing time = 00:00:00.00

Unobservable events: we assume with the authors that events 51,60,70 have become unobservable. As a simplifying assumption on supervisor design, we consider that controllable event 51 will now not be subject to disablement. Thus we could (but will not) relabel event 51 as 50 throughout. Our new assumption allows us to treat the problem as an instance of SCOP (these Notes, Sect. 6.5). We therefore compute as follows.

N	=	Supnorm(SPEC,PLANT,[51,60,70]) (24,39) Computing time = 00:00:00.22
NO	=	Project(N,Null[51,60,70]) (15,24) Computing time = 00:00:00.00
PLANTO	=	Project(PLANT,Null[51,60,70]) (60,129) Computing time = 00:00:00.11
SUPERO	=	Supcon(PLANTO,NO) (15,24) ["Observer's supervisor"] Computing time = 00:00:00.00
SUPERO	=	Condat(PLANTO,SUPERO) Controllable. ["Observer's supervisor"] Computing time = 00:00:00.00
OSUPER	=	Selfloop(SUPERO,[51,60,70]) (15,69) [Feasible supervisor] Computing time = 00:00:00.05

true = Nonconflict(PLANT,OSUPER)
Computing time = 00:00:00.00

K = Meet(PLANT,OSUPER) (24,39) [Controlled behavior using feasible supervisor]
Computing time = 00:00:00.00

SIMSUPO = SupReduce(PLANTO,SUPERO,SUPERO) (2,7; lo_bnd = 2)[5]

Thus SIMSUPO is strictly minimal.

SIMSUPO = Condat(PLANTO,SIMSUPO) Controllable.
Computing time = 00:00:00.05

TESTO = Meet(PLANTO,SIMSUPO) (15,24)
Computing time = 00:00:00.00

true = Isomorph(TESTO,SUPERO;identity)
Computing time = 00:00:00.00

We'll check that, as expected, controlled behavior K using the feasible supervisor is more restricted than the original controlled behavior SUPER (which of course was computed without assuming any observational constraint). Nevertheless, K is adequate for performance of the assembly process: for instance the K-string [10,20,30,40,41,51,60,70,80] is a full assembly cycle.

COSUPER = Complement(SUPER,[]) (34,306)
Computing time = 00:00:00.00

X = Meet(K,COSUPER) (24,39)
Computing time = 00:00:00.00

TX = Trim(X) (0,0)
Computing time = 00:00:00.00

Some routine checks, in principle redundant:

true = Nonconflict(PLANT,OSUPER)
Computing time = 00:00:00.00

OSUPER = Condat(PLANT,OSUPER) Controllable.
Computing time = 00:00:00.00

As expected, OSUPER never disables unobservable event 51.

---

[5] See footnote 3.

Example 8.14.3: Unreliable Machine (Deadlock Avoidance)

Ref: J.O. Moody, P.J. Antsaklis. Deadlock avoidance in Petri nets with uncontrollable transitions. Proc. 1998 American Automatic Control Conference. Reproduced in J.O. Moody, P.J. Antsaklis. Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer, 1998; Sect. 8.3 (pp.122-129).

This is a problem which, in the authors' Petri net formulation, requires finding the system's two "uncontrolled siphons." By contrast, the CTCT solution is fast and immediate, requiring no special analysis.

The system model consists of a machine M1 containing two 1-slot output buffers M1C (for completed workpieces) and M1B (for damaged workpieces, which result when M1 breaks down), together with two dedicated AGVs to clear them. M1 is the conventional RW machine. Event 10 (successful completion) increments M1C, which must be cleared (event 14) by AGV1 before M1 can restart; event 12 (breakdown) increments M1B, which must be cleared (event 16) by AGV2 before M1 can be repaired after breakdown (event 13); these requirements are enforced by SPEC1C, SPEC1B respectively. The workspace near the buffers can be occupied by only one AGV at a time: this is enforced by SPEC1; the final SPEC model is **sync**(SPEC1,SPEC1C,SPEC1B). Blocking would occur if, for instance, AGV1 moved into position to clear its buffer M1C, but M1B rather than M1C was filled; or AGV2 moved into position to clear its buffer M1B, but M1C rather than M1B was filled; in each case the positioned AGV would lock out the other.

Modeling the plant

M1	=	Create(M1,[mark 0],[tran [0,11,1],[1,10,0],[1,12,2],[2,13,0]]) (3,4)
M1C	=	Create(M1C,[mark 0],[tran [0,10,1],[1,14,0]]) (2,2)
M1B	=	Create(M1B,[mark 0],[tran [0,12,1],[1,16,0]]) (2,2)
AGV1	=	Create(AGV1,[mark 0],[tran [0,101,1],[1,14,2],[2,100,0]]) (3,3)
AGV2	=	Create(AGV2,[mark 0],[tran [0,201,1],[1,16,2],[2,200,0]]) (3,3)
P	=	Sync(M1,M1C) (6,10) Blocked_events = None Computing time = 00:00:00.00
P	=	Sync(P,M1B) (12,24) Blocked_events = None Computing time = 00:00:00.00
P	=	Sync(P,AGV1) (36,84) Blocked_events = None Computing time = 00:00:00.00

P = Sync(P,AGV2) (108,288) Blocked_events = None
Computing time = 00:00:00.00

PALL = Allevents(P) (1,10)
Computing time = 00:00:00.00

Modeling the specification

SPEC1 = Create(SPEC,[mark 0],[tran [0,101,1],[0,201,2],[1,100,0],[2,200,0]]) (3,4)

SPEC1C = Create(SPEC1C,[mark 0],[tran [0,10,1],[0,11,0],[1,14,0]]) (2,3)

SPEC1B = Create(SPEC1B,[mark 0],[tran [0,12,1],[0,13,0],[1,16,0]]) (2,3)

SPEC = Sync(SPEC1,SPEC1C) (6,17) Blocked_events = None
Computing time = 00:00:00.00

SPEC = Sync(SPEC,SPEC1B) (12,52) Blocked_events = None
Computing time = 00:00:00.00

SPEC = Sync(SPEC,PALL) (12,52) Blocked_events = None
Computing time = 00:00:00.00

false = Nonconflict(P,SPEC)
Computing time = 00:00:00.00

Blocking could occur in the absence of supervisory control. Some blocking sequences are [11,10,201],[201,11,10], [11,12,101], [101,11,12]. These result in the situations described earlier, where an AGV in the workspace locks out the other, required AGV.

PSPEC = Meet(P,SPEC) (24,40)
Computing time = 00:00:00.00

false = Nonconflict(PSPEC,PALL)
Computing time = 00:00:00.00

MPSPEC = Minstate(PSPEC) (23,40)
Computing time = 00:00:00.06

Computing the supremal supervisor

SUP	=	Supcon(P,SPEC) (16,24) Computing time = 00:00:00.00
SUP	=	Condat(P,SUP) Controllable. Computing time = 00:00:00.00

Computing a simplified supervisor

SIMSUP	=	SupReduce(P,SUP,SUP) (5,23; lo_bnd = 4)[6]
X	=	Meet(P,SIMSUP) (16,24) Computing time = 00:00:00.00
true	=	Isomorph(SUP,X;identity) Computing time = 00:00:00.06
SIMSUP	=	Condat(P,SIMSUP) Controllable. Computing time = 00:00:00.00

It's easy to check by inspection that SIMSUP prohibits the blocking sequences listed above.

8.15 Notes and References

This chapter is based mainly on work of Y. Li [T17, C24, C27, C29, C33, J21, J22], N.-Q. Huang [C35, T16], and S.-L. Chen [C47, T24]. Further developments can be found in [T33]. Exercise 8.13.10 is adapted from Moody & Antsaklis [1998], Sect. 4.5.

[6]See footnote 3.

Chapter 9

Supervisory Control of Timed Discrete-Event Systems

9.1 Introduction

In this chapter we augment the framework of Chapters 3 and 4 with a timing feature. The occurrence of an event, relative to the instant of its enablement, will be constrained to lie between a lower and upper time bound, synchronized with a postulated global digital clock. In this way we are able to capture timing issues in a useful range of control problems. Timing introduces a new dimension of DES modelling and control, of considerable power and applied interest, but also of significant complexity. Nevertheless, it will turn out that our previous concept of controllability, and the existence of maximally permissive supervisory controls, can be suitably generalized. The enhanced setting admits subsystem composition (analogous to synchronous product), and the concept of forcible event as an event that preempts the tick of the clock. An example of a manufacturing cell illustrates how the timed framework can be used to solve control synthesis problems which may include logic-based, temporal and quantitative optimality specifications.

The chapter is organized as follows. The base model of timed discrete-event systems (TDES) is introduced in Sect. 2 and illustrated in Sects. 3 and 4. The role of time bounds as specifications is indicated in Sect. 5. Composition of TDES is defined and illustrated in Sects. 6 and 7. Sect. 8 introduces TDES controllability and forcible events, leading to maximally permissive supervision in Sect. 9. A small academic example ('the endangered pedestrian') is treated in Sect. 10, followed by a simple but nontrivial application to a manufacturing workcell in Sect. 11. Modular supervision is introduced in Sect. 12, and some possible extensions in future work outlined by way of conclusion in Sect. 13.

Our timed framework is amenable to computation in the style of *TCT*; the enhanced package, designed to be used with these notes, is *TTCT*.

9.2 Timed Discrete-Event Systems

To develop the base model we begin with the usual 5-tuple of form

$$\mathbf{G_{act}} = (A, \Sigma_{act}, \delta_{act}, a_o, A_m)$$

except that the 'state set' often designated Q has been replaced with an *activity set* A whose elements are *activities* a. While in principle the state set need not be finite, in applications it nearly always is; here we shall restrict A to be finite for technical simplicity. Σ_{act} is a finite alphabet of *event labels* (or simply, *events*). We stress that, in the interpretation, activities have duration in time, while events are instantaneous. The *activity transition function* is, as expected, a partial function $\delta_{act} : A \times \Sigma_{act} \to A$. An *activity transition* is a triple $[a, \sigma, a']$, with $a' = \delta_{act}(a, \sigma)$. In line with standard terminology, a_o is the *initial activity* and $A_m \subseteq A$ is the subset of *marker* activities. Let $\mathbb{N}$ denote the natural numbers $\{0, 1, 2, ...\}$. In Σ_{act}, each transition (label) σ will be equipped with a *lower time bound* $l_\sigma \in \mathbb{N}$ and an *upper time bound* $u_\sigma \in \mathbb{N} \cup \{\infty\}$. To reflect two distinct possibilities of basic interest we partition Σ_{act} according to

$$\Sigma_{act} = \Sigma_{spe} \dot{\cup} \Sigma_{rem}$$

where 'spe' denotes 'prospective' and 'rem' denotes 'remote'. If an event σ is *prospective*, its upper time bound u_σ is finite ($0 \le u_\sigma < \infty$) and $0 \le l_\sigma \le u_\sigma$; while if σ is *remote*, we set $u_\sigma = \infty$ and require $0 \le l_\sigma < \infty$. The modelling function of time bounds is straightforward: l_σ would typically represent a delay, in communication or in control enforcement; u_σ a hard deadline, imposed by legal specification or physical necessity. The formal role of time bounds will be treated in detail below. The triples $(\sigma, l_\sigma, u_\sigma)$ will be called *timed events*, and for these we write

$$\Sigma_{tim} := \{(\sigma, l_\sigma, u_\sigma) | \sigma \in \Sigma_{act}\}$$

For $j, k \in \mathbb{N}$ write $[j, k]$ for the set of integers i with $j \le i \le k$, and let

$$T_\sigma = \begin{cases} [0, u_\sigma] & \text{if} \quad \sigma \in \Sigma_{spe} \\ [0, l_\sigma] & \text{if} \quad \sigma \in \Sigma_{rem} \end{cases}$$

T_σ will be called the *timer interval* for σ. We can now define the *state set*

$$Q := A \times \prod \{T_\sigma | \sigma \in \Sigma_{act}\}$$

Thus a *state* is an element of form

$$q = (a, \{t_\sigma | \sigma \in \Sigma_{act}\})$$

where $a \in A$ and the $t_\sigma \in T_\sigma$; namely q consists of an activity a together with a tuple assigning to each event $\sigma \in \Sigma_{act}$ an integer in its timer interval T_σ. The component t_σ of q will be called the *timer* of σ in q. If $\sigma \in \Sigma_{spe}$, the *current deadline* for σ is t_σ, while the *current delay* is $\max(t_\sigma + l_\sigma - u_\sigma, 0)$. If $\sigma \in \Sigma_{rem}$, the *current delay* is t_σ (while the current deadline may be regarded as infinite). The value u_σ (resp. l_σ) for a prospective (resp. remote) event σ will be called the *default value* of t_σ. The *initial state* is

$$q_o := (a_o, \{t_{\sigma o} | \sigma \in \Sigma_{act}\})$$

where the t_σ are set to their default values

$$t_{\sigma o} := \begin{cases} u_\sigma & \text{if} \quad \sigma \in \Sigma_{spe} \\ l_\sigma & \text{if} \quad \sigma \in \Sigma_{rem} \end{cases}$$

The *marker state subset* will be taken to be of the form

$$Q_m \subseteq A_m \times \prod \{T_\sigma | \sigma \in \Sigma_{act}\}$$

namely a marker state comprises a marker activity together with a suitable assignment of the timers.

We introduce one additional event, written *tick*, to represent 'tick of the global clock', and take for our total set of events

$$\Sigma := \Sigma_{act} \dot\cup \{tick\}$$

The *state transition function* will be defined in detail below; as expected it will be a partial function

$$\delta : Q \times \Sigma \to Q$$

We now write

$$\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$$

With the above definitions, including the partition of Σ as

$$\Sigma = \Sigma_{spe} \dot\cup \Sigma_{rem} \dot\cup \{tick\}$$

and an assignment of time bounds, $\mathbf{G}$ will be called a *timed discrete-event system (TDES)*. For the purpose of display we may employ the *activity transition graph (ATG)* of $\mathbf{G}$, namely the ordinary transition graph of $\mathbf{G_{act}}$; and the *timed transition graph (TTG)* of $\mathbf{G}$, namely the ordinary transition graph of $\mathbf{G}$, incorporating the *tick* transition explicitly. In addition, by projecting out *tick* from $L(\mathbf{G})$ we can derive the *timed activity* DES $\mathbf{G_{tact}}$ over Σ_{act}, and display its transition structure as the *timed activity transition graph (TATG)* of $\mathbf{G}$. While the TATG suppresses *tick*, it does incorporate the constraints on ordering of activities induced by time bounds. As illustrated by Examples 1 and 2 below, $\mathbf{G_{tact}}$ may be much more complex than $\mathbf{G_{act}}$.

Before defining the behavioral semantics of the TDES $\mathbf{G}$ in detail, we provide an informal summary. As is customary with DES, events are thought of as instantaneous and occurring at quasi-random moments of real time $\mathbb{R}^+ = \{t|0 \le t < \infty\}$. However, we imagine measuring time only with a global digital clock with output *tickcount*: $\mathbb{R}^+ \to \mathbb{N}$, where

$$tickcount(t) := n, \qquad n \le t < n+1$$

Temporal conditions will always be specified in terms of this digital clock time; real-valued time as such, and the clock function *tickcount*, will play no formal role in the model. The temporal resolution available for modelling purposes is thus just one unit of clock time. The event *tick* occurs exactly at the real time moments $t = n$ ($n \in \mathbb{N}$). As usual, **G** is thought of as a generator of strings in Σ^*; intuitively **G** incorporates the digital clock, and thus its 'generating action' extends to the event *tick*.

Events are generated as follows. **G** starts from q_o at $t = 0$ and executes state transitions in accordance with its transition function δ, i.e. by following its TTG. $\delta(q, \sigma)$ is defined at a pair (q, σ), written $\delta(q, \sigma)!$, provided (i) $\sigma = tick$, and no deadline of a prospective event in q is zero (i.e. no prospective event is *imminent*); or (ii) σ is prospective, $q = (a, \_)$, $\delta_{act}(a, \sigma)!$, and $0 \le t_\sigma \le u_\sigma - l_\sigma$; or (iii) σ is remote, $q = (a, \_)$, $\delta_{act}(a, \sigma)!$, and $t_\sigma = 0$. An event $\sigma \in \Sigma_{act}$ is said to be *enabled* at $q = (a, \_)$ if $\delta_{act}(a, \sigma)!$, and to be *eligible* if, in addition, its timer evaluation is such that $\delta(q, \sigma)!$. Only an eligible event 'can actually occur'. If σ is not enabled, it is said to be *disabled*; if σ is not eligible, it is *ineligible*; an enabled but ineligible event will be called *pending*.

The occurrence of *tick* at q causes no change in the activity component a of q; however, the timer components t_σ are altered in accordance with the detailed rules given below. The occurrence of $\sigma \in \Sigma_{act}$ at q always resets t_σ to its default value; again, the effect on other timers will be described below. After $\sigma \in \Sigma_{act}$ first becomes enabled, its timer t_σ is decremented by one at each subsequent *tick* of the clock, until either t_σ reaches zero, or σ occurs, or σ is disabled as a result of the occurrence of some eligible transition (possibly σ itself). If σ occurs, or becomes disabled owing to some transition to a new activity, t_σ is reset to its default value, where it is held until σ next becomes re-enabled, when the foregoing process repeats.

An event $\sigma \in \Sigma_{act}$ cannot become eligible (and so, occur) prior to l_σ *ticks* of the clock after it last became enabled. A prospective event σ cannot be delayed longer than $u_\sigma - l_\sigma$ *ticks* after t_σ has 'ticked down' to $u_\sigma - l_\sigma$; thus when t_σ 'times out' to 0, σ cannot be delayed except by preemptive occurrence of some other eligible event in Σ_{act}. A remote event σ can occur any time (although it need not occur at all), as long as it remains enabled, and provided l_σ ticks have elapsed after it last became enabled. For remote events, our semantics will not distinguish between the assertions 'σ occurs eventually' (but with no hard deadline) and 'σ never occurs at all'; this is a consequence of the viewpoint that 'behavior' is a subset of Σ^* (rather than of the infinite-string language Σ^ω). It is important to note that, because of its possible non-occurrence (even if continuously enabled) a remote event is not just a 'limiting case' of a prospective event.

With the above as guidelines we now provide the formal definition of δ. Write $\delta(q, \sigma) = q'$,

where

$$q = (a, \{t_\tau | \tau \in \Sigma_{act}\}), \quad q' = (a', \{t'_\tau | \tau \in \Sigma_{act}\})$$

Then $\delta(q, \sigma)!$ if and only if

(i) $\sigma = tick$ and $(\forall \tau \in \Sigma_{spe}) \delta_{act}(a, \tau)! \Rightarrow t_\tau > 0$; or

(ii) $\sigma \in \Sigma_{spe}$, $\delta_{act}(a, \sigma)!$, and $0 \leq t_\sigma \leq u_\sigma - l_\sigma$; or

(iii) $\sigma \in \Sigma_{rem}$, $\delta_{act}(a, \sigma)!$, and $t_\sigma = 0$

The entrance state q' is defined as follows.

(i) Let $\sigma = tick$. Then $a' := a$, and

$$\text{if } \tau \text{ is prospective,} \quad t'_\tau := \begin{cases} u_\tau & \text{if } \textit{not} \quad \delta_{act}(a, \tau)! \\ t_\tau - 1 & \text{if} \quad \delta_{act}(a, \tau)! \text{ and } t_\tau > 0 \end{cases}$$

(Recall that if τ is prospective, $\delta_{act}(a, \tau)!$ and $t_\tau = 0$ then *not* $\delta(q, tick)!$)

$$\text{if } \tau \text{ is remote,} \quad t'_\tau := \begin{cases} l_\tau & \text{if } \textit{not} \quad \delta_{act}(a, \tau)! \\ t_\tau - 1 & \text{if} \quad \delta_{act}(a, \tau)! \text{ and } t_\tau > 0 \\ 0 & \text{if} \quad \delta_{act}(a, \tau)! \text{ and } t_\tau = 0 \end{cases}$$

(ii) Let $\sigma \in \Sigma_{act}$. Then $a' := \delta_{act}(a, \sigma)$, and

$$\begin{array}{ll} \text{if } \tau \neq \sigma \text{ and } \tau \text{ is prospective,} & t'_\tau := \begin{cases} u_\tau & \text{if } \textit{not} \quad \delta_{act}(a', \tau)! \\ t_\tau & \text{if} \quad \delta_{act}(a', \tau)! \end{cases} \\ \text{if } \tau = \sigma \text{ and } \sigma \text{ is prospective,} & t'_\tau := u_\sigma \\ \text{if } \tau \neq \sigma \text{ and } \tau \text{ is remote,} & t'_\tau := \begin{cases} l_\tau & \text{if } \textit{not} \quad \delta_{act}(a', \tau)! \\ t_\tau & \text{if} \quad \delta_{act}(a', \tau)! \end{cases} \\ \text{if } \tau = \sigma \text{ and } \sigma \text{ is remote,} & t'_\tau := l_\sigma \end{array}$$

To complete the general definition of TDES we impose a final technical condition, to exclude the physically unrealistic possibility that a *tick* transition might be preempted indefinitely by repeated execution of an activity loop within a fixed unit time interval. A TDES is said to have an *activity loop* if

$$(\exists q \in Q)(\exists s \in \Sigma^+_{act}) \ \delta(q, s) = q$$

We rule this out, and declare that all TDES must be *activity-loop-free (alf)*, namely

$$(\forall q \in Q)(\forall s \in \Sigma^+_{act}) \ \delta(q, s) \neq q$$

It should be stressed that the alf condition refers to the timed transition structure, not to the activity transition structure. The latter may quite safely contain loops provided the time bounds associated with the relevant events in Σ_{act} are appropriate.

With the definition of TDES transition structure now complete, the Σ^* behavioral semantics of $\mathbf{G}$ is defined in the usual way: the *closed behavior* $L(\mathbf{G})$ of $\mathbf{G}$ is the subset of all strings in Σ^* that can be generated by iteration of δ starting from q_o (i.e. the strings s such that $\delta(q_o, s)!$); while the *marked behavior* $L_m(\mathbf{G})$ is the subset of all strings in $L(\mathbf{G})$ for which the terminal state belongs to Q_m (i.e. the strings s such that $\delta(q_o, s) \in Q_m$). Note that a TDES never 'stops the clock': at any state either some transition $(\_, \sigma, \_)$ with $\sigma \in \Sigma_{act}$ is eligible, or at least the *tick* transition is defined. By activity-loop-freedom, no infinite ($\Sigma^\omega-$) string generated by the transition structure of a TDES can be *tick*-free; indeed in any infinite string *tick* must occur infinitely often.[1]

Exercise 9.2.1: Verify the foregoing remark in detail. That is, if Q is finite and the activity-loop-freedom condition holds for $\mathbf{G}$, then every string in $L(\mathbf{G})$ can be extended in $L(\mathbf{G})$ (i.e. by use of the TDES transition structure) to include an additional occurrence of *tick*, and no string can be extended indefinitely without an occurrence of *tick*. This shows that in every infinite string generated by TDES, *tick* must occur infinitely often.

9.3 Example 1

The following example illustrates how timing constraints can strongly influence complexity of the language generated by a TDES. Let

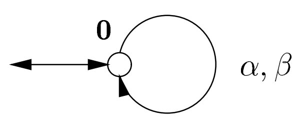
$$\mathbf{G_{act}} = (A, \Sigma_{act}, \delta_{act}, a_o, A_m)$$

with

$$\Sigma_{act} = \{\alpha, \beta\}, \qquad A = A_m = \{0\}, \qquad a_o = 0$$

$$\delta_{act}(0, \alpha) = \delta_{act}(0, \beta) = 0$$

and timed events $(\alpha, 1, 1)$, $(\beta, 2, 3)$, both prospective. The ATG for $\mathbf{G_{act}}$ is simply:



Thus α, β are always enabled. The state set for $\mathbf{G}$ is

$$Q = \{0\} \times T_\alpha \times T_\beta = \{0\} \times [0, 1] \times [0, 3]$$

[1] Here the fact that A, and so Q, are finite sets is crucial.

and has size $|Q| = 8$. We take $Q_m = \{(0, [1,3])\}$. The TTG for **G** is easily constructed and is displayed in Fig. 9.3.1; it has 11 transitions, over the event set $\{\alpha, \beta, tick\}$; the pairs $[t_\alpha, t_\beta]$ corresponding to the states $(0, \{t_\alpha, t_\beta\})$ of **G** are listed below. The event α is pending at states 0,2,5,7 and eligible at states 1,3,4,6, while β is pending at 0,1,2,4 and eligible at 3,5,6,7. Notice that *tick* is preempted by α or β if either of these events has deadline 0 (namely is imminent).

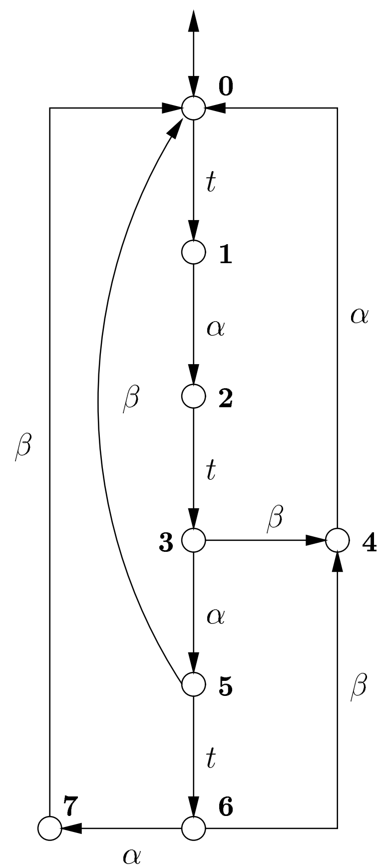


Fig. 9.3.1
Timed transition graph, Ex. 1

State (node of TTG):	0	1	2	3	4	5	6	7
Components $[t_\alpha, t_\beta]$:	[1,3]	[0,2]	[1,2]	[0,1]	[0,3]	[1,1]	[0,0]	[1,0]

To obtain the TATG of **G** we require a projection operation on TDES defined (in outline) as follows. Let **G** be an arbitrary TDES, over the alphabet Σ, with closed and marked behaviors $L(\mathbf{G})$, $L_m(\mathbf{G})$ respectively. Let $\Sigma_{pro} \subseteq \Sigma$ and write $\Sigma_{nul} := \Sigma - \Sigma_{pro}$. Let $P : \Sigma^* \to \Sigma_{pro}^*$ be the natural projection whose action on a string $s \in \Sigma^*$ is just to erase any symbols of Σ_{nul} that appear in s. Now let $\mathbf{G}_{pro}$ be any TDES over Σ_{pro} with closed and marked behaviors

$$L(\mathbf{G}_{pro}) := PL(\mathbf{G}), \qquad L_m(\mathbf{G}_{pro}) := PL_m(\mathbf{G})$$

If the state set of $\mathbf{G}$ is finite, it is convenient in practice to select $\mathbf{G}_{pro}$ such that its state set is of minimal size. In any case, for a suitable determination of $\mathbf{G}_{pro}$ we can define an operation **project** according to

$$\mathbf{G}_{pro} = \mathbf{project}(\mathbf{G}, \Sigma_{nul})$$

In examples Σ_{nul} will be written as a list. We can now specify the TATG of $\mathbf{G}$ as

$$\mathbf{G_{tact}} = \mathbf{project}(\mathbf{G}, [tick])$$

For the example of this section the result is displayed in Fig. 9.3.2. Notice that $\mathbf{G_{tact}}$ happens to have just as many states (8) as $\mathbf{G}$, illustrating the logical complexity that may be induced on the ordering of events by time bounds. This ordering, which could also be thought of as a set of phase relationships, is exhibited in the TATG (Fig. 9.3.2) but not in the ATG (above).

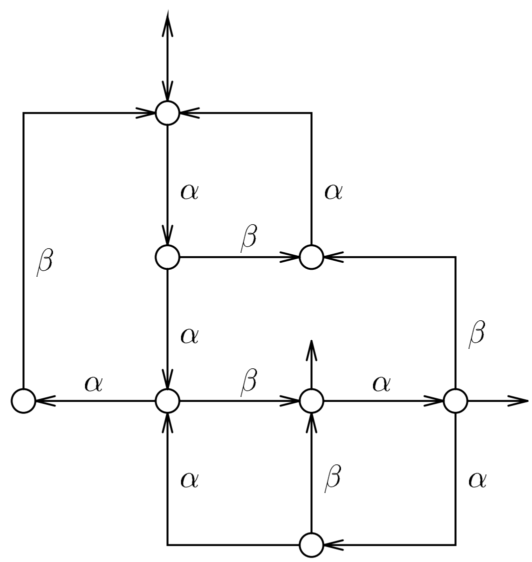


Fig. 9.3.2
Timed activity transition graph, Ex. 1

9.4 Example 2

Let $\Sigma = \{\alpha, \beta, \gamma\}$ with the timed events α, β as in Example 1. Adjoin to the structure of Example 1 the timed remote event $(\gamma, 2, \infty)$ with activity transition $[0, \gamma, 0]$; $\mathbf{G_{act}}$ is otherwise unchanged. The state size of $\mathbf{G}$ turns out to be $|Q| = 24$, with 41 transitions. It is found that $\mathbf{G_{tact}} := \mathbf{project}(\mathbf{G}, [tick])$ has 52 states and 108 transitions, thus being even more complex than $\mathbf{G}$ itself! While at first glance surprising, inasmuch as the occurrence or non-occurrence of γ does not appear to constrain that of α or β, this result can be thought of as a consequence of the nondeterminism created in the transition structure of $\mathbf{G}$ when *tick* is first projected out: an increase in complexity is the penalty exacted (by **project**) for replacing this nondeterministic description by a behaviorally equivalent deterministic one.

9.5 Time Bounds as Specifications

The imposition of time bounds on an event $\sigma \in \Sigma_{act}$ can be thought of as a specification over the alphabet $\{\sigma, tick\}$. If $\sigma \in \Sigma_{spe}$, with bounds $0 \le l_\sigma \le u_\sigma$, then the corresponding DES, say $\mathbf{SPEC}\sigma$, will have state set $\{0, ..., u_\sigma\}$, with transitions $(i, tick, i+1)$ for $0 \le i \le u_\sigma - 1$, together with $(i, \sigma, 0)$ for $l_\sigma \le i \le u_\sigma$. To state i corresponds the evaluation $t_\sigma = u_\sigma - i$. Similarly if $\sigma \in \Sigma_{rem}$, with bound l_σ, $0 \le l_\sigma < \infty$, then $\mathbf{SPEC}\sigma$ has state set $\{0, ..., l_\sigma\}$, with transitions $(i, tick, i+1)$ for $i = 0, ..., l_\sigma - 1$, $(l_\sigma, tick, l_\sigma)$, and $(l_\sigma, \sigma, 0)$. To state i corresponds the evaluation $t_\sigma = l_\sigma - i$. The specifications for the events α, β, γ of Examples 1 and 2 are displayed in Fig. 9.5.1.

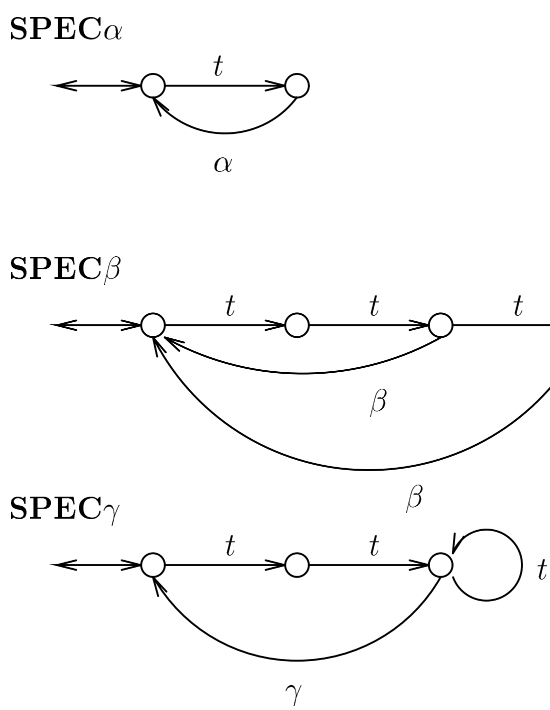


Fig. 9.5.1
Time bound specifications, Exs. 1 and 2

It should be observed that in $\mathbf{SPEC}\sigma$ all events other than σ are ignored, in particular events whose occurrence may reinitialize $\mathbf{SPEC}\sigma$ by transition to an activity where σ is disabled. Unfortunately there is no simple way to obtain $\mathbf{G}$ by straightforward combination of $\mathbf{G_{act}}$ with the $\mathbf{SPEC}\sigma$. In general, to obtain $\mathbf{G}$ (or its reachable subgraph) one must compute the reachable subset of Q, starting from q_o and systematically examining the timed transition rules (for δ) in conjunction with the transition structure (δ_{act}) of $\mathbf{G_{act}}$.

9.6 Composition of TDES

Complex TDES can be built up from simpler ones by a composition operator **comp**. Let $\mathbf{G_1}, \mathbf{G_2}$ be TDES, over alphabets Σ_1, Σ_2 respectively, where $\Sigma_i = \Sigma_{i,act} \cup \{tick\}$. In general $\Sigma_{1,act}, \Sigma_{2,act}$ need not be disjoint. To form the *composition* $\mathbf{G} = \mathbf{comp}(\mathbf{G_1}, \mathbf{G_2})$ we start by defining the alphabet of $\mathbf{G}$ as $\Sigma_1 \cup \Sigma_2$, and the activity transition structure of $\mathbf{G}$ as the synchronous product of the component activity structures:

$$\mathbf{G_{act}} = \mathbf{sync}(\mathbf{G_{1,act}}, \mathbf{G_{2,act}})$$

The time bounds (l_σ, u_σ) in $\mathbf{G}$ of an event

$$\sigma \in (\Sigma_{1,act} - \Sigma_{2,act}) \cup (\Sigma_{2,act} - \Sigma_{1,act})$$

remain unchanged from their definition in the corresponding component structure, while if $\sigma \in \Sigma_{1,act} \cap \Sigma_{2,act}$ then its time bounds in $\mathbf{G}$ are defined in obvious notation to be

$$(l_\sigma, u_\sigma) = (\max(l_{1,\sigma}, l_{2,\sigma}), \min(u_{1,\sigma}, u_{2,\sigma}))$$

provided $l_\sigma \leq u_\sigma$. If the latter condition is violated for any σ then the composition $\mathbf{G}$ is considered undefined. Thus the component TDES with the greater lower time bound (respectively smaller upper time bound) determines the timing behavior of the composition. This convention extends the principle that synchronous product represents an agreement between components that a transition with a shared label can be executed when and only when the conditions imposed on its execution by each component are satisfied.[2]

Provided the time bound conditions as stated above are satisfied, the composition $\mathbf{G}$ is now fully defined; clearly the alf condition will be true for the composition if it is true for each component.

Since synchronous product is associative, as a binary operation on the underlying languages, it follows that composition is associative in this sense as well.

It is important to stress that **comp** (G_1, G_2) is in general quite different from the result of forming the synchronous product of the *timed* transition structures of $\mathbf{G_1}$ and $\mathbf{G_2}$, for the

[2]While this convention respects physical behavior in many applications it need not be considered sacrosanct for all future modelling exercises.

latter would force synchronization of the *tick* transition as it occurs in the component TTGs. Such a rule of composition places a constraint on the interaction of component TDES that proves unrealistic for the modelling requirements in many applications; it may even lead to temporal deadlock ('stopping the clock') as in the example of Sect. 9.7.

Exercise 9.6.1: Let **G1**, **G2** be TDES with $\Sigma_{1,act} \cap \Sigma_{2,act} = \emptyset$. Show that, in this special case,

$$\mathbf{comp}(\mathbf{G1}, \mathbf{G2}) \approx \mathbf{sync}(\mathbf{G1}, \mathbf{G2})$$

where $\approx$ denotes that the closed and marked behaviors of the TDES coincide.

9.7 Example 3

Consider the TDES **G1,G2** with ATGs displayed in Fig. 9.7.1; $\Sigma_{1,act} = \{\alpha, \beta\}$, $\Sigma_{2,act} = \{\beta, \gamma\}$; time bounds are as in Example 2. Let **G = comp (G1,G2)**. The ATG of **G** is also shown in Fig. 9.7.1; as the time bounds for the shared event label β are the same in **G1** and **G2**, the time bounds for $\Sigma_{act} = \{\alpha, \beta, \gamma\}$ are as specified already. While the structure of **G** is fairly rich, and admits strings of arbitrary length, the synchronous product of the *timed* transition graphs of **G1** and **G2** turns out to have a closed behavior which terminates with deadlock (i.e., no subsequent transition is defined) after just 9 transitions.[3] Thus, it does not even represent a TDES.

[3]The language generated is the closure of the string pair $\{tick\ \alpha\ tick^2\ \beta\ tick^2\ (\gamma\ tick \mid tick\ \gamma)\}$.

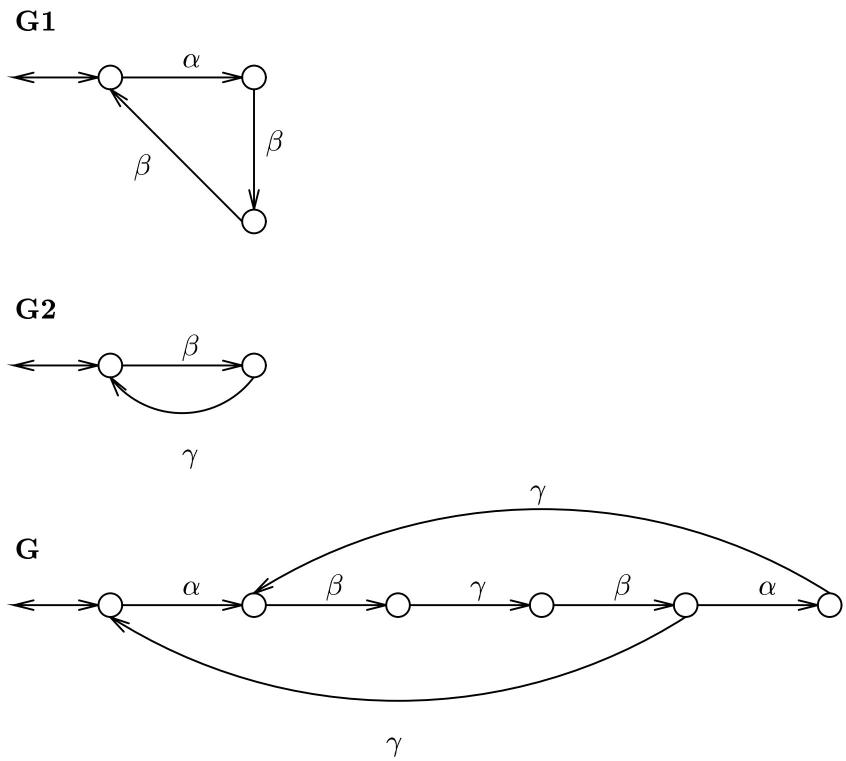


Fig. 9.7.1
Activity transition graphs, Ex. 3

Exercise 9.7.1: Verify this example using *TTCT*.

9.8 Controllability of TDES

To use TDES as models for supervisory control, it is necessary to specify the ways in which TDES transitions can be controlled by an external agent or supervisor. From a theoretical viewpoint it is natural and convenient to impose two criteria on our 'control technology': (i) control should at most restrict uncontrolled behavior, never enlarge it; and (ii) controlled behavior subject to a specification constraint should admit optimization in the sense of maximal permissiveness.

By analogy with our theory of untimed DES, we first seek the counterpart of 'controllable events', namely transitions that can be disabled. Intuitively, if an event can be 'disabled', then it can be prevented indefinitely from occurring. In view of (i) this suggests that only remote events may belong to this category, for if a prospective event were disabled then it might be prohibited from occurring even when imminent and when no competing event is eligible to preempt it. This situation would result in behavior that could never be realized in the absence of control. On this basis we bring in a new subset $\Sigma_{hib} \subseteq \Sigma_{rem}$ to label the *prohibitible* events. Our 'technology' will permit the supervisor to erase a prohibitible event from the current list of eligible transitions at a given state q of the supervised TDES. Of course, just as in the original model, the erased event may be reinstated if and when **G** revisits q on a subsequent occasion.

Next we consider a new category of events that arises naturally in the presence of timing: the *forcible* events, or elements of a new subset $\Sigma_{for} \subseteq \Sigma_{act}$. A forcible event is one that can preempt a *tick* of the clock. If at a given state of **G**, *tick* is defined and one or more elements of Σ_{for} are eligible, then our supervisory control technology permits the effective erasure of *tick* from the current list of defined events, namely the guaranteed preemptive occurrence of some one of the eligible events in Σ_{act}, whether a member of Σ_{for} or otherwise. Thus forcible events are 'events of last resort to beat the clock'. There is no particular relation postulated *a priori* between Σ_{for} and any of Σ_{hib}, Σ_{rem} or Σ_{spe}. In particular an event in Σ_{rem} might be both forcible and prohibitible.

It is convenient to define the *uncontrollable* event set

$$\begin{aligned} \Sigma_{unc} &:= \Sigma_{act} - \Sigma_{hib} \\ &= \Sigma_{spe} \cup (\Sigma_{rem} - \Sigma_{hib}) \end{aligned}$$

Eligible events in Σ_{unc} can never be erased by control action. Finally, we define the (complementary) *controllable* event set

$$\begin{aligned} \Sigma_{con} &:= \Sigma - \Sigma_{unc} \\ &= \Sigma_{hib} \cup \{tick\} \end{aligned}$$

Note that a forcible event may be controllable or uncontrollable; a forcible event that is uncontrollable cannot be directly prevented from occurring by disablement.[4] Also, while formally designated 'controllable' to simplify terminology, the status of *tick* lies intuitively between 'controllable' and 'uncontrollable': no technology could 'prohibit' *tick* in the sense of 'stopping the clock', although a forcible event, if eligible, may preempt it.

Exercise 9.8.1: ('Delayable' events) Consider an event α that is both prohibitible and forcible, with the requirement that α occur no earlier than 2 ticks (from enablement) and no later than 4 ticks. Provide the corresponding specification (as a language over the full alphabet). More generally, suppose the requirement is that α be delayed until some other event β occurs, but not for longer than 4 ticks, and that when α does occur, β is disabled. Assume, for instance, that β is uncontrollable and has time bounds $(0, \infty)$. Show that this specification can be modelled on a structure with 11 states. ◇

[4] An instance: air defense could force a plane to land within 10 minutes (say) but not prevent it from landing eventually; the landing is forcible but not controllable.

The simplest way to visualize the behavior of a TDES $\mathbf{G}$ under supervision is first to consider the (infinite) reachability tree of $\mathbf{G}$ before any control is operative. Each node n of the tree corresponds to a unique string s of $L(\mathbf{G})$ (of course, over the full alphabet Σ including *tick*). At n we may define the subset of eligible events, say $\mathrm{Elig}_{\mathbf{G}}(s) \subseteq \Sigma$. Thus

$$\mathrm{Elig}_{\mathbf{G}}(s) := \{\sigma \in \Sigma \mid s\sigma \in L(\mathbf{G})\}$$

Henceforth we shall use the term 'eligible' in this extended sense, to apply to *tick* as well as to events in Σ_{act}. By our assumptions on $\mathbf{G}$, $\mathrm{Elig}_{\mathbf{G}}(s) \neq \emptyset$ for all $s \in L(\mathbf{G})$. A supervisor will be considered a decision-maker that, at n, selects a nonempty subset of $\mathrm{Elig}_{\mathbf{G}}(s)$ in accordance with the rules stated above. It is now clear that, under these rules, our criterion (i) is satisfied, and it will later be shown that criterion (ii) is satisfied as well.

To formalize the rules we proceed as follows. Define a *supervisory control* to be any map $V : L(\mathbf{G}) \to Pwr(\Sigma)$ such that, for all $s \in L(\mathbf{G})$,

$$V(s) \supseteq \begin{cases} \Sigma_{unc} \cup (\{tick\} \cap \mathrm{Elig}_{\mathbf{G}}(s)) \\ \qquad \text{if } V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{for} = \emptyset \\ \Sigma_{unc} \text{ if } V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Notice that if V' and V'' are both supervisory controls, then so is $V := V' \vee V''$, defined by $V(s) := V'(s) \cup V''(s)$. This property will imply the satisfaction of criterion (ii).

Fix a supervisory control V. The remainder of the discussion proceeds by analogy with [RW87]. Write $V/\mathbf{G}$ to denote the pair $(\mathbf{G}, V)$ ('$\mathbf{G}$ under the supervision of V'). The *closed behavior* of $V/\mathbf{G}$ is the language $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ defined inductively according to:

(i) $\epsilon \in L(V/\mathbf{G})$.

(ii) If $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$, and $s\sigma \in L(\mathbf{G})$ then $s\sigma \in L(V/\mathbf{G})$.

(iii) No other strings belong to $L(V/\mathbf{G})$.

Thus $\{\epsilon\} \subseteq L(V/\mathbf{G}) \subseteq L(\mathbf{G})$, and $L(V/\mathbf{G})$ is nonempty and closed. The *marked behavior* of $V/\mathbf{G}$ is

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G})$$

and thus $\emptyset \subseteq L_m(V/\mathbf{G}) \subseteq L_m(\mathbf{G})$. As usual we say V is *nonblocking for* $\mathbf{G}$ provided

$$\bar{L}_m(V/\mathbf{G}) = L(V/\mathbf{G})$$

Exercise 9.8.2: Show that, for all $s \in L(\mathbf{G})$,

$$V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s) \neq \emptyset \qquad \diamond$$

We shall characterize those sublanguages of $L_m(\mathbf{G})$ that qualify as the marked behavior of some supervisory control V. First let $K \subseteq L(\mathbf{G})$ be arbitrary, and write

$$\text{Elig}_K(s) := \{\sigma \in \Sigma \mid s\sigma \in \bar{K}\}, \qquad s \in \Sigma^*$$

We define K to be *controllable (with respect to* $\mathbf{G}$ *)* if, for all $s \in \bar{K}$,

$$\text{Elig}_K(s) \supseteq \begin{cases} \text{Elig}_{\mathbf{G}}(s) \cap (\Sigma_{unc} \cup \{tick\}) & \text{if } \text{Elig}_K(s) \cap \Sigma_{for} = \emptyset \\ \text{Elig}_{\mathbf{G}}(s) \cap \Sigma_{unc} & \text{if } \text{Elig}_K(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Thus K controllable means that an event σ (in the full alphabet Σ including *tick*) may occur in K if σ is currently eligible in $\mathbf{G}$ and either (i) σ is uncontrollable, or (ii) $\sigma = tick$ and no forcible event is currently eligible in K. The effect of the definition is to allow the occurrence of *tick* (when it is eligible in $\mathbf{G}$) to be ruled out of K only when a forcible event is eligible in K and could thus (perhaps among other events in Σ_{act}) be relied on to preempt it. Notice, however, that a forcible event need not preempt the occurrence of competing non-*tick* events that are eligible simultaneously. In general our model will leave the choice of *tick*-preemptive transition nondeterministic.

In one form or another, the notion of forcing as preemption is inherent in control. Our notion of forcing is 'weakly preemptive', in that only the clock *tick* is assuredly preempted if forcing is invoked; however, the preemptive occurrence of a competing non-forcible but eligible event is not ruled out. A more conventional notion of forcing would require 'strong preemption', namely that a forcible event actually preempt any competing eligible event. If the control technology to be modelled actually admits 'forcing' in the strongly preemptive sense just indicated, then that feature would be modelled in our setup by suitably defining the activity transition structure.[5]

Notice finally that the controllability of K is a property only of $\bar{K}$, and that the languages $\emptyset$, $L(\mathbf{G})$, and Σ^* are all trivially controllable.

Our first main result is the analog of Theorem 3.4.1 (Chapt. 3). Since the *tick* transition needs special treatment, the proof will be given in full.

Theorem 9.8.1

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. There exists a nonblocking supervisory control V for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$ if and only if

(i) K is controllable with respect to $\mathbf{G}$, and

(ii) K is $L_m(\mathbf{G})$-closed.

[5]For instance, if a forcible event σ = 'stop' is to strictly preempt κ = 'collision', the model requires interposing at least one *tick* between σ and κ, and a structure in which σ causes transition to an activity where κ is disabled. This seems quite intuitive on physical grounds.

Proof

(If) For $s \in \bar{K}$ define

$$V(s) := \Sigma_{unc} \cup (\Sigma_{con} \cap \mathrm{Elig}_K(s))$$

while if $s \in L(\mathbf{G}) - \bar{K}$ assign the 'don't care' value

$$V(s) := \Sigma$$

First of all, V really is a supervisory control. Indeed, $V(s) \supseteq \Sigma_{unc}$ always. Next, if

$$V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{for} = \emptyset$$

then

$$\Sigma_{unc} \cap \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{for} = \emptyset$$

and

$$\Sigma_{con} \cap \mathrm{Elig}_K(s) \cap \Sigma_{for} = \emptyset$$

Therefore

$$(\Sigma_{uncon} \cup \Sigma_{con}) \cap \mathrm{Elig}_K(s) \cap \Sigma_{for} = \emptyset$$

i.e.

$$\mathrm{Elig}_K(s) \cap \Sigma_{for} = \emptyset$$

By controllability of K,

$$\{tick\} \cap \mathrm{Elig}_{\mathbf{G}}(s) \subseteq \mathrm{Elig}_K(s)$$

and so

$$\{tick\} \cap \mathrm{Elig}_{\mathbf{G}}(s) \subseteq V(s)$$

as required.

Next we show that $L(V/\mathbf{G}) = \bar{K}$ and begin with $L(V/\mathbf{G}) \subseteq \bar{K}$. We have $\epsilon \in L(V/\mathbf{G})$ by definition, and $\epsilon \in \bar{K}$ since $K \neq \emptyset$. Arguing by induction, let $s \in L(V/\mathbf{G})$, $s \in \bar{K}$, $s\sigma \in L(V/\mathbf{G})$. Then $\sigma \in V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s)$. If $\sigma \in \Sigma_{unc}$ then $\sigma \in \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{unc}$, so $\sigma \in \mathrm{Elig}_K(s)$ since K is controllable. If $\sigma \in \Sigma_{con} \cap \mathrm{Elig}_K(s)$ then again $\sigma \in \mathrm{Elig}_K(s)$. In either case $\sigma \in \mathrm{Elig}_K(s)$, so $s\sigma \in \bar{K}$. For $\bar{K} \subseteq L(V/\mathbf{G})$, we proceed similarly, letting $s \in \bar{K}$, $s \in L(V/\mathbf{G})$, $s\sigma \in \bar{K}$. If $\sigma \in \Sigma_{unc}$ then $\sigma \in V(s)$. Since $s\sigma \in \bar{K}$ we have $s\sigma \in L(\mathbf{G})$ and so $s\sigma \in L(V/\mathbf{G})$. If $\sigma \in \Sigma_{con}$ then $\sigma \in \Sigma_{con} \cap \mathrm{Elig}_K(s)$, i.e. $\sigma \in V(s)$, and again $s\sigma \in L(V/\mathbf{G})$. We have now proved $L(V/\mathbf{G}) = \bar{K}$.

Finally

$$\begin{aligned} L_m(V/\mathbf{G}) &= L(V/\mathbf{G}) \cap L_m(\mathbf{G}) \qquad \text{(by definition)} \\ &= \bar{K} \cap L_m(\mathbf{G}) \\ &= K \qquad \text{(since } K \text{ is } L_m(\mathbf{G})\text{-closed)} \end{aligned}$$

and $\bar{L}_m(V/\mathbf{G}) = \bar{K} = L(V/\mathbf{G})$, so V is nonblocking for $\mathbf{G}$.

(Only if) Let V be a nonblocking supervisory control for $\mathbf{G}$ with $L_m(V/\mathbf{G}) = K$. Since V is nonblocking, we have $L(V/\mathbf{G}) = \bar{K}$, so

$$K = L(V/\mathbf{G}) \cap L_m(\mathbf{G}) = \bar{K} \cap L_m(\mathbf{G})$$

i.e. K is $L_m(\mathbf{G})$-closed. To show that K is controllable let $s \in \bar{K}$, so $s \in L(V/\mathbf{G})$, and by definition of $L(V/\mathbf{G})$,

$$\mathrm{Elig}_K(s) = V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s)$$

Thus

$$\mathrm{Elig}_K(s) \supseteq \Sigma_{unc} \cap \mathrm{Elig}_{\mathbf{G}}(s)$$

always. Also if $\mathrm{Elig}_K(s) \cap \Sigma_{for} = \emptyset$ then

$$V(s) \cap \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{for} = \emptyset$$

Because V is a supervisory control,

$$V(s) \supseteq \{tick\} \cap \mathrm{Elig}_{\mathbf{G}}(s)$$

hence

$$\mathrm{Elig}_K(s) \supseteq \{tick\} \cap \mathrm{Elig}_{\mathbf{G}}(s)$$

as required. So K is controllable, as claimed. $\square$

For brevity we refer to a nonblocking supervisory control (for $\mathbf{G}$, understood) as an NSC. It is useful to introduce a slight generalization of NSC in which the supervisory action includes marking as well as control. For this, let $M \subseteq L_m(\mathbf{G})$. Define a *marking nonblocking supervisory control for the pair* $(M, \mathbf{G})$, or *MNSC*, as a map $V : L(\mathbf{G}) \to Pwr(\Sigma)$ exactly as before; but now for the marked behavior of $V/\mathbf{G}$ we define

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M \ .$$

One may think of the marking action of the MNSC V as carried out by a recognizer for M that monitors the closed behavior of $V/\mathbf{G}$, sounding a beep exactly when a string in M has been generated. As a sublanguage of $L_m(\mathbf{G})$, these strings could be thought of as representing a subset of the 'tasks' that $\mathbf{G}$ (or its underlying physical referent) is supposed to accomplish. For instance in a manufacturing problem, one might define a 'batch' to consist of 10 fully processed workpieces. M might then be taken as the set of strings that represent the successful processing of N integral batches, $N \geq 0$, with all machines returned to the idle state and all buffers empty.

The counterpart result to Theorem 9.8.1 actually represents a simplification, as the condition of $L_m(\mathbf{G})$-closedness can now be dropped.

Theorem 9.8.2

Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. There exists an MNSC V for $(K, \mathbf{G})$ such that

$$L_m(V/\mathbf{G}) = K$$

if and only if K is controllable with respect to $\mathbf{G}$.

Proof

(If) With V defined as in the proof of Theorem 9.8.1, it may be shown as before that $L(V/\mathbf{G}) = \bar{K}$. Then

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap K = \bar{K} \cap K = K$$

so that $\bar{L}_m(V/\mathbf{G}) = \bar{K} = L(V/\mathbf{G})$, namely V is nonblocking for $\mathbf{G}$.

(Only if) We have $\bar{K} = \bar{L}_m(V/\mathbf{G}) = L(V/\mathbf{G})$. Then the proof that K is controllable is unchanged from that of Theorem 9.8.1. $\square$

9.9 Supremal Controllable Sublanguages and Optimal Supervision

Let $\mathbf{G} = (\_, \Sigma, \_, \_, \_)$ be a controlled TDES with Σ partitioned as in the previous section. Let $E \subseteq \Sigma^*$. As in Chapter 3, we introduce the set of all sublanguages of E that are controllable with respect to $\mathbf{G}$:

$$\mathcal{C}(E) = \{K \subseteq E \mid K \text{ is controllable with respect to } \mathbf{G}\}$$

Proposition 9.9.1

$\mathcal{C}(E)$ is nonempty and is closed under arbitrary unions.

Proof

Since the empty language is trivially controllable, $\mathcal{C}(E) \neq \emptyset$. Suppose $K_1, K_2 \in \mathcal{C}(E)$. Let $K = K_1 \cup K_2$; then $\bar{K} = \bar{K}_1 \cup \bar{K}_2$. For any $s \in \Sigma^*$, clearly

$$\mathrm{Elig}_K(s) = \mathrm{Elig}_{K_1}(s) \cup \mathrm{Elig}_{K_2}(s)$$

Let $s \in \bar{K}$. Since at least one of the two subsets on the right satisfies the inclusion condition appearing in the definition of controllability, so does $\mathrm{Elig}_K(s)$, and therefore K is controllable. Extension of the argument to an arbitrary union is obvious. □

We may now assert the existence of a unique supremal element sup $\mathcal{C}(E)$ in E.

Let $E, L \subseteq \Sigma^*$. We say that E is L-*marked* if $E \supseteq \bar{E} \cap L$, namely any prefix of E that belongs to L must also belong to E.

Proposition 9.9.2

Let $E \subseteq \Sigma^*$ be $L_m(\mathbf{G})$-marked. Then sup $\mathcal{C}(E \cap L_m(\mathbf{G}))$ is $L_m(\mathbf{G})$-closed.

Proof

We have $E \supseteq \bar{E} \cap L_m(\mathbf{G})$, from which there follows in turn

$$\begin{aligned}
\bar{E} \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G}) \\
\bar{E} \cap \bar{L}_m(\mathbf{G}) \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G}) \\
\overline{E \cap L_m(\mathbf{G})} \cap L_m(\mathbf{G}) &\subseteq E \cap L_m(\mathbf{G})
\end{aligned}$$

so that $F := E \cap L_m(\mathbf{G})$ is $L_m(\mathbf{G})$-closed. Let $K = \sup \mathcal{C}(F)$. If K is not $L_m(\mathbf{G})$-closed, i.e. $K \subsetneqq \bar{K} \cap L_m(\mathbf{G})$, there is a string $s \in \bar{K} \cap L_m(\mathbf{G})$ with $s \notin K$. Let $J = K \cup \{s\}$. Since $\bar{J} = \bar{K}$ we have that J is controllable. Also $K \subseteq F$ implies that

$$\bar{K} \cap L_m(\mathbf{G}) \subseteq \bar{F} \cap L_m(\mathbf{G}) = F$$

so that $s \in F$ and thus $J \subseteq F$. Therefore $J \in \mathcal{C}(F)$ and $J \supsetneqq K$, contradicting the fact that K is supremal. □

Now we can present the main result of this section.

Theorem 9.9.3

Let $E \subseteq \Sigma^*$ be $L_m(\mathbf{G})$-marked, and let $K = \sup \mathcal{C}(E \cap L_m(\mathbf{G}))$. If $K \neq \emptyset$, there exists a nonblocking supervisory control (NSC) V for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$.

Proof

K is controllable and, by Proposition 9.9.2, $L_m(\mathbf{G})$-closed. The result follows by Theorem 9.8.1. □

As in Chapter 3, the result may be paraphrased by saying that K is (if nonempty) the maximally permissive (or minimally restrictive) solution of the problem of supervising $\mathbf{G}$ in such a way that its behavior belongs to E and control is nonblocking. In this sense the supervisory control provided by Theorem 9.9.3 is (qualitatively) optimal.

As might be expected, if we place part of the burden of 'marking action' on the supervisory control itself we may relax the prior requirement on E. By an application of Theorem 9.8.2 the reader may easily obtain the following analog of Theorem 3.4.2.

Theorem 9.9.4

Let $E \subseteq \Sigma^*$ and let $K = \sup \mathcal{C}(E \cap L_m(\mathbf{G}))$. If $K \neq \emptyset$ there exists a marking nonblocking supervisory control (MNSC) for $\mathbf{G}$ such that $L_m(V/\mathbf{G}) = K$. □

Finally, the implementation of supervisory controls by automata is formally no different from procedures already described in Sect. 3.6.

9.10 Example 4: Endangered Pedestrian

Consider two TDES

$$\begin{aligned} \mathbf{BUS} &= (\{a, g\}, \{pass\}, \{[a, pass, g]\}, a, \{g\}), \qquad (pass, 2, 2) \\ \mathbf{PED} &= (\{r, c\}, \{jump\}, \{[r, jump, c]\}, r, \{c\}), \qquad (jump, 1, \infty) \end{aligned}$$

(where in place of δ we merely list the one activity transition). These model respectively a bus that makes a single transition *pass* between the activities 'approaching' and 'gone by', and a pedestrian who may make a single transition *jump* from 'road' to 'curb'. These entities are combined in the TDES

$$\mathbf{BP} = \mathbf{comp}(\mathbf{BUS}, \mathbf{PED})$$

The ATG and TTG of $\mathbf{BP}$ are displayed in Fig. 9.10.1.

We bring in 'control technology', with the assumption

$$\Sigma_{hib} = \Sigma_{for} = \{jump\}$$

However, nothing can stop the bus from passing in the interval between 2 and 3 *ticks* of the clock.

Suppose it is required that the pedestrian be saved. As a first scenario, we specify a TDES that imposes no *a priori* timing constraints on events, but merely requires the pedestrian to jump before the bus passes:

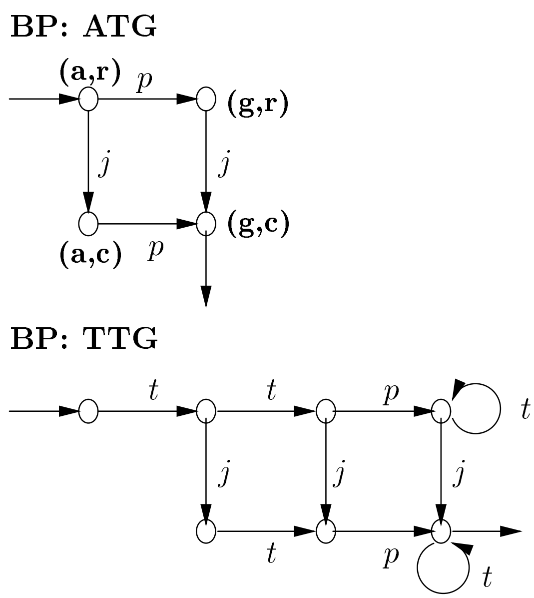


Fig. 9.10.1
Activity and timed transition graphs for BP

$$\mathbf{SAVE} = (\{s0, s1, s2\}, \{jump, pass\}, \{[s0, jump, s1], [s1, pass, s2]\}, s0, \{s2\})$$

with set of timed events $\Sigma_{tim} = \{(jump, 0, \infty), (pass, 0, \infty)\}$. The ATG and TTG are displayed in Fig. 9.10.2; the TTG is obtained from the ATG by selflooping *tick*.

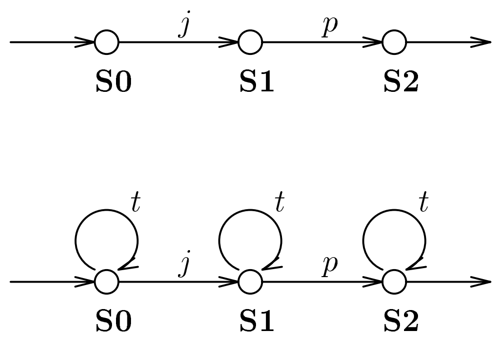


Fig. 9.10.2
Activity and timed transition graphs for **SAVE**

Here and later we use the operation meet, defined on TDES according to

$$\mathbf{G3} = \mathbf{meet}(\mathbf{G1}, \mathbf{G2})$$

where $L(\mathbf{G3}) := L(\mathbf{G1}) \cap L(\mathbf{G2})$, $L_m(\mathbf{G3}) := L_m(\mathbf{G1}) \cap L_m(\mathbf{G2})$. As usual with such operations it is understood that **meet** is uniquely defined at implementation.

Now we can bring in the relevant ('physically possible') strings of $L(\mathbf{SAVE})$ as those shared with **BP**, namely the behavior(s) of

$$\mathbf{BPSAVE} = \mathbf{meet}(\mathbf{BP}, \mathbf{SAVE})$$

as displayed in the TTG of Fig. 9.10.3. The closed behavior $L(\mathbf{BPSAVE})$ is not controllable, since

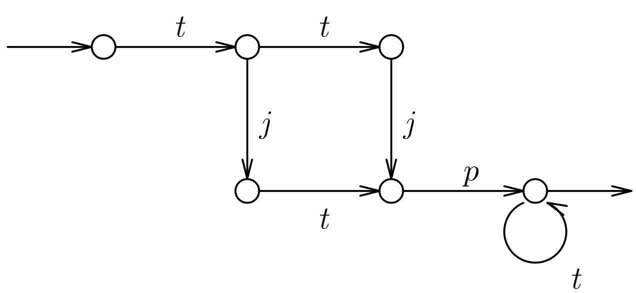


Fig. 9.10.3
Timed transition graph: **BPSAVE**

$$\mathrm{Elig}_{\mathbf{BP}}(tick^2) \cap \Sigma_{for} = \{jump\} \neq \emptyset, \qquad \mathrm{Elig}_{\mathbf{BP}}(tick^2) \cap \Sigma_{unc} = \{pass\}$$

but

$$\mathrm{Elig}_{\mathbf{BPSAVE}}(tick^2) = \{jump\}$$

Evidently, after $tick^2$ nothing can prevent the bus from passing before the pedestrian jumps. But all is not lost: $L_m(\mathbf{BPSAVE})$ has the supremal controllable sublanguage $L_m(\mathbf{BPSAFE})$ as in Fig. 9.10.4. Note that, while $tick \in \mathrm{Elig}_{\mathbf{BP}}(tick)$, nonetheless

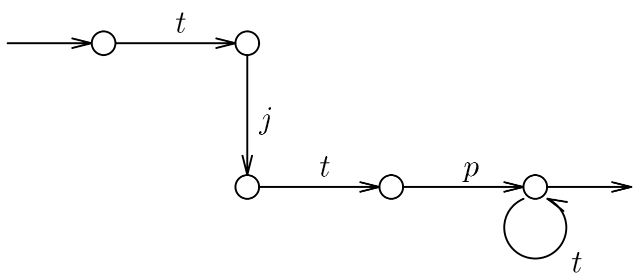


Fig. 9.10.4
Timed transition graph: **PSAFE**

$$\mathrm{Elig}_{\mathbf{BP}}(tick) \cap \Sigma_{for} = \{jump\} \neq \emptyset, \qquad \mathrm{Elig}_{\mathbf{BP}}(tick) \cap \Sigma_{unc} = \emptyset$$

and thus the second *tick* can be reliably preempted by the forcible event *jump* (i.e., the pedestrian can be 'forced' to jump between the first *tick* and the second).

In a less optimistic scenario the pedestrian is again supposed to be saved, but at least 2 ticks must elapse from initialization before a jump (perhaps the pedestrian is handicapped); since $jump \in \Sigma_{hib}$, the string *tick,jump* can surely be prohibited as a prefix of controlled behavior. The resulting specification **PRISK** is shown in Fig. 9.10.5. Just as for **BPSAVE**, it is uncontrollable; but now the supremal controllable sublanguage of $L_m(\mathbf{PRISK})$ is empty, and the control problem is unsolvable.

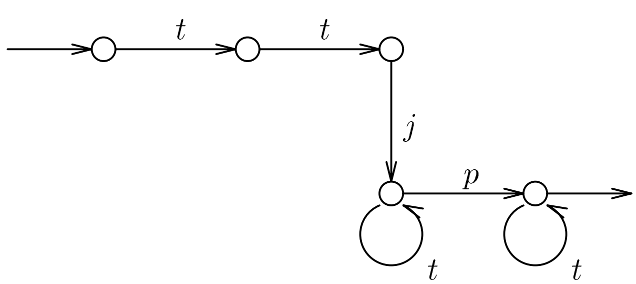


Fig. 9.10.5
Timed transition graph: **PRISK**

As a more complicated variation, suppose the bus can be stopped (by a traffic officer) and held up for 1 *tick*. Introduce **NEWBUS**, like **BUS** but with new timed events $(stop, 0, \infty)$ with $stop \in \Sigma_{for} \cap \Sigma_{hib}$, and $(wait, 1, 1)$, having ATG in Fig. 9.10.6. With the string $delay1 := stop, tick, wait$ the TTG of **NEWBUS** can be displayed as in Fig. 9.10.7. In effect, *delay1* can be used to preempt *tick* at any state of the TTG where both are defined, although it cannot be relied on to preempt *pass* if *pass* is imminent. By use of *delay1* safety of the handicapped pedestrian can be guaranteed, for instance by forcing *stop* initially but disabling *stop* thereafter.

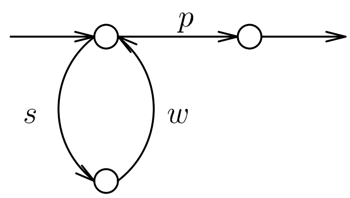


Fig. 9.10.6
Activity transition graph: **NEWBUS**

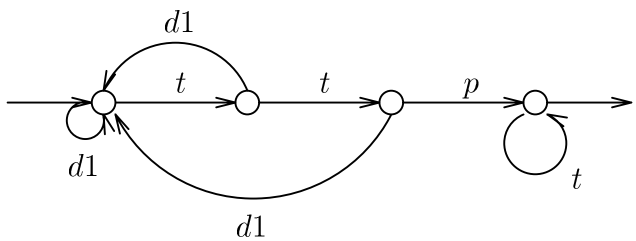
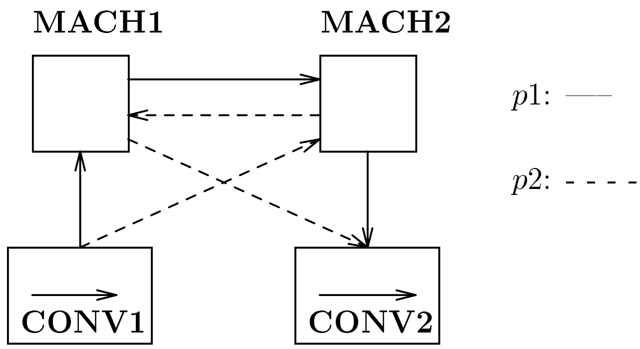


Fig. 9.10.7
Timed transition graph: **NEWBUS**

9.11 Example 5: Manufacturing Cell

The manufacturing cell of Fig. 9.11.1 consists of machines **MACH1,MACH2**, with an input conveyor **CONV1** as an infinite source of workpieces and output conveyor **CONV2** as an infinite sink. Each machine may process two types of parts, $p1$ and $p2$; and each machine is liable to break down, but then may be repaired. For simplicity, the transfer of parts between machines will be absorbed as a step in machine operation. The machine ATGs (identical up to event labelling) are displayed in Fig. 9.11.2 and the timed events listed below.



MACH1,MACH2: numerically controlled machines
CONV1: incoming conveyor (infinite source)
CONV2: outgoing conveyor (infinite sink)

Fig. 9.11.1
The manufacturing cell

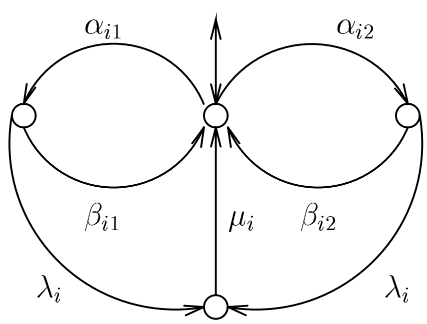


Fig. 9.11.2
Numerically controlled machines

MACH1 :	$(\alpha_{11}, 1, \infty)$	$(\beta_{11}, 3, 3)$	$(\alpha_{12}, 1, \infty)$	$(\beta_{12}, 2, 2)$
	$(\lambda_1, 0, 3)$	$(\mu_1, 1, \infty)$		
MACH2 :	$(\alpha_{21}, 1, \infty)$	$(\beta_{21}, 1, 1)$	$(\alpha_{22}, 1, \infty)$	$(\beta_{22}, 4, 4)$
	$(\lambda_2, 0, 4)$	$(\mu_2, 1, \infty)$		

Here α_{ij} is the event '**MACHi** starts work on a pj-part', while β_{ij} is '**MACHi** finishes working on a pj-part'; λ_i, μ_i represent respectively the breakdown[6] and repair of **MACHi**.

[6]Since breakdown may occur only when a machine is working, the upper time bound u_λ assigned to a breakdown event need not exceed the (finite) upper time bound u_β for completion of the corresponding work cycle. The u_λ could be replaced by anything larger, including ∞, without affecting behavior.

We take

$$\Sigma_{for} = \{\alpha_{ij} \mid i,j = 1,2\}, \qquad \Sigma_{unc} = \{\lambda_i, \beta_{ij} \mid i,j = 1,2\}$$

$$\Sigma_{hib} = \Sigma_{for} \cup \{\mu_1, \mu_2\}$$

The TTGs of **MACH1** and **MACH2** are shown in Figs. 9.11.3, 9.11.4.

We shall impose (i) *logic-based* specifications, (ii) a *temporal* specification, and (iii) a *quantitative optimality* specification as follows:

(i)
- a given part can be processed by just one machine at a time
- a $p1$-part must be processed first by **MACH1** and then by **MACH2**
- a $p2$-part must be processed first by **MACH2** and then by **MACH1**
- one $p1$-part and one $p2$-part must be processed in each production cycle
- if both machines are down, **MACH2** is always repaired before **MACH1**

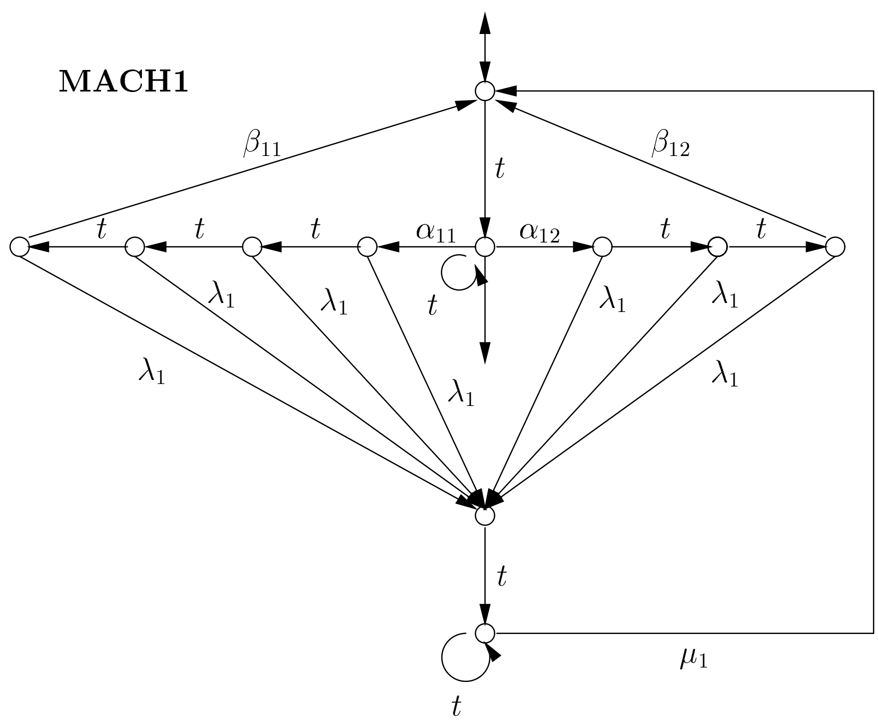


Fig. 9.11.3
The TTG of **MACH1**

MACH2

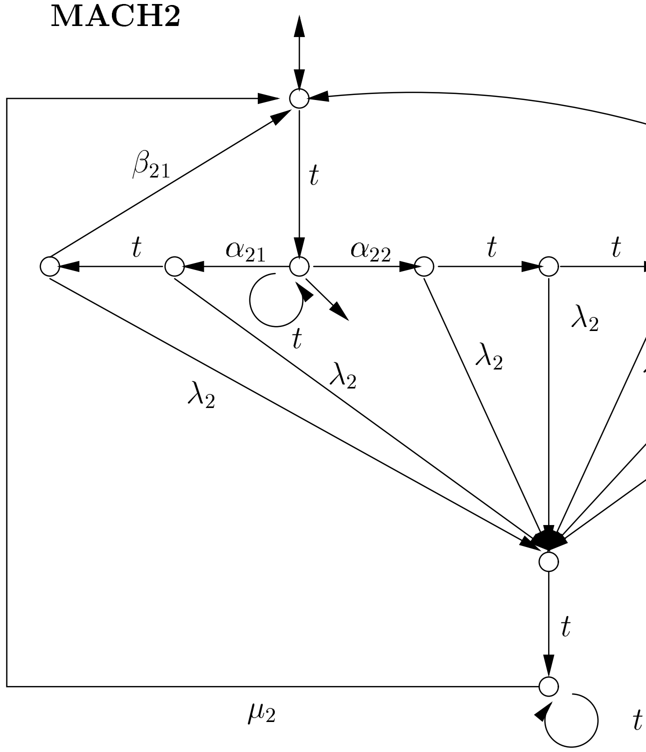


Fig. 9.11.4
The TTG of **MACH2**

(ii) • in the absence of breakdown/repair events a production cycle must be completed in at most 10 time units

(iii) • subject to (ii), production cycle time is to be minimized

The first three specifications (i) are formalized as TDES **SPEC1-SPEC4**, displayed in Fig. 9.11.5, while the fourth specification (i) is formalized as **SPEC5**, Fig. 9.11.6, and the fifth (breakdown/repair) as **SPEC6**, Fig. 9.11.7. It can be verified that, in fact, **SPEC1** and **SPEC2** are automatically enforced by **SPEC3** and **SPEC4** together. We therefore define the complete logic-based specification

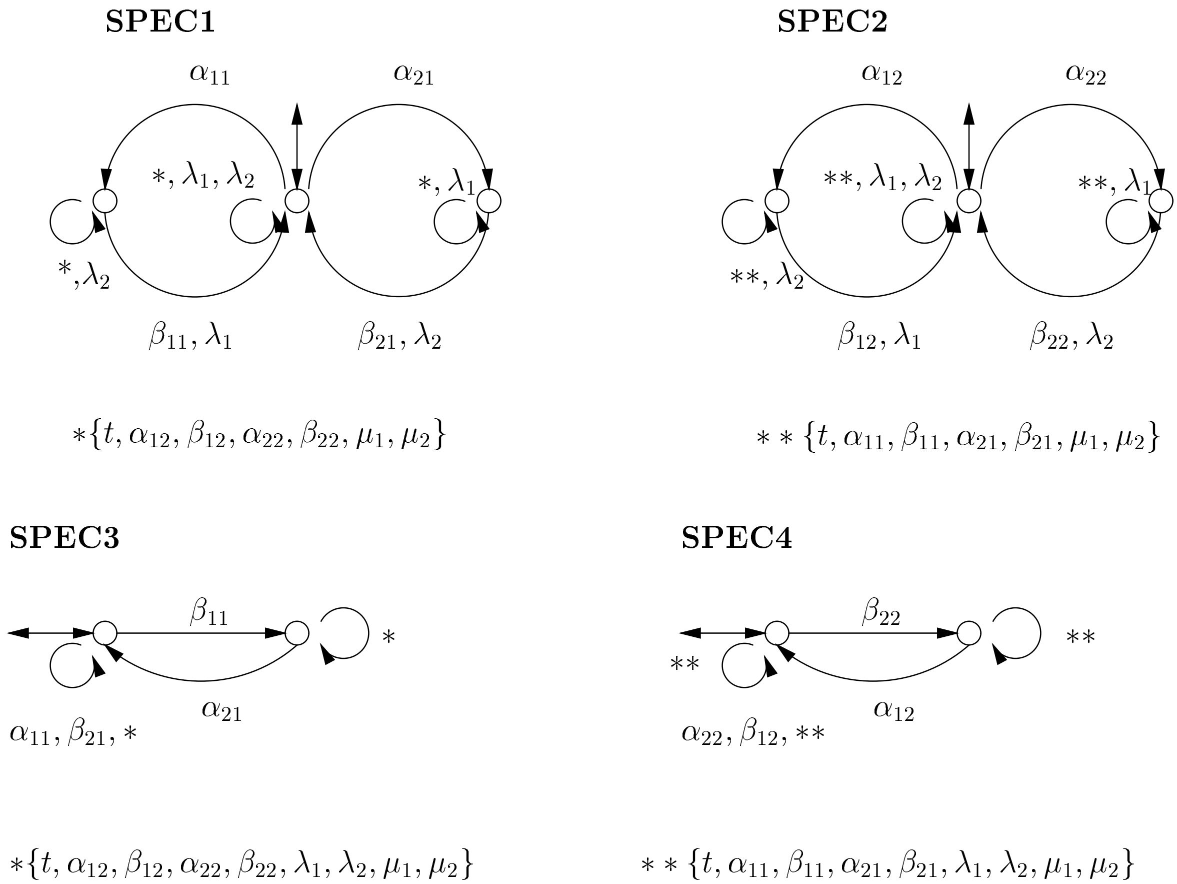
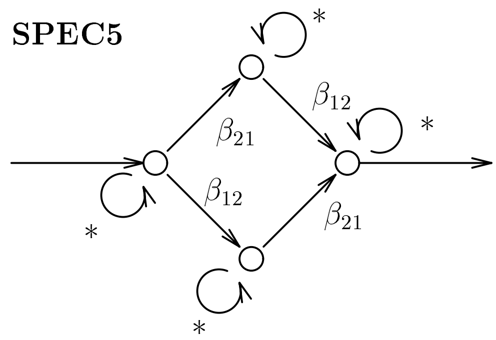
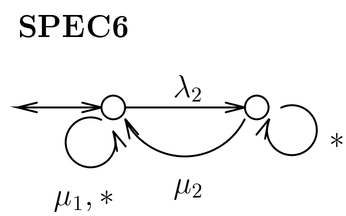


Fig. 9.11.5
SPEC1 – SPEC4



$*\{t, \alpha_{11}, \beta_{11}, \alpha_{12}, \alpha_{21}, \alpha_{22}, \beta_{22}, \lambda_1, \lambda_2, \mu_1, \mu_2\}$

Fig. 9.11.6
SPEC5



$*\{t, \alpha_{11}, \beta_{11}, \alpha_{12}, \beta_{12}, \alpha_{21}, \beta_{21}, \alpha_{22}, \beta_{22}, \lambda_1\}$

Fig. 9.11.7
SPEC6

$$\mathbf{SPECLOG} = \mathbf{meet}(\mathbf{SPEC3}, \mathbf{SPEC4}, \mathbf{SPEC5}, \mathbf{SPEC6})$$

a TDES with 32 states and 224 transitions. Define the cell's open-loop behavior as the composition **MACH** of **MACH1** and **MACH2**:

$$\mathbf{MACH} = \mathbf{comp}(\mathbf{MACH1}, \mathbf{MACH2})$$

(121 states, 345 transitions).

Here and below we write **G3 = supcon(G1,G2)** to denote the operation that returns a TDES **G3** whose marked behavior $L_m(\mathbf{G3})$ is the supremal controllable sublanguage $\sup \mathcal{C}(L_m(\mathbf{G1}), L_m(\mathbf{G2}))$; while its closed behavior $L(\mathbf{G3}) = \bar{L}_m(\mathbf{G3})$.

The maximally permissive proper supervisor for **MACH** that enforces **SPECLOG** can now be computed as

$$\mathbf{SUPLOG} = \mathbf{supcon}(\mathbf{MACH}, \mathbf{SPECLOG})$$

(264 states, 584 transitions).

To address the temporal specification (ii) we first recompute the results for (i), under the stated assumption that breakdowns are absent. For this we define new machines **NMACH1**, **NMACH2** with simplified ATGs, Fig. 9.11.8. The new logic-based specification is

NMACHi, $i = 1, 2$

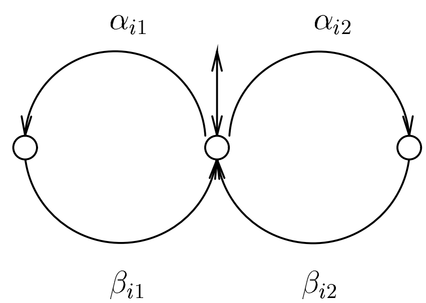


Fig. 9.11.8
The new machine activity transition graphs

$$\mathbf{NSPECLOG} = \mathbf{meet}(\mathbf{SPEC3}, \mathbf{SPEC4}, \mathbf{SPEC5})$$

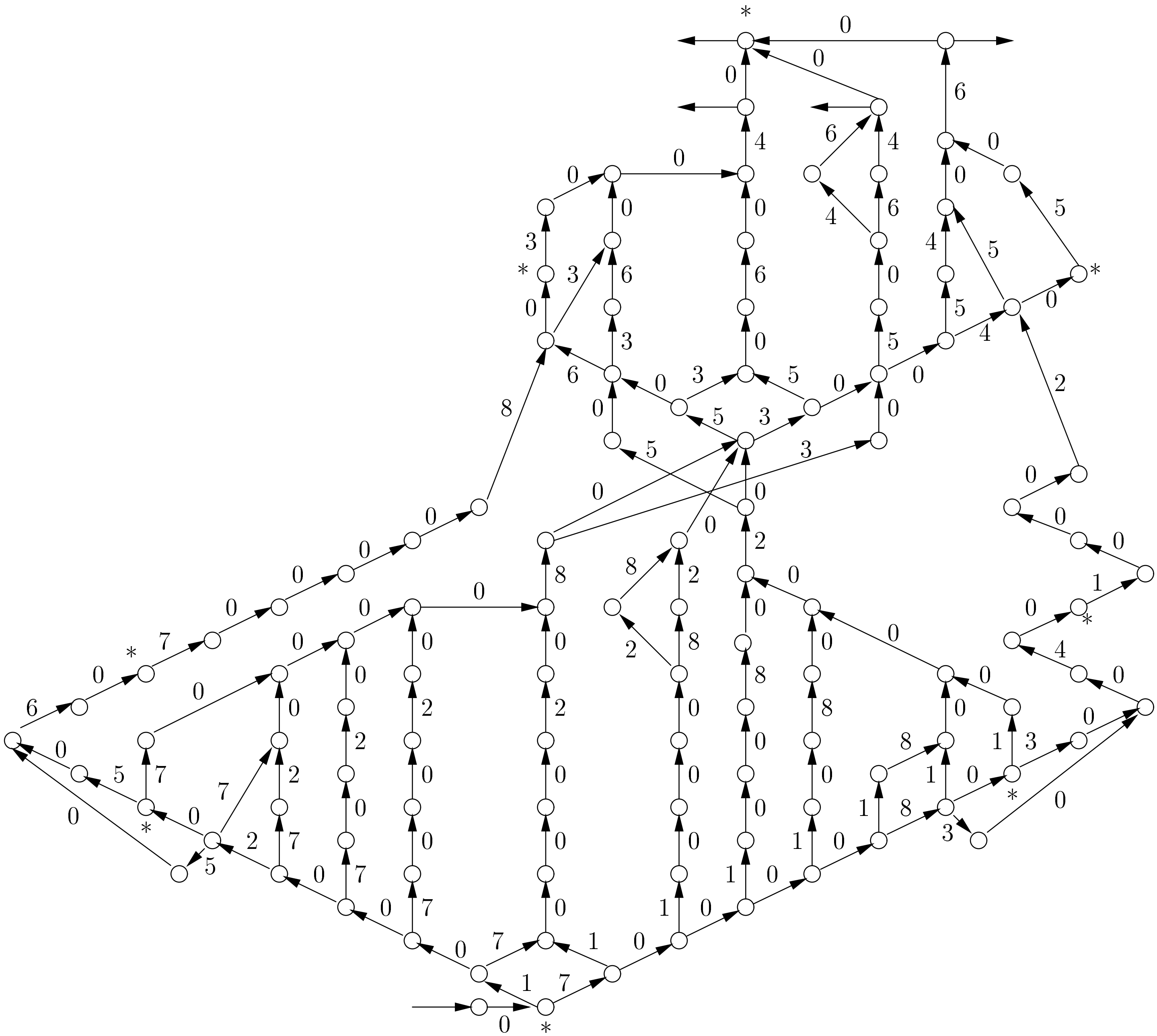
(16 states, 72 transitions). The open-loop behavior of the simplified cell is

$$\mathbf{NMACH} = \mathbf{comp}(\mathbf{NMACH1}, \mathbf{NMACH2})$$

(81 states, 121 transitions). These definitions yield the new supervisor

$$\mathbf{NSUPLOG} = \mathbf{supcon}(\mathbf{NMACH}, \mathbf{NSPECLOG})$$

(108 states, 144 transitions), displayed in Fig. 9.11.9. Now we consider the temporal specification itself. Bring in the TDES **TIMER_10** displayed in Fig. 9.11.10. **TIMER_10** is simply an 11-*tick* sequence all of whose states are marked. **TIMER_10** forces any TDES with which it is synchronized by **meet** to halt after at most 10 *ticks*, i.e. after 11 *ticks* to execute no further event whatever except the *tick* event. Thus it extracts the marked strings (if any) which satisfy this constraint, namely the 'tasks' of TDES which can be accomplished in at most 10 *ticks*. Of course, the designer must guarantee that the 10-*tick* deadline is actually met, if necessary by suitable forcing action. To determine whether such a guarantee is feasible, it suffices to check that the corresponding supremal controllable sublanguage is nonempty. The result is



$1:\alpha_{11}, 2:\beta_{11}, 3:\alpha_{12}, 4:\beta_{12}, 5:\alpha_{21}, 6:\beta_{21}, 7:\alpha_{22}, 8:\beta_{22}, 0:$ *tick*, $^{*}:$ selfloop in *tick*.

Fig. 9.11.9

NSUPLOG

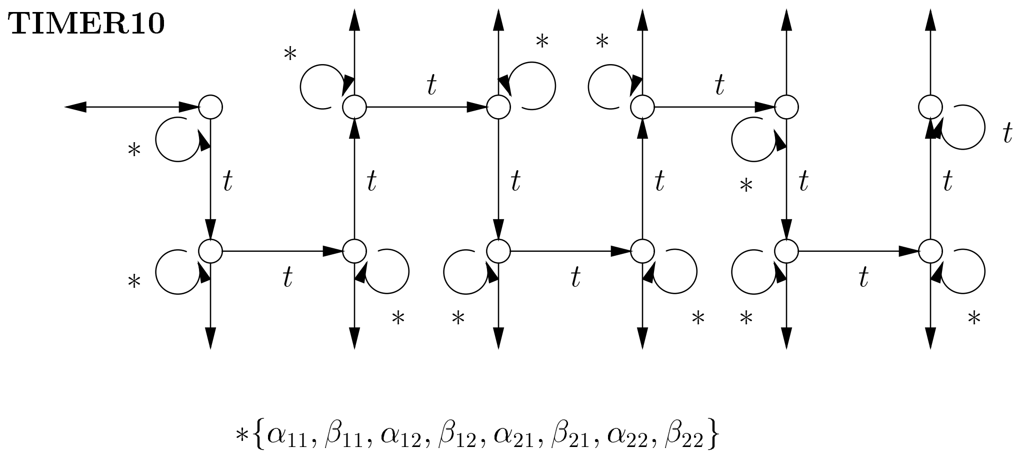


Fig. 9.11.10
TIMER10

$$\mathbf{TNSUPLOG} = \mathbf{supcon}(\mathbf{NSUPLOG}, \mathbf{TIMER\_10})$$

(209 states, 263 transitions), so the check succeeds. We conclude that, in the absence of breakdowns, a production cycle can indeed be forced to complete in 10 *ticks* or less.

Finally, to address specification (iii), we proceed as in (ii) with successive timer sequences of *tick*-length 9,8,... until **supcon** returns an empty result. For this example the minimum enforceable production time turns out to be 7 *ticks*, with behavior

$$\mathbf{OTNSUPLOG} = \mathbf{supcon}(\mathbf{SUPLOG}, \mathbf{TIMER\_7})$$

(19 states, 21 transitions) shown in Fig. 9.11.11. Initially both **NMACH1** and **NMACH2** are forced to start work on a $p1$-part and $p2$-part respectively (events α_{11}, α_{22}). Forcing occurs as soon as these events become eligible, thus preempting a *tick* which would take the system along a suboptimal (slower) path (see Fig. 9.11.9). **NMACH1** (**NMACH2**) finishes work on its $p1$-part ($p2$-part) within 3 (resp. 4) time units. As soon as **NMACH2** has finished with its $p2$-part (event β_{22}), **NMACH1** is forced to start working on it (α_{12}), again preempting a *tick* that would take the system along a suboptimal path. Finally, **NMACH2** is forced to work on the $p1$-part, enabling the system to finish its production cycle in minimum time.

OTNSUPLOG

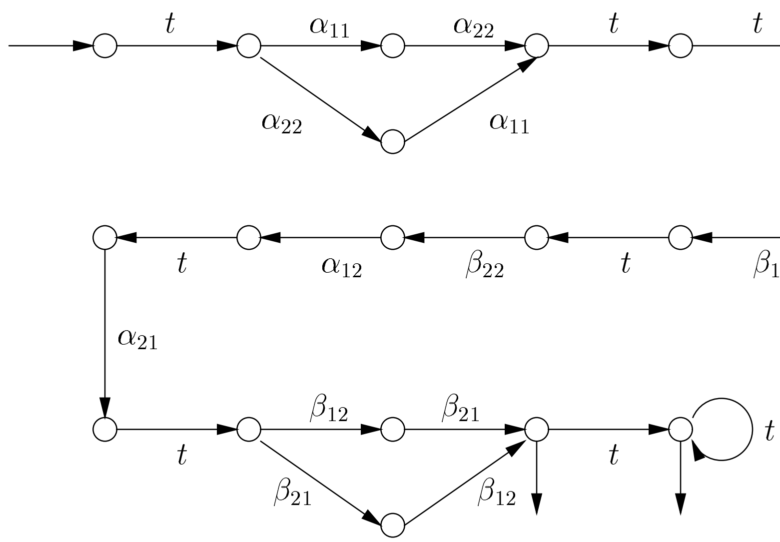


Fig. 9.11.11
OTNSUPLOG

9.12 Modular Supervision of TDES

Let

$$\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m), \qquad \mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$$

be TDES, with $\Sigma = \Sigma_{act} \cup \{tick\}$. We assume that $\mathbf{G}$ is equipped with control structure, as in Sect. 9.8, and consider when $\mathbf{S}$ can be used a supervisor for $\mathbf{G}$. As in Chapter 4 write $\mathbf{S} \wedge \mathbf{G}$ for the conjunction of $\mathbf{S}$ and $\mathbf{G}$ (implemented by TCT **meet**), so

$$L_m(\mathbf{S} \wedge \mathbf{G}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}), \qquad L(\mathbf{S} \wedge \mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G})$$

As in Sect. 3.6 we say that $\mathbf{S}$ is a *proper supervisor* for $\mathbf{G}$ if

(i) $\mathbf{S}$ is trim

(ii) $\mathbf{S}$ is controllable with respect to $\mathbf{G}$ (i.e. $L_m(\mathbf{S} \wedge \mathbf{G})$ is controllable)

(iii) $\mathbf{S} \wedge \mathbf{G}$ is nonblocking

Since by (iii), $\overline{L_m(\mathbf{S} \wedge \mathbf{G})} = L(\mathbf{S} \wedge \mathbf{G})$, (ii) means that

$$\mathrm{Elig}_{L(\mathbf{S}\wedge\mathbf{G})}(s) \supseteq \begin{cases} \mathrm{Elig}_{\mathbf{G}}(s) \cap (\Sigma_{unc} \cup \{tick\}) \quad \text{if } \mathrm{Elig}_{L(\mathbf{S}\wedge\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ \mathrm{Elig}_{\mathbf{G}}(s) \cap \Sigma_{unc} \quad \text{if } \mathrm{Elig}_{L(\mathbf{S}\wedge\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

We remark that if $L_m(\mathbf{S} \wedge \mathbf{G}) \neq \emptyset$ then

$$(\forall s \in L(\mathbf{S} \wedge \mathbf{G}))\ \mathrm{Elig}_{L(\mathbf{S}\wedge\mathbf{G})}(s) \neq \emptyset$$

Exercise 9.12.1: Justify this statement. $\diamond$

Let $K \subseteq L(\mathbf{G})$. The following definition extracts the feature of controllability that expresses the preemption of *tick* by a forcible event. We say that K is *coercive* with respect to $\mathbf{G}$ if

$$(\forall s \in \bar{K}) tick \in \mathrm{Elig}_{\mathbf{G}}(s) - \mathrm{Elig}_K(s) \Rightarrow \mathrm{Elig}_K(s) \cap \Sigma_{for} \neq \emptyset$$

i.e.

$$(\forall s \in \bar{K})\ \mathrm{Elig}_K(s) \cap \Sigma_{for} = \emptyset \quad \& \quad tick \in \mathrm{Elig}_{\mathbf{G}}(s) \Rightarrow tick \in \mathrm{Elig}_K(s)$$

We say that languages $K_1, K_2 \subseteq L(\mathbf{G})$ are *jointly coercive* with respect to $\mathbf{G}$ if $K_1 \cap K_2$ is coercive with respect to $\mathbf{G}$. Now let **S1**, **S2** be proper supervisors for $\mathbf{G}$.

Theorem 9.12.1

S1 $\wedge$ **S2** is a proper supervisor for **G** if

(i) **S1** $\wedge$ **S2** is trim,

(ii) $L_m(\mathbf{S1} \wedge \mathbf{G})$, $L_m(\mathbf{S2} \wedge \mathbf{G})$ are nonconflicting, and

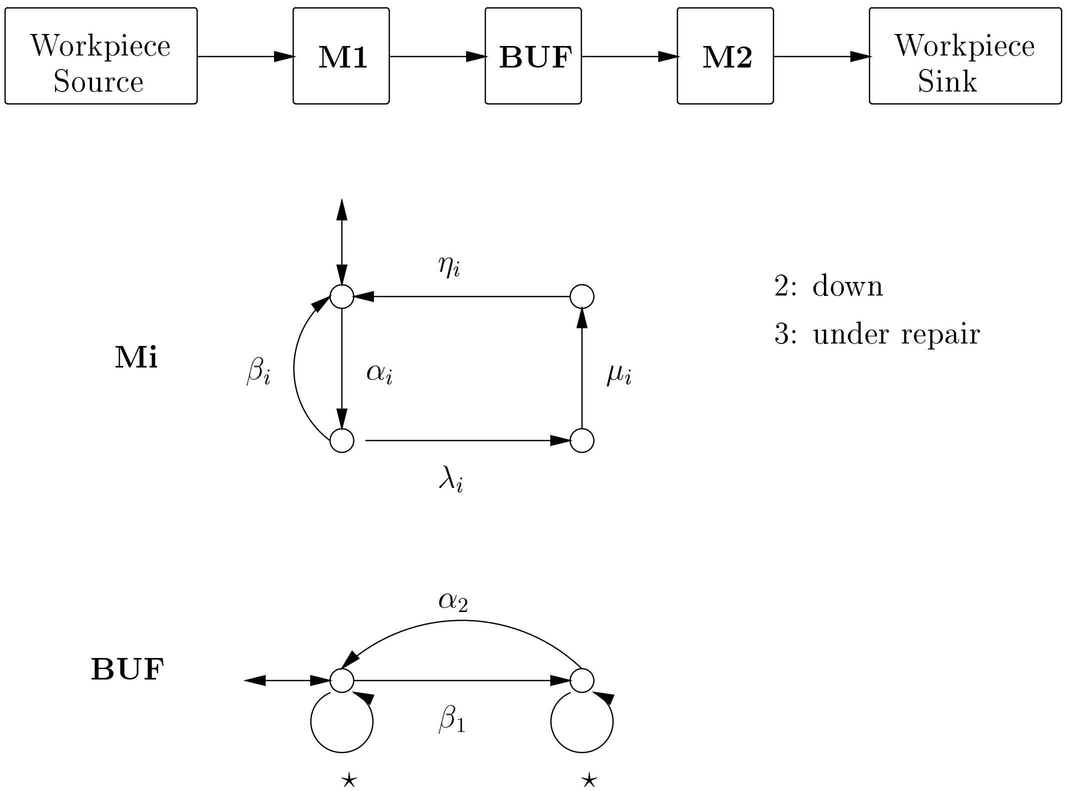
(iii) $L_m(\mathbf{S1} \wedge \mathbf{G})$, $L_m(\mathbf{S2} \wedge \mathbf{G})$ are jointly coercive with respect to **G**.

$\square$

Exercise 9.12.2: Prove Theorem 9.12.1. $\diamond$

Example 9.12.1: Timed Workcell

Consider a workcell consisting of two machines **M1,M2** linked by a one-slot buffer **BUF** as shown below.



$$\star = \{\alpha_1, \lambda_1, \mu_1, \eta_1, \beta_2, \lambda_2, \mu_2, \eta_2\}$$

Fig. 9.12.1

Let

$$\Sigma_{unc} = \{\beta_i, \lambda_i, \eta_i \mid i = 1, 2\}, \quad \Sigma_{for} = \Sigma_{hib} = \{\alpha_i, \mu_i \mid i = 1, 2\}$$

with corresponding timed events

$$(\alpha_1, 0, \infty), \quad (\beta_1, 1, 2), \quad (\lambda_1, 0, 2), \quad (\mu_1, 0, \infty), \quad (\eta_1, 1, \infty)$$

$$(\alpha_2, 0, \infty), \quad (\beta_2, 1, 1), \quad (\lambda_2, 0, 1), \quad (\mu_2, 0, \infty), \quad (\eta_2, 2, \infty)$$

For the TDES to be controlled we take **WORKCELL = comp(M1,M2)**, under the informal specifications

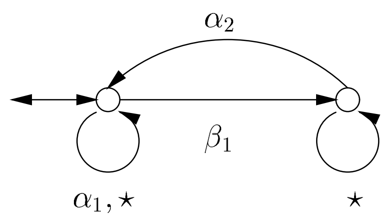
1. **BUF** must not overflow or underflow
2. If **M2** goes down, its repair must be started "immediately", and prior to starting repair of **M1** if **M1** is currently down.

These specifications are formalized by **R1**, **R2** respectively, as in Fig. 9.12.2; the final specification is **R = meet(R1,R2)**. It turns out that

$$\mathbf{SUPER} := \mathbf{meet}(\mathbf{WORKCELL}, \mathbf{R}) \quad (49, 110)$$

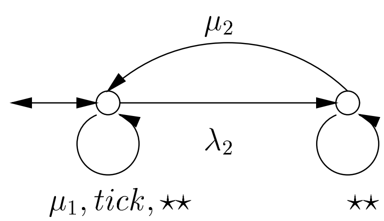
is controllable and qualifies as a proper supervisor for **WORKCELL**.

R1



$$\star = \{\lambda_1, \mu_1, \eta_1, \beta_2, \lambda_2, \mu_2, \eta_2, tick\}$$

R2



$$\star\star = \{\alpha_1, \beta_1, \lambda_1, \eta_1, \alpha_2, \beta_2, \eta_2\}$$

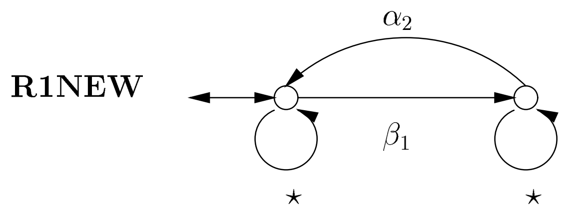
Fig. 9.12.2

Now let us try **R1** and **R2** as modular supervisors. We leave the reader to verify that the same result is achieved as for **SUPER**, namely the modular approach succeeds and is optimal.

Exercise 9.12.3: Check the conditions of Theorem 9.12.1, taking **Si** = **Ri**, **G** = **WORKCELL**. Use *TTCT* **condat** to check joint coerciveness.

Exercise 9.12.4: In the example above, show that the jointly coercive property of **R1, R2** comes into play on the occurrence of λ_2: the repair of **M2** must be initialized without delay, thus preempting the occurrence of *tick* (by the forcible event μ_2) in the transition structure of **R2** and consequently of **R1**. $\diamond$

Finally let us replace the specification **R1** with the less stringent specification **R1NEW**, as displayed in Fig. 9.12.3.



$$\star = \{\alpha_1, \lambda_1, \mu_1, \eta_1, \beta_2, \lambda_2, \mu_2, \eta_2, tick\}$$

Fig. 9.12.3

R1NEW does not qualify as a modular supervisor, because the uncontrollable event β_1 has been erased at state 1, leaving **R1NEW** uncontrollable. Let **SUP1NEW** represent the supremal controllable sublanguage of $L_m(\mathbf{WORKCELL} \wedge \mathbf{R1NEW})$. It can be verified that correct modular control is achieved with the conjunction of **SUP1NEW** with **R2**, and the resulting controlled behavior is a little freer than the behavior we obtained initially. It should be noted that the 'improved' behavior depends critically on the forcibility of event α_2 and could not be achieved in the corresponding untimed framework.

Exercise 9.12.5: Verify the preceding comments in detail.

9.13 Conclusions

The chapter provides a framework for the study of theoretical issues in the design of supervisory controls for timed discrete-event systems. The model incorporates both time delays and hard deadlines, and admits both forcing and disablement as means of control. In addition it supports composition of modular subsystems and systematic synthesis. In particular, the model retains the concept of design that is qualitatively optimal in the sense of minimally restricting the behavior of the underlying controlled system, subject to constraints imposed by formal specifications of performance.

Because the theory is expressed in the elementary framework of regular languages and finite automata, only rudimentary control scenarios can be treated directly. For instance, no explicit provision is made for program variables, or such constructs of a real-time programming language as interrupts and logically conditioned events and procedures. In higher-level approaches where such constructs are supported, design approaches tend to be heuristic. With the introduction of suitable architecture the present framework may supply a basis for rendering such approaches more formal and systematic.

9.14 Notes and References

The timed DES framework of this chapter is a simplified version of that for 'timed transition models' treated by Ostroff ([T06], Ostroff [1989,1990], [J13]); the new controllability features, including forcing, are due to Brandin ([T11], [T26], [C51], [C52], [J22]). Time bounds on events in timed Petri nets were previously employed by Merlin & Farber [1976] and Berthomieu & Diaz [1991], while forcing in DES was investigated by Golaszewski & Ramadge [1987]. From a different perspective, timed automata are described in Alur & Dill [1990] and applied to supervisory control in Wong-Toi & Hoffmann [1991].

Bibliography

TEXTBOOKS, MONOGRAPHS AND PROCEEDINGS

Discrete-Event Systems

Baccelli, F., G. Cohen, G.J. Olsder, J.-P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.

Balemi, S., P. Kozak, R. Smedinga (Eds.). *Discrete Event Systems: Modeling and Control.* Birkhaüser, 1993.

Boel, R., G. Stremersch (Eds.). *Discrete Event Systems: Analysis and Control.* [Proceedings of WODES2000, Ghent, August 21-23, 2000]. Kluwer, 2000.

Caillaud, B., P. Darondeau, L. Lavagno, X. Xie (Eds.). *Synthesis and Control of Discrete Event Systems.* [Proceedings of SCODES'2001, INRIA, Paris, July 2001]. Kluwer, 2002.

Cassandras, C.G. *Discrete Event Systems.* Irwin, 1993.

Cassandras, C.G., S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer, 1999.

Cohen, G., J.-P. Quadrat (Eds). *Proc. 11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems.* Sophia-Antipolis, June 1994. Lecture Notes in Control and Information Sciences vol. 119, Springer-Verlag, 1994.

Ho, Y.-C. (Ed.). *Special Issue on Dynamics of Discrete-Event Systems.* Proc. IEEE 77 (1), January 1989.

Ho, Y.-C., Y.-P. Zheng (Eds.). *Proc. 1991 IFAC Workshop on Discrete Event System Theory and Applications in Manufacturing and Social Phenomena.* International Academic, 1991.

International Workshop on Discrete Event Systems – WODES'96. Institute of Electrical Engineers, London, UK, August 1996.

International Workshop on Discrete Event Systems – WODES'98. Institute of Electrical Engineers, London, UK, August 1998.

Varaiya, P., A.B. Kurzhanski (Eds.). *Discrete Event Systems: Models and Applications.* Lecture Notes in Control and Information Sciences 103, Springer-Verlag, 1987.

Viswanadham, N., Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Prentice-Hall, 1992.

Automata and Languages

Arnold, A. *Finite Transition Systems*, Prentice-Hall, 1994.

Carroll, J., D. Long. *Theory of Finite Automata.* Prentice-Hall, 1989.

Hopcroft, J.E., J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

Lewis, H.R., C.H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

Algebra

Burris, S., H.P. Sankappanavar. *A Course in Universal Algebra.* Springer-Verlag, 1981.

Davey, B.A., H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

Mac Lane, S., G. Birkhoff. *Algebra.* Third ed., Chelsea, 1993.

Szasz, G. *Introduction to Lattice Theory.* Academic, 1963.

Wechler, W. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992.

Temporal Logic

Manna, Z., A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

Ostroff, J.S. *Temporal Logic for Real-Time Systems.* Research Studies Press, 1989.

Petri Nets

David, R., H. Alla. *Petri Nets and Grafcet.* Prentice-Hall, 1992.

Desrochers, A.A., R.Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems.* IEEE Press, 1995.

Moody, J., P. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets.* Kluwer, 1998.

Peterson, J.L. *Petri Net Theory and Modeling of Systems.* Prentice-Hall, 1981.

Reisig, W. *Petri Nets.* Springer-Verlag, 1985.

Zhou, M.C., F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems.* Kluwer, 1993.

Zhou, M.C., K. Venkatesh. *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach.* World Scientific, Singapore, 1999.

Zhou, M.C. (Ed.). *Petri Nets in Flexible and Agile Automation.* Kluwer, 1995.

Discrete-Event Simulation

Birtwistle, G.M. *DEMOS - A System for Discrete Event Modelling on Simula.* Springer-Verlag, 1987.

Fishman, G.S. *Principles of Discrete Event Simulation.* Wiley, 1978.

Law, A.M., W.D. Kelton. *Simulation, Modeling & Analysis.* McGraw-Hill, 1991.

Zeigler, B.P. *Multifacetted Modeling and Discrete Event Simulation.* Academic, 1984.

Zeigler, B.P. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic, 1990.

Programming and Computation

Ben-Ari, M. *Principles of Concurrent Programming.* Prentice-Hall International, 1982.

Dijkstra, Edsger W. *A Discipline of Programming.* Prentice-Hall, 1976.

Germundsson, R. *Symbolic Systems - Theory, Computation and Applications.* Ph.D. thesis No. 389, Dept. of Electrical Engineering, Linköping University, 1995.

Gunnarsson, J. *On Modeling of Discrete Event Dynamic Systems, Using Symbolic Algebraic Methods.* Thesis No. 502, Division of Automatic Control, Lund University, 1997.

Holt, R.C., G.S. Graham, E.D. Lazowska, M.A. Scott. *Structured Concurrent Programming With Operating Systems Applications.* Addison-Wesley, 1978.

Lynch, N. *Distributed Algorithms.* Morgan Kaufmann, 1996.

Magee, J., J. Kramer. *Concurrency: State Models and Java Programs.* Wiley, 1999.

Mesarovic, M.D., D. Macko, Y. Takahara. *Theory of Hierarchical, Multilevel, Systems.* Academic, 1970.

Implementation and Engineering

Bennett, S. *Real-Time Computer Control: An Introduction.* Prentice-Hall, 1988.

Burns, A., A. Wellings. *Real-Time Systems and Their Programming Languages.* Sec. ed., Addison-Wesley, 1997.

Fleming, P.I. *Parallel Processing in Control: The Transputer and Other Architectures.* Peregrinus, 1988.

Gray, D. *Introduction to the Formal Design of Real-Time Systems.* Springer, 1999.

Krishna, C.M., K.G. Shin. *Real-Time Systems.* McGraw-Hill, 1997.

Raynal, M. *Algorithms for Mutual Exclusion.* The MIT Press, 1986.

SUPPLEMENTARY REPORTS AND ARTICLES

R.Y. Al-Jaar, A.A. Desrochers. A modular approach for the performance analysis of automated manufacturing systems using generalized stochastic Petri nets. Rpt. RAL #116, Robotics and Automation Lab., Rensselaer Polytechnic Institute, Troy NY, 1988.

R. Alur, D. Dill. Automata for modeling real-time systems. *In Proc. 17th International Colloquium on Automata, Languages and Programming. Lecture Notes on Computer Science vol. 443*, Springer-Verlag 1990, pp. 322-335.

B. Berthomieu, M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering* 17 (3), March 1991, pp. 259-273.

E.W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica* 1, 1971, pp. 115-138.

A. Falcione, B. Krogh. Design recovery for relay ladder logic. *IEEE Control Systems* 13 (2) April 1993, pp. 90-98.

M. Heymann. Concurrency and discrete event control. *IEEE Control Systems* 10 (4) 1990, pp. 103-112.

P. Hubbard, P.E. Caines. Dynamical consistency in hierarchical supervisory control. *IEEE Trans. on Automatic Control* 47 (1) January 2002, pp. 37-52.

A. Kay, J.N. Reed. A relay and guarantee method for timed CSP: a specification and design of a telephone exchange. *IEEE Trans. on Software Engineering* 19 (6) June 1993, pp. 625-639.

P.M. Merlin, D.J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Trans. on Communications* 24, Sept. 1976, pp. 1036-1043.

J.S. Ostroff. Deciding properties of timed transition models. *IEEE Trans. on Parallel and Distributed Systems* 1 (2), April 1990, pp. 170-183.

H. Simon. The architectecture of complexity. *Proc. Amer. Phil. Soc.* 106, December 1967, pp. 467-482. Reprinted in: Herbert A. Simon, *The Sciences of the Artificial* (Chapt. 7, pp. 193-229), Second ed., The MIT Press, Cambridge MA, 1981.

E.V. Sørensen, J. Nordahl, N.H. Hansen. From CSP models to Markov models. *IEEE Trans. on Software Engineering* 19 (6) June 1993, pp. 554-570.

K.C. Wong. On the complexity of projections of discrete-event systems. *Proc. International Workshop on Discrete Event Systems (WODES' 98).* IEE, London, 1998; pp. 201-206.

H. Wong-Toi, G. Hoffmann. The control of dense real-time discrete event systems. *Proc. of the 30th IEEE Conference on Decision and Control*, Brighton, U.K., December 1991, pp. 1527-1528.

SYSTEMS CONTROL GROUP PUBLICATIONS AND THESES ON DISCRETE-EVENT SYSTEMS

Journal Publications

[J01] A.F. Vaz, W.M. Wonham. On supervisor reduction in discrete-event systems. International J. Control 44 (2), 1986, pp. 475-491.

[J02] J.G. Thistle, W.M. Wonham. Control problems in a temporal logic framework. International J. Control 44 (4), 1986, pp. 943-976.

[J03] P.J. Ramadge, W.M. Wonham. Supervisory control of a class of discrete event processes. SIAM J. Control and Optimization 25 (1), 1987, pp. 206-230.

[J04] W.M. Wonham. Some remarks on control and computer science. Control Systems Magazine 7 (2), 1987, pp. 9-10.

[J05] W.M. Wonham, P.J. Ramadge. On the supremal controllable sublanguage of a given language. SIAM J. Control and Optimization 25 (3), 1987, pp. 637-659.

[J06] P.J. Ramadge, W.M. Wonham. Modular feedback logic for discrete event systems. SIAM J. Control and Optimization, 25 (5), 1987, pp. 1202-1218.

[J07] W.M. Wonham, P.J. Ramadge. Modular supervisory control of discrete event systems. Maths. of Control, Signals & Systems 1 (1), 1988, pp. 13-30.

[J08] F. Lin, W.M. Wonham. Decentralized supervisory control of discrete-event systems. Information Sciences 44 (2), 1988, pp. 199-224.

[J09] F. Lin, W.M. Wonham. On observability of discrete-event systems. Information Sciences 44 (2), 1988, pp. 173-198.

[J10] F. Lin, A. Vaz, W.M. Wonham. Supervisor specification and synthesis for discrete event systems. International J. Control 48 (1), 1988, pp. 321-332.

[J11] Y. Li, W.M. Wonham. On supervisory control of real-time discrete-event systems. Information Sciences 46 (3), 1988, pp. 159-183.

[J12] P.J. Ramadge, W.M. Wonham. The control of discrete event systems. Proc. IEEE, Special Issue on Discrete Event Dynamic Systems, 77 (1), January 1989, pp. 81-98.

[J13] J.S. Ostroff, W.M. Wonham. A framework for real-time discrete event control. IEEE Trans. on Automatic Control 35 (4) 1990, pp. 386-397.

[J14] H. Zhong, W.M. Wonham. On consistency of hierarchical supervision in discrete-event systems. IEEE Trans. on Automatic Control 35 (10) 1990, pp. 1125-1134.

[J15] R.D. Brandt, V. Garg, R. Kumar, F. Lin, S.I. Marcus, W.M. Wonham. Formulas for calculating supremal controllable and normal sublanguages. Systems & Control Letters 15, 1990, pp. 111-117.

[J16] F. Lin, W.M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. IEEE Trans. on Automatic Control 35 (12), 1990, pp. 1330-1337.

[J17] K. Rudie, W.M. Wonham. The infimal prefix-closed and observable superlanguage of a given language. Systems & Control Letters 15, 1990, pp. 361-371.

[J18] F. Lin, W.M. Wonham. Verification of nonblocking in decentralized supervision. Control Theory and Advanced Technology 7 (1) 1991, pp. 19-29.

[J19] T. Ushio, Y. Li, W.M. Wonham. Concurrency and state feedback in discrete-event systems. IEEE Trans. on Automatic Control 37 (8) 1992, pp. 1180-1184.

[J20] K. Rudie, W.M. Wonham. Think globally, act locally: decentralized supervisory control. IEEE Trans. on Automatic Control 37 (11) 1992, pp. 1692-1708. Reprinted in F.A. Sadjadi (Ed.), Selected Papers on Sensor and Data Fusion, 1996; ISBN 0-8194-2265-7.

[J21] Y. Li, W.M. Wonham. Control of vector discrete-event systems: I - The base model. IEEE Trans. on Automatic Control 38 (8), August 1993, pp. 1214-1227. Correction: IEEE Trans. on Automatic Control 39 (8) August 1994, p. 1771.

[J22] B.A. Brandin, W.M. Wonham. Supervisory control of timed discrete-event systems. IEEE Trans. on Automatic Control 39 (2) February 1994, pp. 329-342.

[J23] Y. Li, W.M. Wonham. Control of vector discrete-event systems: II - controller synthesis. IEEE Trans. on Automatic Control 39 (3) March 1994, pp. 512-531.

[J24] J.G. Thistle, W.M. Wonham. Control of infinite behavior of finite automata. SIAM J. on Control and Optimization 32 (4) July 1994, pp. 1075-1097.

[J25] J.G. Thistle, W.M. Wonham. Supervision of infinite behavior of discrete-event systems. SIAM J. on Control and Optimization 32 (4) July 1994, pp. 1098-1113.

[J26] F. Lin, W.M. Wonham. Supervisory control of timed discrete event systems under partial observation. IEEE Trans. on Automatic Control 40 (3) March 1995, pp. 558-562.

[J27] Y. Li, W.M. Wonham. Concurrent vector discrete-event systems. IEEE Trans. on Automatic Control 40 (4) April 1995, pp. 628-638.

[J28] M. Lawford, W.M. Wonham. Equivalence preserving transformations for timed transition models. IEEE Trans. on Automatic Control 40 (7) July 1995, pp. 1167-1179.

[J29] P. Kozak, W.M. Wonham. Fully decentralized solutions of supervisory control problems. IEEE Trans. on Automatic Control 40 (12) December 1995, pp. 2094-2097.

[J30] K.C. Wong, W.M. Wonham. Hierarchical control of discrete-event systems. Discrete Event Dynamic Systems 6 (3) July 1996, pp. 241-273.

[J31] K.C. Wong, W.M. Wonham. Hierarchical control of timed discrete-event systems. Discrete Event Dynamic Systems 6 (3) July 1996, pp. 274-306.

[J32] P. Kozak, W.M. Wonham. Design of transaction management protocols. IEEE Trans. Automatic Control 41 (9) September 1996, pp. 1330-1335.

[J33] K.C. Wong, W.M. Wonham. Modular control and coordination of discrete event systems. Discrete Event Dynamic Systems 8 (3) October 1998, pp. 247-297.

[J34] S. Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Supremum operators and computation of supremal elements in system theory. SIAM J. Control & Optimization 37 (3) March 1999, pp. 695-709.

[J35] P. Gohari, W.M. Wonham. On the complexity of supervisory control design in the RW framework. IEEE Trans. on Systems, Man and Cybernetics; Part B: Cybernetics. (Special Issue on Discrete Systems and Control) 30 (5) October 2000, pp. 643-652.

[J36] P.C.Y. Chen, W.M. Wonham. Stable supervisory control of flexible manufacturing systems with fixed supply and demand rates. International J. of Production Research: Special Issue on Modeling, Specification and Analysis of Manufacturing Systems 39 (2) January 2001, pp. 347-368.

[J37] P.C.Y. Chen, W.M. Wonham. Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. Real Time Systems J. [to appear].

Conference Papers

[C01] P.J. Ramadge, W.M. Wonham. Supervisory control of discrete event processes. Joint Workshop on Feedback & Synthesis of Linear & Nonlinear Systems. Istituto di Automatica, Univ. di Roma, June 1981. In: D. Hinrichsen, A. Isidori (eds.), Feedback Control of Linear and Nonlinear Systems, Lecture Notes on Control and Information Sciences No. 39, Springer-Verlag, Berlin, 1982, pp. 202-214.

[C02] P.J. Ramadge, W.M. Wonham. Algebraic decomposition of controlled sequential machines. Eighth Triennial World Congress, Intntl. Fedn. Aut. Control (IFAC), Kyoto, August 1981. Preprints, vol. 3, pp. 37-41.

[C03] P.J. Ramadge, W.M. Wonham. Supervision of discrete event processes. Proc. 21st IEEE Conf. on Decision and Control, IEEE Control Systems Society, New York, December 1982, pp. 1228-1229.

[C04] P.J. Ramadge, W.M. Wonham. Supervisory control of a class of discrete event processes. Proc. Sixth Intnl. Conference on Analysis and Optimization of Systems, Nice, June 1984. In: A. Bensoussan, J.L. Lions (eds.), Analysis and Optimization of Systems, Lecture Notes on Control and Information Sciences No. 63, Springer-Verlag, Berlin, 1984; Part 2, pp. 477-498.

[C05] W.M. Wonham, P.J. Ramadge. On the supremal controllable sublanguage of a given language. Proc. 23rd IEEE Conf. on Decision and Control, IEEE Control Systems Society, New York, December 1984, pp. 1073-1080.

[C06] W.M. Wonham, P.J. Ramadge. On modular synthesis of supervisory controls for discrete event processes. Proc. Intnl. Conf. on Computers, Systems and Signal Processing, IEEE and I.I.Sc., Bangalore, December 1984, pp. 500-504.

[C07] W.M. Wonham. On control of discrete event systems. Seventh Intnl. Symp. on the Mathematical Theory of Networks and Systems (MTNS-85), Stockholm, June 1985. In: C.I. Byrnes, A. Lindquist (eds.), Computational and Combinatorial Methods in Systems Theory, North-Holland, Amsterdam, 1986, pp. 159-174.

[C08] F. Lin, W.M. Wonham. On the computation of supremal controllable sublanguages. Proc. 23rd Annual Allerton Conf. on Communication, Control and Computing. Univ. of Illinois, Urbana, October 1985, pp. 942-950.

[C09] J.G. Thistle, W.M. Wonham. On the use of temporal logic in control theory. Proc. 23rd Annual Allerton Conf. on Communication, Control and Computing. Univ. of Illinois, October 1985, pp. 961-970.

[C10] A. Vaz, W.M. Wonham. On supervisor reduction in discrete event systems. Proc. 23rd Annual Allerton Conf. on Communication, Control and Computing. Univ. of Illinois, Urbana, October 1985, pp. 933-939.

[C11] J.S. Ostroff, W.M. Wonham. A temporal logic approach to real time control. Proc. 24th IEEE Conf. on Decision and Control, IEEE Control Systems Society, New York, December 1985, pp.656-657.

[C12] P.J. Ramadge, W.M. Wonham. Modular supervisory control of discrete event systems. Seventh Intnl. Conf. on Analysis and Optimization of Systems, Antibes, June 1986. In: A. Bensoussan, J.L. Lions (eds.), Analysis and Optimization of Systems, Lecture Notes on Control and Information Sciences, vol.83, Springer-Verlag, New York, 1986, pp. 202-214.

[C13] P.J. Ramadge, W.M. Wonham. Modular feedback logic for discrete event systems. Fourth IFAC/IFORS Symp., Large Scale Systems: Theory and Applications, Zurich, August 1986. In: H.P. Geering, M. Mansour (eds.), Large Scale Systems: Theory and Applications. Pergamon Press, Oxford, August 1986, vol.1, pp. 83-88.

[C14] W.M. Wonham. Some remarks on control and computer science. Workshop on Future Directions in System Theory and Applications, Univ. of Santa Clara, September 1986.

[C15] Y. Li, W.M. Wonham. Supervisory control of real-time discrete-event systems. Proceedings, 1987 American Control Conference, Minneapolis, June 1987, pp. 1715-1720.

[C16] F. Lin, W.M. Wonham. Supervisory control and observation of discrete-event systems. MTNS '87 - Mathematical Theory of Networks and Systems - International Symposium, Phoenix, June 1987. In: Byrnes, C.I., Martin, C.F., Saeks, R.E. (eds.): Analysis and Control of Nonlinear Systems. North-Holland, Amsterdam, 1988, pp. 337-348.

[C17] J.G. Thistle, W.M. Wonham. Supervisory control with infinite-string specifications. Proc. Twenty-Fifth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1987, vol. 1, pp. 327-334.

[C18] W.M. Wonham. Logic and language in control theory. Proc. Twenty-Fifth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1987, vol. 1, pp. 1-3.

[C19] J.S. Ostroff, W.M. Wonham. State machines, temporal logic and control: a framework for discrete event systems. Proc. 26th IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, 1987, pp. 681-686.

[C20] J.S. Ostroff, W.M. Wonham. Modelling, specifying and verifying real-time embedded computer systems. Proc. Eighth Real-Time Systems Symposium, IEEE Computer Society, New York, December 1987, pp. 124-132.

[C21] H. Zhong, W.M. Wonham. On hierarchical control of discrete-event systems. Proc. 1988 Conference on Information Sciences and Systems, Dept. of Electrical Engineering, Princeton University, 1988, pp. 64-70.

[C22] Y. Li, W.M. Wonham. Deadlock issues in supervisory control of discrete-event systems. Proc. 1988 Conference on Information Sciences and Systems, Dept. of Electrical Engineering, Princeton University, 1988, pp. 57-63.

[C23] J.G. Thistle, W.M. Wonham. On the synthesis of supervisors subject to ω-language specifications. Proc. 1988 Conference on Information Sciences and Systems, Dept. of Electrical Engineering, Princeton University, 1988, pp. 440-444.

[C24] Y. Li, W.M. Wonham. A state-variable approach to the modeling and control of discrete-event systems. Proc. Twenty-Sixth Annual Allerton Conference on Communication, Control, and Computing. University of Illinois, September 1988, pp. 1140-1149.

[C25] W.M. Wonham. A control theory for discrete-event systems. In: M.J. Denham, A.J. Laub (eds.), Advanced Computing Concepts and Techniques in Control Engineering, NATO ASI Series, vol. F47, Springer-Verlag, Berlin, 1988; pp. 129-169.

[C26] W.M. Wonham. A language-based control theory of discrete-event systems. Preprints, Shell Conference on Logistics, Appeldoorn, The Netherlands, October 1988. In C.F.H. van Rijn (Ed.), Logistics - Where Ends Have To Meet, Pergamon, Oxford (UK), 1989, pp. 158- 169.

[C27] Y. Li, W.M. Wonham. Controllability and observability in the state-feedback control of discrete-event systems. Proc. 27th IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, December 1988, pp. 203-208.

[C28] F. Lin, W.M. Wonham. Decentralized control and coordination of discrete-event systems. Proc. 27th IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, December 1988, pp. 1125-1130.

[C29] Y. Li, W.M. Wonham. Composition and modular state-feedback control of vector discrete-event systems. Proc. 1989 Conf. on Information Sciences and Systems, The Johns Hopkins University, Baltimore, March 1989, pp. 103-111.

[C30] Y. Li, W.M. Wonham. Strict concurrency and nondeterministic control of discrete-event systems. Proc. 28th IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, December 1989, pp. 2731-2736.

[C31] T. Ushio, Y. Li, W.M. Wonham. Basis feedback, weak interaction, and concurrent well-posedness in concurrent discrete-event systems. Proc. 28th IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, December 1989, pp. 127-131.

[C32] F. Lin, R.D. Brandt, W.M. Wonham. A note on supremal controllable and normal sublanguages. Proc. Twenty-Seventh Annual Allerton Conf. on Communication, Control and Computing, September 1989, pp. 491-500.

[C33] Y. Li, W.M. Wonham. Linear integer programming techniques in the control of vector discrete-event systems. Proc. Twenty-Seventh Annual Allerton Conf. on Communication, Control and Computing, Univ. of Illinois, September 1989, pp. 528-537.

[C34] H. Zhong, W.M. Wonham. Hierarchical control of discrete-event systems: computation and examples. Proc. Twenty-Seventh Annual Allerton Conf. on Communication, Control and Computing, Univ. of Illinois, September 1989, pp. 511-519.

[C35] N.Q. Huang, Y. Li, W.M. Wonham. Supervisory control of vector discrete-event systems. Proc. Twenty-Seventh Annual Allerton Conf. on Communication, Control and Computing, Univ. of Illinois, September 1989, pp. 925-934.

[C36] S.D. O'Young, W.M. Wonham. Object-oriented computation and simulation of large-scale discrete event systems. Proc. Twenty-Seventh Annual Allerton Conf. on Communication, Control and Computing, Univ. of Illinois, September 1989, pp. 945-954.

[C37] K. Rudie, W.M. Wonham. Supervisory control of communicating processes. Tenth International IFIP WG 6.1 Symposium on Protocol Specification, Testing and Verification. Ottawa, June 1990. [In L. Logrippo, R.L. Probert, H. Ural (Eds.). Protocol Specification, Testing and Verification, X. Elsevier (North-Holland), 1990, pp. 243-257.]

[C38] H. Zhong, W.M. Wonham. Hierarchical coordination. Proc. Fifth IEEE International Symposium on Intelligent Control. Philadelphia, September 5-7, 1990, pp. 8-14.

[C39] K. Rudie, W.M. Wonham. Think globally, act locally: decentralized supervisory control. Proc. 1991 American Control Conference, Boston, June 1991, pp. 898-903.

[C40] W.M. Wonham. Some current research directions in control of discrete-event systems. Presented at European Control Conference ECC 91, Grenoble, July 1991. [preprint available separately from Proceedings]

[C41] B.A. Brandin, B. Benhabib, W.M. Wonham. Discrete event system supervisory control applied to the management of manufacturing workcells. Proc. Seventh International Conference on Computer-Aided Production Engineering, Cookeville TN USA, August 1991; Elsevier, Amsterdam, 1991, pp. 527-536.

[C42] J.G. Thistle, W.M. Wonham. Control of ω-automata, Church's problem, and the emptiness problem for tree ω-automata. Proc. Computer Science Logic CSL '91 (Institut fur Informatik und angewandte Mathematik, Universitat Bern, Bern, Switzerland, October 1991). In E.Börger, G. Jäger, H.K Büning, M.M. Richter (Eds.). Lecture Notes in Computer Science No. 626, Springer-Verlag, Berlin, 1992, pp. 367-381.

[C43] M. Lawford, W.M. Wonham. An application of real-time transformational equivalence. Proc. 1992 Conference on Information Sciences and Systems, vol. 1, pp. 233-238, Princeton University, Princeton NJ, 1992.

[C44] B.A. Brandin, W.M. Wonham, B. Benhabib. Manufacturing cell supervisory control - a timed discrete event system approach. Proc. 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 1992, pp. 931-936.

[C45] B.A. Brandin, W.M. Wonham, B. Benhabib. Modular supervisory control of timed discrete-event systems. Proc. Thirtieth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1992, pp. 624-632.

[C46] K.C. Wong, W.M. Wonham. Hierarchical and modular control of discrete-event systems. Proc. Thirtieth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1992, pp. 614-623.

[C47] S.-L. Chen, W.M. Wonham. Existence and design of supervisors for vector discrete-event systems. Proc. Thirtieth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1992, pp. 604-613.

[C48] T.-J. Ho, W.M. Wonham. A framework for timed discrete-event systems. Proc. Thirtieth Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 1992, pp. 650-651.

[C49] K. Rudie, W.M. Wonham. Protocol verification using discrete-event systems. Proc. 31st IEEE Conference on Decision and Control, Tucson, Arizona, December 1992, pp. 3770-3777.

[C50] M. Lawford, W.M. Wonham. Equivalence preserving transformations for timed transition models. Proc. 31st IEEE Conference on Decision and Control, Tucson, Arizona, December 1992, pp. 3350-3356.

[C51] B.A. Brandin, W.M. Wonham. Supervisory control of timed discrete-event systems. Proc. 31st IEEE Conference on Decision and Control, Tucson, Arizona, December 1992, pp. 3357-3362.

[C52] B.A. Brandin, W.M. Wonham, B. Benhabib. Manufacturing cell supervisory control - a modular timed discrete-event system approach. Proc. IEEE International Conference on Robotics and Automation, Atlanta, Georgia, May 1993, pp. 931-936.

[C53] K.C. Wong, W.M. Wonham. Hierarchical control of timed discrete-event systems. Proc. Second European Control Conference 1993, Groningen, The Netherlands, June-July 1993, pp. 509-512.

[C54] M. Lawford, W.M. Wonham. Supervisory control of probabilistic discrete Event Systems event systems. Proc. 36th Midwest Symposium on Circuits and Systems, Detroit, MI, August 1993, pp. 327-331.

[C55] B.A. Brandin, W.M. Wonham. Modular supervisory control of timed discrete-event systems. Proc. 32nd IEEE Conference on Decision and Control, IEEE Control Systems Society, New York, December 1993, pp. 2230-2235.

[C56] M. Lawford, W.M. Wonham, J.S. Ostroff. State-event observers for labeled transition systems. Proc. 33rd IEEE Conference on Decision and Control, December 1994, pp. 3642-3648.

[C57] W.M. Wonham. Supervisory control automata for discrete event systems. Preprints, ADEDOPS Workshop (Analysis and Design of Event-Driven Operations in Process Systems), Centre for Process Systems Engineering, Imperial College of Science Technology and Medicine, London UK, April 1995; 34 pp.

[C58] S.L. Chen, W.M. Wonham. Existence and design of supervisors for vector discrete-event systems. Proc. 1995 Canadian Conference on Electrical and Computer Engineering, Montreal, September 1995, pp. 805-808.

[C59] R.J. Leduc, W.M. Wonham. Discrete event systems modeling and control of a manufacturing testbed. Proc. 1995 Canadian Conference on Electrical and Computer Engineering, Montreal, September 1995, pp. 793-796.

[C60] S.L. Chen, W.M. Wonham. Supervisory control of finite automata under dynamic predicate specifications. Proc. Thirty-Third Annual Allerton Conference on Communication, Control and Computing, University of Illinois, October, 1995, pp. 501-509.

[C61] R.J. Leduc, W.M. Wonham. PLC implementation of a DES supervisor for a manufacturing testbed. Proc. Thirty-Third Annual Allerton Conference on Communication, Control and Computing, University of Illinois, October, 1995, pp. 519-528.

[C62] K.C. Wong, W.M. Wonham. Modular control and coordination of discrete-event systems. Proc. 4th IEEE Mediterranean Symposium on New Directions in Control & Automation, Chania, Crete, Greece, June 1996; pp. 595-599.

[C63] K.C. Wong, W.M. Wonham. Modular control and coordination of a transfer line: a tutorial example. Pre-Proceedings of the 1996 Workshop on Theoretical Problems on Manufacturing Systems Design and Control, Univ. of Minho and Univ. of Parma, Lisbon, June 1996; pp. 21-29.

[C64] M. Lawford, J.S. Ostroff, W.M. Wonham. Model reduction of modules for state-event temporal logics. Workshop on Application of Formal Methods to System Development: Telecommunications, VLSI and Real-Time Computerized Control Systems. Jacques Cartier Centre, Montreal, October 1996; pp. 281-287.

[C65] M. Lawford, J.S. Ostroff, W.M. Wonham. Model reduction of modules for state-event temporal logics. In R. Gotzhein, J. Bredereke (Eds.), Formal Description Techniques IX, Chapman & Hall, London, 1996; pp. 263-278. [IFIP TC6/6.1 International Conference on Formal Description Techniques IX/ Protocol Specification, Testing and Verification XVI, Kaiserslautern, Germany, 8-11 October 1996]

[C66] Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Supremum operators and computation of supremal elements in system theory. Proc. 1997 IEEE Conference on Decision and Control (CDC'97), December 1997, pp. 2946-2951.

[C67] W.M. Wonham, P. Gohari M. A linguistic framework for controlled hierarchical DES. International Workshop on Discrete Event Systems (WODES '98). IEE, London, August 1998; pp. 207-212.

[C68] S. Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. Proc. 1998 IEEE Conference on Decision and Control (CDC'98), December 1998, pp. 3769-3774.

[C69] C.Y.P. Chen, W.M. Wonham. Non-preemptive scheduling of periodic tasks: a discrete-event control approach. Proc. Fifth International Conference on Control, Automation, Robotics and Vision. Nanyang Technological University, Singapore, December 8-11, 1998; pp. 1674-1678.

[C70] C.Y.P. Chen, W.M. Wonham. Real-time supervisory control of a processor for non-preemptive execution of periodic tasks. Proc. IFAC-99 World Congress, Beijing, July 1999, Vol. J, pp. 13-18.

[C71] S. Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Fault diagnosis in timed discrete-event systems. Proc. 1999 IEEE Conference on Decision and Control (CDC'99), pp. 1756-1761.

[C72] R. Minhas, W.M. Wonham. Modelling of timed discrete event systems. Proc. Thirty-Seventh Annual Allerton Conference on Communication, Control and Computing, Allerton, IL, September 1999, pp. 75-84.

[C73] S. Abdelwahed, W.M. Wonham. Interacting discrete event systems. Proc. Thirty-Seventh Annual Allerton Conference on Communication, Control and Computing, Allerton. IL, September 1999, pp. 85-92.

[C74] S. Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Fault diagnosis in finite-state automata and timed discrete-event systems. In: D. Miller, L. Qiu (Eds.), Topics in Control and Its Applications, pp. 81-105. Springer-Verlag, 1999.

[C75] W.M. Wonham. Supervisory control of discrete-event systems: an introduction. Proc. IEEE Intnl. Conf. on Industrial Technology 2000 (ICIT2000), Goa, India, January 19-22, 2000; pp. 474-479.

[C76] K.C. Wong, W.M. Wonham. On the computation of observers in discrete-event systems. 2000 Conference on Information Sciences and Systems, Princeton University, March 15-17, 2000, pp. TP1.7-TP1.12.

[C77] R. Su, W.M. Wonham. Decentralized fault diagnosis for discrete-event systems. 2000 Conference on Information Sciences and Systems, Princeton University, March 15-17, 2000, pp. TP1.1-TP1.6.

[C78] R.J. Leduc, B.A. Brandin, W.M. Wonham. Hierarchical interface-based non-blocking verification. Canadian Conference on Electrical and Computer Engineering (CCECE 2000), Halifax, May 7-10, 2000, pp. 1-6.

[C79] P. Gohari, W.M. Wonham. Reduced supervisors for timed discrete-event systems. In R. Boel, G. Stremersch (Eds.), Discrete Event Systems: Analysis and Control [Proc. WODES2000], Kluwer, 2000; pp. 119-130.

[C80] S. Hashtrudi Zad, R.H. Kwong, W.M. Wonham. Fault diagnosis and consistency in hybrid systems. Proc. Thirty-Eighth Annual Allerton Conference on Communications, Control, and Computing, Allerton, IL, October 2000; pp. 1135-1144.

[C81] S.E. Bourdon, W.M. Wonham, M. Lawford. Invariance under scaling of time bounds in timed discrete-event systems. Proc. Thirty-Eighth Annual Allerton Conference on Communications, Control, and Computing, Allerton, IL, October 2000; pp. 1145-1154.

[C82] R. Su, W.M. Wonham. Supervisor reduction for discrete-event systems. 2001 Conference on Information Sciences and Systems, The Johns Hopkins University, March 21-23, 2001 [6 pp.].

[C83] P. Dietrich, R. Malik, W.M. Wonham, B.A. Brandin. Implementation considerations in supervisory control. In B. Caillaud, X. Xie (Eds.), Proc. Symposium on the Supervisory Control of Discrete Event Systems (SCODES'2001), INRIA, Paris, July 2001; pp. 27-38.

[C84] Z.H. Zhang, W.M. Wonham. STCT: an efficient algorithm for supervisory control design. In B. Caillaud, X. Xie (Eds.), Proc. Symposium on the Supervisory Control of Discrete Event Systems (SCODES'2001), INRIA, Paris, July 2001; pp. 82-93. See also: B. Caillaud et al. (Eds.). Synthesis and Control of Discrete Event Systems, Kluwer 2002; pp. 77-100.

[C85] R.J. Leduc, B.A. Brandin, W.M. Wonham, M. Lawford. Hierarchical interface-based supervisory control: serial case. Proc. 40th IEEE Conference on Decision & Control, Orlando, Fla., December 2001, pp. 4116-4121.

[C86] R.J. Leduc, W.M. Wonham, M. Lawford. Hierarchical interface-based supervisory control: parallel case. Proc. Thirty-Ninth Annual Allerton Conference on Communications, Control, and Computing, Allerton, IL, October 2001, pp. 386-395.

[C87] R.J. Leduc, M. Lawford, W.M. Wonham. Hierarchical interface-based supervisory control: AIP example. Proc. Thirty-Ninth Annual Allerton Conference on Communications, Control, and Computing, Allerton, IL, October 2001, pp. 396-405.

[C88] W.M. Wonham. Supervisory control of discrete-event systems. In Asit K. Datta, Anish Deb, Samarjit Sengupta (Eds.): Proc. Intnl. Conf. on Control, Instrumentation and Information Communication (CIIC 2001), Dept. of Applied Physics, U. of Calcutta, 13-15 December 2001; pp. 321-330.

[C89] S.E. Bourdon, M. Lawford, W.M. Wonham. Robust nonblocking supervisory control of discrete-event systems. Proc. 2002 American Control Conference, Anchorage, May 2002, pp. 730-735.

Graduate Theses

[T01] P.J. Ramadge. Control and Supervision of Discrete Event Processes. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, May 1983.

[T02] F. Lin. Supervisor Synthesis for Discrete Event Processes. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, August 1984.

[T03] R.S.W. Chan. Simulation and Modelling of a Supervisory Control System Using Concurrent Euclid. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, September 1984.

[T04] J.G. Thistle. Control Problems in a Temporal Logic Setting. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, January 1985.

[T05] Y. Li. Supervisory Control of Real-Time Discrete Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, July 1986.

[T06] J. Ostroff. Real-Time Computer Control of Discrete Systems Modelled by Extended State Machines: A Temporal Logic Approach. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, January 1987.

[T07] J. Shifman. A Hierarchical Approach to Path Planning for Robots. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, September 1987.

[T08] F. Lin. On Controllability and Observability of Discrete Event Systems. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, November 1987.

[T09] H. Zhong. Control of Discrete-Event Systems: Decentralized and Hierarchical Control. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, December 1987.

[T10] K. Rudie. Software for the Control of Discrete-Event Systems: A Complexity Study. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, March 1988.

[T11] B.A. Brandin. The Supervisory Control of Discrete Event Systems with Forcible Events. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, October 1989.

[T12] K.C. Wong. An Algebraic Description of Hierarchical Control in Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, June 1990.

[T13] T.-M. Pai. Real Time Implementation of Discrete-Event Control Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, June 1990.

[T14] R.K. Wong. State-Based Discrete-Event Modeling and Control of Concurrent Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, November 1990.

[T15] J.G. Thistle. Control of Infinite Behaviour of Discrete-Event Systems. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, January 1991.

[T16] N.-Q. Huang. Supervisory Control of Vector Discrete-Event Processes. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, April 1991.

[T17] Y. Li. Control of Vector Discrete-Event Systems. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, April 1991.

[T18] I.E.H. Caulder. Applications of Decentralized Hierarchical DES Control to Telecommunications Network Protocols. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, August 1991.

[T19] M.S. Lawford. Transformational Equivalence of Timed Transition Models. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, January 1992.

[T20] H. Zhong. Hierarchical Control of Discrete-Event Systems. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, April 1992.

[T21] J. Liao. Hierarchical Control in Vector Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, April 1992.

[T22] C.-Y. Yuen. Control Synthesis for Timed Discrete Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, April 1992.

[T23] K. Rudie. Decentralized Control of Discrete-Event Systems. Ph.D. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, June 1992.

[T24] S.-L. Chen. Existence and Design of Supervisors for Vector Discrete Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, July 1992.

[T25] B. Schwartz. State Aggregation of Controlled Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. Engrg., Univ. of Toronto, July 1992.

[T26] B.A. Brandin. Real-Time Supervisory Control of Automated Manufacturing Systems. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, January 1993.

[T27] L. Zhu. Control Theory of Stochastic Discrete Event Systems. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 1993.

[T28] K.-C. Wong. Discrete-Event Control Architecture: An Algebraic Approach. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, June 1994.

[T29] B. Wang. Top-Down Design for RW Supervisory Control Theory. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, June 1995.

[T30] R.J. Leduc. PLC Implementation of a DES Supervisor for a Manufacturing Testbed: An Implementation Perspective. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, January 1996.

[T31] Y.-Q. Zhang. Software for State-Event Observation Theory and its Application to Supervisory Control. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, January 1996.

[T32] T.Y.L. Chun. Diagnostic Supervisory Control: A DES Approach. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, August 1996.

[T33] S.-L. Chen. Control of Discrete-Event Systems of Vector and Mixed Structural Type. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, September 1996.

[T34] M.S. Lawford. Model Reduction of Discrete Real-Time Systems. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, January 1997.

[T35] X.-Q. Zhang. Control of Boolean Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 1997.

[T36] Y.-C. Guan. Implementation of Hierarchical Observer Theory. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 1997.

[T37] T.-J. Ho. The Control of Real-Time Discrete-Event Systems Subject to Predicate-Based Constraints. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, August 1997.

[T38] P. Gohari-Moghadam. A Linguistic Framework for Controlled Hierarchical DES. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 1998.

[T39] C. Ma. A Computational Approach to Top-Down Hierarchical Supervisory Control of DES. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 1999.

[T40] S. Hashtrudi Zad. Fault Diagnosis in Discrete-Event and Hybrid Systems. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, August 1999.

[T41] K.Q. Pu. Modeling and Control of Discrete-Event Systems with Hierarchical Abstraction. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, March 2000.

[T42] R. Su. Decentralized Fault Diagnosis for Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 2000.

[T43] Y.W. Wang. Supervisory Control of Boolean Discrete-Event Systems. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, June 2000.

[T44] Z.H. Zhang. Smart TCT: An Efficient Algorithm for Supervisory Control Design. M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 2001.

[T45] S. Abdelwahed. Interacting Discrete-Event Systems: Modeling, Verification, and Supervisory Control. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, March 2002.

[T46] R.J. Leduc. Hierarchical Interface-based Supervisory Control. Ph.D. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, April 2002.