Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

H. Fierz

SCSM – Synchronous Composition of Sequential Machines

TIK-Report
Nr. 14, Juni 1994

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

H. Fierz
SCSM – Synchronous Composition of Sequential Machines
Juni 1994
Version 1
TIK-Report Nr. 14

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich

Institut für Technische Informatik und Kommunikationsnetze,
Eidgenössische Technische Hochschule Zürich

Gloriastrasse 35, ETH-Zentrum, CH-8092 Zürich, Switzerland

SCSM – Synchronous Composition of Sequential Machines

Hugo Fierz

Institut für Technische Informatik und Kommunikationsnetze
ETH-Zentrum, CH-8092 Zürich, Switzerland
e-mail: fierz@tik.ee.ethz.ch

Abstract

The SCSM formalism allows to define event driven synchronous systems by means of expressions formed by sum, product and restriction operators applied to non-deterministic sequential machines. Deterministic expressions are obtained through global product restrictions, which correlate transitions of non-deterministic components with states of other components. SCSM is usefull as denotation domain for synchronous system description languages, especially for models concerning reactive systems.

Introduction

A reactive system is a system which interacts permanently with its environment. A typical environment consists of many physical objects, thus the structure of a reactive system is expected to be highly parallel in nature. The formal semantics of a general system description language for reactive systems should therefore be based on a model of concurrency like Petri-Nets, CCS or process algebras. However, *highly parallel in nature* does not necessairely mean *highly disributed*. In the real environment of a reactive system one will always find groups of objects which have to be controlled with respect to a common local time. The control of such a local group can usefully be specified by a subsystem consisting of a set of synchronously cooperating processes. The SCSM formalism adresses the formal description of such synchronous subsystems which in the following will be called *clusters*.

A cluster generally represents a sequential subsystem of a distributed reactive system. The parallellism of a cluster's components is expressed through the respective orthogonality of the component's state spaces. An occurring event for a cluster's component leads through generated software events to transitions of other components. The occurring chain reaction is always considered as instantaneous, i.e. a temporal atomic state transition of the cluster. Instantaneous interaction mechanisms are in fact the basic ingredients of several well known synchronous system description languages: ESTEREL[1], LUSTRE[6], SIGNAL[11], STATECHARTS[13].

Many questions related to safety and reliability are much easier to treat in synchronous systems than in concurrent ones. However, the formal semantics of the cited synchronous languages is more complex and difficult to understand than expected from the evident intuitive meaning of particular system descriptions. The seemingly unneccessary complexity of the defined semantic functions is partly a consequence of the adopted traditional way of language development,

where the syntax of a language is given ahead. The semantics is then obtained by formalizing previously given informal descriptions [2], [6], [12], [14]. Most synchronous language approaches adopt an operational semantics, compiling specifications into extended automata. We propose instead to define a system directly as executable composition of sequential machines.

SCSM allows to base the definition of a synchronous language on a predefined mathematical model. SCSM expressions describe synchronous compositions of sequential machines. Starting the development of a language with a compositional mathematical model as denotation domain naturally leads to specification languages supporting a constructive system development process, c.f. [3] [4].

The origin of SCSM was the need of a formalism allowing to define a model for synchronous subsystems within the framework of CIP (Communicating Interacting Processes) [9, 10], a recently developed specification method for distributed reactive systems. CIP unifies synchronous and asynchronous cooperation of sequential machines using two different semantic models. Language expressions for local compositions are interpreted as SCSM expressions, while for language constructs describing distributed behaviour one uses Petri Nets as denotation domain.

Our Goals

As basic elements of the formalism we take non-deterministic sequential machines (output automata) because of their evident modelling power of reactive behaviour.

- The formalism must allow the description of the following interaction mechanisms:
  A component of a system can generate output which is used instantanously as input to other components. Such a transmission mechanism will be called *pulse passing*.
  In order to get deterministic compositions the execution of a non-deterministic branching of a component is allowed to depend on the states of other components. The evaluation of the corresponding conditions is called *state inspection.*
- A specific system must be defined by an expression of the formalism. Complex expressions are obtained by means of operators applied to simpler expressions. State inspection has to be described by projections on deterministic sequential machines.

Instantaneous pulse passing is the common interaction mechanism of the above mentioned synchronous languages. State inspection mechanisms have been proposed by guarded transitions in STATECHARTS, state inquiry operations in "Synchronization of Hard-Real-Time Systems" [8] and status vector inspection in Jackson System Development [5], although JSD-systems consist of asynchronously cooperating processes.

In Section 1 of this paper non-deterministic sequential machines are defined as partial functions on transition relations. Section 2 contains the definition of operators acting on sequential machines. Section 3 explains the construction of determining projectors for the

formal description of state inspection. In Section 4 we define the formal semantics of a graphically specified system example by an expression of the SCSM formalism.

1. Sequential Machines

1. Sequential Machines viewed as Partial Functions

Our definition of product operators is a generalization of the notion for products of output modules [7] or Mealy Machines [15] which are complete and deterministic sequential machines. We shall work with incomplete and non-deterministic machines. The notion of completeness and determinism of sequentiel machines is defined below.

We show briefly how Mealy Machines generalize to non-deterministic machines.

A Mealy Machine with state space Q, input alphabet B and output alphabet F is given by a next state function σ and an output function λ

$$\sigma \in Q \times B \to Q \qquad\qquad \lambda \in Q \times B \to F\ .$$

For a non-deterministic machine the next state function has to be defined as a map from prestate-input pairs to sets of poststates

$$\sigma \in Q \times B \to \mathbb{P}(Q) \qquad \text{where } \mathbb{P}(Q) \text{ denotes the powerset of Q.}$$

The output function λ is then defined as a function from the defined set T of input labeled transitions into the output alphabet F

$$\lambda \in T \to F \quad \text{where } T = \{(q, b, q') \in Q \times B \times Q \mid q' \in \sigma(q, b)\}\ .$$

The implicit dependence of the output function λ on the next state function σ seems to lead to complications in the definition of product operators for non-deterministic machines. However, in taking a more static view a simple notion for product operators is possible. We define a sequential machine as a partial function which maps the input labeled transitions into the output alphabet. Operations on sequential machines can then be defined by operations on partial functions.

NOTATION: $\xrightarrow{p}$ is used for partial functions, $\to$ for total functions.

DEFINITION Sequential Machines $\mathbb{SM}$(Q, B, F)

$\mathbb{SM}$(Q, B, F) is the class of sequential machines with state space Q, input alphabet B, and output alphabet F. The symbol sets Q, B and F are supposed to be non-empty.

A *sequential machine* $\mathcal{M} \in \mathbb{SM}$(Q, B, F) is defined as a partial function

$$\mathcal{M} \in \mathbf{Q \times B \times Q} \xrightarrow{p} \mathbf{F} \qquad \blacksquare$$

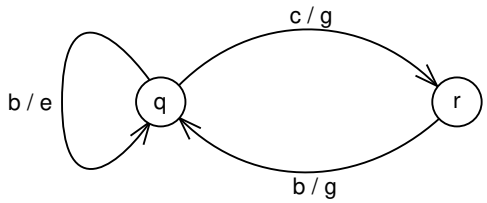
The domain $DOM(\mathcal{M}) \subset Q \times B \times Q$ represents a labeled transition system describing the dynamic behaviour of the machine. The function $\mathcal{M} \in DOM(\mathcal{M}) \to F$ defines the reaction of the machine. $\mathcal{M}(t)$ is the unique output generated by transition t.

EXAMPLE $\mathcal{M} \in \mathbb{SM}(\{q, r\}, \{b, c\}, \{e, g\})$ is defined by

$\mathrm{DOM}(\mathcal{M}) := \{(q, b, q), (q, c, r), (r, b, q)\}$

$\mathcal{M}(q, b, q) := e, \qquad \mathcal{M}(q, c, r) := g,$

$\mathcal{M}(r, b, q) := g$



A sequential machine $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ is called ***deterministic***,

iff $|\mathrm{DOM}(\mathcal{M}) \cap (q, b, Q)| \leq 1$ for all $(q, b) \in Q \times B$.

i.e. for each prestate-input pair there exists at most one transition $(q, b, q') \in \mathrm{DOM}(\mathcal{M})$.

A sequential machine $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ is called ***complete***,

iff $|(q, b, Q) \cap \mathrm{DOM}(\mathcal{M})| \geq 1$ for all $(q, b) \in Q \times B$.

i.e. for each prestate-input pair there exists at least one transition $(q, b, q') \in \mathrm{DOM}(\mathcal{M})$.

If $\mathcal{M}$ is complete and deterministic, $\mathcal{M}$ may be taken as *Mealy Machine* $\mathcal{M} \in B \times Q \to Q \times F$.

(The machine of the above example is deterministic but not complete.)

2. Operators on Sequential Machines

The operators are defined by constructive descriptions of the resulting sequential machines, although the effect of the operators can be described independently of the state transition structures of the involved machines:

-] **restriction**: the input of a machine is restricted
- $\oplus$ **sum**: joins machines with different input acting on the same state space
- $\otimes$ **parallel product**: two machines act as independent components
- • **serial product**: the output of one machine is taken as input to the other one

Partial functions

Our main mathematical tool consists of operations on partial functions. In order to fix notations we give a short summary:

A partial function $f \in A \overset{p}{\to} B$ is a functional relation $f \subset A \times B$.

$\mathrm{DOM}(f) := \{a \in A \mid \text{there exists } b \in B \text{ with } f(a) = b\}$ is called the domain of f.

For $R \subset A$, the restriction $f|_R$ is defined by

$\mathrm{DOM}(f|_R) := \mathrm{DOM}(f) \cap R$ and $f|_R(a) := f(a)$ for $a \in \mathrm{DOM}(f|_R)$.

For $\mathrm{DOM}(f) \cap \mathrm{DOM}(g) = \varnothing$ the sum $f + g$ is defined by

$\mathrm{DOM}(f + g) := \mathrm{DOM}(f) \cup \mathrm{DOM}(g)$

$(f + g)(a) := f(a)$ for $a \in \mathrm{DOM}(f)$, $(f + g)(a) := g(a)$ for $a \in \mathrm{DOM}(g)$.

A partition $A = \sum_k R_k$ leads to an expansion $f = \sum_k f|_{R_k}$. ($\sum_k R_k$ denotes disjoint union)

$f \cup g$ is defined as binary relation $f \cup g \subset A \times B$.

The functional composition $(g \circ f)(\cdot) := g(f(\cdot))$ of two partial functions is defined as composed binary relation, which is again a partial function.

2.1. Restrictions and Sums of Sequential Machines

DEFINITION Restriction $_{C]}\mathcal{M}$

Let $C \subset B$ be a subset of the input alphabet of a machine $\mathcal{M} \in \mathbb{SM}(Q, B, F)$

The *restriction* $_{C]}\mathcal{M}$ is defined as restriction of the

partial function $\mathcal{M} \in Q \times B \times Q \overset{p}{\rightarrow} F$ by

$$_{C]}\mathcal{M} := \mathcal{M}|_{(Q, C, Q)}$$ ∎

The machine $_{C]}\mathcal{M}$ is restricted to transitions triggered by input symbols of the subset C.

Presuppose two sequential machines acting on the same state space. The union of their sets of labeled transitions gives a further transition system with joined dynamic behaviour. To be able to take over the output functions of the original machines, one has to suppose that their domains do not have any common element. We even suppose disjoint input alphabets in order to rely on a condition independent of transition structures.

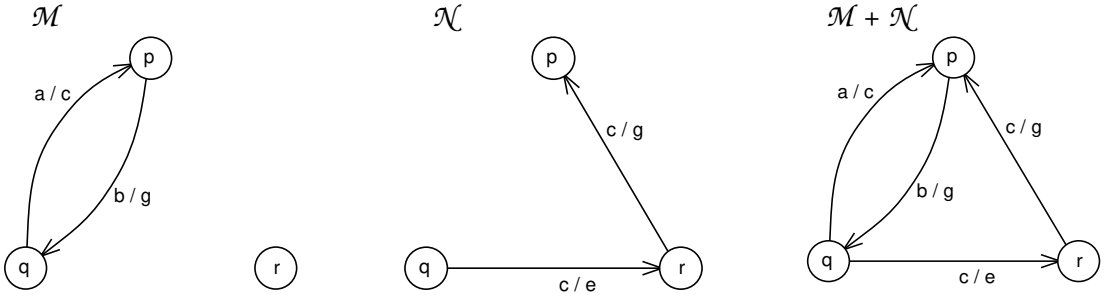
DEFINITION Sum $\mathcal{M} \oplus \mathcal{N}$

Let $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ and $\mathcal{N} \in \mathbb{SM}(Q, C, G)$ be two sequential machines acting on the same state space Q with disjoint input alphabets $B \cap C = \varnothing$.

The *sum* $\mathcal{M} \oplus \mathcal{N} \in \mathbb{SM}(Q, B \cup C, F \cup G)$ is defined as a sum of the partial functions $\mathcal{M}, \mathcal{N} \in Q \times (B \cup C) \times Q \overset{p}{\rightarrow} F \cup G$ by

$$\mathcal{M} \oplus \mathcal{N} := \mathcal{M} + \mathcal{N}$$ ∎

EXAMPLE



The sum operator is clearly commutative and associative. $\mathcal{M}$ and $\mathcal{N}$ are called parts of $\mathcal{M} \oplus \mathcal{N}$. The application of the sum operator gives a machine with an enlarged set of transitions.

Input Expansion

A partition $B = \sum_k B_k$ of the input alphabet of $\mathcal{M}$ leads to an expansion $\mathcal{M} = \bigoplus_k {}_{B_k]}\mathcal{M}$.

2.2. Products of Sequential Machines

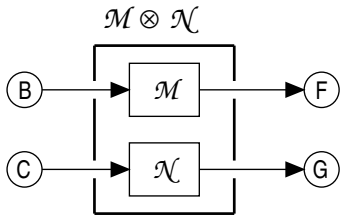
A product of two sequential machines represents a synchronous composition acting on the combined state space. We define *parallel* and *serial* products of sequential machines. Our definitions generalize the notion of products of output modules of Eilenberg[7].

DEFINITION Parallel Product $\mathcal{M} \otimes \mathcal{N}$

Let $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ and $\mathcal{N} \in \mathbb{SM}(R, C, G)$ be two sequential machines.
The *parallel product* $\mathcal{L} = \mathcal{M} \otimes \mathcal{N} = \mathcal{N} \otimes \mathcal{M}$ *acting on* $Q \times R$
is a machine of $\mathbb{SM}(Q \times R, B \times C, F \times G)$:
$\mathcal{L} \in (Q \times R) \times (B \times C) \times (Q \times R) \xrightarrow{P} F \times G$ is defined by

$$\mathrm{DOM}(\mathcal{L}) := \{((q, r), (b, c), (q', r')) \mid (q, b, q') \in \mathrm{DOM}(\mathcal{M}), (r, c, r') \in \mathrm{DOM}(\mathcal{N})\},$$

$$\mathcal{L}((q, r), (b, c), (q', r')) := (\mathcal{M}(q, b, q'), \mathcal{N}(r, c, r'))$$ ∎



In a transition of a parallel product (often called direct product) each component accepts an input symbol and generates an output symbol. There is no functional dependence between the components of a parallel product.

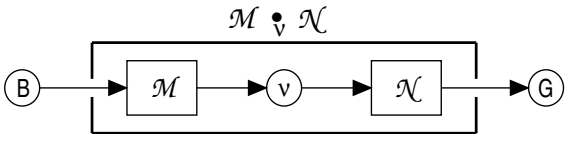
On the other hand, a serial product of two sequential machines represents a coupling, where the output of one machine is taken as input to the other one. The serial product is defined as functional composition of partial functions. The definition relies on a translation function relating producer output to consumer input.

DEFINITION Serial Product $\mathcal{M} \overset{\nu}{\bullet} \mathcal{N}$

Let $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ and $\mathcal{N} \in \mathbb{SM}(R, C, G)$ be two sequential machines.
The output alphabet F of $\mathcal{M}$ is translated into the input alphabet C of $\mathcal{N}$ by a presupposed translation function $\nu \in F \to C$.
The *serial product* $\mathcal{L} = \mathcal{M} \overset{\nu}{\bullet} \mathcal{N}$ *acting on* $Q \times R$
is a machine of $\mathbb{SM}(Q \times R, B, G)$:
$\mathcal{L} \in (Q \times R) \times B \times (Q \times R) \xrightarrow{P} G$ is defined by

$$\mathrm{DOM}(\mathcal{L}) := \{((q, r), b, (q', r')) \mid (q, b, q') \in \mathrm{DOM}(\mathcal{M}) \text{ and } (r, \nu(\mathcal{M}(q, b, q')), r') \in \mathrm{DOM}(\mathcal{N})\},$$

$$\mathcal{L}((q, r), b, (q', r')) := \mathcal{N}(r, \nu(\mathcal{M}(q, b, q')), r')$$ ∎



In the serial composition $\mathcal{M} \overset{\nu}{\bullet} \mathcal{N}$, the chained machine $\mathcal{N}$ is functionally dependent on $\mathcal{M}$. The serial product is clearly not commutative.

NOTATION

When ν is understood the serial product is simply denoted by $\mathcal{M} \bullet \mathcal{N}$.
The defined operators have precedence in the following order from left to right: $]$, $\bullet$, $\otimes$, $\oplus$

Laws

The following product laws can be verified.

associativity

$(\mathcal{L} \otimes \mathcal{M}) \otimes \mathcal{N} = \mathcal{L} \otimes (\mathcal{M} \otimes \mathcal{N})$ $\qquad$ $(\mathcal{L} \overset{\bullet}{\nu} \mathcal{M}) \overset{\bullet}{\mu} \mathcal{N} = \mathcal{L} \overset{\bullet}{\nu} (\mathcal{M} \overset{\bullet}{\mu} \mathcal{N})$

distributivity

$(\mathcal{M} \oplus \mathcal{N}) \otimes \mathcal{L} = (\mathcal{M} \otimes \mathcal{L}) \oplus (\mathcal{N} \otimes \mathcal{L}) = (\mathcal{L} \otimes \mathcal{M}) \oplus (\mathcal{L} \otimes \mathcal{N}) = \mathcal{L} \otimes (\mathcal{M} \oplus \mathcal{N})$

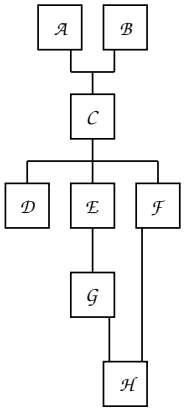
$(\mathcal{M} \oplus \mathcal{N}) \overset{\bullet}{\nu} \mathcal{L} = (\mathcal{M} \overset{\bullet}{\nu} \mathcal{L}) \oplus (\mathcal{N} \overset{\bullet}{\nu} \mathcal{M})$ (only distributivity to the right)

parallel chaining

$(\mathcal{M}_1 \overset{\bullet}{\nu} \mathcal{N}_1) \otimes (\mathcal{M}_2 \overset{\bullet}{\mu} \mathcal{N}_2) = (\mathcal{M}_1 \otimes \mathcal{M}_2) \overset{\bullet}{(\nu,\mu)} (\mathcal{N}_1 \otimes \mathcal{N}_2)$

Example of a mixed product expression

$$(\mathcal{A} \otimes \mathcal{B}) \bullet \mathcal{C} \bullet (\mathcal{D} \otimes (\mathcal{E} \bullet \mathcal{G} \otimes \mathcal{F}) \bullet \mathcal{H})$$



The functional dependence between the various components defines a partial order on the set of components. The corresponding graphical representation is usually interpreted as data flow diagram.

Examples of expressions involving also restriction and sum operators are contained in the specification example of Section 4 of the paper.

3. Determining Projectors

On one hand, serial products describe interaction through *pulse transmission*. On the other hand, *state inspection* allows to specify dependence between synchronous components through a kind of shared variable coupling:

The choice of a non-deterministic component's transition can
depend on the actual prestate of other components of the composition.

State inspection represents a far weaker coupling between components than pulse passing, because inspected components are not influenced by the inspection mechanism. We shall describe state inspection formally by restrictions of expanded products. The product restrictions lead to correlations between non-deterministic transitions and prestates of complementary components. We define such restrictions by means of *determining projectors* which do not destroy any completeness property of the involved sequential machines.

EXAMPLE

$\mathcal{M} \in \mathbb{SM}(\{x, y, z\}, \{a, b\}, \{e, f\})$:

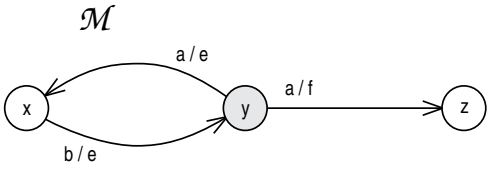
DOM($\mathcal{M}$) := $\{m_1, m_2, m_3\}$ where

$m_1 = (y, a, x)$, $m_2 = (y, a, z)$,

$m_3 = (x, b, y)$

$\mathcal{M}(m_1) := e$, $\mathcal{M}(m_2) := f$, $\mathcal{M}(m_3) := e$

$\mathcal{M}$



$\mathcal{M}$ is *non-deterministic* :

The transitions m1 and m2 have prestate y and are triggerd by the same input a.

$\mathcal{N} \in \mathbb{SM}(\{u, v\}, \{c, d\}, \{g, h\})$:

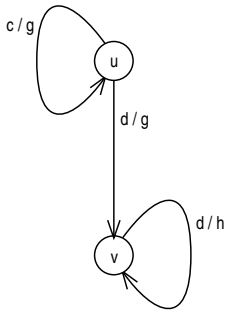
DOM($\mathcal{N}$) := $\{n_1, n_2, n_3\}$ where

$n_1 = (u, c, u)$, $n_2 = (u, d, v)$, $n_3 = (v, d, v)$

$\mathcal{N}(n_1) := g$, $\mathcal{N}(n_2) := g$, $\mathcal{N}(n_3) := h$

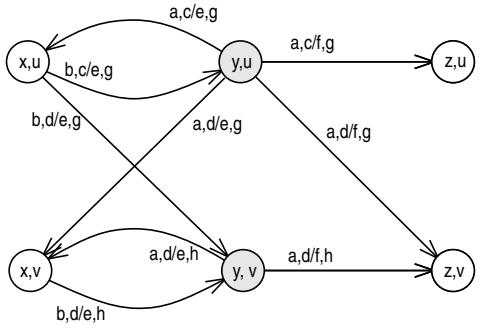
$\mathcal{N}$ is *deterministic*.

$\mathcal{N}$



Let us consider the parallel product $\mathcal{L} = \mathcal{M} \otimes \mathcal{N}$:

$\mathcal{L} = \mathcal{M} \otimes \mathcal{N}$



$\mathcal{L}$ is *non-deterministic*

for input symbol (a, c) in state (y, u),

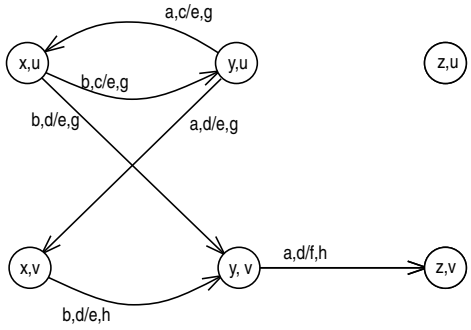
for input symbol (a, d) in state (y, u) and

for input symbol (a, d) in state (y, v).

Deterministic behaviour of $\mathcal{L}$ can be specified through additional branching conditions. We want to make the execution of the non-deterministic transitions of the component $\mathcal{M}$ dependent of the actual state of the component $\mathcal{N}$:

m_1 is enabled, if $\mathcal{N}$ is in state u m_2 is enabled, if $\mathcal{N}$ is in state v

$\mathcal{L}_{det}$



The restricted product $\mathcal{L}_{det}$ satisfies the additional branching conditions.

$\mathcal{L}_{det}$ is *deterministic*.

State inspection is formally described by restrictions of the partial functions describing the involved machines. The restriction of a partial function f to A is going to be denoted by $f|_A$, while is f+g used to denote the sum of f and g. Restrictions and sums of partial functions are more general operations than the defined restriction] and sum $\oplus$ of sequential machines.

The formal description of the determining restriction of the given graphical example is obtained as follows:

We first expand the partial function defining $\mathcal{M}$ into its two *branchings*

$$\mathcal{L} = (\mathcal{M}|_{B_y} + \mathcal{M}|_{B_x}) \otimes \mathcal{N} = \mathcal{M}|_{B_y} \otimes \mathcal{N} + \mathcal{M}|_{B_x} \otimes \mathcal{N}$$

where $B_y = \{m_1, m_2\}$ and $B_x = \{m_3\}$

The first term only is non-deterministic.

An expansion of $\mathcal{N}$ into two terms restricted to transitions with prestate u resp. v gives

$$\mathcal{L} = \mathcal{M}|_{B_y} \otimes \mathcal{N}|_{\{n_1,n_2\}} + \mathcal{M}|_{B_y} \otimes \mathcal{N}|_{\{n_3\}} + \mathcal{M}|_{B_x} \otimes \mathcal{N}$$

The desired deterministic machine is now obtained by appropriately restricting the terms containing the non-deterministic branching $\mathcal{M}|_{B_y}$

$$\mathcal{L}_{det} := (\mathcal{M}|_{B_y})|_{\{m_1\}} \otimes \mathcal{N}|_{\{n_1,n_2\}} + (\mathcal{M}|_{B_y})|_{\{m_2\}} \otimes \mathcal{N}|_{\{n_3\}} + \mathcal{M}|_{B_x} \otimes \mathcal{N}$$

$$= \mathcal{M}|_{\{m_1\}} \otimes \mathcal{N}|_{\{n_1,n_2\}} + \mathcal{M}|_{\{m_2\}} \otimes \mathcal{N}|_{\{n_3\}} + \mathcal{M}|_{B_x} \otimes \mathcal{N}$$

3.1. Branchings

The non-deterministic behaviour of a machine $\mathcal{M} \in \mathbb{SM}(B, Q, F)$ is analyzed by expanding the partial function $\mathcal{M} \in Q \times B \times Q \xrightarrow{p} F$ in its *branchings*:

The set of *valid* prestate-input pairs of $\mathcal{M}$ is given by

$$V_{\mathcal{M}} = \{(q, b) \in Q \times B \mid (q, b, Q) \cap \mathrm{DOM}(\mathcal{M}) \neq \emptyset\}.$$

For $(q, b) \in V_{\mathcal{M}}$, the restriction $\mathcal{M}|_{(q,b,Q)}$ is called a *branching* of $\mathcal{M}$ and will be denoted by $\mathcal{M}|_{(q,b)}$.

The branching $\mathcal{M}|_{(q,b)}$ represents the part of $\mathcal{M}$ which accepts input b in state q.
If $\mathcal{M}$ is deterministic, then $|\mathrm{DOM}(\mathcal{M}|_{(q,b)})| = 1$ for all $(q, b) \in V_{\mathcal{M}}$.
$\mathcal{M}$ is called non-deterministic in (q, b), iff $|\mathrm{DOM}(\mathcal{M}|_{(q,b)})| > 1$ (non singular branching).

$\mathcal{M}$ can be expanded in its branchings $\mathcal{M} = \sum_{(q,b) \in V_{\mathcal{M}}} \mathcal{M}|_{(q,b)}$.

To turn $\mathcal{M}$ into a deterministic machine, conditions have to be associated with all its non singular branchings. We allow conditions which are dependent on the states of other product components, as illustrated in the above example.

3.2. Determining Projectors

Let $\mathcal{M}_1, ..., \mathcal{M}_N$ be a finite family of sequential machines $\mathcal{M}_k \in \mathbb{SM}(Q_k, B_k, F_k)$. The corresponding state spaces Q_k, output alphabets B_k and output alphabets F_k are supposed to be mutually disjoint.

A *composition* of $\mathcal{M}_1, ..., \mathcal{M}_N$ is a sequential machine acting on the combined state space $Q_1 \times ... \times Q_N$, described by an expression involving the SCSM-operators], $\oplus$, $\otimes$ and $\bullet$.

A composition $\Theta(\mathcal{M}_1, ... \mathcal{M}_N)$ may be taken as a machine of the form $\Theta \in \mathbb{SM}(Q, B_\Theta, F_\Theta)$, where

$Q = Q_1 \times ... \times Q_N$ is the combined state space of all components,

$B_\Theta \subset B_1 \cup B_2 \cup ... \cup B_1 \times B_2 \cup B_1 \times B_3 \cup ... \cup B_1 \times ... \times B_N$ is the effective input alphabet and $F_\Theta \subset F_1 \cup F_2 \cup ... \cup F_1 \times F_2 \cup F_1 \times F_3 \cup ...$ is the effective output alphabet.

EXAMPLE

$$\Theta(\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3) = \mathcal{M}_1 \bullet (\mathcal{M}_2 \otimes \mathcal{M}_3) \quad \oplus \quad \mathcal{M}_3 \bullet (\mathcal{M}_2 \bullet \mathcal{M}_1 \oplus \mathcal{M}_1 \bullet \mathcal{M}_2)$$

Θ is effectively an element of $\mathbb{SM}(Q_1 \times Q_2 \times Q_3, B_1 \cup B_3, F_2 \times F_3 \cup F_1 \cup F_2)$.

In order to describe state inspection independently of product structures, we introduce *determining projectors* acting on compositions of $\mathcal{M}_1, ..., \mathcal{M}_N$. The definition of a determining projector for component $\mathcal{M}_r$ relies on an *enabling relation* ω_r, which is given by a set of *enabling conditions* for the non-deterministic transitions of $\mathcal{M}_r$.

An enabling condition for a transition of a branching of $\mathcal{M}_r$ is specified by a subset of states of the complementary state space $Q_r^\perp = Q_1 \times ... \times Q_{r-1} \times Q_{r+1} \times ... \times Q_N$, namely the states which enable the transition. The sets of enabling states associated with a branching are supposed to form a partition of $Q_r^\perp$. The mutual disjointness is necessairy for the unique resolution of the non-determinism, while the covering property of the partition ensures the conservation of the machine's completeness properties. The enabling conditions of a branching can therefore be defined through a *switch function* which maps the complementary state space $Q_r^\perp$ into the set of transitions of the branching. For formal reasons we define the switch function on the complete product state space Q. Correspondingly, the switch functions associated with component $\mathcal{M}_r$ have to be supposed invariant on Q_r.

DEFINITION Enabling Relation ω_r

An *enabling relation* for component $\mathcal{M}_r$ is a relation $\omega_r \subset Q \times \mathrm{DOM}(\mathcal{M}_r)$ given as union of switch functions $\omega_{r,(s,b)}$ associated with the branchings of $\mathcal{M}$:

$$\omega_r := \bigcup_{(s,b) \in V_{\mathcal{M}_r}} \omega_{r,(s,b)} \ ,$$

where $\omega_{r,(s,b)} \in Q \to \mathrm{DOM}(\mathcal{M}_r|_{(s,b)})$

with $\omega_{r,(s,b)}(q_1, ..., q_r, ...) = \omega_{r,(s,b)}(q_1, ..., q_r', ...)$ for all $q_r, q_r' \in Q_r$ ∎

The union of the switch functions is defined as union of the corresponding binary relations. The union runs over the set $V_{\mathcal{M}_r}$ of the valid prestate-input pairs of $\mathcal{M}_r$.
$\omega_{r,(s,b)}$ represents the switch function associated with branching $\mathcal{M}_r|_{(s,b)}$.

We shall use the enabling relation $\omega_r \subset Q \times \mathrm{DOM}(\mathcal{M}_r)$ as a function

$\omega_r \in Q \to \mathbb{P}(\mathrm{DOM}(\mathcal{M}_r))$, where $\mathbb{P}(\mathrm{DOM}(\mathcal{M}_r))$ is the set of subsets of transitions of $\mathcal{M}_r$.

The set of transitions enabled by q is then given by $\omega_r(q)$.

PROPOSITION 1

For fixed $q \in Q$,
the partial function restriction $\mathcal{M}_r|_{\omega_r(q)}$ is a deterministic sequential machine.

Proof:
The set of $\mathcal{M}_r$-transitions enabled by product state $q \in Q$ is given by a disjoint union of singular transitions, i.e. the transitions enabled by the corresponding switch functions.

$$\omega_r(q) = \bigcup_{(s,b) \in V_{\mathcal{M}_r}} \omega_{r,(s,b)}(q) \ .$$

and therefore $$\mathcal{M}_r|_{\omega_r(q)} = \sum_{(s,b) \in V_{\mathcal{M}_r}} \mathcal{M}_r|_{\omega_{r,(q_r,b)}(q)}$$

which shows that $\mathcal{M}_r|_{\omega_r(q)}$ is a sum partial functions, each representing sequential machine consisting of one transition only, and each with a different prestate-input pair. ∎

Let $\mathbb{C}(\mathcal{M}_1, ..., \mathcal{M}_N)$ be the class of sequential machines which contains all compositions of $\mathcal{M}_1, ..., \mathcal{M}_N$, and which is extended to the sequential machines obtained by the application of the partial function operators | and +.

In order to obtain deterministic behaviour of a component $\mathcal{M}_r$ of a composition of $\mathcal{M}_1, ..., \mathcal{M}_N$, the composition is expanded with respect to its prestates. The component $\mathcal{M}_r$ appearing in the individual terms of the expansion have then to be restricted appropriately:

DEFINITION Determining Projector Δ_{ω_r}

The *determining projector*

$$\Delta_{\omega_r} \in \mathbb{C}(\mathcal{M}_1, ..., \mathcal{M}_N) \to \mathbb{C}(\mathcal{M}_1, ..., \mathcal{M}_N)$$

associated with the enabling relation $\omega_r \subset Q \times \mathrm{DOM}(\mathcal{M}_r)$ is defined by

$$\Delta_{\omega_r}\Theta := \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_r|_{\omega_r(q)}, ..., \mathcal{M}_N)\Big|_{(q,\, B_\Theta,\, Q)}$$

where $\Theta \in \mathbb{C}(\mathcal{M}_1, ..., \mathcal{M}_N)$ is supposed to be a composition of $\mathcal{M}_1, ..., \mathcal{M}_N$. ∎

The effect of Δ_{ω_r} applied to Θ is to restrict component $\mathcal{M}_r$ deterministically in every term of the state expansion $\Theta = \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_r, ..., \mathcal{M}_N)\big|_{(q,\, B_\Theta,\, Q)}$:

If in a product transition with prestate $q = (q_1, ..., q_r, ..., q_N)$ input b to component $\mathcal{M}_r$ occurs, then $\omega_{r,(q_r,b)}(q)$ is the only enabled transition of $\mathcal{M}_r$. The behaviour of the complementary components $\mathcal{M}_1, ..., \mathcal{M}_{r-1}, \mathcal{M}_{r+1}, ..., \mathcal{M}_N$ is not influenced by the determining projector Δ_{ω_r}.

In order to get full deterministic behaviour of $\Theta \in \mathbb{C}(\mathcal{M}_1, ... \mathcal{M}_N)$, several determining projectors have to be applied to Θ, namely one for each non-deterministic component.

Determining projectors show the commutation property of orthogonal projection operators.

PROPOSITION 2

$$\Delta_{\omega_r}\Delta_{\omega_s}\Theta = \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_r|_{\omega_r(q)}, ..., \mathcal{M}_s|_{\omega_s(q)}, ..., \mathcal{M}_N)\Big|_{(q,\, B_\Theta,\, Q)}$$

$$\Delta_{\omega_r}\Delta_{\omega_s}\Theta = \Delta_{\omega_s}\Delta_{\omega_r}\Theta \qquad \Delta_{\omega_r}\Delta_{\omega_r}\Theta = \Delta_{\omega_r}\Theta$$

Proof:

The description of combined restrictions of a partial function as a restriction to the intersection of the restriction domains allows to reduce the combined restricted expansion:

$$\begin{aligned}
\Delta_{\omega_r}\Delta_{\omega_s}\Theta &= \Delta_{\omega_r} \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_s|_{\omega_s(q)}, ..., \mathcal{M}_N)\Big|_{(q,\, B_\Theta,\, Q)} \\
&= \sum_{q' \in Q} \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_r|_{\omega_r(q')}, ..., \mathcal{M}_s|_{\omega_s(q)}, ..., \mathcal{M}_N)\Big|_{(q',\, B_\Theta,\, Q)\ \cap\ (q,\, B_\Theta,\, Q)} \\
&= \sum_{q \in Q} \Theta(\mathcal{M}_1, ..., \mathcal{M}_r|_{\omega_r(q)}, ..., \mathcal{M}_s|_{\omega_s(q)}, ..., \mathcal{M}_N)\Big|_{(q,\, B_\Theta,\, Q)}
\end{aligned}$$

The verification of the commutation property and of the idempotence of determining projectors is straightforward. ∎

The following theorem states that a complete determining restriction of a composition is deterministic, and that completeness properties of its components are conserved.

THEOREM 3

Let $\mathcal{M}_1, ..., \mathcal{M}_N$ be a finite family of sequential machines $\mathcal{M}_k \in \mathbb{SM}(Q_k, B_k, F_k)$ with an associated set of enabling relations $\omega_1, ..., \omega_N$

where $\omega_k \subset Q \times \text{DOM}(\mathcal{M}_k)$.

For a composition $\Theta \in \mathbb{C}(\mathcal{M}_1, ..., \mathcal{M}_N)$ the projection $\Delta\omega_1 \ldots \Delta\omega_N \Theta$ is a deterministic sequential machine.

If all components $\mathcal{M}_k$ are complete, then $\Delta\omega_1 \ldots \Delta\omega_N \Theta$ is still a complete sequential machine.

Proof:

Let us fix a product prestate $q \in Q$ and presuppose that $\Theta \in \mathbb{SM}(Q, B_\Theta, F_\Theta)$.

PROPOSITION 2 is extended straightforwardly to multiple products of determining projectors:

$$\Delta\omega_1 \ldots \Delta\omega_N \Theta\big|_{(q,\, B_\Theta,\, Q)} = \Theta\big(\mathcal{M}_1|_{\omega_1(q)}, ..., \mathcal{M}_N|_{\omega_N(q)}\big)\big|_{(q,\, B_\Theta,\, Q)}$$

$$= \Theta\big(\mathcal{M}_1|_{\omega_1(q)}, ..., \mathcal{M}_N|_{\omega_N(q)}\big)$$

which due to PROPOSITION 1 is a composition of deterministic machines.

Because sums and products of deterministic sequential machines are deterministic $\Delta\omega_1 \ldots \Delta\omega_N \Theta\big|_{(q,\, B_\Theta,\, Q)}$ is also deterministic.

Thus $\big|\text{DOM}(\Delta\omega_1 \ldots \Delta\omega_N \Theta) \cap (q, e, Q)\big| \leq 1$ for all $(q, e) \in Q \times B_\Theta$ which shows that $\Delta\omega_1 \ldots \Delta\omega_N \Theta$ is deterministic.

For a complete component $\mathcal{M}_r \in \mathbb{SM}(Q_r, B_r, F_r)$, every input $b_r \in B_r$ is valid in every state of the machine. This holds also for the restricted machine $\mathcal{M}_r|_{\omega_r(q)}$ due to the completeness of the involved switch conditions.

Thus $\mathcal{M}_r|_{\omega_r(q)}$ can be represented as a total function $\mathcal{M}_r|_{\omega_r(q)} \in \{q_r\} \times B_r \to Q_r \times F_r$.

If all components of the composition are complete, it follows that the product restriction

$$\Delta\omega_1 \ldots \Delta\omega_N \Theta\big|_{(q,\, B_\Theta,\, Q)} = \Theta\big(\mathcal{M}_1|_{\omega_1(q)}, ..., \mathcal{M}_N|_{\omega_N(q)}\big)$$

is also a total function, namely

$$\Delta\omega_1 \ldots \Delta\omega_N \Theta\big|_{(B_\Theta,\, q,\, Q)} \in \{q\} \times B_\Theta \to Q \times F_\Theta \quad \text{for all } q \in Q.$$

Thus $\big|\text{DOM}(\Delta\omega_1 \ldots \Delta\omega_N \Theta) \cap (q, e, Q)\big| = 1$ for all $(q, e) \in Q \times B_\Theta$ and hence $\Delta\omega_1 \ldots \Delta\omega_N \Theta$ is complete. ∎

4. Example: DoorControlSystem

We define a small reactive system by a SCSM expression. The example indicates how a graphical system description language could be designed with SCSM expressions as denotation domain.

In the construction of the expression we have been led by the development concepts of the CIP method mentioned in the introduction. We shall start with a graphical system description borrowing from CIP some graphical language constructs. In order to get a corresponding mathematical expression, some additional technical notions have to be introduced such as null symbols, identity machines and auxiliary translation functions.

Environment Description

A door can be opened and closed by a motor which is controllable by digital commands. A sensor is activated when the door gets fully open, another one when it gets fully closed.
A button can be pushed and released.
A timer device can be set and reset. The timer responds three seconds after the last setting with a "time is up" signal.
A lamp can be turned on at a medium or at a high level, or it can be turned off.

Requirement Description

A button push must open the door. The door must stay fully open for three seconds. However, when the button remains pushed or when it is pushed again, the door must stay fully open until three seconds have passed after the release of the button.

The lamp must indicate the button and the door status. When the button is pushed, the lamp must light at the high level, when the button is released it must light at the medium level. When the door is fully closed the lamp must be turned off.

System Description with CIP Processes

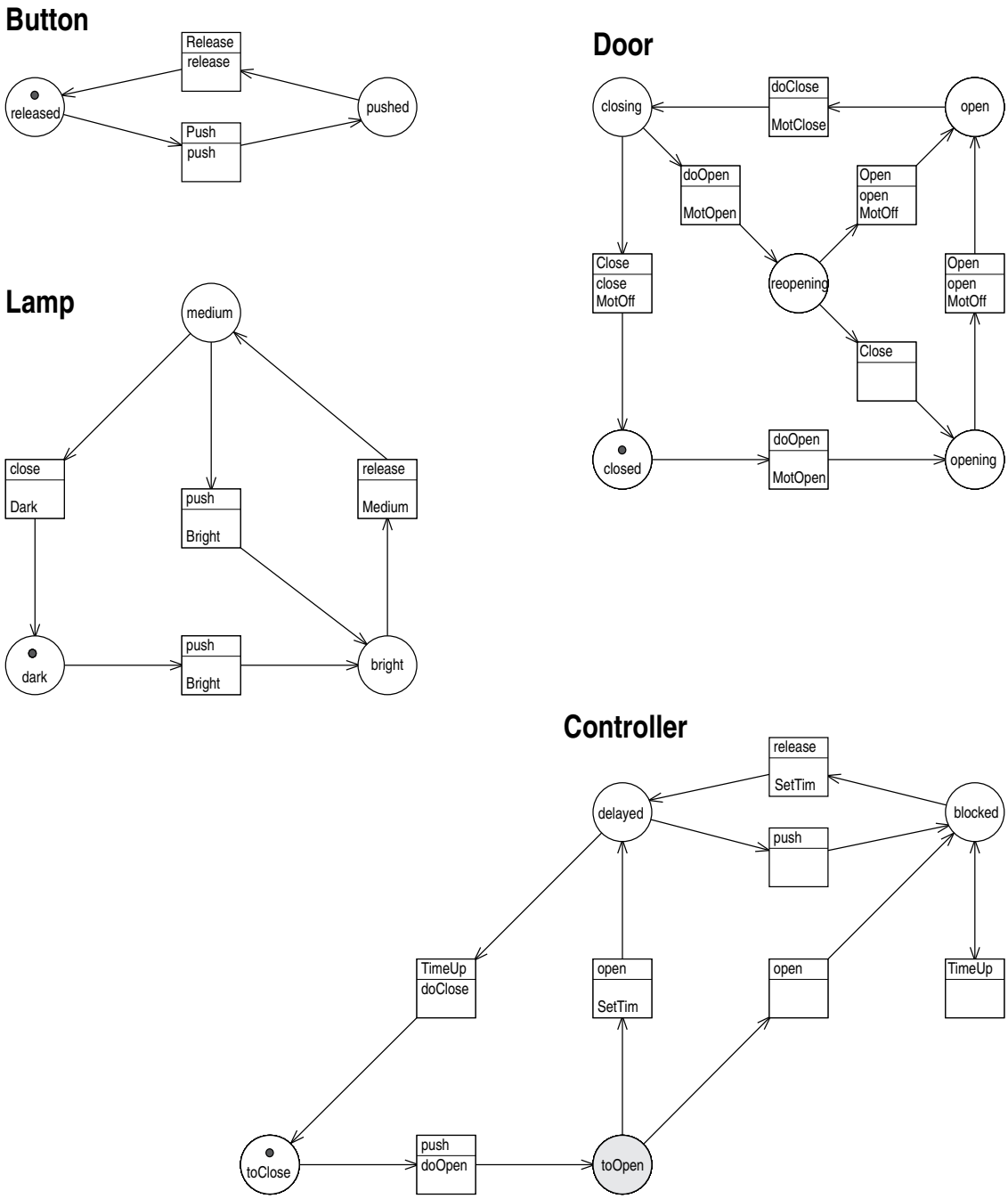
The components of our system are called processes. A process interface distinguishes clearly between *event* symbols used to connect the process to the environment and *pulse* symbols used for interaction with other processes.

A *process* corresponds to a sequential machine of the form $\mathcal{P} \in \mathbb{SM}(P, E+I, O \times A)$

The *Events* **E** represent input from the environment,
the *Inpulses* **I** denote input from other system components,
The *Outpulses* **O** represent output for other system components,
the *Actions* **A** are used as output to the environment.

The various alphabets are supposed to contain null symbols $\mathbb{n} \in I$, $\mathbb{n} \in O$ and $\mathbb{n} \in A$. A null input to a process leads always to a trivial transition and a null output.

The process structures of the DoorControlSystem are given by the following state transition diagrams:



Glossary

As a mnemonic help, event and action names begin with an uppercase, pulse names with a lowercase. The transmission of pulses is defined by serial process couplings graphically described by the cascades given below.

The processes have been found by following the real *world modeling* approach of the CIP development method. **Button**, **Door** and **Lamp** are model processes reflecting the behaviour of the corresponding objects of the system environment. An *event* signalizes a state transition of such an object while an *action* generates a state transition of the corresponding object:

Button corresponds to the active object button with events "Pushed" and "Released".

Door is an image of the reactive object door with events "Open" and "Closed", and actions "MotOpen", "MotClose" and "MotOff".

Lamp corresponds to the passive lamp with actions "Medium", "Bright" and "Dark".

The model processes are controlled by the function process **Controller**. The Controller sets and resets the timer using the "SetTim" action and consumes occurring "TimeUp" events.

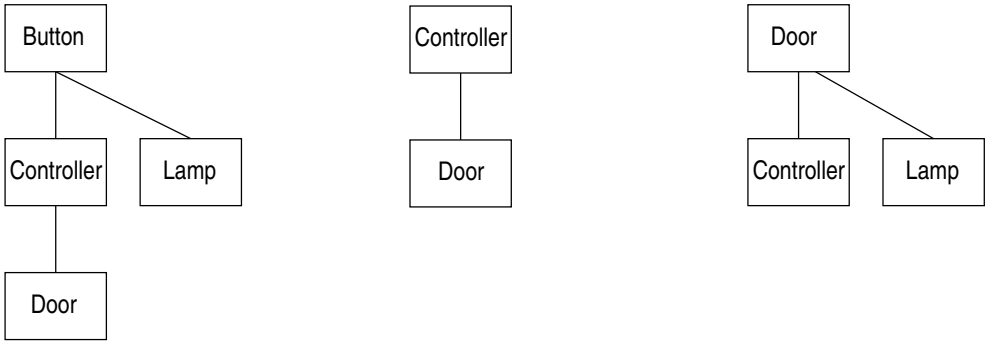
Pulse Transmission

The specified processes are still independent. Interaction by pulse transmission is going to be defined by serial process couplings. For the non-deterministic Controller process, we shall give enabling conditions depending on the Button state.

The system is supposed to be sequentially driven by singular process events (interleaving of simultaneous events). An activated process can activate further processes and thus cause several actions for the environment.

Every process with a non-empty event set can receive external input and activate the complementary components. The system is thus expected to consist of three compositions, each one activable by a different event triggered process. The process compositions are described graphically by *cascades* viewing the pulse flow structure of the composed components. The translation of a transmitted outpulse into a receiver inpulse is specified implicitly by pulse name identification, i.e. we have named each outpulse like the inpulse it has to be translated into. The translation functions are given explicitely in the formal description below.

Cascades of the DoorControlSystem



The description of the system by cascades corresponds to an "SCSM expression" like

Button • (Controller • Door ⊗ Lamp)
⊕ Controller • Door
⊕ Door • (Controller ⊗ Lamp)

Although the meaning of the "expression" is clear, it is of course not consistent due to different state space dimensions of the various terms. A consistent expression will be given in the formal system description below.

State Inspection

The process Controller is non-deterministic due to the branching for inpulse "open" in state "toOpen" (shadowed state in the diagram).

The process is rendered deterministic through inspection of the state of process Button:

If the Button is "pushed", the Controller has to branch to "blocked",

else it has to branch to "delayed".

Formal System Description

In order to get an internally complete composition we first complete our processes with respect to their inpulses. The processes need not to be completed with respect to events in order to allow the recognition of invalid events (exception handling).

For a general sequential machine $\mathcal{M} \in \mathbb{SM}(Q, B, F)$ with a null output symbol $\mathbb{n} \in F$ the *completion* $\mathcal{M}^c \in \mathbb{SM}(Q, B, F)$ is defined by

$\mathcal{M}^c|_{DOM(\mathcal{M})} := \mathcal{M}$

If $(q, b, Q) \cap DOM(\mathcal{M}) = \varnothing$ set $(q, b, q) \in DOM(\mathcal{M}^c)$ and define $\mathcal{M}^c(q, b, q) := \mathbb{n}$.

The graphically specified processes lead to the definition of four sequential machines $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{L}$ with associated initial states b_o, c_o, d_o and l_o. Pulse and action alphabets are extended by a null symbol $\mathbb{n}$.

$\mathcal{B} := {}_{E_B}]\textbf{Button} + ({}_{I_B}]\textbf{Button})^c \in \mathbb{SM}(B, E_B+I_B, O_B \times A_B)$ where

$B = \{\text{released, pushed}\}$, $b_o = \text{released}$,

$E_B = \{\text{Push, Release}\}$, $A_B = \{\mathbb{n}\}$,

$I_B = \{\mathbb{n}\}$, $O_B = \{\text{push, release, } \mathbb{n}\}$.

$\mathcal{C} := {}_{E_C}]\textbf{Controller} + ({}_{I_C}]\textbf{Controller})^c \in \mathbb{SM}(C, E_C+I_C, O_C \times A_C)$ where

$C = \{\text{toClose, toOpen, delayed, blocked}\}$, $c_o = \text{toClose}$,

$E_C = \{\text{TimeUp}\}$, $A_C = \{\text{SetTim, } \mathbb{n}\}$,

$I_C = \{\text{push, release, open, } \mathbb{n}\}$, $O_C = \{\text{doOpen, doClose, } \mathbb{n}\}$.

$\mathcal{D} := {}_{E_D}]\textbf{Door} + ({}_{I_D}]\textbf{Door})^c \in \mathbb{SM}(D, E_D+I_D, O_D \times A_D)$ where

$D = \{\text{closed, opening, open, closing, reopening}\}$, $d_o = \text{closed}$,

$E_D = \{\text{Open, Close}\}$, $A_D = \{\text{MotOpen, MotClose, MotOff, } \mathbb{n}\}$,

$I_D = \{\text{doOpen, doClose, } \mathbb{n}\}$, $O_D = \{\text{open, close, } \mathbb{n}\}$.

$\mathcal{L} := ({}_{I_L}]\textbf{Lamp})^c \in \mathbb{SM}(L, E_L+I_L, O_L \times A_L)$ where

$L = \{\text{dark, medium, bright}\}$, $l_o = \text{dark}$,

$E_L = \varnothing$, $A_L = \{\text{Medium, Bright, Dark, } \mathbb{n}\}$,

$I_L = \{\text{push, release, close, } \mathbb{n}\}$, $O_L = \{\mathbb{n}\}$.

If in a state transition of the graphical description no outpulse, resp. no action is specified, it is defined to be the null output $\mathbb{n}$.

The effective system input is then given by the disjoint union of all event sets. The effective system output consists of tuples of process actions, thus the system is of the form

$\mathcal{S} \in \mathbb{SM}(B\times C\times D\times L,\ E_B+E_C+E_D,\ A_B\times A_C\times A_D\times A_L)$

i.e. a partial function $\mathcal{S} \in B\times C\times D\times L\times(E_B+E_C+E_D)\times B\times C\times D\times L \overset{p}{\to} A_B\times A_C\times A_D\times A_L$

From the graphical cascade specification one expects an "expression" of the form

$${}_{E_B}]\mathcal{B}\bullet(\mathcal{C}\bullet\mathcal{D}\otimes\mathcal{L}) \ \oplus\ {}_{E_C}]\mathcal{C}\bullet\mathcal{D} \ \oplus\ {}_{E_D}]\mathcal{D}\bullet(\mathcal{C}\otimes\mathcal{L})$$

In order to be able to form a consistent sum of cascades we introduce identity machines and auxiliary input and output translation functions.

Identitiy Machines

The identity machine $\mathbb{1}_G \in \mathbb{SM}(\{q\}, G, G)$ on the alphabet G is defined by

$DOM(\mathbb{1}_G) := (q, G, q)$ and $\mathbb{1}_G((q, g, q)) := g$ for all $g \in G$.

q is an invariant dummy state.

We introduce identity machines $\mathbb{1}_{A_B}$, $\mathbb{1}_{A_C}$ and $\mathbb{1}_{A_D}$ on the action alphabets A_B, A_C and A_D.

Auxiliary input translations are trivial extensions of the identity map:

$\delta_C \in E_C \to E_C\times\{\mathbb{n}\}\times\{\mathbb{n}\}$, $\delta_D \in E_D \to E_D\times\{\mathbb{n}\}$ generate a null input to inactiv processes.

Auxiliary output translations project a process output on its action alphabet:

$\tau_B \in O_B\times A_B \to A_B$, $\tau_C \in O_C\times A_C \to A_C$, $\tau_C \in O_D\times A_D \to A_D$, $\tau_L \in O_L\times A_L \to A_L$

They are used to absorb outpulses of processes at the bottom of a cascade.

In the following expression for the DoorControlSystem, the auxiliary input, resp. output translations are simply written before, resp. after, the corresponding sequential machine. The trivial extensions of the product state space due to the dummy states of the used identity machines are omitted in the given expression.

The non-deterministic system is defined by

$$\mathcal{S} = {}_{E_B}]\mathcal{B}\bullet\Big(\big({}_{I_C}]\mathcal{C}\bullet({}_{I_D}]\mathcal{D}\tau_D\otimes\mathbb{1}_{A_C}\big)\otimes{}_{I_L}]\mathcal{L}\tau_L\Big)\otimes\mathbb{1}_{A_B}\Big) \oplus$$
$$\delta_C\Big({}_{E_C}]\mathcal{C}\bullet({}_{I_D}]\mathcal{D}\tau_D\otimes\mathbb{1}_{A_C}\big)\otimes{}_{\mathbb{n}}]\mathcal{B}\tau_B\otimes{}_{\mathbb{n}}]\mathcal{L}\tau_L\Big) \oplus$$
$$\delta_D\Big({}_{E_D}]\mathcal{D}\bullet\big(({}_{I_C}]\mathcal{C}\tau_C\otimes{}_{I_L}]\mathcal{L}\tau_L\big)\otimes\mathbb{1}_{A_D}\big)\otimes{}_{\mathbb{n}}]\mathcal{B}\tau_B\Big)$$

where in the specified serial couplings the following translation functions have been applied. The functions are given by sets of associations of output and input symbols:

$\nu_{BC} \in O_B \to I_C$, $\nu_{BC} = \{(\text{push}, \text{push}), (\text{release}, \text{release}), (\mathbb{n}, \mathbb{n})\}$

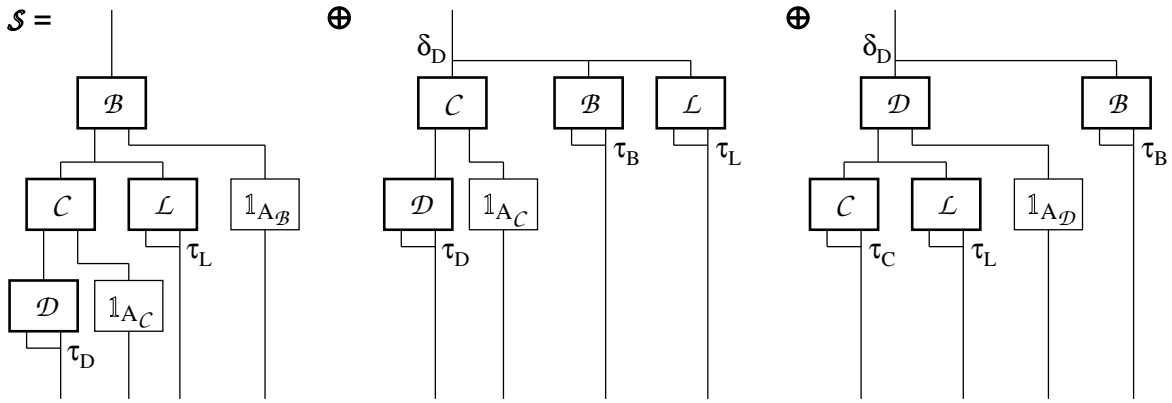
$\nu_{BL} \in O_B \to I_L$, $\nu_{BL} = \{(\text{push}, \text{push}), (\text{release}, \text{release}), (\mathbb{n}, \mathbb{n})\}$

$\nu_{CD} \in O_C \to I_D$, $\nu_{CD} = \{(\text{doOpen}, \text{doOpen}), (\text{doClose}, \text{doClose}), (\mathbb{n}, \mathbb{n})\}$

$\nu_{DC} \in O_D \to I_C$, $\nu_{DC} = \{(\text{open}, \text{open}), (\text{close}, \mathbb{n}), (\mathbb{n}, \mathbb{n})\}$

$\nu_{DL} \in O_D \to I_L$, $\nu_{DL} = \{(\text{open}, \mathbb{n}), (\text{close}, \text{close}), (\mathbb{n}, \mathbb{n})\}$

Graphical representation of $\mathcal{S}$



The final deterministic system $\mathcal{S}_{\text{det}}$ is obtained by application of a determining projector corresponding to the inspection of the states of the Button process through the Controller process:

$$\mathcal{S}_{\textbf{det}} := \Delta_{\omega_{\mathcal{C}}}\mathcal{S}$$

where the enabling relation $\omega_{\mathcal{C}} \subset (B\times C\times D\times L) \times \text{DOM}(\mathcal{C})$ for component $\mathcal{C}$ is given by the switch function assigned to the only non-singular branching $\mathcal{C}|_{(\text{toOpen, open})}$.

Thus $\omega_{\mathcal{C}} = \omega_{\mathcal{C},(\text{toOpen, open})}$

where $\omega_{\mathcal{C},(\text{toOpen, open})} \in B\times C\times D\times L \rightarrow \text{DOM}(\mathcal{C}|_{(\text{toOpen, open})})$

is defined by

$\omega_{\mathcal{C},(\text{toOpen, open})}(\text{pushed, C, D, L}) := (\text{opening, open, blocked})$

$\omega_{\mathcal{C},(\text{toOpen, open})}(\text{released, C, D, L}) := (\text{opening, open, delayed})$

REMARK

If one described the DoorControlSystem completely in the formal specification language of the mentioned CIP method, then the denotation in the semantic model of the language would correspond exactly to the given expression.

Concluding Remarks

SCSM is suited for the modelling of synchronous cooperation of sequential processes. The formalism supports the description of instantantenous interaction of sequential machines by means of pulse passing and state inspection. The transition structures of the specified sequential machines describe, how these components react on inputs. The internal interaction of a system is defined through parallel and serial compositions of these machines. A composition of machines defines a static data flow structure representing a restriction of the possible interactions. A solely operational description of instantaneous pulse passing between arbitrary

components of a system would easily lead to an unstructured collection of possible chain reactions, thus resulting in system specifications with bad transparency and bad manageability. Furthermore, the formalism does not allow internal feedback cycles in order to ensure bounded response times for implemented systems.

We propose to use SCSM expressions as denotation domain for synchronous specification languages. The development concepts of a method can be supported by restricting the corresponding language to a class of well chosen composition structures. Technical standard constructs and the formal meaning of default assumptions must appear only in the definition of semantic functions. Such a language may typically include graphical specification constructs, support of multiple instances of processes and syntactic sugar for standardized specification objects such as "timers", "watchdogs" and "external feedback mechanisms". Behavioural models for a language are easily derived from the transformation monoids generated by the specified sequential machines.

For the description of data handling and algorithmic functions the sequential machines can be extended in the usual way by variables, by data structures for input and output symbols, and by annotation of transitions by operations expressed in an appropriate functional language. State inspection can correspondingly be extended to inspection of machine variables.

The proposed formal description of state inspection through determining projectors is based on switch functions which associate subsets of system states with transitions of non-deterministic components. Our view of these functions is a purely static and existential one. A specification language should apply a more procedural approach for the definition of those functions. Abstraction and modularity can be supported by the use of nestable state vector access procedures, well known as access methodes in object oriented languages.

REMARK

The mentioned CIP method is already applied in several industrial projects. A commercially available version of a specification tool supporting visual specification, check of consistency and code generation for CIP systems is planned for release in November 1993.

Acknowledgement

The work presented in this paper has been accomplished in the CIP project at the Federal Institute of Technologie of Switzerland (CIP is a new software development method for distributed reactive systems, CIP stands for Communicating Interacting Processes). The support by grant No. 2268 of the KWF (Kommission zur Förderung der wissenschaftlichen Forschung in der Schweiz) is gratefully acknowledged here. The autor wants to thank Prof. Albert Kündig, Hans Peter Gisiger, Hansruedi Müller, Spiros Netos, Daniel Schweizer, Hans Otto Trutmann and Andreas Wuertz for inspirating discussions and support of this work.

References

[1] G. Berry, S. Moisan and J.P. Rigault, ESTEREL: Towards a Synchronous and Semantically Sound High Level Language for Real Time Applications, in: Proc. IEEE 1983 Real-Time Systems symposium (1983) 30-37.

[2] G. Berry and L. Cosserat, The ESTEREL Synchronous Programming Language and its Mathematical Semantics, in: S.D. Brookes, A.W. Roscoe and G. Winskel, Eds., Seminar in Concurrency (Springer, New York 1985) 389-448.

[3] A. Blikle, Denotational Engeneering or From Denotations to Syntax, in: D. Bjorner, Ed., Proc. VDM-Europe Symposium 1987, (LNCS 252, Springer, Berlin 1987) 151-209.

[4] R.T. Boute, System Semantics: Principles, Applications and Implementations, ACM Transactions on Programming. Languages and Systems 10, 1 (1988) 118-155.

[5] J.R. Cameron, The modelling phase of JSD, in: J.R. Cameron, Ed., JSP and JSD: The Jackson Approach to Software Development, IEEE Comp. Soc. Press (1989) 282-292.

[6] P. Caspi, D. Pilaud, N. Halbwachs and J.A. Plaice, LUSTRE: A declarative language for programming synchronous systems, Symposium on Principles of Progr. Languages, in: Fourteenth Annual ACM SIGACT-SIGPLAN (Munich 1987) 178-188.

[7] S. Eilenberg, Automata, Languages and Machines, (Academic Press, New York 1973).

[8] S.R. Faulk and D.L. Parnas, On Synchronization in Hard-Real-Time Systems, Comm. ACM, 31, 3 (1988) 274-287.

[9] H. Fierz, The Synchronous System Description Language IPL, in preparation.

[10] H. Fierz, CIP-Communicating Interacting Processes, in proceedings SAFECOMP'93 Poznan, Poland, October 1993, to appear.

[11] T. Gautier, Le Guernic and L. Besnard, SIGNAL: A Declarative Language for Synchronous Programming of Real-Time Systems, in: G. Kahn, Ed., Proc. third Conference on Functional Programming Languages and Computer Architecture, (LNCS 274, Springer, Berlin 1978) 257-277.

[12] A. Benviste and P. Le Guernic, Synchronous Programming with Events and Relations: the SIGNAL Language and its Semantics, Science of Computer Programming 16 (1991) 103-149.

[13] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8 (1987) 231-274.

[14] D. Harel, A. Pnueli, J.P. Schmidt and R. Sherman, On the Formal Semantics of Statecharts, in: Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, IEEE Press, New York (1987) 54-64.

[15] M.W. Shields, An Introduction to Automata Theorie (Blackwell Scientific Publications, Oxford 1987).

TIK-Reports

A Relational Data Base Design for an X.500 Directory System Agent

F. Perruchoud, C. Lanz, B. Plattner, July 1990, TIK-Report No. 1

Model and Functionality Definition for the Collaborative Editing Conferencing System MultimETH.

H. Lubich, July 1990, TIK-Report No. 2

X.400 Security Capabilities: Evaluation and Constructive Criticism

M. Müller, August 1990, TIK-Report No. 3

CPU Evaluation for ADaM

Schibli, M. Tadjan Januar 1992, TIK-Report No. 4

Aspekte computergestützter Kooperation - Schriftliches Material eines Seminars an der ETH Zürich

Hannes Lubich, Januar 1993, TIK-Report No. 5

Extensible Attribute Grammars

R. Marti, T. Murer, December 1992, TIK-Report No. 6

GIPSY: A Generator for Incremental Programming Systems

R. Marti, T. Murer, June 1992, TIK-Report No. 7

Test Case Validation - TTCN Test Case Validation Against SDL Specifications

F. Kristoffersen, T. Walter, May 1994, TIK-Report No. 8

Conformance and Interoperability - A critical assessment

T. Walter, B. Plattner, September 1994, TIK-Report No. 9

OOP-Softwarearchitektur für Multimediakommunikation

Serge Hoffmann, März 1995, TIK-Report No.10

A Comparison of Selection Schemes used in Genetic Algorithms

Tobias Blickle, Lothar Thiele, April 1995, TIK-Report No. 11

Spezifizieren und Generieren von integrierten Umgebungen mit GIPSY

A. Helbling, T.Murer, Mai 1995, TIK-Report No. 12

Die Spezifizierungssprache GIPSY/L

A. Helbling, T.Murer, Mai 1995, TIK-Report No. 13