Resource Contention in Real-time Systems

Thesis

UNRESTRICTED

Resource Contention in Real-time Systems

Robert John Smart, B.Sc., M.Eng.

A thesis submitted to the
Department of Telematics
of the Open University
for the degree of
Doctor of Philosophy,
May 2003

Submission date: 12 May 2003
Award date: 30 October 2003

Abstract

The *divide—and—conquer* method is extensively used for system design. For real-time systems the separated components execute concurrently using some common computational infrastructure and this can lead to contention for system resources, such as processors, memory, communication channels, and so on. Unless the resource contention is accommodated, then a system built from the composition of components may not function as expected and the "proven" behaviour of the components can be invalid. To overcome this uncertainty a *divide—conquer—and—system-composition* method is required.

This thesis takes a different approach to many of the existing notations which focus on descriptions of behaviour. The Composite Transition System notation and algebra presented here enables the resource usage of the components to be specified and combined to form a composite system of concurrently executing components. By relating the composite system to the realisable behaviour of the system resources provided by the common infrastructure it becomes possible to determine any violation of the constraints imposed by the system resources. If the composite system model is then constrained by the resource behaviours then it is possible through an *extraction* operation to determine the modified behaviour of the components that will yield a system free of resource contention.

Component specification, concurrent composition, the application of system level constraints and extraction are applied in this thesis to a system encountered in a commercial application. The purpose of this example is to demonstrate contention modelling and the mathematics of the notation, rather than to prove any specific properties of the application. Deployment of the notation to more complex applications will require the development of software tools to compute concurrent composition and extraction, and this is the motivation for the mathematical treatment in this thesis.

Acknowledgements

I really must thank Mandy and my two sons, Jamie and Benjamin, for their tremendous patience and tolerance, especially during the closing stages of this research. There will now be that long awaited time for all those outstanding model kits to be built and the house to be decorated.

I must also thank my supervisor, Professor John Monk, for his valuable input and support during the course of this research. I now have less fear of reading papers, articles and books that have a mathematical content that exceeds the normal expectations of any engineering manager. Additionally I would like to thank Stan Hurst for his support during the early days, and Paul Edwards for his initial support and proof reading.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advances in microelectronic technology have led to the widespread embedding of programmable devices in systems built with diverse technologies [97]. This has brought with it significant design, verification, implementation and validation problems [82, 90]. Traditional ad-hoc approaches to the implementation of *embedded* systems are no longer adequate.

The programming of any system must incorporate an algorithm, but an embedded system adds the problem of interaction through its interfaces. Interface interaction often requires the timely reaction to an input, which is fundamentally a matter of *timing*, and often requires a timely output response, which is fundamentally a matter of (algorithmic) *performance*. The existence of these time related attributes often leads to the term *real-time* as an interchangeable alternative for *embedded*.

Many definitions of a real-time system can be found in the literature, and, not surprisingly, the notion of response times is a common theme. Mathai and Goswami [56] describe a real-time system as comprising an *external environment*, with its sensors and actuators, and a *programmable system* which registers events from the sensors and responds by producing actions to drive the actuators. It is because the sensor events are distributed in time that such a system is often classified as a real-time system. Young's definition [100] focuses on the temporal aspects and considers a real-time system to be any processing system which responds to externally generated input stimuli within a finite and specified period. Stankovic [90] though defines a real-time system to be one where the correctness of the system depends not only on the logical result of the computation, but also on the time at which the result occurs.

Other researchers [10, 69, 82, 92] embellish Stankovic's definition by categorising failures to meet timing requirements as *hard*, *firm*, or *soft*. Shin [82], for example, defines a *hard* timing requirement as one where the impact of a failure to meet the requirement is "catastrophic", but a *firm* timing requirement is one where the result ceases to be "useful" if the requirement is not met. Burns [10] describes a *soft* requirement by applying a utility measure which begins to decay once the required deadline has been missed, so allowing a distinction between a soft deadline and no deadline.

The common themes of this genre of definitions are the existence of timing requirements, or the significance of failure to meet timing requirements. But such themes are questionable because these definitions may be applied to many systems that embedded system engineers would not consider to have real-time properties. For example, a word processor may be required to respond to user input within a specified time. The consequence of failure might be that the application is not very useful.

Timing, and not performance, is often seen as important, yet Turski [94] asserts that *time* inordinately pre-occupies the designers of so called real-time systems. There is often confusion between timing and performance, often because the problems of meeting a timing requirement are a consequence of inadequate performance. In other words, given enough performance, then the timing requirements can be met. For example, some implementations have a single path of execution with a control loop which cycles through a set of input—compute—output operations. If the order of the input stimuli is predictable, then the program structure will be determined by this predictable order. Provided that there is sufficient performance, the program will always be waiting for the next anticipated stimulus. Thus the ability to meet the timing and performance requirements of the set of operations is fundamentally determined by the structure of the program, the speed at which the processor executes instructions and an assertion about the minimum time interval between one stimulus and the next stimulus.

Very often it is not possible to predict the temporal order of input stimuli. However, it is realistic to assume that the set of inputs might include both some periodic stimuli with known repetition intervals and some aperiodic stimuli with known shortest repetition intervals. For example, consider an asynchronous serial communication channel used to communicate a string of characters. Arrival of the first character is aperiodic, that is, it is not possible to predict when it will arrive. The subsequent characters of the string are (quasi) periodic as the communication data rate is (quasi) constant.

The possibility of aperiodic and quasi-periodic stimuli, or, more generally independent stimuli, makes it impossible to predict the time of occurrence of a stimulus or guarantee a minimum time between any pair of stimuli. Consequently a program might be unable to respond to a stimulus because it is still processing the previous stimulus. In other words, there is contention for the processor resource because the processor is performance bound.

A very common approach to solving the problem of dealing with independent stimuli is to design a system by decomposition into (largely) independent components, where each component is individually testable and analysable [45]. This is the *divide—and—conquer* method, and relies on the premise that a system built by integrating proven components is likely to be functionally correct.

In a real-time system, each component may well be implemented as a single thread of execution which repeatedly deals with stimuli from a single input. The design of the integrated system then comprises a collection of concurrent threads of execution assumed to be largely independent of one another. Underlying this decomposition method are the concepts of *concurrent execution* and *independent action* (freedom of constraints from other components). However, interdependence must exist for the composite to be considered a system [57], in other words, the components are not totally independent and some sort of interaction is essential. Interaction may occur between the components, although loose coupling of components is a common design objective. At least one of the components must interact with the application environment, if the system is to provide a useful function [81] and interaction may take place between components through the environment and infrastructure.

A composite system is established by the *concurrent composition* of components, and the interdependence is achieved through *interaction* between components.

Consider concurrent composition. Many mathematical notations based around concurrent composition (including some reviewed in Chapter 2) assume *maximum parallelism* in their computational model. This model ensures that if two processes are ready to communicate then communication is not delayed by a shortage of computational resource. Maximum parallelism enables "true" concurrency, that is, simultaneous execution and that assumes unbounded computational resources. Such an assumption is overtly optimistic [25] and very often invalid because generally there are more processes than processing resources. Instead, the processes compete for processor resource [56].

Many systems include a *real-time kernel* [89] which schedules the processes to the available processing resources according to a *scheduling algorithm*. The scheduler gives an illusion of concurrency by interleaving the execution of processes on a shared processor. Not surprisingly, *scheduling theory* is a significant body of real-time systems research [10, 19, 35, 43, 72, 77]. Indeed Stankovic and Ramamrithan in [89] make the assertion that the most critical part of supporting real-time systems is the scheduling algorithm (and the design of the operating system). The maximum parallelism view avoids the need for scheduling, but does not guarantee that real-time constraints are met because it says nothing about the required processor performance to ensure that the program will always be waiting for the next anticipated stimulus.

Thus the scheduler in a real-time kernel provides an example of a practical concurrency resolution mechanism, where the scheduling algorithm is used to resolve the contention that arises for a processing resource. Scheduling algorithms often use a priority associated with each process as the basis of choice in the resolution of contention for processor resource. Real-time kernels often implement a *priority pre-emptive* scheduling policy where the highest priority contending process is chosen and immediately allocated to the processor resource. In general, the higher the priority of a process the sooner it will be allocated to the processor in response to a stimuli.

Where a practical concurrency operation is not modelled accurately with a concurrency operator in a modelling notation, then the behaviour of a system implementation may differ from the behaviour determined by the system model. Thus the implementation behaviour may not necessarily meet the requirements of a system given in a specification even though a model derived from the same specification (but assuming an idealised computational model) indicates that the required behaviour is met.

Now consider interaction, the other fundamental characteristic of the decomposition and concurrent composition method. The specification of components and their interaction is not necessarily sufficient to guarantee a functionally correct system. Milner [57] provides a simple example to illustrate the problem.

Milner defines a *jobber* which may use (non-deterministically) either a *mallet* or a *hammer* to perform its job. Both tools are part of the modelled environment of the jobber, that is, there is a defined interaction between the jobber and the mallet, and between the jobber and the hammer. The mallet and the hammer do not interact. Now consider the existence of a second jobber with identical behaviour. As neither jobber has the need to interact, the existence of another jobber does not alter the modelled jobber behaviour and both are independent. Since the jobbers have the same behaviour, they can both use either the mallet or the hammer. However, the availability of the hammer or the mallet to one jobber is now influenced by the other jobber. In other words, there is *unintended interaction* when there is contention for shared resources.

This simple example, which illustrates a class of problems encountered in building real-time systems, can be attributed to the *resource contention* which arises from the unforseen interaction between concurrently combined processes. The concurrency operators provided by the process specification notations reviewed in Chapter 2 allow independent processes to be combined to execute concurrently. In many practical systems, these concurrent processes cannot be simply combined because they then make simultaneous demands for shared resources which cannot necessarily be fulfilled. It is not surprising that the component behaviour is in some way distorted when the implementation of the composition does not match the properties of the composition operator in the notation used to describe the system. This behavioural distortion is difficult to predict.

Surprisingly, *resource contention* has attracted very little research. This thesis focuses not on the specification of process behaviour, but rather on the modelling of process interaction resulting from shared resources and the subsequent prediction of behavioural distortion. This requires a machine algebra which assumes the overtly optimistic maximum parallelism view of computation, but which can be restricted through the explicit modelling of the constraining behaviour of the shared resources. A notation, called Composite Transition Systems (CTS), is defined in Chapter 3, and the operations of *merge composition* and *concurrent composition* of Composite Transition Systems are defined in Chapter 4. The operation of *extraction* determines the distorted behaviour of a Composite Transition System component and is defined in Chapter 5.

The Composite Transition System notation is applied in Chapter 6 to a data acquisition system which samples and de-multiplexes a quadrature multiplexed signal. De-multiplexing is in response to external stimuli. The de-multiplexed signal is then sampled using stimuli from a position transducer. This application is interesting because the intuitive solution is to have one de-multiplexing process and a position sampling process. A design objective is to execute both processes on the same processor. Further, one process provides a data set that may be read by the other process, so the processes share some memory. However, the read and write access to the shared memory must not be interleaved as then a partially updated data set may be sampled. Finally, neither process may miss a stimulus, so there are timing constraints. The objective of the example of Chapter 6 is to illustrate the modelling of resource contention through the use of the Composite Transition System notation and its operators, rather than to prove any specific property of the system.

To conclude, Chapter 7 discusses the results of the research presented in thesis, reviews the definition of the operators, and discusses the important concepts and issues that arose during this research. Finally, areas for further work are identified.

Chapter 2

Formal Notations

Many notations can be found in the published literature that aim to provide some formal basis to the design of systems and, in particular, systems that must exploit concurrency in order to fulfil the real-time properties introduced in Chapter 1. This chapter is a selected review of some of those notations that provided the motivation for the development of the Composite Transition System (CTS) notation presented in Chapter 3 of this thesis.

First, a brief review of Graph Theory is given in section 2.1 as a precursor to the introduction of Labelled Transition Systems in section 2.2. Graph theory is a long established branch of mathematics concerned with the structure, and patterns within that structure, that results from the relationship between entities. It is of interest here because this thesis is concerned with the structure of processes modelled as a set of related entities. Published applications of the use of graphs include the study of system behaviour [14, 29], the specification of concurrency in the Ada programming language [42], task scheduling [35, 60], distributed process scheduling and load balancing [19, 96, 99], scheduling input and output operations [43], communication resource constrained scheduling [79, 83] and system fault diagnosis [73].

Graph theory permits a description of systems without the distraction of computer science derived notations, some of which are introduced later in this chapter. A problem with such notations is that they often incorporate operators with semantics motivated by the objectives of the notation creators and these semantics are not necessarily equivalent to the semantics of the corresponding operators in the tools available to system builders. Indeed, some researchers assert that the objectives of formal notations often limit the bounds of their applicability [64, 84] and this will be corroborated in the context of the notations reviewed in this chapter.

Labelled Transition Systems (LTS), reviewed in section 2.2 (page 12), are a form of a graph with a structural and executional interpretation and which allow the description of sequential discrete event systems. Published applications of Labelled Transition Systems include the specification of Ada tasking [11], the study of Communicating Sequential Processes (CSP) (section 2.4), Calculus of Communicating Systems (CCS) [57] and concurrent programming languages [91]. Finite State Automata (FSA) are a classic form of Labelled Transition System used, for example, in formal language theory [31, 38].

Critical examination of Labelled Transition Systems shows their limited applicability to the description of concurrent systems and this has motivated the development of the Composite Transition System (CTS) notation presented in Chapter 3 and the operators presented in Chapters 4 and 5. Little published literature exists on the algebraic manipulation of Labelled Transition Systems, especially for concurrent composition, and this too motivated the definition of the Composite Transition System notation.

A further three notations are of interest for their treatment of concurrency, their definition of algebraic operators, and their treatment of resources and consequent restriction on system behaviour. Although two of the notations define a wider set of operators, this review is restricted to the treatment of concurrency, communication and choice. Concurrency is a clear requirement. Communication is of interest because it is used to describe interaction between concurrent processes. Choice is of interest because the resolution of resource contention requires some choice to made.

Communicating Real-Time State Machines (CRSM) [81], reviewed in section 2.3, is a "specification" notation based on the concurrent execution of sequential discrete event machines that interact through explicit communication. Each machine is an obvious form of Labelled Transition System, although augmented to specify communication and timing requirements on the execution of transitions. Although the notation allows the specification of concurrency, it defines no operators for the algebraic manipulation of machines.

Communicating Sequential Processes (CSP) [36, 37], reviewed in section 2.4 (page 23), is probably one of the more significant contributions for specifying concurrent systems arising from computer science research. The notation defines a "Process Algebra" for describing the behaviour of discrete event sequential processes and their algebraic composition to form a system. A timed model for CSP has been developed to explicitly include a model of time [18, 76]. Applications have included the modelling of digital electronic circuits [74].

Communicating Shared Resources (CSR) [25], reviewed in section 2.5 (page 29), is also based on the concurrent execution of discrete event processes that interact through communication. This notation is of interest since it recognises that the behaviour of a system depends not only on any communication but also on the resource requirements and any resultant execution scheduling. Therefore, the Communicating Shared Resources notation assumes a resource limited computational model and uses event priority to resolve resource contention where events simultaneously occur on a shared resource.

The semantics of priority in CSR does not "lend itself to an equational characterisation" [26] and this led to the development of the Calculus of Communicating Shared Resources (CCSR) [26, 27, 28] and, more recently, the Algebra of Communicating Shared resources (ACSR) [51]. Both CCSR and ACSR are based on Milner's Calculus of Communicating Systems although the notion of communication is closer to that of Communicating Sequential Processes.

Section 2.6 (page 34) provides a brief overview of some other formalisms and techniques of some related interest to the specification of real-time systems and, therefore, the work presented in this thesis. Finally, section 2.7 (page 40) summarises a crucial deficiency of the reviewed notations that limits their application to the problem of modelling resource contention.

2.1 Graph Theory

A graph, $G = (V, E)$, has a finite non-empty set of *vertices*, V, and a possibly empty set of *edges*, E. Each $v_i \in V$ is a vertex and each $e_i \in E$ an edge. An edge connects two *adjacent* vertices and identifies a relationship between the two vertices. Useful texts on Graph Theory include Chartrand and Oellerman [14] and Gibbons [29].

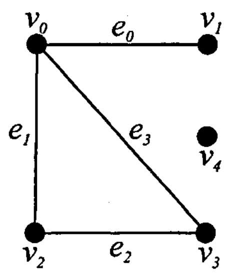


Figure 2.1: A diagram of an undirected graph

One possible interpretation of the graph $G = (\{v_0, v_1, v_2, v_3, v_4\}, \{e_0, e_1, e_2, e_3\})$ is illustrated in figure 2.1. In this interpretation the vertices v_0 and v_3 are adjacent because of the edge e_3, but vertices v_1 and v_3 are not adjacent. Other interpretations of G are possible because G gives only the vertex set and the edge set, specifically it does not state a relationship between the vertices. Each edge is often specified by a set of adjacent vertices. For example, e_0 in figure 2.1 is an edge between the vertices v_0 and v_1, hence e_0 can be written as $\{v_0, v_1\}$, that is $e_0 = \{v_0, v_1\}$. Thus, E can be written as $E = \{\{v_0, v_1\}, \{v_0, v_2\}, \{v_2, v_3\}, \{v_0, v_3\}\}$. In this example, there is no relationship between v_4 and any other vertex.

From set theory [30, 52], the edges $\{v_m, v_n\}$ and $\{v_n, v_m\}$ are the same edge, in other words, there is no direction implied in any relationship, thus the term *undirected graph* is often used. Gibbons, in [29], uses the set theoretic pair notation (v_m, v_n), but, for undirected graphs, an edge (v_m, v_n) cannot be distinguished from an edge written as (v_n, v_m). However, in set theory, these pairs are not the same, that is $(v_m, v_n) \neq (v_n, v_m)$.

A *directed* graph, or *digraph*, is used to define the direction of an edge. Edge direction can be specified by an ordered pair $e_i = (v_m, v_n)$ which states that vertex v_m, the first vertex in the pair, is *adjacent to* vertex v_n and v_n, the second vertex in the pair, is *adjacent from* vertex v_m. Each vertex may have zero or more adjacent to edges and zero or more adjacent from edges. Figure 2.2 illustrates the directed graph given by $V = \{v_0, v_1, v_2, v_3\}$ and $E = \{(v_0, v_1), (v_0, v_2), (v_2, v_3), (v_3, v_0)\}$, where vertex v_1 has one adjacent from edge and vertex v_0 has two adjacent to edges.

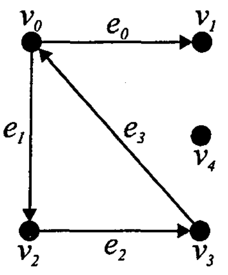


Figure 2.2: A diagram of a directed graph

Graphs and directed graphs define a relationship between vertices but neither offer an interpretation of an edge or a vertex. Labelled Transition Systems offer an interpretation.

2.2 Labelled Transition Systems

A Labelled Transition System (LTS) is an interpretation of a directed graph where each vertex models a system *state* and each edge models a state-to-state *transition.*

A state is often a statement about the "present". In other words, a state can be interpreted as the current value of a variable or set of variables [34, 61]. A state can also be considered to be an abstraction for detail where a state might model some activity with a duration with internal states and transitions [61]. Whatever interpretation is adopted, a state provides the opportunity for alternative behaviour, that is, from any state there may be several transitions that lead to different possible successive states.

Each transition describes an indivisible (atomic) state change and some Labelled Transition System definitions define a transition to be instantaneous. A system is only observed to progress from state to state as the result of an event, identified by the transition label, that represents environmental interaction. This environmental interaction is abstract in the sense that no mechanism is visible. Further, the states that a system passes through are intended to be internal in that the states are not visible to the environment [1].

This thesis adopts the formal definition of a Labelled Transition System given by Stark [90]. An LTS is defined to be a tuple $M = (Q, q_0, \Sigma, \Delta)$. The term Q is a finite non-empty set of states, $q_0 \in Q$ is a distinguished start state, and Σ is a finite event set. Note that Σ does not contain a distinguished *identity* event ϵ which represents non-progress of the transition system. Informally, Q is the vertex set and Δ the edge set in the graphical interpretation. The transition set Δ is bounded by $Q \times (\Sigma \cup \{\epsilon\}) \times Q$. Thus each transition $\sigma \in \Delta$ is an ordered triple specifying the *adjacent to* state (vertex), the event label (edge), and the *adjacent from* state (vertex).

Figure 2.3 is a diagram of a Labelled Transition System, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$ and $\Delta = \{(q_0, \sigma_0, q_1), (q_0, \sigma_1, q_2), (q_2, \sigma_2, q_3), (q_3, \sigma_3, q_0)\}$.

Each state may have zero or more *adjacent to* states. For the case where there is no adjacent to state, the system cannot progress, for example, state q_1 in figure 2.3. Such a state is often called a *terminal* state. Where there is exactly one adjacent to state then

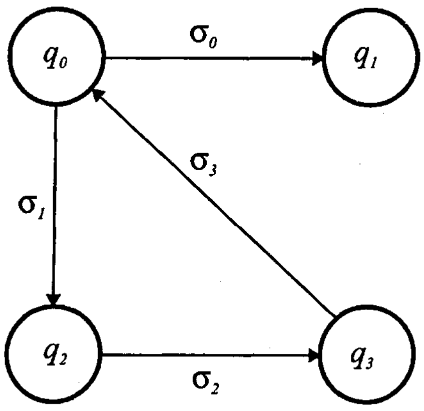


Figure 2.3: A diagram of a Labelled Transition System

there is only one possible next state and the from and to states are ordered. For example, states q_2 and q_3 are totally ordered because state q_3 can only follow from q_2, similarly state q_0 can only follow from q_3.

For states with more than one adjacent to state the system is said to be *deterministic* and the states partially ordered if for each state with more than one adjacent to state, no two transition labels are the same. However, if two or more labels are the same then the system is said to be *non-deterministic* because it is not possible for the environment to determine the next state. Droste in [20] defines the *disambiguation property*, such that, if $(q, \sigma, r) \in \Delta$ and $(q, \sigma, r') \in \Delta$ then $r = r'$. In other words, no two event labels from any state are the same.

Various other definitions of a Labelled Transition System can be found, for example, Droste [20], Khendek and Bochman [45], LOTOS [41] and Peng and Purushothaman in [63]. Henzinger (*et.al.*) in [34] give a definition which elaborates the definition of a state. Specifically, $S = (V, \Sigma, \Theta, T)$, where V is a finite non-empty set of variables, Σ is a finite non-empty set of states, Θ a non-empty set of initial states (such that $\Theta \in \Sigma$), and T is a set of transitions. Every state $\sigma \in \Sigma$ is an assignment to all the variables $v \in V$, in other words, a state is a particular assignment of the variables.

Peng and Purushothaman in [63] use finite state machines to describe processes where the events are treated as messages. A network of processes is built and the messages represent communication. They define the *shuffle product*, denoted by $P \otimes Q$, as a composition of the processes P and Q which defines the global state and message buffers. For example, let $p \overset{\alpha}{\to} p'$ denote a transition of process P and $q \overset{\beta}{\to} q'$ denote a transition of Q. The transition $[p, q] \overset{\langle\alpha,\epsilon\rangle}{\to} [p', q]$ is a transition of $P \otimes Q$, where process P has communicated message α but process Q has not communicated because ϵ denotes an empty message. And similarly for the transition $[p, q] \overset{\langle\epsilon,\beta\rangle}{\to} [p, q']$.

Peyravian and Lea in [68] also use a form of communicating finite state machine, on which they define the *Cartesian Cross Product Forming Algorithm* (CCPFA) to form a composite of machines P and Q. The state set of a composite machine is the set product of its components, however the message set is the set union $M_P \cup M_Q$ (where M denotes a message set), provided that the message sets M_P and M_Q are disjoint, that is, $M_P \cap M_Q = \{\}$. This latter condition ensures that the transitions of a composite machine only describe a state change of one of the components.

The *shuffle product* and the CCPFA, which have some similarity with the concurrent composition operator introduced in section 4.3 (page 74), cannot describe the simultaneous communication of both P and Q. This limits the applicability of these forms of composition for the purposes explored in this thesis.

2.2.1 LTS Summary

Labelled Transition Systems are a well established technique applicable to sequential event systems, however, it is an assumed property of concurrent systems that some events may occur simultaneously. As a sequential formalism a Labelled Transition System can act on just one event at any one instant and progress to just one next state. Thus, an LTS describes the existence of a system by just one state at any one instance.

Like the other sequential formalisms reviewed in this chapter, event simultaneity is either prohibited or the system is non-deterministic because a non-deterministic choice must be made, even if the disambiguation property is met. Prohibition is a weak requirement in that it does not enforce non-simultaneity and this is a restriction that limits the applicability of Labelled Transition Systems. The non-determinism approach is a failure to reason about simultaneity and this too is unrepresentative of systems which often in practice are required to exhibit deterministic behaviour. The disambiguation property can be thought of as ensuring a system with a deterministic structure, but it does not ensure non-deterministic execution. This problem arises because many existing definitions of Labelled Transition Systems do not deal with simultaneity and, therefore, do not recognise the difference between structure and execution.

Some researchers have explored the manipulation of Labelled Transition Systems to merge behaviours [45], or to describe concurrency. Stark in [91] observes that the parallel composition of two transition systems to form a new transition system requires the use of an interleaved execution model. Stark, and also Droste in [20], add concurrency relations to describe how pairs of transitions commute, that is, they interleave to form concurrent progression. Droste augments a definition of an automata with a collection of irreflexive, symmetric binary relations, denoted by $\|$. These relations define the concurrency information on each state q in the state set Q for all pairs of events in the event set E. Thus the relation $\|_q$ is a relation indexed on the states q. For example, if $q \in Q$ and $a, b \in E$, then the concurrent relation $a \|_q b$ describes the interleaving of the transitions (q, a, r) and (r, b, p) with (q, b, s) and (s, a, p) for the system to progress by interleaved execution of the events a and b from state q to state p.

Cattani and Sassone in [13] use *Higher Dimensional Transition Systems* to describe concurrency where transitions are labelled with finite *multi-sets* of actions (events) that represent the "simultaneous performance of their component actions". Using the example from the last paragraph, a simultaneous transition from state q to state p by the multi-set

$\{a, b\}$, denoted $(q, \{a, b\}, p)$, represents the simultaneous performance of the component actions a and b. The concept of a multi-set of events to describe simultaneity has some similarity with the Composite Transition System notation presented in Chapter 3.

Regrettably, much of the published research on concurrent transitions systems is motivated by the proof of mathematical properties and this is often to the detriment of the applicability to all but the simplest systems.

2.3 Communicating Real-Time State Machines

Shaw in [81] describes Communicating Real-Time State Machines (CRSM) as a notation for the specification of concurrency, communication, synchronisation, timing and environmental interactions of real-time systems. Further, Shaw's development goals included an executable notation enabling simulation where formal reasoning would prove difficult or intractable, and a simple (graphical) notation that would be familiar to system designers.

The CRSM notation has some similarities with Labelled Transition Systems, outlined in section 2.2. Specifically, both notations include a state set and a transition set. Further, both are sequential formalisms which cannot describe either state concurrency or event simultaneity. A formal definition of a CRSM would be closer to that of a Timed Transition System as defined by Henziger (*et.al.*) [34]. Additionally, Shaw asserts in [81] that a CRSM "bears some resemblance to a CSP process"; the CSP notation is outlined in section 2.4 (page 23).

A CRSM may be defined by the tuple $m = (S, T, C, V)$, where S is a set of states, T a set of transitions, C a set of communication channels, and V is a set of variables local to m. For any state $s \in S$, s defines a set of possible values of the variables $v \in V$. Transitions $t \in T$ model the execution of *commands* which make assignments to the variables v or which communicate on a channel $c \in C$ with another CRSM.

A system model is a set of CRSMs each executing independently except when they interact through explicit acts of communication. Such communication is the only mechanism to pass data as there are no shared variables, which, Shaw asserts, are the characteristics of distributed processing. Further, each CRSM is mapped to its own processor, thus the notation implicitly assumes a computational model with maximum parallelism.

Each command has an optional *guard* which is a predicate over the variables v. When a state $s \in S$ is reached, the guards on all the transitions from that state are evaluated and only those commands whose guards evaluate true are eligible to execute. Of those eligible commands, a command that makes assignment to the variables v will be executed immediately the state is entered. Communication commands will execute only when the communicating partner is also ready to execute (section 2.3.2, page 19). Note that the sequential composition of commands, denoted $A; B$, describes the execution of A immediately followed by the execution of B. There is no intermediate state between A and B and, therefore, the execution of a sequential composition is indivisible.

Where timing must be described, time parameters can be applied to a command. These define either the minimum and maximum execution times of a command, or for communication commands, the earliest to latest times that the command can execute after entering the *from* state. For two CRSMs to communicate, the combination of the arrival time in their respective to states and the timing specified on their respective input and output commands must overlap, that is both must contemporaneously be willing to communicate. If this condition is not met then deadlock can occur. Timing in CRSM is not explored further in this thesis, except in the summary in section 2.3.4 (page 21).

2.3.1 Choice

States may be viewed as opportunities for alternative behaviour. Where there are two or more eligible commands, then the choice is made on the basis of the first command that is able to execute. In the event of two or more commands being able to execute (or simultaneously becoming able) then the choice is non-deterministic. Whilst a non-deterministic choice avoids the need to define a choice algorithm or policy, it is unrealistic as most designs assume a deterministic behaviour.

Note that the evaluation of any guard holds until the machine executes a command. Thus the set of transitions from a state that may actually be executed is invariant until the state is re-entered. This arises because the guard predicates range over the local variables of the machine and these variables can be changed only by the execution of the machine and not by the execution of any other machine.

2.3.2 Communication

Communication is specified with input and output commands which act on one-to-one uni-directional channels, where each channel connects exactly two machines. A command that outputs the value x to channel c is written $c(x)!$. A command that assigns to the variable y a value input from channel c is written $c(y)?$. The result of this communication over channel c is equivalent to the assignment $y := x$.

The machine that first executes an input or output command on a channel must wait for the communicating partner to execute the corresponding output or input command on that same channel. Thus, a $c(x)!$ command will not progress until a $c(y)?$ command is executed and *vice-versa.* Such a communication model is often referred to as *synchronous* because the progress of the communicating machines is synchronised by the act of the communication (rather than by the value of the data communicated). This means that no data are lost and data buffering is not required. Note that self communication would lead to deadlock because the sequential nature of the machines means any one machine cannot simultaneously execute an input and an output command.

Restricting a channel to one-to-one communication avoids the specification of a choice policy that a many-to-one channel would necessitate. However, the CRSM notation is devoid of any mathematical foundation yet it is the mathematical foundation of other notations which justifies a non-deterministic choice. Note that although the CRSM communication model is based on the Communicating Sequential Processes model, CSP assumes a one-to-one restriction only as a convention [37].

The motivation for uni-directional channels is not given by Shaw in [81]. Such a restriction prevents the specification of a many-to-one channel and avoids the need for the CRSM notation to define a choice of channel from which to input data. Note that the Ada **accept** statement is an example of a many-to-one channel with a first-come-first-served choice policy [5]. Further, a bi-directional many-to-one channel would require a one-to-many return channel. For a one-to-many channel it is necessary to either "broadcast" to all possible recipients, or to include some address protocol which identifies the intended recipient. In practical systems, one-to-many constructs typically represent the case of broadcast (to the many) where receipt is not guaranteed and acknowledgement is not required; that is, there is no return channel.

Alternative communication topologies can be modelled by a CRSM, for example, one-to-any-one (1 to 1-of-n) and broadcast (1 to n). A possible form of asynchronous communication might allow a writer to progress even if no reader is awaiting a communication event. In this case the data must be buffered for the future reader and since an unbounded number of writes may occur before a read, the buffer must be infinitely large. Given a finite buffer then it is possible that the buffer will overflow and data will be lost. Such an asynchronous channel must be modelled by synchronous communication and message buffering within an intermediate CRSM, that is, a CRSM that specifically models an asynchronous communications channel.

2.3.3 Concurrency

Concurrency within a CRSM cannot be described and concurrency between CRSMs assumes a maximum parallelism model of computation. There is no specific formalisation of concurrency within the CRSM notation and there is no defined operator for the concurrent composition of CRSMs.

Where command execution is assumed to occur in zero time then it is possible for two consecutive commands to execute simultaneously. In other words, they execute at the same time, rather than sequentially, and this gives apparent concurrency. To "reflect the reality that it always takes some non-zero amount of time" and to avoid this apparent concurrency, the CRSM notation assumes that there is some non-zero time interval between the execution of commands [81]. A similar mechanism is used in Timed CSP (TCSP) [18, 76], whilst CSP [36, 37] uses arbitrary sequentialisation of simultaneous events with some non-zero time gap.

2.3.4 CRSM Summary

Although based on the concept of state machines, the syntax and semantics of the CRSM notation do not possess the rigour of state machines. Specifically, undefined choice mechanisms, the sequential composition of commands, synchronous communication, and timing parameters can all lead to problems. Further, state ambiguity can arise from the sequential composition of commands that take a non-zero time to execute. During execution a machine is neither in the *from* state nor in the *to* state. Further, should a communication command not be matched by its partner then both communicating CRSMs can deadlock between states.

The undefined choice mechanisms make it difficult to analyse the behaviour of a system of CRSMs, and any timing parameters exacerbate this difficulty. Consider figure 2.4 which shows two states and possible transitions to some next state. For the left hand state the first communication command that is able to progress is chosen. Where both commands could progress the choice is non-deterministic. For the right hand state, the internal command c does not have to wait for a partner and is, therefore, always ready to execute. In this case the notation is unclear, but it would seem that the choice would be to execute c, unless $a?$ or $b?$ are also ready and then the choice is non-deterministic.

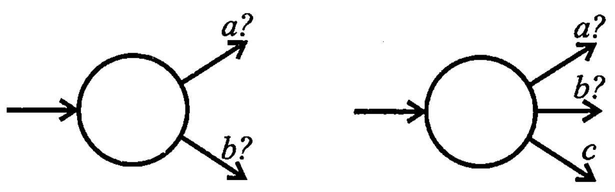


Figure 2.4: Choice on input (left) and input and internal commands (right)

The actual choice seems to be an "earliest-to-execute" policy, but where this policy is inconclusive then the choice is non-deterministic. However, an earliest-to-execute policy introduces some difficulties with command sequences. A choice based only on the first command in a sequence to be able to execute does not necessarily lead to a progressing machine. Consider a choice between the sequential command $a; b?$ and the command $c?$. If the input command $c?$ is not immediately able to execute then the command $a; b?$ will execute. After the execution of a, the input $b?$ may not be able to execute because it awaits its communicating partner, but perhaps $c?$ could now execute. This now leaves the CRSM in an unspecified state.

An alternative approach could be to choose an "earliest-to-complete" policy, where a command or sequence of commands is deemed to complete only when the last command places the CRSM in the *from* state of the sequence. However, the ability to determine which sequence requires consideration of the behaviour of every other CRSM that participates in communication within the sequence. This is likely to be difficult to determine. Finally, in the case of internal only commands with timing parameters, then an earliest-to-complete policy would (unfairly) always choose the quickest executing sequence.

It is clear that the CRSM notation lacks a formal definition and the behaviour of the constructs lacks rigour. None-the-less, it is a potentially useful graphical tool for simply expressing a design of a system, but without greater rigour and an algebra for the manipulation of machines it is very difficult to reason about the overall behaviour of a system of CRSMs.

2.4 Communicating Sequential Processes

Communicating Sequential Processes is a notation for describing a process as a mathematical abstraction of the interactions between a system and its environment [36, 37]. CSP is an event based notation based on the concept of indivisible interaction [57]. An event represents an observation of the behaviour of a process where a process behaviour is all possible execution paths of the process. The environment engages in these observations. Communication between processes are also environmental observations.

Based on set theory and predicate calculus, CSP defines a notation for describing processes and a set of operators for composition of those processes. Every CSP process P has an alphabet of events in which it will engage and this alphabet is a set denoted by αP. The notation $(x \rightarrow P)$ states that the process $(x \rightarrow P)$ engages in event x and then behaves like process P, and the alphabet of $(x \rightarrow P)$ is denoted by $\alpha(x \rightarrow P)$. CSP requires $\alpha(x \rightarrow P) = \alpha P$, rather than the more intuitive $\alpha(x \rightarrow P) = \alpha P \cup \{x\}$, thus $x \in \alpha P$. Recursion can then simply be defined without alphabet expansion and the notation $P = (x \rightarrow P)$ can be expanded by substitution;

$$\begin{aligned} P &= (x \rightarrow P) \\ &= (x \rightarrow (x \rightarrow P)) \\ &= (x \rightarrow (x \rightarrow (x \rightarrow P))) = (x \rightarrow x \rightarrow x \rightarrow P) \\ &\vdots \end{aligned}$$

Executed behaviour is described in CSP by a *trace*, for example, the trace $\langle a, b\rangle$ describes two events occurrences, an a event followed by a b event. For example, the trace of $P = (x \rightarrow P)$, denoted $trace(P)$, after three occurrences of the event x is written $\langle x, x, x\rangle$. Traces are used in the following sections to illustrate the operation. Observe that simultaneity of events cannot be recorded in a trace, and where events do occur simultaneously CSP defines that a non-deterministic choice is made.

2.4.1 Choice Operators

CSP defines three choice operators. Let P and Q denote two arbitrary processes, with alphabets αP and αQ respectively. The first choice operator, denoted $|$, states that if x and y are distinct events, that is $x \neq y$, then the process $(x \rightarrow P \mid y \rightarrow Q)$ is prepared to engage in event x or event y. If event x occurs then the behaviour is defined by process P, however, if event y occurs then the behaviour is defined by process Q. Note that it is required that $\alpha(x \rightarrow P \mid y \rightarrow Q) = \alpha P = \alpha Q$.

The second choice operator, denoted $[]$, states that the process $(x \rightarrow P[]y \rightarrow Q)$ is prepared to engage in event x or event y as determined by the environment, thus if $x \neq y$ then $(x \rightarrow P[]y \rightarrow Q) = (x \rightarrow P \mid y \rightarrow Q)$. Conversely, if x and y are the same event, then the choice of which process follows, P or Q, is not determined by the environment so is non-deterministic. Note that $\alpha(x \rightarrow P[]y \rightarrow Q) = \alpha P = \alpha Q$.

The third choice operator allows for the case where the system behaviour should not be determined by the environment, resulting, for example, from a specific implementation. Choice in this case is expressed by the non-deterministic choice operator, $\sqcap$, which states that the process $P \sqcap Q$ behaves either like process P or process Q and the environment must be prepared to engage in the first events from both P and Q. If the initial events of P and Q are the same then $(x \rightarrow P) \sqcap (y \rightarrow Q) = (x \rightarrow P[]x \rightarrow Q)$. Note that $\alpha(P \sqcap Q) = \alpha P = \alpha Q$.

2.4.2 Concurrent Composition Operators

Two concurrent composition operators are defined by CSP. Concurrent composition of processes P and Q by the operator, $||$, describes a process $P \parallel Q$ such that processes P and Q may progress simultaneously, but in a lock-stepped fashion for events common to the alphabets αP and αQ. That is, process interaction is modelled by the simultaneous engagement of the processes in the same *interaction event.* The alphabet of $P \parallel Q$ is $\alpha(P \parallel Q) = \alpha P \cup \alpha Q$, thus the alphabets αP and αQ are not necessarily the same and alphabet expansion can result. An *interaction event,* however, must be a member of both αP and αQ, thus the set of interacting events is $\alpha P \cap \alpha Q$.

Four cases arise in the composition $P \parallel Q$. If $\alpha P = \{a, x, y\}$ and $\alpha Q = \{b, x, y\}$, then the *interacting events* are x and y, but a and b are independent events. Processes P and Q are considered to execute in *lockstep* for interacting events, that is, both processes must engage in an interacting event to progress. Any event known only to one process allows that process to progress independently of the other process.

1. The process $(x \rightarrow P) \parallel (x \rightarrow Q)$ will cause both processes to simultaneous engage in the interacting event x, and then concurrently proceed hence $(x \rightarrow P) \parallel (x \rightarrow Q) = x \rightarrow (P \parallel Q)$. Thus the trace over the next event is $\langle x \rangle$.

2. The process $(x \rightarrow P) \parallel (y \rightarrow Q)$ will stop because x and y are different interacting events. This is denoted as $(x \rightarrow P) \parallel (y \rightarrow Q) = STOP$, where $STOP$ denotes a process that cannot engage in any events. Since the process cannot progress, the trace must be $\langle\rangle$.

3. The process $(a \rightarrow P) \parallel (x \rightarrow Q)$ can only progress on the independent event a, hence $(a \rightarrow P) \parallel (x \rightarrow Q) = a \rightarrow (P \parallel (x \rightarrow Q))$. Thus the trace over the next event is $\langle a \rangle$. Note that the $||$ operator is commutative, thus $trace((a \rightarrow P) \parallel (x \rightarrow Q)) = trace((x \rightarrow Q) \parallel (a \rightarrow P))$. Hence, $trace((x \rightarrow Q) \parallel (a \rightarrow P)) = \langle a \rangle$.

4. Finally $(a \rightarrow P) \parallel (b \rightarrow Q)$ defines a process of two independent processes and leads to either $a \rightarrow (P \parallel (b \rightarrow Q))$ if an a event occurs first, or $b \rightarrow ((a \rightarrow P) \parallel Q)$ if a b event occurs first. Thus, $trace((a \rightarrow P) \parallel (b \rightarrow Q)) = \langle a \rangle$ if an a event occurs first, or $\langle b \rangle$ if a b event occurs first.

Concurrent composition with the $\parallel$ operator describes the simultaneous progress of the processes P and Q only for interacting events. Hoare, in [37], shows that such a process is equivalent to a single process without the concurrency operator. Should non-interacting events occur simultaneously then the behaviour of the system $P \parallel Q$ is non-deterministic.

The second concurrent composition operator, $|||$, defines a process $P \mathbin{|||} Q$ such that the processes P and Q progress in an interleaved fashion without process interaction. Moreover, only one process progresses on any one event. Note that $\alpha(P \mathbin{|||} Q) = \alpha P = \alpha Q$. Consider the process $(x \rightarrow P) \mathbin{|||} (y \rightarrow Q)$, where $\alpha P = \alpha Q = \{x, y\}$.

1. Where x and y are distinct, that is $x \neq y$, the environment determines which process progresses. An x event will lead to $x \rightarrow (P \mathbin{|||} (y \rightarrow Q))$, that is, progression of $x \rightarrow P$ to P, but $y \rightarrow Q$ has not progressed because it is still awaiting a y event. Likewise, a y event will lead to $y \rightarrow ((x \rightarrow P) \mathbin{|||} Q)$.

2. Where x and y are not distinct, that is $x = y$, then the choice is not determined by the environment, but is non-deterministic. One possible outcome of $(x \rightarrow P) \mathbin{|||} (x \rightarrow Q)$ is $x \rightarrow (P \mathbin{|||} (x \rightarrow Q))$, that is, only $(x \rightarrow P)$ has progressed.

The two possible outcomes follow from the definition of the $\Box$ operator (section 2.4.1) and is written formally as follows.

$$(x \rightarrow P) \mathbin{|||} (y \rightarrow Q) = (x \rightarrow (P \mathbin{|||} (y \rightarrow Q)) \Box y \rightarrow ((x \rightarrow P) \mathbin{|||} Q))$$

2.4.3 Communication

Communication, an important element in describing the interaction between sequential processes, is a special form of event. Moreover, the act of communication requires synchronisation between the communicating processes.

Communicating the value v over the communication channel c is denoted by the $c.v$ event. As an event, the (familiar) notation $(c.v \rightarrow P)$ specifies a process that engages in the event $c.v$ and then behaves like process P. The process P can communicate on channel c those messages in the alphabet denoted by $\alpha c(P)$ which is defined to be a subset of the alphabet αP. In other words, process P can communicate only those messages defined in the alphabet $\alpha c(P)$. Two processes that are composed concurrently in the system $P \parallel Q$ and communicate (and synchronise) via the channel c must have the same alphabet at both ends of the channel, that is $\alpha c(P) = \alpha c(Q)$. By convention, a communication channel is uni-directional and connects exactly two processes.

The notation $c!v$ describes the event $c.v$ where the value v is output to the channel c. Hence, a process $(c!v \rightarrow P)$ behaves the same as the process $(c.v \rightarrow P)$. The notation $c?v$ describes the event $c.v$ where the a value v is input from the channel c. Thus the process $(v?c \rightarrow P(v))$ describes a process that inputs a value v and then behaves like the process $P(v)$. This output and input notation has been adopted by many other notations, including some reviewed in this chapter. Note that the **occam** programming language [40, 70], which is based on the concepts of CSP, uses this notation although the CSP convention of uni-directional channels between exactly two processes is enforced.

2.4.4 CSP Summary

Communicating Sequential Processes is a language for describing the behaviour of processes and for the composition of these processes to build systems. Unlike the CRSM notation (section 2.3), the behaviour of the operators is rigorously defined by a set of laws, giving a good basis for mathematical analysis. Some of the operators have been reviewed in this chapter and in particular the behaviour associated with the operators. However, there are some aspects of the behaviour of the language constructs which do not aid the analysis of resource induced constraints and, more generally, do not match the known behaviour of implementations.

Most notably, any simultaneity of events is treated by a non-deterministic choice. Whilst this is mathematically convenient, since it avoids reasoning about the behaviour, it is not necessarily representative of the behaviour of an implemented system. For example, if each process is executed on a separate processor then each process can progress simultaneously, that is, no choice is necessary. Should, instead, the processes be executed on a shared processor then a deterministic choice will be made either, for example, by some hardware or perhaps by the operating system. In other words, in both the separate and shared processor cases, a non-deterministic choice will lead to a specification that is not representative of a real implementation of that system.

An attempt to address this limitation of CSP is presented by Lowe in [53]. Non-deterministic choices are replaced by probabilistic choices to model naturally probabilistic phenomena in practical systems, such as unreliable network communication. Of more relevance to the work presented in this thesis is the specification of priority on a choice, arguably the extreme case of probabilistic choice [78]. Fidge, in [23], also describes priority in CSP as a formal basis for the prioritising constructs `PRI PAR` and `PRI ALT` in `Occam`, and the `PRIORITY` pragma in Ada [9]. Priorities can be used, for example, to model interrupts, process priorities, and competition for shared resources.

CSP has undoubtedly been significant in influencing the development of other languages developed in the domain of computer science research, of international specifications such as LOTOS [41] and implementations such as `Occam` [40, 70]. Despite the specification of priorities in the work presented by Lowe, itself an extension of Timed CSP [18, 76], CSP and its derivatives remain a language for the specification of process behaviour and the composition of those processes. Further, there is little published work describing its application to implemented systems.

2.5 Communicating Shared Resources

Gerber and Lee in [25, 28] describe Communicating Shared Resources (CSR) as a language for specifying distributed real-time systems. Motivation for the development of CSR came from the recognition that most models are sufficiently abstract that resource details are not considered. Indeed, process based models often treat the execution of processes without consideration of their operating environments, yet these environments often have a significant effect on the (timing) behaviour of the system. For example, assumptions about the underlying computational model vary from the optimistic maximum parallelism model to the pessimistic maximum interleaving view and such assumptions cannot be ignored [26].

Gerber and Lee have developed the *Calculus of Communicating Shared Resources* (CCSR) process algebra and a proof system [26, 27] to enable algebraic manipulation of CSR processes derived by translation into the CCSR language. Automatic translation is presented in [28]. The treatment of time is extended in the *Algebra of Communicating Shared Resources* [51] by defining the behaviour of a process not just as a sequence of events, but a sequence of event-time pairs (in a similar manner to [85]).

Communicating Shared Resources uses a resource based computational model. Each resource can execute only a single action at any one time, thus, each resource can be considered to be inherently sequential. A resource may host a set of processes and at any instance any of these processes may compete for the resource. Hence the actions of multiple processes must be interleaved on any one resource, where arbitration between competing processes is resolved by a priority ordering scheme. True concurrency, that is, maximum parallelism, can only take place between processes executing on different resources.

Timing is expressed in the CSR language either as an implicit property of an operator or through explicit timing parameters. For example, the statement **wait** t is a delay of t time units, whilst the statement $\mathbf{exec}(a, t_{min}, t_{max})$ denotes the execution of the event a where t_{min} is the lower bound on the execution time and t_{max} the upper bound.

The CSR language also includes constructs to specify periodic processes, temporal scope, time-outs and interrupt handlers. Execution of an interrupt handler is determined by the priority of the interrupting event, thus, only if the event has the highest priority will the handler execute immediately.

2.5.1 Choice Operators

Alternative behaviours in a process are permitted only with guarded statements. Each guard is either a local computation event denoted a, an input event denoted $a?$, or an output event denoted $a!$. A local computation is an event that is confined to a resource. Associated with each event is a priority and functions of the form $PRI \in \Sigma \rightarrow \mathcal{N}$, where Σ is the set of events and $\mathcal{N}$ is the set of natural numbers, maps an event to its priority. The function $PRI_i(a)$ returns the priority of the input event $a?$, $PRI_o(a)$ returns the priority of the output event $a!$, and $PRI_I(a)$ returns the priority of the internal event a.

In the case of two or more guards simultaneously being matched by their communicating partners then the statement associated with the highest priority matching guard is executed. Whilst the statement executes it does so with the priority of the guard. If there are any computation guards then any priority arbitration takes place immediately.

It is assumed that no two event priorities are equal, and this avoids the non-deterministic choice seen in many other notations.

Additionally, there is a construct that allows alternative behaviours but which also provides a timeout mechanism. This is a simple syntactic extension with a statement preceded with a **wait** *t* guard.

2.5.2 Concurrent Composition Operators

There are two concurrency operators and these arise naturally from the mapping of processes to resources. Consider the following section of the grammar of the CSR notation;

$$\begin{aligned}
\langle\text{system}\rangle &\overset{def}{=} \langle\text{resource}\rangle \mid \langle\text{system}\rangle \,\|\, \langle\text{system}\rangle \\
\langle\text{resource}\rangle &\overset{def}{=} \{\langle\text{process}\rangle, \ldots\} \\
\langle\text{process}\rangle &\overset{def}{=} \langle\text{statement}\rangle \mid \langle\text{process}\rangle \& \langle\text{process}\rangle
\end{aligned}$$

The symbol || denotes the true concurrency composition operator, thus a system comprises one or more resources which execute with true concurrency. However, a resource is a set of processes, and processes are defined to execute with interleaved concurrency, denoted by the symbol &. The symbol | indicates a syntactic alternative.

Arbitration between resources is not necessary, the maximum parallelism view of computation means that guarded alternative commands are the only model of competition for a resource. However, arbitration between processes is necessary. The use of guard priority to determine which statement is executed when matching guards are simultaneously satisfied also appears to resolve the simultaneous need for the resource. First consider that the alternatives are expressed as alternatives of a process and not alternatives between processes. Recall that processes are interleaved, thus if two processes are simultaneously satisfied by two other resources, then distinct events are involved because of the one-to-one communication and because all events are unique a unique priority can be resolved.

Gerber and Lee in [28] present a *configuration language*, motivated by the need to define the relationship between CSR processes, which are without "concurrent context", and the overall system. Configuration schema map processes to resources, assign priorities to events, and create channels between processes. Mapping processes to resources with a configuration schema means that the CSR language no longer includes the $||$ concurrency operator. However, the interleaved concurrency operator & is retained for intra-process concurrency, that is, the operands of & are always bound to the same resource.

2.5.3 Communication

The (familiar) notation $a?$ is used to represent an input (read) event, $a!$ an output (write) event, and a a computation event. When both processes agree to communicate then both simultaneously execute the event a. A computation event a requires possession of the resource.

All communication between resources, is defined as one-to-one and performed by synchronising on an event with a shared label. Thus $a?$ must be matched by exactly one $a!$. Note that all events are taken from the global event set Σ. However, the interleaved execution of processes makes it impossible for matching $a?$ and $a!$ actions to simultaneously execute if the two processes are allocated to the same resource. Gerber and Lee in [25] suggest that such communication can be modelled by using another resource as an intermediary.

In practice many systems do have communication between interleaved processes. Extensions to the basic CSR notation provide an asynchronous communication *channel* through a communication system. This communication system is simply an abstraction of another resource, however, this approach does allow the behaviour of the communication system to be explicitly modelled.

2.5.4 CSR Summary

The motivation behind the development of CSR was the recognition that existing formal models treated the execution of a process without regard for the significant effect the system resources have on the behaviour of the individual processes in the system. As a process specification language, CSR seeks to address resource specification through the explicit mapping of processes to resources. Execution of those processes mapped to a resource must be interleaved, whilst true concurrency can occur between processes executing on different resources. This treatment of resources leads only to an indirect specification of interleaved concurrency, the & operator, or true concurrency, the || operator.

The CSR notation requires all communication between processes, regardless of the mapping to resources, to be via matching input and output events. The behaviour of such communication is defined by the notation, yet it may be unrealistic for the communication behaviour of the implementation to be equivalent. Indeed, the example application presented in Chapter 6 requires different communication behaviour.

In many practical systems, communication through shared memory is deployed for reasons of performance, and this makes the communication channel model less appropriate. For example, consider the implementation of the software for a serial communication port. There is likely to be an executing thread that requests data and an asynchronous interrupt service routine that delivers data. The interface between the thread and the interrupt service routine will likely be via shared memory with some form of locking to ensure the correct operation where the interrupt service routine pre-empts the executing thread. This form of communication is not unrepresentative of techniques required in practical real-time systems, yet it does not fit easily with a distributed model of computation.

However, CSR adopts a *distributed system* view of real-time systems and so denies communication through memory shared between processor resources. Denying shared memory, even between processes mapped to the same resource, is a limitation that can detract from the application of CSR to practical real-time systems.

Mapping processes to resources, adopting the interprocess communication strategy, and the application of priorities for resolving simultaneity of events leads to a notation that can be used to specify certain classes of real-time system. Yet, it is not clear how this leads to full understanding of the interaction between processes, the problem that this thesis seeks to address. Gerber and Lee, in [25], suggest that the mapping of CSR to communicating finite state machines might enable state exploration techniques to detect the presence of properties such as live-lock or deadlock, however the "complexities" of CSR gave them cause for concern.

2.6 Related Research

Many other notations and techniques for describing concurrent systems can be found in the published literature. This section gives a short summary of some of those briefly explored for the purpose of modelling resource contention. Finally, this section includes a brief summary of related reading.

Petri Nets

Petri's Net Theory is perhaps the earliest general theory of concurrency [57] and one of its uses has been in the modelling of discrete event systems that may exhibit asynchronous and concurrent activities [66]. Peterson provides an introductory text in [67]. Applications of Petri Nets include the analysis of deadlock in the Ada programming language [22, 80], the design of complex synchronous controllers where concurrency is present [33], the specification and analysis of real-time, embedded systems or parallel systems [44, 62, 95], and the modelling of LOTOS ([41]) expressions [84].

A Petri Net graph comprises *places* and *transitions* and models the static properties of a system. Dynamic properties result from the execution of a "marked net" where one or more "tokens" move from place to place. A system exhibits concurrency when there are two or more tokens.

A place is similar to a state in a Labelled Transition System, however transitions are dissimilar in that they may have more than one input and more than one output. One or more places input to a transition, and a transition outputs to one or more places. When there is a token on all the input places of a transition, the "firing" of that transition is enabled. Where several transitions are enabled, a choice must be made and this is usually non-deterministic. When a transition fires, the tokens on the input places are moved to the output places and where there is more than one output place then each input token is replicated in each output place. Note that each place can be marked with more than one token. Enabled transitions are said to be "in conflict" if the firing of one transition moves an enabling token of another transition.

The non-deterministic choice of which transition to fire complicates the analysis of Petri Nets. A typical simplification is to assume that transition firing is instantaneous and the probability that any two or more events occurring simultaneously is zero.

Generalised and sub-classes of Petri Nets have been defined depending upon the modelling or analysis objectives. One such sub-class is a state machine where each transition has exactly one input and one output and where labels are applied to the transitions. Cortadella (*et.al.*) in [17] describe a technique for deriving a place-irredundant Petri Net from a transition system such that the behaviours are *bisimiliar*. Their motivation is the belief that the Petri Net representation simplifies the "representation of concurrency and causality" in the system. Further, their technique is claimed to support the automatic generation of Labelled Petri Nets not only from finite state machines but also CSP processes and Milner's CCS agents ([57]). Other variants include Predicate/Transition Nets, Coloured Petri Nets, Timed Petri Nets, Extended Timed Petri Nets and Higher Level Petri Nets.

Z Specification Language

The Z language is a mathematical notation for the specification of data or information systems through the description of state and logical conditions. Spivey and Ince [88, 39] provide introductory texts and a complete reference can be found in [86]. Applications of Z include the specification of a real-time kernel [87], analysis of the implications of priority in process scheduling [69], and the specification and verification hardware [75].

The language is based on mathematical data types, rather than those naturally found in computer systems, and a collection of operators defined by predicates with unambiguous mathematical properties. A system can be decomposed into *schemas* which describe both static and dynamic properties. Each schema defines any variables, any included schema(ta), any pre-conditions that must hold for a state change to occur, and the definition of the operation on the variables that reflect a state change. The values after the state change are called the post-conditions and the state values are called observations.

State descriptions in Z and the interpretation of a CSP specification as a state machine prompted Benjamin in [8] to use a combination of CSP and Z to describe a system with communication and concurrency. Duke and Smith in [21] use a combination of Z and *Temporal Logic* to explore properties such as *fairness* and *progress* in communication protocols modelled as event driven state transition systems.

Temporal Logic

Temporal logics have been applied to the specification and proof of the correctness properties of concurrent programs. This leads some researchers to observe that reasoning about the subtle timing properties of a system concurrent programs is easier using an abstraction such as temporal logic than "imperative" programming languages such as *Pascal* [32, 49, 59]. Lamport in [49] expresses the belief that an algorithm specified and proven using an abstract form should be compilable and the need to code the algorithm in a programming language is unnecessary.

Ben-Ari in [7] provides a simple introduction to temporal logic. The expression $\Box p$ asserts that p is at all times true and is often called the *safety* property. The expression $\Diamond p$ asserts that p is true either now or at some time in the future and is often called the *liveness* property. Hence, the expression $\Diamond\Box p$ asserts that at now or at some time in the future, p will become true.and stay true. Kröger in [46] provides an extensive mathematical treatment.

Moszkowski in [59] provides a good introductory text of "Interval Temporal Logic" (ITL), which includes, amongst others, the operators $\Box$, $\Diamond$ and $\bigcirc$ (where $\bigcirc p$ asserts that p will hold true at the next interval). Moszkowski also introduces *Tempura*, a logic programming language based on ITL, which is then illustrated by application to a multiplier circuit, a simple SR latch, and synchronised communication between two parallel processes. Hale, in [32], uses *Tempura* to illustrate its application to the "Towers of Hanoi" problem and the specification of the RS-232 asynchronous communications protocol.

Dealing with real-time constraints has led to extensions to temporal logics, typically by specifying interval bounds to temporal operators (for example, $\Diamond_{[a,b]}p$ asserts that p will hold true at some interval between a and b), or by adding expressions that allow timing bounds to be specified against a global clock [4]. This latter approach was adopted by Ostroff in [61] who uses *Real-Time Temporal Logic* as a proof system for parallel discrete event systems specified using his *Extended State Machines* notation, itself a form of *Timed Transition System.*

Protocol Conversion

Protocol conversion is a research topic that has some synergy with the research presented in this thesis. Pengelly and Ince, in [65], have followed research in *quotient machines* and developed a technique to solve the *interface equation.* Although their work was motivated by the application to protocol conversion, they believed that the interface equation was applicable more generally to concurrent systems that interact via an interface described by observable events or actions.

The objective was to construct a converter $\mathcal{M}_r$, such that the parallel composition of the protocol $\mathcal{M}_p$ with the converter is in some way equivalent to the protocol $\mathcal{M}_q$. In other words, the required behaviour is defined by $\mathcal{M}_q$, and the behaviour of $\mathcal{M}_p$ must be modified in some way to be equivalent to $\mathcal{M}_q$. This modification is achieved by the derived converter $\mathcal{M}_r$.

The interface equation takes the form $(\mathcal{M}_p|\mathcal{M}_r)\backslash A \sim \mathcal{M}_q$, where the protocols $\mathcal{M}_p$ and $\mathcal{M}_q$, and the protocol converter $\mathcal{M}_r$ are all described by labelled directed graphs. The operators are those defined by Milner's CCS notation [57]; that is, $|$ is parallel composition, the set A defines the interaction between $\mathcal{M}_p$ and $\mathcal{M}_r$, $\backslash A$ hides the interaction events, and $\sim$ defines some form of observational equivalence.

Pengelly determined that the definition of the CCS parallel composition operator was closely related to the graph Cartesian product (GCP) from which a *quotient machine* can be determined. Informally, $\mathcal{M}_r$ is the quotient machine of $\frac{\mathcal{M}_p}{\mathcal{M}_q}$. Unfortunately, the interaction results in directed arcs in a graph of $\mathcal{M}_p|\mathcal{M}_r$ that describe simultaneous state changes of $\mathcal{M}_p$ and $\mathcal{M}_r$. These simultaneous state changes cannot be formed by the GCP and thus a quotient machine cannot be extracted. The introduction of interaction breaks the symmetry of the graph Cartesian product and the technique presented in [65] is based on restoring the symmetry.

Background Reading

In the field of concurrent and real-time systems there are many texts. Laplante in [50], Burns and Wellings in [12] and Ben-Ari in [7] are comprehensive texts aimed at an engineer. Magee and Kramer in [55] also deal with concurrency and use the Java programming language (a language not often associated with concurrent or real-time systems) in their examples. A more mathematical or formal method approach has been taken by Hoare for CSP (section 2.4) and Milner for CCS [57] and for the *Polyadic π-Calculus* [58]. Like Hoare's CSP, Milner's CCS has been significant in stimulating further work, for example, Chen in [15] defines *Timed CCS* and Cleaveland and Hennessy in [16] introduce the notion of process priority.

The mathematical approach taken in the work presented this thesis uses set theory and predicates. Green in [30] and Lipschutz in [52] provide good introductory texts and Ayers in [6] covers more on the theory of groups and rings. Quine in [71] discusses logic largely through the analysis of written phrases (and so of value to those involved in translating written specifications into an implementation).

There is much published work that takes a broad or philosophical view of real time systems. Kurki-Suonio [47] makes the observation that all criteria for distinguishing "real time" from "non-real time" in the real world are artificial and depend on what we decide to consider "real time". Kurki-Suonio then questions some of the commonly accepted attributes of real time systems, for example, the need for deterministic constructs in programming languages, and the applicability of interleaving models. Lamport in [48] deals with the ordering of events in a distributed system, a point also touched on by Kurki-Suonio.

2.7 Summary of Formal Notations

Of the many published notations, Labelled Transition Systems (LTS), Communicating Real-Time State Machines (CRSM), Communicating Sequential Processes (CSP) and Communicating Shared Resources (CSR) were reviewed not only for comparison of their treatment of concurrency, choice and interaction, but also because each has some of the attributes required (see page 42) of a notation applicable to modelling resource contention. Notably, the CRSM notation, as a form of Labelled Transition System, is fairly intuitive and immediately applicable to an engineer. The CSP process algebra naturally takes an algebraic approach which is supported by stated mathematical laws for the operators, and the CSR notation is one of few that incorporates the notion of resources.

However, there is a significant impediment to the applicability of all the reviewed notations. Concurrent composition is limited to an assertion that the sequential components execute concurrently; this is clear with the CRSM notation, but is true also of CSP and CSR despite their syntax including a concurrent composition operator. Therefore, concurrent composition does not generate a system model that can then be restricted by the application of system level resource constraints. Additionally, the components are always explicit which obviates the need for a component extraction operator. Without component extraction from a restricted system, the method of resource modelling cannot take the significant step from composition to generation of the required components.

This significant impediment is true also of Labelled Transition Systems. However, Labelled Transition Systems are defined using set theory, and these definitions can be refined to describe concurrency and operators can be defined for their algebraic manipulation. Also, Labelled Transition Systems have the constructs necessary to describe the use of a resource by a sequential process without the distraction of guarded execution, communication channels, input and output operations, sequential composition, and so on, found in the other reviewed notations.

Chapter 3

Composite Transition Systems

This chapter defines the Composite Transition System (CTS) notation which has been developed to enable the description of the behaviour of (real-time) systems that comprise multiple processes which can involve concurrency, simultaneity, and synchronisation. In this thesis, concurrency implies the contemporaneous coexistence of independent components, thus, the concurrent composite is a description of all the possible combinations of the states of existence of the components. Moreover, independence implies asynchronous progress and the possibility that progress occurs simultaneously. It will be seen later in this chapter that concurrency relates to the "states" of a Composite Transition System, and asynchrony and simultaneity to the "events".

Simultaneity is the recognition that it is possible that two or more components will, by coincidence, progress. Many notations, including some of those reviewed in Chapter 2, deny the possibility or choose to make a non-deterministic choice about the consequent system behaviour in the case of simultaneity. Simultaneity is a real phenomenon in implemented systems, moreover, any choice made in the case of simultaneity is likely to be the outcome of some determined arbitration policy, typically priority, and thus a non-deterministic choice is not representative of an implemented system. The ability to describe simultaneity is an important requirement and it will be seen later that simultaneity relates to the "events" of a Composite Transition System.

Synchronisation is required when otherwise independent components must be constrained, typically to ensure co-ordination to achieve the system requirements. Dependence can arise in the interaction between components originally designed in isolation, often through the modelled interfaces of the component. Examples include function calls, access to shared data, any semaphore or mutex structures, any message queues, and so on. Further, there are often un-modelled system level constraints such as the access to shared memory, the competition for processor resource, or the hardware behaviour if two interrupts occur either simultaneously or contemporaneously. It will be seen later that synchronisation relates to the "events" of a Composite Transition System.

The objectives for the Composite Transition System notation developed in this thesis are as follows;

1. The notation must be able to represent concurrency, asynchrony, simultaneity and synchronicity because these are properties that characterise real-time systems.
2. The notation should not assume any specific computational model. In particular, a distributed model should not be assumed since the modelling of interaction through (synchronous) communication channels is not always applicable.
3. The notation must have algebraic operators to add behaviour to a component, to form a model that describes the concurrent composition of components, and to extract a component from a system. This latter operator is a consequence of the recognition that components are specified in isolation, but system level limitations may constrain the behaviour of the components and hence redefine them.
4. The creation of software tools to aid the system designer in describing a system algebraically is essential. Creation of such software requires thorough formalisation of the definition of the notation and of the operators of the notation.
5. The notation must be applicable to an engineer tasked with designing and implementing a system for deployment.

The design of the Composite Transition System notation follows from the objectives and this chapter describes the notation. Chapter 4 defines operators to perform *merge composition* and *concurrent composition*, and Chapter 5 defines the *extraction* of components. Merge extends the behaviour of a machine and provides the basis for the concurrent composition and extraction operators. Concurrent composition deals with combining the specifications of component machines to form a system which may then have system level resource constraints applied which restricts the system behaviour. The extraction operator deals with determining the required specification of the components such that their integration meets the restricted system behaviour.

Although the Composite Transition System notation enables the description of systems with asynchronous, simultaneous and synchronous progress, for a variety of reasons a CTS may be unrealisable. The use of the algebra of CTS's may therefore involve stages in which realisability has to be confirmed before an implementation of a Composite Transition System is attempted.

The remainder of this chapter is organised as follows. Composite Transition Systems and the concepts of composition and extraction are introduced in section 3.1. A formal definition of a CTS is presented in section 3.2 (page 56). Widespread use of Labelled Transition Systems makes it useful to consider the translation of a Labelled Transition System into a CTS, and *vice-versa*, and this is presented in section 3.3 (page 65). Except for formal definition or where ambiguity might arise, the terms *concurrent state* and *state*, the terms *simultaneous event* and *event*, and the terms *simultaneous transition* and *transition* are interchangeable.

3.1 Overview

This section is an overview of the Composite Transition Systems concept, notation and operators that are defined fully in the subsequent sections of this chapter.

Every *concurrent state* of a CTS is a set of one or more states, one from each component machine, and gives a system view of the contemporaneous state of existence of the components. Every *simultaneous event* is a set of one or more events, one from each component machine, that simultaneously occur in those components. The progression of a Composite Transition System occurs because of *simultaneous transitions*, that is, a simultaneous transition is a collection of transitions simultaneously executed by each of the component machines.

Figure 3.1 is a diagram of a Composite Transition System; at this stage it is not necessary to know how such a system may be formed. In such diagrams, the states are represented by circles, the transitions by directed arcs and the extent of any CTS is given by a bounding rectangle. Note that throughout this thesis, the state and event identifiers, and their ordering within the sets, have been chosen to aid readability only.

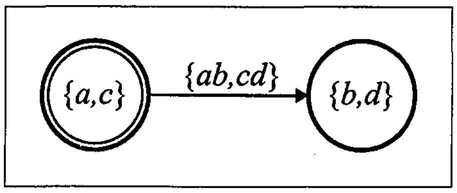


Figure 3.1: A Composite Transition System

The concurrent state $\{a, c\}$ represents the contemporaneous existence of two components, one component in state $\{a\}$, the other component in state $\{c\}$. Likewise the concurrent state $\{b, d\}$ represents the contemporaneous existence of one component in state $\{b\}$ with the other component in state $\{d\}$. Initial concurrent states are identified by a double circle, hence, there is one initial concurrent state, $\{a, c\}$. The example will react to one simultaneous event $\{ab, cd\}$, representing simultaneity of the event $\{ab\}$ from one component with the event $\{cd\}$ from the other component. Thus, should the CTS be in the concurrent state $\{a, c\}$ and the events $\{ab\}$ and $\{cd\}$ from the two components occur simultaneously then the CTS will progress to the concurrent state $\{b, d\}$.

3.1.1 Asynchronous and Coincidental Progression

Concurrency within systems may require the asynchronous progression of components, in other words, not all components may progress at the same instant. The CTS notation uses *idle events*, denoted by $\{\gamma_n\}$ (for component n), which can be thought of as an event which is always prepared to occur simultaneously with any non-idle event. Idle events provide the basis for describing asynchronous progress. Alternatively some components may coincidently and simultaneously progress and such progress can be thought of as coincidental asynchronous progress. In other words, the components happen to progress at the same instance, where this is neither a design objective nor is it enforced.

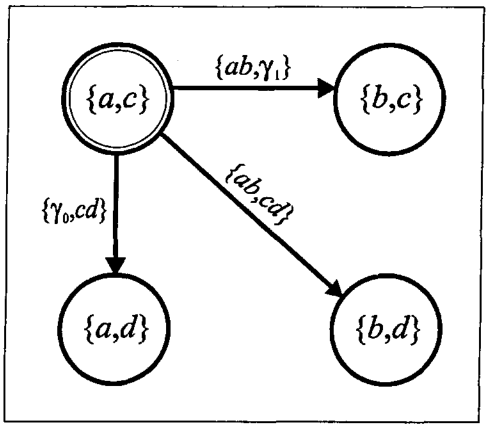


Figure 3.2: Asynchronous and simultaneous progress

Consider figure 3.2 which illustrates a CTS that exhibits asynchronous and simultaneous progress; at this stage it is not necessary to know how such a system may be formed. Let component C_0 contribute a transition labelled $\{ab\}$ from state $\{a\}$ to state $\{b\}$ and component C_1 contribute a transition labelled $\{cd\}$ from state $\{c\}$ to state $\{d\}$. Should the system be in the concurrent state $\{a,c\}$ and the component events $\{ab\}$ and $\{cd\}$ occur simultaneously then the system will progress to the concurrent state $\{b,d\}$. However, should only the event $\{ab\}$ occur then the system will progress to the concurrent state $\{b,c\}$ because the simultaneous event $\{ab,\gamma_1\}$ includes the idle event γ_1. In other words, component C_0 has asynchronously progressed from state $\{a\}$ to state $\{b\}$ because component C_1 has idled and stayed in state $\{c\}$. Similarly, if only the event $\{cd\}$ occurs then the system will progress to state $\{a, d\}$ because of the simultaneous event $\{\gamma_0, cd\}$.

Any event that includes a component idle event and a component non-idle event, such as $\{\gamma_0, cd\}$, may occur when the non-idle component event occurs. In other words, the occurrence of the component event $\{cd\}$ is sufficient for the event $\{\gamma_0, cd\}$ to occur. However, any non-idle event contributed by a component will take precedence over an idle event also contributed by that component. Consider again figure 3.2. Should the system be in the state $\{a, c\}$ and the events $\{ab\}$ and $\{cd\}$ occur simultaneously then the system will always progress simultaneously to state $\{b, d\}$. The system will not progress asynchronously to either the state $\{a, d\}$ by event $\{\gamma_0, cd\}$, because $\{ab\}$ takes precedence over $\{\gamma_0\}$, or the state $\{b, c\}$ by event $\{ab, \gamma_1\}$ because $\{cd\}$ takes precedence over $\{\gamma_1\}$.

Given that an idle event is always prepared to execute, then if the events $\{ab\}$ and $\{cd\}$ occur simultaneously then the events $\{ab, cd\}$, $\{ab, \gamma_1\}$ and $\{\gamma_0, cd\}$ all appear to occur simultaneously. The consequence of the precedence of a non-idle event over an idle event is that only the simultaneous event $\{ab, cd\}$ is deemed to occur, rather than the events $\{ab, \gamma_1\}$ and $\{\gamma_0, cd\}$. Thus the description of asynchronous and simultaneous progress does not require the non-deterministic choice often found in other notations.

Now consider figure 3.3 (page 47) which shows the composite system, $C_0 \| C_1$, formed by concurrent composition of the components C_0 (top) and C_1 (left). The concurrent composition operator, $\|$, is described in detail in section 4.3 (page 74). The states of the concurrent system are constructed by *pairing* states from the components. For example, the pairing of component states $\{a\}$ with $\{e\}$ gives the concurrent composite state $\{a, e\}$, the pairing of $\{a\}$ with $\{f\}$ gives $\{a, f\}$, and so on.

If the transitions of the components are considered to be asynchronous, then C_0 should be able to progress from state $\{a\}$ to state $\{b\}$ whatever the state of C_1, that is, it does

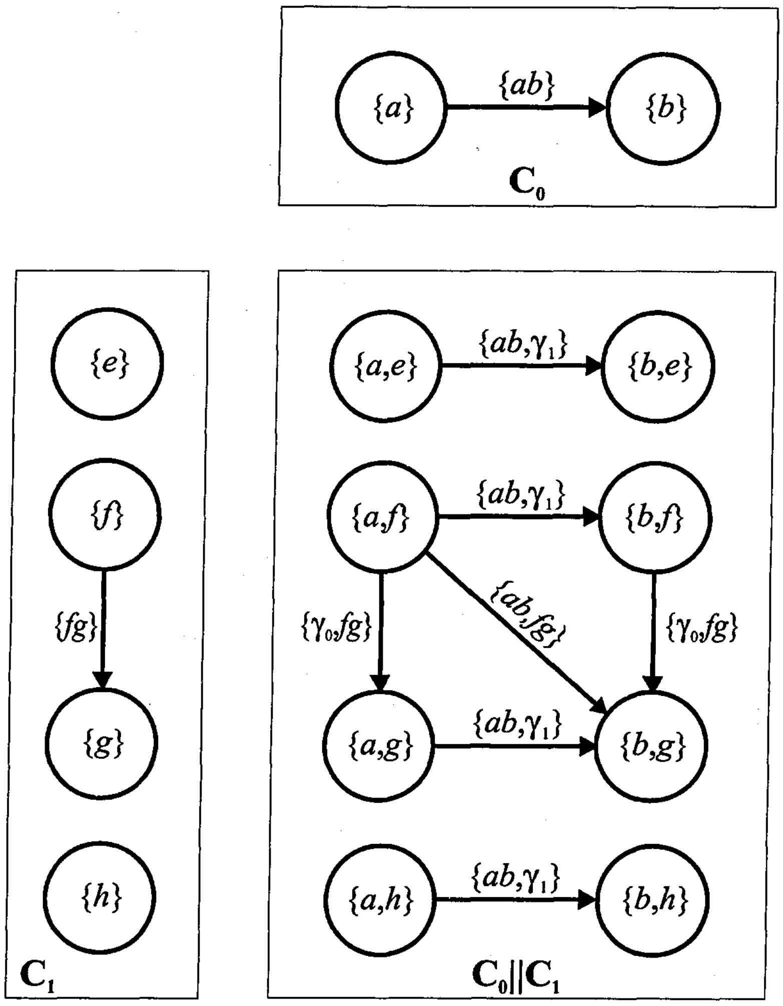


Figure 3.3: Asynchronous and simultaneous concurrent composition

not matter if C_1 is in state $\{e\}$, $\{f\}$, $\{g\}$ or $\{h\}$. Likewise C_1 must be able to progress from $\{f\}$ to $\{g\}$ whatever the state of C_0. This means that the system must describe simultaneous progress when both components can progress and asynchronous progress when one component progresses and the other idles.

The asynchronous transitions of the composite system are constructed by the pairing of transitions from one component with implied idle transitions on each state of the other component. For example, the pairing of the C_0 transition from $\{a\}$ to $\{b\}$ labelled $\{ab\}$, with an implied C_1 idle transition from $\{e\}$ to $\{e\}$ labelled $\{\gamma_1\}$ leads to the transition from $\{a, e\}$ to $\{b, e\}$ labelled $\{ab, \gamma_1\}$. The remaining five asynchronous transitions of $C_0||C_1$ are formed from the other five possible pairings.

The simultaneous transitions of the composite system are constructed by the pairing of explicit transitions of the components. In this example there is only the pairing of the C_0 transition from $\{a\}$ to $\{b\}$ labelled $\{ab\}$ with the C_1 transition from $\{f\}$ to $\{g\}$ labelled $\{fg\}$. This pairing gives the simultaneous transition from $\{a, f\}$ to $\{b, g\}$ labelled $\{ab, fg\}$. Observe, for example, that there is no transition from $\{a, e\}$ to $\{b, f\}$ because there is no C_1 transition from $\{e\}$ to $\{f\}$ to pair with the C_0 transition from $\{a\}$ to $\{b\}$.

3.1.2 Synchronous Progression

All the previous examples have used event names that are unique to a component. Under concurrent composition, the Composite Transition System notation uses the convention that event names common to the components are synchronous. Synchronisation is required for two reasons. First, many systems comprise components that by design synchronise on certain events, indeed, this is how they co-ordinate to meet the overall system requirements. Second, section 3.1.4 (page 50) introduces the concept of behaviour restriction and the extract operator which may introduce transitions with common event names specifically to coerce synchronisation in order to avoid, for example, resource contention.

Where component transitions progress on a common event name, then those transitions must be synchronously executed, in other words, there can be neither coincidental nor asynchronous progress. This means that a common event name cannot be paired with any other event name, including idle event names. Consider figure 3.4 (page 49) where the event name $\{s\}$ is common to both C_0 (top) and C_1 (left). The synchronous transitions are constructed by pairing transitions from each component provided that the pairing has a common event name. For example, the pairing of C_0 transition from $\{a\}$ to $\{b\}$, labelled with the event name $\{s\}$, with the C_1 transition from $\{f\}$ to $\{g\}$, also labelled with the event name $\{s\}$ leads to the transition from $\{a, f\}$ to $\{b, g\}$ labelled $\{s\}$. Since the event name $\{s\}$ is common to both components, $\{s\}$ has not, unlike asynchronous pairings, been paired with any implied idle transitions.

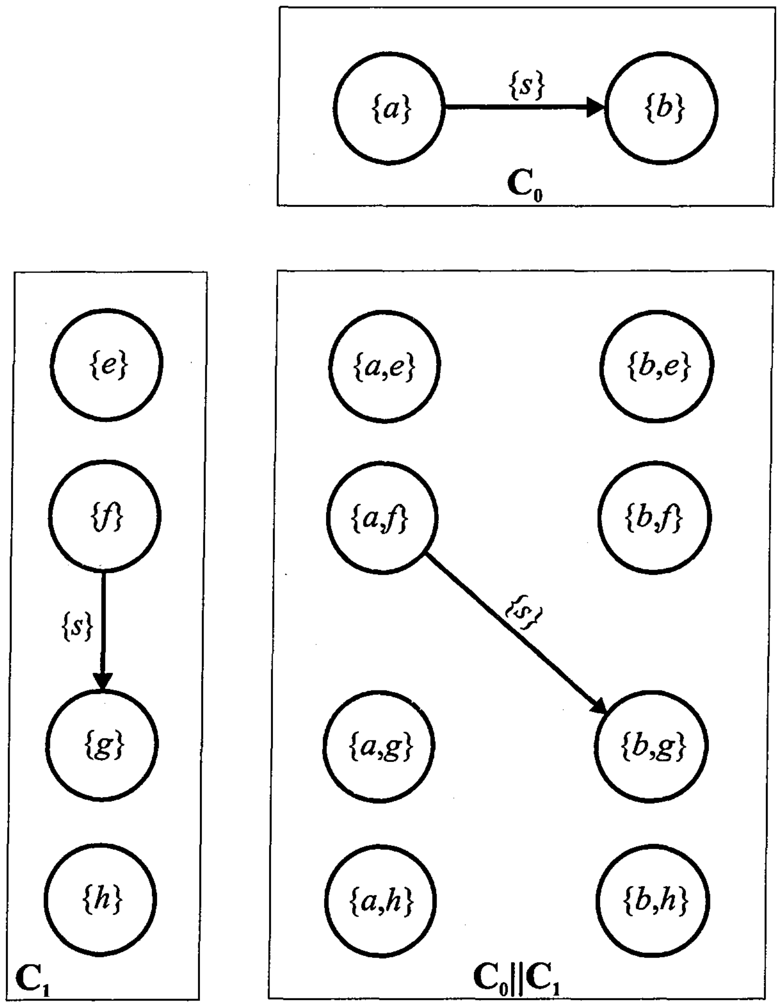


Figure 3.4: Concurrent composition of synchronous transitions

3.1.3 Concurrent Composition

The CTS notation defines a *concurrent composition* operator that takes two components, which together may incorporate asynchronous, simultaneous or synchronous transitions, and forms a system that describes their concurrent composition. However complex the components, the concurrent composition of such components is the "superposition" of each asynchronous, simultaneous and synchronous pairing as previously described in sections 3.1.1 and 3.1.2. Additionally, the Composite Transition System notation defines a *merge composition* operator which provides the formal basis of superposition. These operators are defined in Chapter 4 (page 68).

3.1.4 Restriction and Extraction

Asynchronous and simultaneous progress describe component behaviour where there is no interaction. Conversely, synchronisation describes component behaviour where interaction occurs and the behaviour of a component that exhibits interaction will be constrained by the components with which it interacts.

Consider, for example, a *writer* component that repeatedly writes data to a shared buffer, and a *reader* component that repeatedly reads the data from the shared buffer. If these components do not synchronise then their execution can result in an arbitrary write and read ordering. Thus it is possible that the writer over-writes data that has not been read or, perhaps, written data is re-read by the reader before new data has been written. This may well be the required behaviour. If, instead, the required behaviour is to allow a write if and only if the previous data has been read, then this implies that the behaviour of the writer must be restricted to prevent over-writing and therefore the behaviour of the writer is restricted by the behaviour of the reader. Further, if a strict write-read-write-read (and so on) behaviour is required then the behaviour of the reader must also be restricted to prevent re-reading.

The above example may be modelled with writer and reader components that are independent and the system may be modelled by their concurrent execution. Access to the buffer, a shared resource, will impose some restriction on the concurrent system, depending upon the required resource behaviour. Further, the allocation of the components to either their own processor or a shared processor will also determine any restrictions on the system behaviour.

Introducing system level constraints has the effect of removing transitions from the system model, and these system constraints also force constraints on the specification of the components of the system. The objective of the extraction operator is to determine the required modified behaviour of the components such that the behaviour of the constrained system is met. In other words, the removal of transitions from a system $C_0||C_1$ gives a restricted system denoted $\widetilde{C_0||C_1}$. Extraction of C_0 and C_1 from $\widetilde{C_0||C_1}$ will yield modified components $\widetilde{C_0}$ and $\widetilde{C_1}$. The behaviour of these modified components is such that their concurrent composition, $\widetilde{C_0}||\widetilde{C_1}$, gives a behaviour that is in some way congruent with the restricted system $\widetilde{C_0||C_1}$.

The CTS notation imposes no rules on which transitions can be removed as this might unnecessarily constrain the designer and so limit the applicability of the notation to describe a system. For example, it is possible that the system shown in figure 3.2 (page 45) describes a complete system or it could be the specification of a restricted system which is to be formed from interacting components. Therefore, a restricted system might comprise transitions that can be formed by asynchronous, simultaneous or synchronous pairings, as introduced in sections 3.1.1 and 3.1.2, under concurrent composition, but also transitions that cannot. Indeed, the example system of figure 3.2 cannot be formed by the concurrent composition of independent components, specifically, there are two absent asynchronous transitions. Further, the system cannot be formed by the concurrent composition of dependent components, specifically, the presence of the two asynchronous transitions causes difficulties. Consequently, the required interaction between components, which cannot always be determined by a simple set of interactions, can be determined by the extraction operator. However, required interaction may not always lead to a viable implementation.

Concurrent composition creates a form of symmetry in a concurrent model that might be broken by the removal of transitions in the formation of a restricted system. The extract operation introduces synchronisation to re-assert that symmetry while ensuring that the concurrent composition of the extracted components leads to a composite model with a behaviour congruent to that of the restricted system. This is achieved by the introduction of *State Dependent Synchronisation* and *Progressive Synchronisation.*

State Dependent Synchronisation

State dependent synchronisation is introduced when one or more of the anticipated transitions of the concurrent composition of independent components do not exist in a composite system. In other words, there are absent simultaneous or asynchronous transitions. Extraction creates reflexive state dependent synchronisation transitions which synchronise with transitions of the other component of the concurrent composition as a consequence of a common event name. (In a reflexive transition, the from and to states are the same.)

1. Absent Simultaneous Transition. Consider the example of the system in figure 3.5 where the component C_0 contributed a transition from state $\{a\}$ to state $\{b\}$ by event $\{ab\}$ and C_1 contributed a transition from state $\{c\}$ to state $\{d\}$ by event $\{cd\}$.

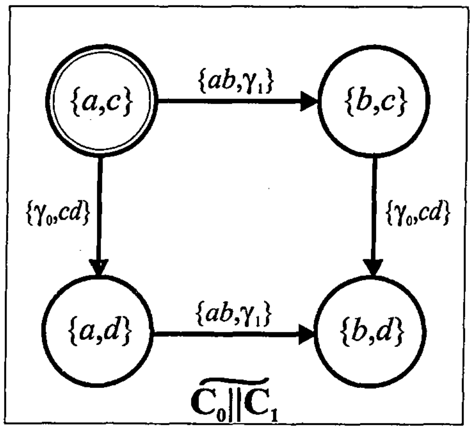


Figure 3.5: $\widetilde{C_0||C_1}$ with absent simultaneous transition

The anticipated simultaneous transition from state $\{a, c\}$ to $\{b, d\}$ labelled $\{ab, cd\}$ is absent, thus the simultaneous progression of C_0 from state $\{a\}$ to $\{b\}$ by event $\{ab\}$ and of C_1 from state $\{c\}$ to state $\{d\}$ by event $\{cd\}$ is denied. However, the progression of C_0 from state $\{a\}$ to $\{b\}$ by event $\{ab\}$ while C_1 stays in state $\{c\}$ or stays in state $\{d\}$ is not denied. Similarly, the progression of C_1 from state $\{c\}$ to $\{d\}$ by event $\{cd\}$ while C_0 stays in state $\{a\}$ or stays in state $\{b\}$ is not denied.

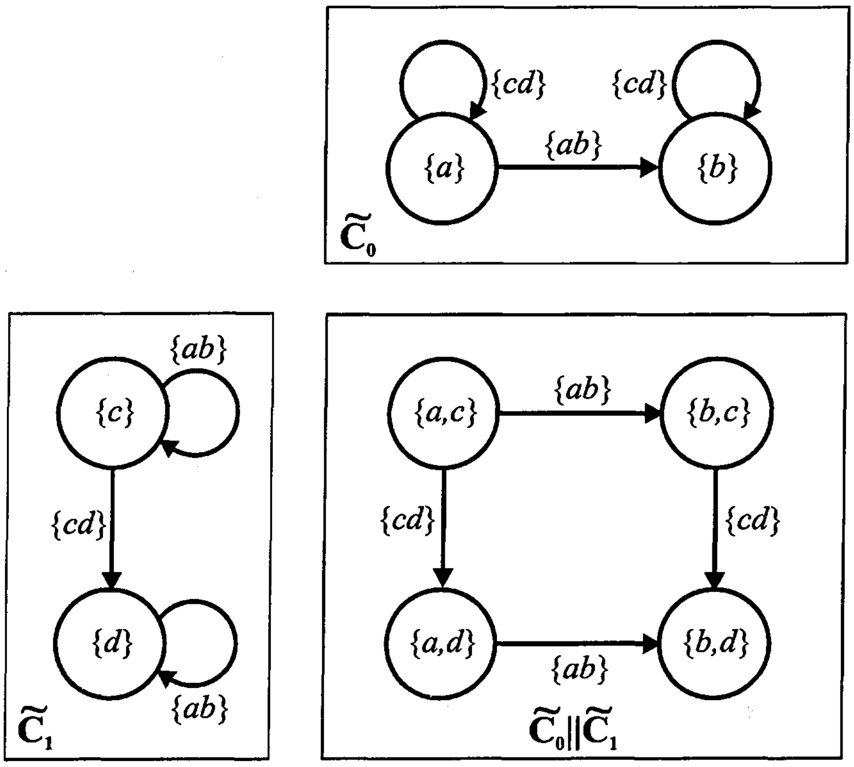


Figure 3.6: $\widetilde{C_0}$, $\widetilde{C_1}$ and $\widetilde{C_0}||\widetilde{C_1}$ for absent simultaneous transition

Figure 3.6 illustrates $\widetilde{C_0}$, $\widetilde{C_1}$ and their composition $\widetilde{C_0}||\widetilde{C_1}$. Included in $\widetilde{C_0}$ are two state dependent transitions, one from state $\{a\}$ to state $\{a\}$ and one from state $\{b\}$ to state $\{b\}$, both labelled with the C_1 event name $\{cd\}$. Thus the event name $\{cd\}$ has become common to both $\widetilde{C_0}$ and $\widetilde{C_1}$, and hence synchronous. The $\widetilde{C_1}$ transition $(\{c\}, \{cd\}, \{d\})$ can only progress if $\widetilde{C_0}$ is in and stays in state $\{a\}$ or in state $\{b\}$. Hence, for example, the synchronous transition from state $\{b, c\}$ to state $\{b, d\}$ labelled $\{cd\}$ in $\widetilde{C_0}||\widetilde{C_1}$ is formed in place of the transition from state $\{b, c\}$ to state $\{b, d\}$ labelled with the asynchronous event name $\{\gamma_0, cd\}$ of $\widetilde{C_0||C_1}$ in figure 3.5.

Similarly, $\widetilde{C_1}$ includes the state dependent synchronising transitions $(\{c\}, \{ab\}, \{c\})$ and $(\{d\}, \{ab\}, \{d\})$. Thus the $\widetilde{C_0}$ transition $(\{a\}, \{ab\}, \{b\})$ can only progress if $\widetilde{C_1}$ stays in the state $\{c\}$ or state $\{d\}$.

2. Absent Asynchronous Transition. Consider the example of the system in figure 3.7. Again, component C_0 contributes a transition from state $\{a\}$ to state $\{b\}$ by event $\{ab\}$ and C_1 contributes a transition from state $\{c\}$ to state $\{d\}$ by event $\{cd\}$.

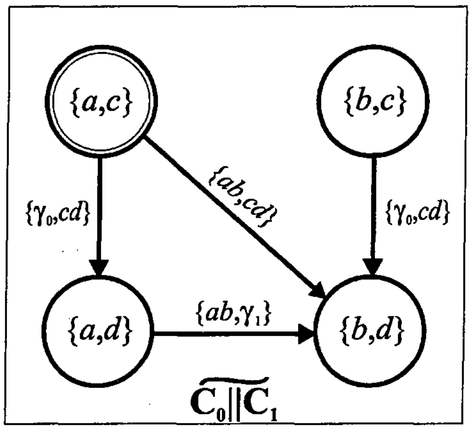


Figure 3.7: $\widetilde{C_0||C_1}$ with absent asynchronous transition

The anticipated asynchronous transition from state $\{a,c\}$ to $\{b,c\}$ labelled $\{ab,\gamma_1\}$ is absent, thus the progression of C_0 from state $\{a\}$ to $\{b\}$ by event $\{ab\}$ while C_1 idles in state $\{c\}$ is denied. However, the progression of C_0 from state $\{a\}$ to $\{b\}$ by event $\{ab\}$ while C_1 stays in state $\{d\}$ is not denied because C_1 is prepared to idle in state $\{d\}$. Hence, the progress of C_0 from state $\{a\}$ to $\{b\}$ is no longer asynchronous as it depends upon the state of C_1.

Figure 3.8 illustrates $\widetilde{C_0}$, $\widetilde{C_1}$ and their composition $\widetilde{C_0}||\widetilde{C_1}$. Included in $\widetilde{C_1}$ is a state dependent transition from state $\{d\}$ to state $\{d\}$ labelled with the C_0 event name $\{ab\}$. Thus the event name $\{ab\}$ has become common to both $\widetilde{C_0}$ and $\widetilde{C_1}$, and hence synchronous. The $\widetilde{C_0}$ transition $(\{a\},\{ab\},\{b\})$ can only progress if $\widetilde{C_1}$ is in and stays in state $\{d\}$. Hence, the synchronous transition from state $\{a,d\}$ to state $\{b,d\}$ labelled $\{ab\}$ in $\widetilde{C_0}||\widetilde{C_1}$ is formed in place of the transition labelled with the asynchronous event name $\{ab,\gamma_1\}$ of $\widetilde{C_0||C_1}$ in figure 3.7. Concurrent composition will not form a transition from $\{a,c\}$ to $\{b,c\}$ labelled $\{ab,\gamma_1\}$ because $\{ab\}$ will not be paired with $\{\gamma_1\}$. Note that transitions labelled $\{\sigma\}$ arise from progressive synchronisation (page 55).

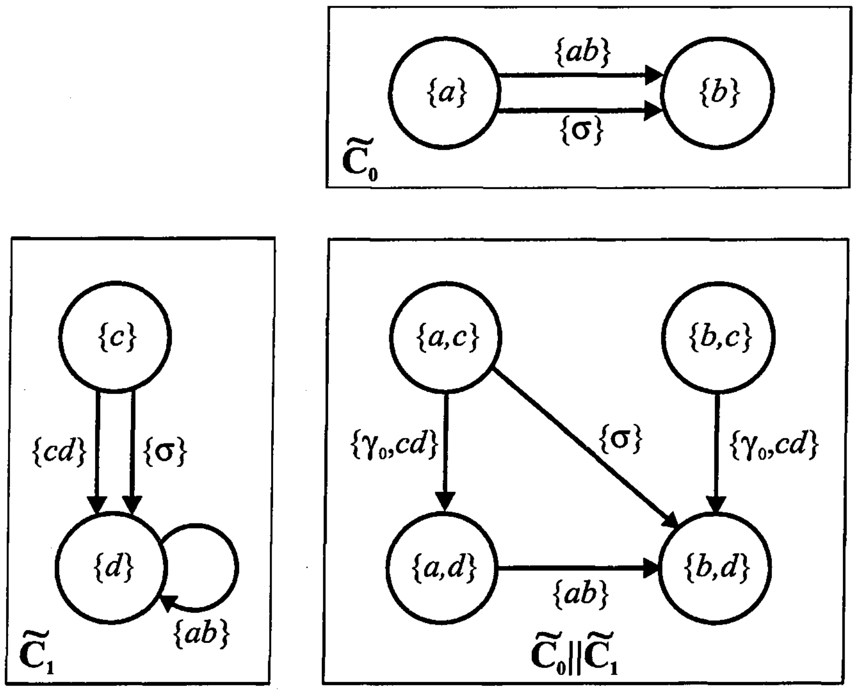


Figure 3.8: $\widetilde{C_0}$, $\widetilde{C_1}$ and $\widetilde{C_0}||\widetilde{C_1}$ for absent asynchronous transition

Progressive Synchronisation

Since the event name $\{ab\}$ of figure 3.7 has become common to $\widetilde{C_0}$ and $\widetilde{C_1}$ due to state dependent synchronisation, the event names $\{ab\}$ and $\{cd\}$ will not be paired and no simultaneous transition from state $\{a, c\}$ to state $\{b, d\}$ labelled $\{ab, cd\}$ will be formed, yet this transition is required because it exists in $\widetilde{C_0||C_1}$ in figure 3.7. As a substitute a simultaneous transition from $\{a, c\}$ to $\{b, d\}$ can be formed by using a new common event $\{\sigma\}$ derived from $\{ab\}$ and $\{cd\}$, as illustrated in figure 3.8. This is called *progressive synchronisation* and requires the extracted component $\widetilde{C_0}$ to include a transition from $\{a\}$ to $\{b\}$ labelled $\{\sigma\}$, and $\widetilde{C_1}$ to include a transition from $\{c\}$ to $\{d\}$ also labelled $\{\sigma\}$. These transitions are illustrated in figure 3.8.

Every composite system reveals the set of events to which the components of that composition will react. Progressive synchronisation introduces a new event name to the event name set, expanding the system interface. Unless the environment of the system is also changed, it is not obvious that any instances of the new event will occur and, therefore, progressive synchronisation transitions may never execute. Instead, interpretation of progressive synchronisation event names is required.

Consider figure 3.8 which illustrates $\widetilde{C_0}$ and $\widetilde{C_1}$, and their concurrent composition $\widetilde{C_0}||\widetilde{C_1}$. The new event name $\{\sigma\}$ asserts that if the events $\{ab\}$ and $\{cd\}$ occur simultaneously then the composite system (and the components) will progress. Now if the events $\{ab\}$ and $\{cd\}$ occur simultaneously when the system is in state $\{a, c\}$, then the events $\{cd\}$ and $\{\sigma\}$ appear to occur simultaneously. Consequently, progressive synchronisation appears to introduce non-determinism.

In an analogous way to the precedence of explicit events over idle events (page 46), let the simultaneous occurrence of both events take precedence over the individual events. For example, if the composite system is in state $\{a, c\}$, then the simultaneous occurrence of the events $\{ab\}$ and $\{cd\}$ causes the system only to progress from state $\{a, c\}$ to $\{b, d\}$ by event $\{\sigma\}$ and not from state $\{a, c\}$ to $\{a, d\}$ by event $\{\gamma_0, cd\}$. This interpretation of progressive synchronisation avoids the apparent introduction of new events to the interface of a system, and avoids the non-deterministic choice often found in other notation.

Comparison of the composite system $\widetilde{C_0}||\widetilde{C_1}$ illustrated in figure 3.8 with the composite system $\widetilde{C_0||C_1}$ illustrated in figure 3.7 shows that both have the same structure, that is, they are congruent, but they are not identical. Therefore the system designer has to determine if the system $\widetilde{C_0}||\widetilde{C_1}$ in fact meets the requirements.

3.2 Formal Definition of a CTS

A Composite Transition System (CTS) is defined as a quin-tuple comprising a *concurrent state* set, $\hat{Q}$, an *initial concurrent state* set, $\ddot{Q}$, a *simultaneous event name* set, $\hat{\Sigma}$, a *simultaneous idle event*, Γ, and a *simultaneous transition* set, $\hat{\Delta}$.

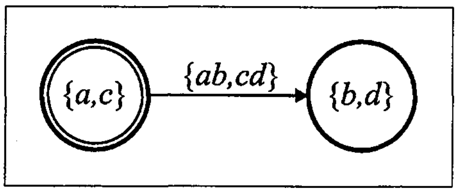


Figure 3.9: Concurrent CTS

Consider the CTS of figure 3.9. The concurrent state set comprises the concurrent states $\{a, c\}$ and $\{b, d\}$, thus $\hat{Q} = \{\{a, c\}, \{b, d\}\}$. The initial state set comprises the initial states, thus $\ddot{Q} = \{\{a, c\}\}$. The event name set comprises the event names, in this case the single event name $\{ab, cd\}$, hence $\hat{\Sigma} = \{\{ab, cd\}\}$. The idle event name cannot be read from figure 3.9, however, for now it is sufficient to state that it is of the form $\{\gamma', \gamma''\}$, noting that it is not, unlike the previous terms, a set of sets. Finally, each transition is written as an ordered triple, the first term defining the *from* state, the second term the event name and the third term the *to* state. Thus the single transition in figure 3.9 is written as $(\{a, c\}, \{ab, cd\}, \{b, d\})$. Hence the transition set is $\hat{\Delta} = \{(\{a, c\}, \{ab, cd\}, \{b, d\})\}$.

In the example of figure 3.9 the states and the event name comprise two identifiers, called *component* identifiers, for example ab and cd in the event name $\{ab, cd\}$. Where there are two or more component identifiers then concurrency and simultaneity are described. However, a component may be described where each state and event name comprises a single component identifier. For example, the state set of C_0 in figure 3.3 (page 47) is $\{\{a\}, \{b\}\}$.

A Composite Transition System is formally defined in Definition 3.1. Much of the motivation for this definition becomes apparent when considering concurrent composition later in this chapter.

Definition 3.1 *CTS Definition.*

$$C \overset{def}{=} (\hat{Q}, \ddot{Q}, \hat{\Sigma}, \Gamma, \hat{\Delta})$$

1. $\hat{Q}$ is the *concurrent state* set, where each concurrent state $Q \in \hat{Q}$ is a set. Each $q \in Q$ is called a *component state identifier.* Both $\hat{Q}$ and Q may be empty.
2. $\ddot{Q}$ is the *initial concurrent state* set such that $\ddot{Q} \subseteq \hat{Q}$, where each initial concurrent state $\dot{Q} \in \ddot{Q}$ is a set. Each $\dot{q} \in \dot{Q}$ is called a *component initial state identifier.* This definition allows a set of initial concurrent states. Both $\ddot{Q}$ and $\dot{Q}$ may be empty.
3. $\hat{\Sigma}$ is the *simultaneous event name* set, where each simultaneous event name $\Sigma \in \hat{\Sigma}$ is a set. Each $\sigma \in \Sigma$ is a called a *component event identifier.* Both $\hat{\Sigma}$ and Σ may be empty.
4. Γ is the *simultaneous idle event name* such that $\Gamma \notin \hat{\Sigma}$. Each $\gamma \in \Gamma$ is called a *component idle event identifier.* By convention simultaneous idle event names should be unique and distinguishable from all other idle event names of every other Composite Transition System in a system.
5. $\hat{\Delta}$ is the *simultaneous transition* set, where $\hat{\Delta} \subseteq \hat{Q} \times \hat{\Sigma} \times \hat{Q}$. Each simultaneous transition $\Delta \in \hat{\Delta}$ is an ordered triple, $(Q, \Sigma, P) \in \hat{Q} \times \hat{\Sigma} \times \hat{Q}$, which, in order, specifies a *from* concurrent state, Q, a simultaneous event name, Σ, and a *to* concurrent state, P. $\hat{\Delta}$ may be empty.

3.2.1 Concurrent State Set $\hat{Q}$

The concurrent state set, $\hat{Q}$, is a set of concurrent states Q. A concurrent state is a set of component identifiers, q, thus $q \in Q$. A concurrent state Q exhibits state concurrency when it specifies two or more component state identifiers, that is $\#Q \geq 2$. The Composite Transition System notation can be also used to describe a non-concurrent component.

The concurrent state set may be empty, that is $\hat{Q} = \{\}$. This property is not useful in describing a realisable system because without any states there can be no transitions and, therefore, no progress. A concurrent state may also be empty, that is $Q = \{\}$. This property is not useful either in describing a realisable system as it contains no identified component state. The value of these properties is in the exploration of the mathematical properties of the calculus of CTS's and this is discussed in Chapter 7. Note that the empty concurrent state is called the *anonymous state.* From set theory, $\{\} \neq \{\{\}\}$, thus if the anonymous state is specified, then the concurrent state set $\hat{Q}$ is not empty because it contains at least the anonymous state, that is $\{\} \in \hat{Q}$.

3.2.2 Initial State Set $\ddot{Q}$

The Composite Transition System notation defines a set of initial states. Three interesting cases arise based on the cardinality of the initial state set. The case of a single initial state, that is $\#\ddot{Q} = 1$, is well understood from Labelled Transition System notations which permit only a single initial state [20, 45, 55, 91]. When a single initial state is specified the *actual* initial state can be uniquely determined from the specified single initial state.

The case of two or more initial states, that is $\#\ddot{Q} \geq 2$, is permitted in the Composite Transition System notation. Some ω-automata notations also define a set of initial states. When two or more initial states are specified then such automata are considered to be non-deterministic [3]. Non-determinism arises because the notation is interpreted as defining the *actual* initial state(s) and not the *possible* initial states. Therefore, it is not possible to determine the *actual* initial state should two or more (possible) initial states be specified.

An alternative interpretation implies that before the first transition the system simultaneously exists in all of the specified initial states. Consequently the system is prepared to execute any transition from any of those states. (After the first transition, a system is considered to exist in exactly one state at any one time and therefore will execute only the transitions from that state.) However, this interpretation does not avoid non-determinism because the behaviour of the model is not defined if any pair of transitions from any two or more of the initial states have a common event. Consider, for example, the transitions $(\{a\}, \{ab\}, \{b\})$ and $(\{c\}, \{ab\}, \{d\})$, where both $\{a\}$ and $\{c\}$ are initial states. Should the common event $\{ab\}$ occur then the model does not determine whether the system progresses to state $\{b\}$ or to state $\{d\}$. Further, if some interpretation is given to each state, for example, the value of a variable x, then the initial states $\{a\}$ and $\{c\}$ might have contradictory interpretations, perhaps $x = 1$ in state $\{a\}$, but $x = 2$ in state $\{c\}$.

One further possible interpretation of initial states is that a system is not in any state and it is the first event, Σ, that places the system momentarily into an initial state, $\dot{Q}$, the *from* state, before the immediate transition $(\dot{Q}, \Sigma, P)$ is executed to progress to the *to* state P. This interpretation only overcomes the contradictory state problem, but does not overcome the problem of non-deterministic behaviour in the case of common events.

In practice, many implemented systems have several *potential* initial states, for example, the initialisation parameters of some software or the setting of configuration registers of some hardware give different initial states. Contrary to the non-determinism assumed in some notations, each of these initial states is often well defined, indeed, it is probably only these initial states that are fully determined.

Removal of the ambiguity over the semantics of initial states is important, and the Composite Transition System notation achieves this by defining the initial state set as the set of possible initial states. Thus the Composite Transition System notation can be used to model implementable systems, where the *actual* initial state is determined by the environment. For example, the environment provides initialisation parameters to software or sets the configuration registers of some hardware.

Finally, for an empty initial state set, that is $\#\ddot{Q} = 0$, then no initial state is defined. Progression of a Composite Transition System requires execution of a transition from the current state. Without an initial state a CTS is never able to progress. However, an empty initial state is of use in extending the behaviour of a CTS by merge composition (section 4.1). Conversely, concurrent composition (section 4.3) with a component without an initial state will result in a CTS without an initial state, and so it is unable to progress. Also, the value of an empty initial state set is in the exploration of the mathematical properties of the calculus of CTS's and this is discussed in Chapter 7.

3.2.3 Simultaneous Event Name Set $\hat{\Sigma}$

The simultaneous event name set, $\hat{\Sigma}$, is a set of simultaneous event names Σ. A simultaneous event name, Σ, is a set of component event identifiers, σ, thus $\sigma \in \Sigma$. A simultaneous event name Σ describes event simultaneity when it specifies two or more component event identifiers, that is $\#\Sigma \geq 2$. However, concurrent composition of event names with common component event identifiers will lead to a CTS which describes synchronised progression but where $\#\Sigma < 2$. The example shown in figure 3.10 (page 63) illustrates a CTS where the components used in its formation have a common event name. In this example $\#\{\sigma, \gamma_1\} = 2$, but $\#\{\sigma\} = 1$.

The simultaneous event name set may be empty, that is $\hat{\Sigma} = \{\}$. This property is not useful in describing a realisable system because without any events there can be no transitions and, therefore, no progress. A simultaneous event name may also be empty, that is $\Sigma = \{\}$. This property is not useful either in describing a realisable system as it contains no identified component event. The value of these properties is in the exploration of the mathematical properties of the calculus of CTS's and this is discussed in Chapter 7. Note that the empty simultaneous event name is called the *anonymous simultaneous event.*

3.2.4 Idle Event Name Γ

The idle event name, Γ, is used in concurrent composition (see section 4.3) to form reflexive *simultaneous idle transitions* that enable a Composite Transition System to describe asynchronous progress. Idle transitions do not, by convention, have to be shown in a diagram of a CTS. Execution of an idle transition does not constitute progress of a CTS, thus idle transitions are reflexive, taking the form (Q, Γ, Q), that is, the *to* and *from* states are the same state. Non-reflexive idle transitions, that is (Q, Γ, P) where $Q \neq P$, are not permitted because they represent progression by an idle event. In other words, a CTS is permitted to progress only with those events in the event set $\hat{\Sigma}$.

Further, the exclusion of the idle event name from the event name set, asserted by $\Gamma \notin \hat{\Sigma}$ in Definition 3.1, prevents the specification of an idle transition and so ensures that there is no confusion with reflexive (but non-idle) transitions of the form (Q, Σ, Q), where $\Sigma \neq \Gamma$. Some notations [3, 45, 57] define transitions in the (CTS) form $Q \times (\Sigma \cup \Gamma) \times Q$, but such a definition requires rules to prohibit the specification of transitions which contain the idle event name. Definition 3.1 obviates the need for such rules.

The idle event name may be empty, that is $\Gamma = \{\}$, and is called the *anonymous idle event*. This property is only of use in describing a realisable system if the anonymous event does not exist, that is, the condition $\Gamma \notin \hat{\Sigma}$ must hold. The value of these properties is in the exploration of the mathematical properties of the calculus of CTS's and this is discussed in Chapter 7.

3.2.5 Simultaneous Transition Set $\hat{\Delta}$

The simultaneous transition set, $\hat{\Delta}$, is a set of simultaneous transitions, Δ, where a simultaneous transition is an ordered triple comprising a *from* concurrent state, a simultaneous event name, and a *to* concurrent state. Thus transitions are written in the form (Q, Σ, P), for example, $(\{a, c\}, \{ab, cd\}, \{b, d\})$ is a transition of the system illustrated in figure 3.9 (page 57).

Some transition system and automata notations allow transitions of the form (Q, Σ, P) and (Q, Σ, P') only if $P = P'$. In other words, no two transitions from a state may have the same event name unless the transitions progress to the same *to* state. This is called the *disambiguation property* and it is often defined to ensure that there is no non-determinism in the *to* state [20]. Disambiguation is not defined for the CTS notation. Non-deterministic Composite Transition Systems can be specified, but, this does not prevent specification of a deterministic CTS. In other words, the CTS notation places no restrictions on the possible behaviours that may be described, rather, any restrictions are determined by the form of the specific description.

Consider the possible progression of the following two examples. The first, illustrated in figure 3.10, implies two components but because the asynchronous transitions $(\{a,c\},\{\sigma,\gamma_1\},\{b,c\})$ and $(\{a,c\},\{\gamma_0,\sigma\},\{a,d\})$ exist, the common event $\{\sigma\}$ has not been treated as synchronising. The treatment of shared events as synchronising is a convention adopted in the definition of the concurrent composition operator (section 4.3) but is not required by Definition 3.1. Thus the system of figure 3.10 may be specified but it cannot be created by the concurrent composition operator.

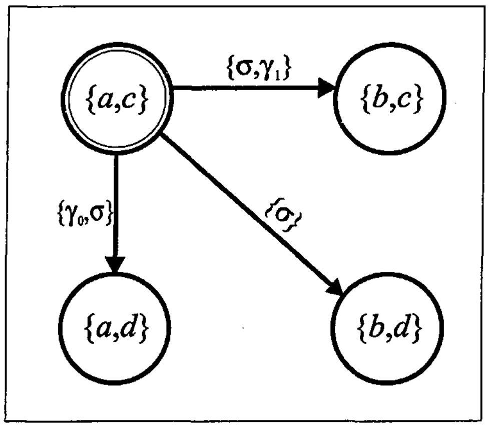


Figure 3.10: Common event, asynchronous progress

This example system is deterministic because the same *from* state is found in all three transitions and no two of the three share the same event name, that is, $\{\sigma\} \neq \{\sigma,\gamma_1\} \neq \{\gamma_0,\sigma\}$. Further, the system is deterministic because priority is always given to the execution of non-idle events over idle events. Hence, the common event $\{\sigma\}$, as a C_0 event, takes priority over the C_0 idle event $\{\gamma_0\}$. Similarly, the common event $\{\sigma\}$, as a C_1 event, takes priority over the C_1 idle event $\{\gamma_1\}$. Consequently, the transitions $(\{a,c\},\{\sigma,\gamma_1\},\{b,c\})$ and $(\{a,c\},\{\gamma_0,\sigma\},\{a,d\})$ are redundant as they will never execute.

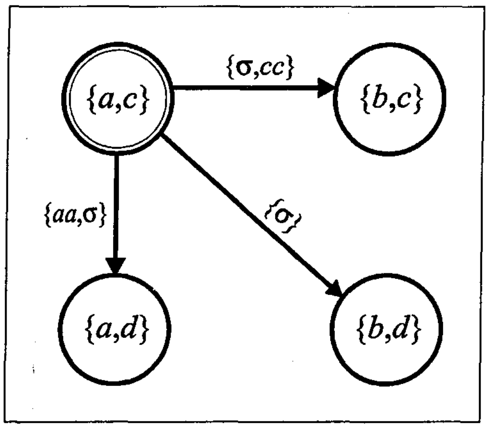


Figure 3.11: Common event, non-synchronous CTS

Now consider the system illustrated in figure 3.11 which, like the previous system, may be specified but cannot be created using the concurrent composition operator. This example is non-deterministic in the sense that it meets the *dis-ambiguation property* because $\{\sigma\} \neq \{\sigma, cc\} \neq \{aa, \sigma\}$. However, if the component events $\{\sigma\}$ and $\{cc\}$ occur simultaneously when in the state $\{a,c\}$ then it is not determined if the system progresses to $\{b,c\}$ or to $\{b,d\}$. In other words, the definition of a CTS means that a system specification can be non-deterministic even if the disambiguation property is met. Any computer based tool might, however, detect and indicate any non-determinism in a system specification.

If the concurrent state set includes the *anonymous concurrent state*, that is $\{\} \in \hat{Q}$, and the simultaneous event name set includes the *anonymous simultaneous event*, that is $\{\} \in \hat{\Sigma}$, then the transition set may include the *anonymous simultaneous transition* $(\{\},\{\},\{\}) \in \hat{\Delta}$. Both the empty simultaneous transition set and the anonymous simultaneous transition are only useful in the exploration of the mathematical properties of the calculus of CTS's and this is discussed in Chapter 7.

3.3 Translation between an LTS and a CTS

This thesis presents the CTS notation for the manipulation of machines and not as an intermediate formulation in the manipulation of Labelled Transition Systems. However, translation of a Labelled Transition System into a Composite Transition System, denoted $C \leftarrow L$, is of value because Labelled Transition Systems have been used to describe the behaviour of implemented systems and recent software design methods, such as the Unified Modelling Language [2, 24], include transition system models. Translation of a Composite Transition System into a Labelled Transition System, denoted $L \leftarrow C$, is shown to be possible only under the condition of no concurrency and no simultaneity. This translation can be useful for making comparisons with existing Labelled Transition System models and with examples published in the literature.

Various formal definitions of a Labelled Transition System can be found, for example, Droste [20], Khendek and Bochman [45] and Stark [91]. Stark defines a Labelled Transition System as the tuple $M = (Q, q_0, \Sigma, \Delta)$. To aid comparison with the definition of a Composite Transition System, the symbols in the definition from Stark have been changed in this thesis to $L = (Q_L, \dot{q}_L, \Sigma_L, \Delta_L)$. The term Q_L is the state set, $\dot{q}_L$ is a distinguished start state (where $\dot{q}_L \in Q_L$), Σ_L is the event set, which does not contain the distinguished event ϵ_L, and $\Delta_L \subseteq Q_L \times (\Sigma_L \cup \{\epsilon_L\}) \times Q_L$ is the transition relation.

A *trace* is an event sequence $\sigma_0, \sigma_1, \ldots, \sigma_n$, for any $\sigma_k \in \Sigma_L$, that may be observed when a machine executes [3]. Note that the events in a trace do include the symbol ϵ_L which represents an unobservable (or internal) event. Such unobservable events are also defined in other notations. Milner's CCS [57], for example, uses unobservable actions (events) as an abstraction of those actions considered irrelevant to the purpose of the model, thus allowing overall simplification of the model. There is no equivalent of the distinguished event ϵ_L in the CTS definition. In translation, ϵ_L could be included in the event set $\hat{\Sigma}$, enabling it to be retained in the CTS model, without imparting any interpretation on ϵ_L.

Translation of an LTS into a CTS is defined in Definition 3.2 and illustrated with the example shown in figure 3.12. Note that an LTS is drawn without a bounding rectangle. In this example, $Q_L = \{a, b\}$, $\dot{q} = a$, $\Sigma_L = \{ab, ba\}$ and $\Delta_L = \{(a, ab, b), (b, ba, a)\}$.

Definition 3.2 *LTS to CTS translation, $C \leftarrow L$.*

$(\hat{Q}, \ddot{Q}, \hat{\Sigma}, \Gamma, \hat{\Delta}) \leftarrow (Q_L, \dot{q}_L, \Sigma_L, \Delta_L)$

1. $\hat{Q} \leftarrow Q_L$ is defined by $\hat{Q} \overset{def}{=} \{\{q\} | q \in Q_L\}$, for example, $\{\{a\}, \{b\}\} \leftarrow \{a, b\}$.
2. $\ddot{Q} \leftarrow \dot{Q}_L$ is defined by $\ddot{Q} \overset{def}{=} \{\{\dot{q}_L\}\}$, for example, $\{\{a\}\} \leftarrow a$.
3. $\hat{\Sigma} \leftarrow \Sigma_L$ is defined by $\hat{\Sigma} \overset{def}{=} \{\{\sigma\} | \sigma \in \Sigma_L \cup \{\epsilon_L\}\}$, for example, $\{\{ab\}, \{ba\}\} \leftarrow \{ab, ba\}$.
4. $\Gamma \overset{def}{=} \{\gamma\}$. Note that the distinguished symbol ϵ_L is not used in this definition.
5. $\hat{\Delta} \leftarrow \Delta_L$ is defined by $\hat{\Delta} \overset{def}{=} \{(\{q\}, \{\sigma\}, \{q'\}) | (q, \sigma, q') \in \Delta_L\}$. Hence, for the given example, $\{(\{a\}, \{ab\}, \{b\}), (\{b\}, \{ba\}, \{a\})\} \leftarrow \{(a, ab, b), (b, ba, a)\}$.

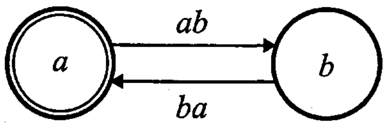
Thus, the LTS and CTS definitions of the example can be written as follows;

$L = (\{a, b\}, a, \{ab, ba\}, \{(a, ab, b), (b, ba, a)\})$

$C = (\{\{a\}, \{b\}\}, \{\{a\}\}, \{\{ab\}, \{ba\}\}, \{\gamma\}, \{(\{a\}, \{ab\}, \{b\}), (\{b\}, \{ba\}, \{a\})\}$

A Composite Transition System, unlike a Labelled Transition System, can describe concurrency and simultaneity. This advantage of a Composite Transition System renders translation of a CTS into an LTS impossible, except in the case where a CTS does not exhibit concurrency and simultaneity. However, translation is briefly explored in this section because of the extensive use of Labelled Transition Systems in modelling systems.

Translation of a CTS into an LTS, denoted $L \leftarrow C$, is straightforward for a Composite Transition System that describes neither concurrency nor simultaneity because every



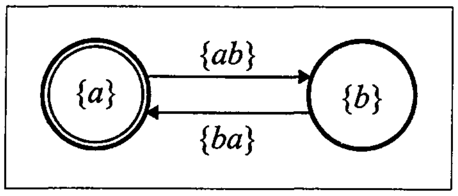


Figure 3.12: Example LTS (top) and its corresponding CTS (bottom)

concurrent state and simultaneous event is itself a singleton set, that is, $\#Q = 1$, for all $Q \in \hat{Q}$, and $\#\Sigma = 1$ for all $\Sigma \in \hat{\Sigma}$. The translation $Q_L \leftarrow \hat{Q}$, can be defined as $Q_L \overset{def}{=} \{q \mid q \in Q \land Q \in \hat{Q}\}$, for example, the Composite Transition System concurrent state set $\hat{Q} = \{\{a\},\{b\}\}$ yields the Labelled Transition System state set $Q_L = \{a, b\}$. Likewise, the translation $\Delta_L \leftarrow \hat{\Delta}$ can be defined as $\Delta_L \overset{def}{=} \{(q,\sigma,p) \mid q \in Q \land \sigma \in \Sigma \land p \in P \land (Q,\Sigma,P) \in \hat{\Delta}\}$. Thus if $\Delta = (\{a\},\{ab\},\{b\})$ then $\Delta_L = (a, ab, b)$. Translation of the other terms follows a similar form.

Since a Labelled Transition System cannot describe simultaneity and concurrency, any translation $L \leftarrow C$ when C exhibits simultaneity and concurrency would necessarily lose information. Any likely translation leads to the result $C' \neq C$, rather than the anticipated $C' = C$, for the operation $C' \leftarrow (L \leftarrow C)$. This problem is of no relevance to the objective of this thesis and will not be explored further.

Chapter 4

Composition

The composition operators presented in this chapter act on Composite Transition Systems and allow expressions incorporating Composite Transition Systems to be written. The *merge composition* and *concurrent composition* operators, respectively denoted by the symbols $+$ and $||$, and expressions based on these operators, denoted by the forms $C' + C''$ and $C'||C''$ respectively, yield Composite Transition Systems. The mathematical properties of these operators are examined in Appendix B (page 196).

A Composite Transition System, C, is a quin-tuple $(\hat{Q}, \ddot{Q}, \hat{\Sigma}, \Gamma, \hat{\Delta})$ formed by some expression incorporating Composite Transition Systems. Let the functions, $\hat{Q}(C)$, $\ddot{Q}(C)$, $\hat{\Sigma}(C)$, $\Gamma(C)$ and $\hat{\Delta}(C)$ respectively give the *concurrent state* set, the *concurrent initial state* set, the *simultaneous event name* set, the *simultaneous idle event name* and the *simultaneous transition* set from C. For example, the function $\hat{Q}(C)$ gives the concurrent state set from the Composite Transition System, C, and the function $\hat{\Sigma}(C'||C'')$ gives the simultaneous event set from the Composite Transition System, $C'||C''$.

The remainder of this chapter is organised as follows. Merge composition is presented in section 4.1 with an example in section 4.2 (page 72). Concurrent composition is presented in section 4.3 (page 74) with an example of the composition of independent components in section 4.4 (page 84), and of dependent components in section 4.5 (page 88).

4.1 Merge Composition

The motivation for defining the merge composition operator, $+$, is to introduce new behaviour to a Composite Transition System, in other words, merge composition is expected to change the behaviour of a CTS by adding states, events and transitions. Merge is important because it provides the basis for the "superposition" of Composite Transition Systems formed under concurrent composition (section 4.3, page 74). The mathematical properties of merge are presented in Appendix B.1 (page 196).

Merge composition of the components C' and C'' is denoted by $C' + C''$, where the components C' and C'' may have common states and may have common events. An example is presented in section 4.2 (page 72).

4.1.1 Merge State Set $\hat{Q}(C' + C'')$

The *merge state* set, $\hat{Q}(C'+C'')$, of the merge composition of the components C' and C'', is defined in Definition 4.1.

Definition 4.1 *Merge state set.*

$$\hat{Q}(C'+C'') \overset{def}{=} \hat{Q}(C') \cup \hat{Q}(C'')$$

States common to the concurrent state sets of the components, that is $Q \in \hat{Q}(C') \cap \hat{Q}(C'')$, yield a merge composite system that, from the common states, can follow the behaviour of either C' or C''. If there are no common states, that is $\hat{Q}(C') \cap \hat{Q}(C'') = \{\}$, then the merge composite system exhibits only the behaviour of C' if the actual initial state is $\dot{Q}' \in \ddot{Q}(C')$, or only the behaviour of C'' if the actual initial state is $\dot{Q}'' \in \ddot{Q}(C'')$.

4.1.2 Merge Initial State Set $\ddot{Q}(C' + C'')$

The *initial concurrent state* set, $\ddot{Q}(C' + C'')$, of the merge composition of the components C' and C'', is defined in Definition 4.2.

Definition 4.2 *Merge initial state set.*

$$\ddot{Q}(C' + C'') \stackrel{def}{=} \ddot{Q}(C') \cup \ddot{Q}(C'')$$

Definition 3.1 defines the initial state set $\ddot{Q}$ to be a subset of the state set $\hat{Q}$, that is $\ddot{Q} \subseteq \hat{Q}$, thus the merged initial state set $\ddot{Q}(C'+C'')$ must be a subset of the merged state set $\hat{Q}(C'+C'')$, that is $\ddot{Q}(C'+C'') \subseteq \hat{Q}(C'+C'')$. Consider as an example the case where C_0 and C_1 specify the initial state sets $\ddot{Q}(C_0) = \{\{a\}\}$ and $\ddot{Q}(C_1) = \{\{e\}\}$. The merged initial state set $\ddot{Q}(C_0+C_1)$ is $\{\{a\}\} \cup \{\{e\}\} = \{\{a\}, \{e\}\}$. From Definition 4.1, the merge composite state set $\hat{Q}(C_0 + C_1)$ is $\{\dots, \{a\}, \dots\} \cup \{\dots, \{e\}, \dots\} = \{\dots, \{a\}, \{e\}, \dots\}$. Hence $\ddot{Q}(C_0 + C_1) \subseteq \hat{Q}(C_0 + C_1)$ because $\{\{a\}, \{e\}\} \subseteq \{\dots, \{a\}, \{e\}, \dots\}$.

The Composite Transition System notation allows a set of possible initial states. If instead a single initial state had been adopted, as found in the definition of a Labelled Transition System in [91], then the merge operator would have to choose an initial state if the components did not specify the same initial state. For the above example, the merge operator would have to make the choice between $\{a\}$ or $\{e\}$. Thus the merge operator would influence the system description. The CTS notation overcomes this limitation inherent in notations which define only a single initial state.

4.1.3 Merge Event Name Set $\hat{\Sigma}(C' + C'')$

The *simultaneous event name* set, $\hat{\Sigma}(C'+C'')$, of the merge composition of the components C' and C'', is defined in Definition 4.3.

Definition 4.3 *Merge event name set.*

$$\hat{\Sigma}(C' + C'') \overset{def}{=} \hat{\Sigma}(C') \cup \hat{\Sigma}(C'')$$

Unlike the concurrent composition of event names (section 4.3.3, page 76), merge composition does not result in a set of event names where each event name describes simultaneity of events from the components C' and C''. Thus an event name common to both C' and C'' is not considered to be synchronising.

4.1.4 Merge Idle Event Name $\Gamma(C' + C'')$

The *idle event name*, $\Gamma(C' + C'')$, of the merge composition of the components C' and C'', is defined in Definition 4.4 to be a new unique idle event name determined by the function $\mathcal{N}(\Gamma(C') \cup \Gamma(C''))$.

Definition 4.4 *Merge idle event name.*

$$\Gamma(C' + C'') \overset{def}{=} \mathcal{N}(\Gamma(C') \cup \Gamma(C''))$$

Since the idle event name is defined as a set, rather than a set of sets, a definition that followed the form of the event set, thus a definition of the form $\Gamma(C') \cup \Gamma(C'')$, would incorrectly yield an idle event that described simultaneity of the component idle event names. For example, if $\Gamma(C') = \{\gamma'\}$ and $\Gamma(C'') = \{\gamma''\}$ then $\Gamma(C') \cup \Gamma(C'') = \{\gamma', \gamma''\}$, which describes the simultaneity of $\{\gamma'\}$ with $\{\gamma''\}$. However, merge composition does not introduce simultaneity.

4.1.5 Merge Transition Set $\hat{\Delta}(C' + C'')$

The *simultaneous transition* set, $\hat{\Delta}(C' + C'')$, of the merge composition of the components C' and C'', is defined in Definition 4.5.

Definition 4.5 *Merge transition set.*

$$\hat{\Delta}(C' + C'') \stackrel{def}{=} \hat{\Delta}(C') \cup \hat{\Delta}(C'')$$

Since the transition set is defined using the other terms of the CTS, the *from* and *to* states specified in each merge transition must be a member of the merge concurrent state set. Likewise, the event name specified in each merge transition must be a member of the merge event name set. The merge composite transition set is $\hat{\Delta}(C') \cup \hat{\Delta}(C'') \subseteq \hat{Q}(C' + C'') \times \hat{\Sigma}(C' + C'') \times \hat{Q}(C' + C'')$. Thus $\hat{\Delta}(C' + C'')$ is a subset of all possible transitions and there need not be a transition from every *from* state to every *to* state.

4.2 Example of Merge Composition

Merge composition is illustrated with the following example which comprises the two components $C_0 = (\hat{Q}_0, \ddot{Q}_0, \hat{\Sigma}_0, \Gamma_0, \hat{\Delta}_0)$ and $C_1 = (\hat{Q}_1, \ddot{Q}_1, \hat{\Sigma}_1, \Gamma_1, \hat{\Delta}_1)$ defined below and illustrated in figure 4.1.

$$\begin{aligned}
\hat{Q}(C_0) &= \{\{a\}, \{b\}\}, \\
\ddot{Q}(C_0) &= \{\{a\}\}, \\
\hat{\Sigma}(C_0) &= \{\{ab\}, \{ba\}\}, \\
\Gamma(C_0) &= \{\gamma_0\}, \\
\hat{\Delta}(C_0) &= \{(\{a\}, \{ab\}, \{b\}), (\{b\}, \{ba\}, \{a\})\}
\end{aligned}$$

$$\begin{aligned}
\hat{Q}(C_1) &= \{\{e\}, \{f\}\} \\
\ddot{Q}(C_1) &= \{\{e\}\} \\
\hat{\Sigma}(C_1) &= \{\{ee\}, \{ef\}\} \\
\Gamma(C_1) &= \{\gamma_1\} \\
\hat{\Delta}(C_1) &= \{(\{e\}, \{ee\}, \{e\}), (\{e\}, \{ef\}, \{f\})\}
\end{aligned}$$

The merge composite $C_0 + C_1$ is defined below and illustrated in figure 4.2.

$$\begin{aligned}
\hat{Q}(C_0 + C_1) &= \{\{a\}, \{b\}, \{e\}, \{f\}\} \\
\ddot{Q}(C_0 + C_1) &= \{\{a\}, \{e\}\} \\
\hat{\Sigma}(C_0 + C_1) &= \{\{ab\}, \{ba\}, \{ee\}, \{ef\}\} \\
\Gamma(C_0 + C_1) &= \mathcal{N}(\{\gamma_0\} \cup \{\gamma_1\}) \\
\hat{\Delta}(C_0 + C_1) &= \{(\{a\}, \{ab\}, \{b\}), (\{b\}, \{ba\}, \{a\}), \\
&\quad (\{e\}, \{ee\}, \{e\}), (\{e\}, \{ef\}, \{f\})\}
\end{aligned}$$

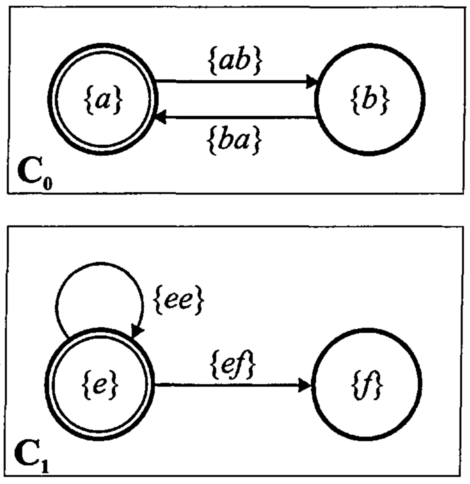


Figure 4.1: C_0 (top) and C_1 (bottom)

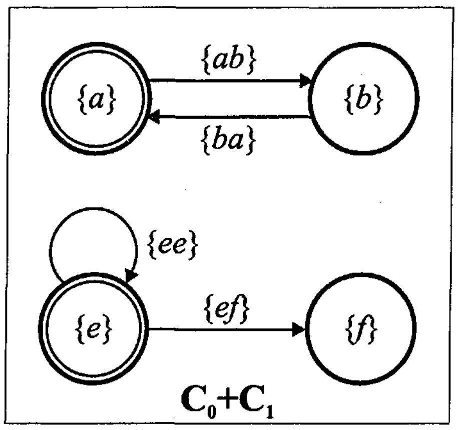


Figure 4.2: $C_0 + C_1$

In this example there are no states common to both components, that is $\hat{Q}_0 \cap \hat{Q}_1 = \{\}$. Merge composition of components with no common states gives a system with no possible transitions between the states contributed by one component and the states contributed by the other component. Thus, if $\hat{Q}(C') \cap \hat{Q}(C'') = \{\}$, then $C' + C''$ exhibits a behaviour the same as C' if the CTS is placed into an initial state contributed by C', or, the same as C'' if the CTS is placed into an initial state contributed by C''. While this example is unrepresentative of the expected use of the merge composition operation, it illustrates the operation and will permit merge to be contrasted with the concurrent composition operation on the same components, performed in section 4.4 (page 84).

4.3 Concurrent Composition

The concurrent composition operator, $||$, constructs the Composite Transition System of a system comprised of components that exhibit asynchronous, simultaneous (co-incidental) and synchronous progress. Indeed, much of the motivation of the formal definition of a Composite Transition System given in Definition 3.1 (page 57) becomes apparent in considering concurrent composition. Concurrent composition of the components C' and C'' is denoted by $C'||C''$.

Under concurrent composition, pairs of terms from the components C' and C'' are formed and the concurrent composite system is the merge composition of all the pairings. These pairings describe state concurrency and event simultaneity. The components should not have common states, that is $\hat{Q}(C') \cap \hat{Q}(C'') = \{\}$. The components may have common event names, that is $\Sigma \in \hat{\Sigma}(C') \cap \hat{\Sigma}(C'')$, and the concurrent composition operator will treat these as synchronous, denying the formation of asynchronous and simultaneous events based on them. This treatment of common event names as synchronising is a convention adopted for the concurrent composition operator.

There are three forms of transition formed under concurrent composition. Asynchronous transition pairings and simultaneous transition pairings are formed to describe the independent progress of the components within the composite. This was illustrated in figure 3.3 (page 47). Synchronous transition pairings are formed to describe the dependent progress of the components within the composite system and is illustrated in figure 3.4 (page 49).

4.3.1 Concurrent State Set $\hat{Q}(C'||C'')$

The *concurrent state* set, $\hat{Q}(C'||C'')$, of the concurrent composition of the components C' and C'', is defined in Definition 4.6.

Definition 4.6 *Concurrent state set.*

$$\hat{Q}(C'||C'') \stackrel{def}{=} \{Q' \cup Q'' | Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'')\}$$

Definition 4.6 forms pairings of states Q' with Q'', that is, every state $Q' \in \hat{Q}(C')$ is paired with every state $Q'' \in \hat{Q}(C'')$. For each pairing, the concurrent state is defined as the union of the pairing and this defines the state concurrency of component pairs.

Consider the following example. Let $\hat{Q}(C_0) = \{\{a\}, \{b\}\}$ and $\hat{Q}(C_1) = \{\{c\}, \{d\}\}$, then $\hat{Q}(C_0||C_1) = \{\{a, c\}, \{b, c\}, \{a, d\}, \{b, d\}\}$. Note that the composite state set can also be written as $\hat{Q}(C_0||C_1) = \{\{a, c\}\} \cup \{\{b, c\}\} \cup \{\{a, d\}\} \cup \{\{b, d\}\}$, which, by definition, is isomorphic to the merge composition $\{\{a, c\}\} + \{\{b, c\}\} + \{\{a, d\}\} + \{\{b, d\}\}$.

To avoid ambiguity in the interpretation of a composite state there should not be more than one state name that is common to the components. If $\hat{Q}(C_0) = \{\{a\}, \{b\}\}$ and $\hat{Q}(C_1) = \{\{a\}, \{b\}\}$, then $\hat{Q}(C_0||C_1) = \{\{a\}, \{a, b\}, \{b\}\}$. The single composite state $\{a, b\}$ describes two situations. It describes the state concurrency of C_0 in state $\{a\}$ with C_1 in state $\{b\}$. It also describes state concurrency of C_0 in state $\{b\}$ with C_1 in state $\{a\}$. Ambiguity arises because the state does not determine which of the two situations is a representation of the current state of the components. This ambiguity is avoided by ensuring that there are no common states.

4.3.2 Concurrent Initial State Set $\ddot{Q}(C'||C'')$

The *initial concurrent state* set, $\ddot{Q}(C'||C'')$, of the concurrent composition of the components C' and C'', is defined in Definition 4.7.

Definition 4.7 *Concurrent initial state set.*

$$\ddot{Q}(C'||C'') \stackrel{def}{=} \{\dot{Q}' \cup \dot{Q}'' | \dot{Q}' \in \ddot{Q}(C') \wedge \dot{Q}'' \in \ddot{Q}(C'')\}$$

Definition 4.7 forms pairings of initial states, and for each pairing defines the initial concurrent state as the union of the elements. From Definition 3.1, the initial state set is defined to be a subset of the concurrent state set, that is $\ddot{Q}(C) \subseteq \hat{Q}(C)$, therefore, the concurrent composite initial state set must be a subset of the concurrent composite state set, that is, $\ddot{Q}(C'\|C'') \subseteq \hat{Q}(C'\|C'')$. This holds because $\ddot{Q}(C') \subseteq \hat{Q}(C')$ and $\ddot{Q}(C'') \subseteq \hat{Q}(C'')$.

4.3.3 Concurrent Event Name Set $\hat{\Sigma}(C'\|C'')$

The *simultaneous event name* set, $\hat{\Sigma}(C'\|C'')$, of the concurrent composition of the components C' and C'', is defined in Definition 4.8. Unlike the operation of merge composition of event names (section 4.1.3, page 70), concurrent composition will generate new event names that describe simultaneity.

Asynchronous events are formed by pairing events from the event set of one component with the idle event of the other component. Let such pairings be denoted by $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ and $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$, where $\mathcal{A}$ denotes "asynchronous" and, for example, $\Sigma_{\Sigma',\Gamma''}$ denotes an event pairing of the component events Σ' with Γ''. Both simultaneous and synchronous events are formed by the pairings of events of one component with the events of the other component. Let simultaneous (coincidental) pairings be denoted $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$, hence $\mathcal{C}$ denotes "coincidental", and synchronous pairings be denoted $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$, hence $\mathcal{S}$ denotes "synchronous".

1. $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ pairings.

   The concurrent composition of an event name Σ' with an idle event name Γ'' is formed by set union, that is, $\Sigma' \cup \Gamma''$. However, for the event name Σ' to be asynchronous, the event name Σ' must not be common with any event name $\Sigma'' \in \hat{\Sigma}(C'')$. This suggests that an asynchronous pairing must only be formed if a conjunction of the form $\Sigma' \notin \hat{\Sigma}(C'')$ holds. As an alternative, a conjunction of the form $\Sigma' \cap \Upsilon = \{\}$ is used, where $\Upsilon \in \hat{\Sigma}(C'')$ and Υ is universally quantified, that is, the conjunction must hold for all event names Υ in $\hat{\Sigma}(C'')$. In this section it is sufficient to note that the conjunction $\Sigma' \cap \Upsilon = \{\}$, unlike $\Sigma' \notin \hat{\Sigma}(C'')$, ensures the associative law of concurrent composition, law B.4 (page 201).

Term 4.1 gives the set of $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ asynchronous pairings.

$$\{ \Sigma' \cup \Gamma(C'') \mid \Sigma' \in \hat{\Sigma}(C') \wedge \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \} \tag{4.1}$$

Consider the following example, where $\hat{\Sigma}(C'') = \{\{c\}, \{d\}\}$ and $\Gamma(C'') = \{\gamma''\}$.

(a) Let $\Sigma' = \{a\}$. The conjunction $\Sigma' \cap \Upsilon = \{\}$ holds for all event names $\Upsilon \in \hat{\Sigma}(C'')$ because the component identifier a is neither common with $\{c\}$ nor $\{d\}$ in $\hat{\Sigma}(C'')$. Therefore, the pairing $\mathcal{A}\Sigma_{\{a\},\{\gamma''\}}$ will give the asynchronous event name $\{a, \gamma''\} \in \hat{\Sigma}(C' \| C'')$.

(b) Let $\Sigma' = \{c\}$. The component identifier c is common with the C'' event name $\{c\}$. Therefore, the conjunction $\Sigma' \cap \Upsilon = \{\}$ does not hold for all event names Υ in $\hat{\Sigma}(C'')$, specifically, it does not hold when $\Upsilon = \{c\}$. Thus the asynchronous pairing $\mathcal{A}\Sigma_{\{c\},\{\gamma''\}}$ will not generate an asynchronous event name $\{c, \gamma''\}$ because c is a synchronous event name.

Note that the idle event names are by Definition 3.1 unique and therefore not common. In other words, it is not necessary to determine if the idle event name of one component is common with any event name of the other component.

2. $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$ pairings.

The derivation of this set of asynchronous pairings follows exactly from the derivation of $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$. Term 4.2 gives the set of $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$ asynchronous pairings.

$$\{ \Gamma(C') \cup \Sigma'' \mid \Sigma'' \in \hat{\Sigma}(C'') \wedge \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\} \} \tag{4.2}$$

3. $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ pairings.

Simultaneous (co-incidental) event names are formed by pairing C' event names with C'' event names. Any specific pairing $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ can only form a simultaneous event name $\Sigma' \cup \Sigma''$ if the event name Σ' is not common with any event name $\Upsilon \in \hat{\Sigma}(C'')$, and the event name Σ'' is not common with any event name $\Upsilon \in \hat{\Sigma}(C')$. The form of the constraints follow directly from the derivation of $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ and $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$. Term 4.3 gives the set of $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ simultaneous pairings.

$$\begin{aligned}\{\,\Sigma' \cup \Sigma'' \mid\ & \Sigma' \in \hat{\Sigma}(C') \wedge \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \wedge \\ & \Sigma'' \in \hat{\Sigma}(C'') \wedge \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}\,\}\end{aligned} \tag{4.3}$$

Note that there are two bound variables, both named Υ, where one is bound to the predicate $\Sigma' \cap \Upsilon = \{\}$ and the other to the predicate $\Sigma'' \cap \Upsilon = \{\}$. In other words the two Υ variables are independent.

4. $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ pairings.

Synchronous event names are formed by pairing C' event names with C'' event names. Any specific pairing $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ can only form a synchronous event name $\Sigma' \cup \Sigma''$ if the event names Σ' and Σ'' have a common event identifier. The derivation of the set of synchronous pairings follows from event name pairings only if there is a common component event identifier in that specific pairing. The conjunction $\Sigma' \cap \Sigma'' \neq \{\}$ will hold if the component event identifier σ is common to Σ' and Σ'', that is $\Sigma' \cap \Sigma'' = \{\sigma\}$. Term 4.4 gives the set of $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ synchronous pairings.

$$\{\,\Sigma' \cup \Sigma'' \mid \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma'' \in \hat{\Sigma}(C'') \wedge \Sigma' \cap \Sigma'' \neq \{\}\,\} \tag{4.4}$$

The event name is of the form $\Sigma' \cup \Sigma''$, because Σ' and Σ'' are not necessarily identical. Consider the formation of the event name set $\hat{\Sigma}(C_0 || (C_1 || C_2))$, where $\hat{\Sigma}(C_0) = \{\{a\}\}$, $\hat{\Sigma}(C_1) = \{\{a\}\}$ and $\hat{\Sigma}(C_2) = \{\{c\}\}$. First, the event name set $\hat{\Sigma}(C_1 || C_2)$ evaluates as $\{\{a, c\}, \ldots\}$ due to the pairing $\mathcal{C}\Sigma_{\{a\},\{c\}}$. Second, the event name set $\hat{\Sigma}(C_0 || (C_1 || C_2))$ evaluates as $\{\{a, c\}, \ldots\}$ due to the synchronous pairing $\mathcal{S}\Sigma_{\{a\},\{a,c\}}$. In this latter pairing the component event identifier a is common, but $\{a\} \neq \{a, c\}$. For completeness, the event name set evaluates as $\{\{a, c\}, \{a, \gamma_2\}, \{\gamma_0, \gamma_1, c\}\}$ where the terms γ_n are the component idle event names.

The concurrent composite event name set, given in Definition 4.8, is the union of the sets of asynchronous pairings, terms 4.1 and 4.2, simultaneous pairings, term 4.3, and synchronous pairings, term 4.4.

Definition 4.8 *Concurrent event name set.*

$$\begin{array}{rlll}
\hat{\Sigma}(C' \| C'') & \overset{def}{=} \{ & \Sigma' \cup \Gamma(C'') & \mid \Sigma' \in \hat{\Sigma}(C') \wedge \\
 & & & \quad \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \} \\
 & \bigcup \{ & \Gamma(C') \cup \Sigma'' & \mid \Sigma'' \in \hat{\Sigma}(C'') \wedge \\
 & & & \quad \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\} \} \\
 & \bigcup \{ & \Sigma' \cup \Sigma'' & \mid \Sigma' \in \hat{\Sigma}(C') \wedge \\
 & & & \quad \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \wedge \\
 & & & \Sigma'' \in \hat{\Sigma}(C'') \wedge \\
 & & & \quad \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\} \} \\
 & \bigcup \{ & \Sigma' \cup \Sigma'' & \mid \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma'' \in \hat{\Sigma}(C'') \wedge \\
 & & & \quad \Sigma' \cap \Sigma'' \neq \{\} \}
\end{array}$$

The following example illustrates the formation of a concurrent event name set. Let $\hat{\Sigma}(C_0) = \{\{ab\}, \{s\}\}$ and $\Gamma_0 = \{\gamma_0\}$. Let $\hat{\Sigma}(C_1) = \{\{cd\}, \{s\}\}$ and $\Gamma_1 = \{\gamma_1\}$. Six of the twelve possible pairings are described in detail. Note that the universal quantification of Υ requires the predicates of the form $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$ to hold true for all values of Υ. Thus, if $\Upsilon \in \hat{\Sigma}(C_1)$, then $\Upsilon \in \{\{cd\}, \{s\}\}$. Hence both $\Sigma' \cap \{cd\} = \{\}$ and $\Sigma' \cap \{s\} = \{\}$ must hold true. This is equivalent to a predicate $\Sigma' \cap \{cd\} = \{\} \wedge \Sigma' \cap \{s\} = \{\}$. Such equivalent predicates are used in the following descriptions.

1. $\mathcal{A}\Sigma_{\{ab\},\{\gamma_1\}}$. The predicate $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$, which is equivalent to $\{ab\} \cap \{cd\} = \{\} \wedge \{ab\} \cap \{s\} = \{\}$, holds true and the asynchronous event name $\{ab, \gamma_1\}$ is formed from this pairing.

2. $\mathcal{A}\Sigma_{\{s\},\{\gamma_1\}}$. The predicate $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$ is equivalent to $\{s\} \cap \{cd\} = \{\} \wedge \{s\} \cap \{s\} = \{\}$. This does not hold true and this pairing does form an event name.

3. $\mathcal{C}\Sigma_{\{ab\},\{cd\}}$. The predicate $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$, which is equivalent to $\{ab\} \cap \{cd\} = \{\} \wedge \{ab\} \cap \{s\} = \{\}$, holds true. Likewise, the predicate $\forall \Upsilon \in \hat{\Sigma}(C') \bullet \Upsilon \cap \Sigma'' =$

$\{\}$, which is equivalent to $\{ab\} \cap \{cd\} = \{\} \wedge \{s\} \cap \{cd\} = \{\}$, holds true. Since the terms are conjoined and both hold true, the simultaneous event name $\{ab, cd\}$ is formed from this pairing.

4. $\mathcal{C}\Sigma_{\{ab\},\{s\}}$. The predicate $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$, which is equivalent to $\{ab\} \cap \{cd\} = \{\} \wedge \{ab\} \cap \{s\} = \{\}$, holds true. However, $\forall \Upsilon \in \hat{\Sigma}(C') \bullet \Upsilon \cap \Sigma' = \{\}$ expands to $\{ab\} \cap \{s\} = \{\} \wedge \{s\} \cap \{s\} = \{\}$, which does not hold. Since the terms are conjoined and only one holds true, no simultaneous event name is formed from this pairing.

5. $\mathcal{S}\Sigma_{\{s\},\{s\}}$. The predicate $\Sigma' \cap \Sigma'' \neq \{\}$ becomes $\{s\} \cap \{s\} \neq \{\}$. This holds true so the simultaneous event name $\{s\}$ is formed from this pairing.

6. $\mathcal{S}\Sigma_{\{s\},\{cd\}}$. The predicate $\Sigma' \cap \Sigma'' \neq \{\}$ becomes $\{s\} \cap \{cd\} \neq \{\}$. This does not hold true and no simultaneous event name is formed from this pairing.

There are a further six pairings to evaluate the composite event name set. Only the pairing $\mathcal{A}\Sigma_{\{cd\},\{\gamma_0\}}$ holds, giving the asynchronous event name $\{\gamma_0, cd\}$. None of the remainder, $\mathcal{A}\Sigma_{\{s\},\{\gamma_0\}}$, $\mathcal{C}\Sigma_{\{s\},\{cd\}}$, $\mathcal{C}\Sigma_{\{s\},\{s\}}$, $\mathcal{S}\Sigma_{\{ab\},\{cd\}}$ nor $\mathcal{S}\Sigma_{\{s\},\{ab\}}$ hold, hence the concurrent composite event name set is $\hat{\Sigma}(C' \| C'') = \{\{ab, \gamma_1\}, \{ab, cd\}, \{s\}, \{\gamma_0, cd\}\}$.

4.3.4 Concurrent Idle Event Name $\Gamma(C' \| C'')$

The *idle event name*, $\Gamma(C' \| C'')$, of the concurrent composition of the components C' and C'', is defined in Definition 4.9. Unlike merge composition of idle event names (section 4.1.4, page 71), the concurrent composite idle event name describes the concurrent idling of the components.

Definition 4.9 *Concurrent idle event name.*

$$\Gamma(C' \| C'') \stackrel{def}{=} \Gamma(C') \cup \Gamma(C'')$$

From Definition 3.1 (page 57), the component idle event names $\Gamma(C')$ and $\Gamma(C'')$ must be unique, that is $\Gamma(C') \cap \Gamma(C'') = \{\}$. Definition 4.9 is sufficient to ensure that $\Gamma(C'\|C'')$ is unique if $\Gamma(C')$ and $\Gamma(C'')$ are unique.

If Definition 3.1 had allowed common idle event names then concurrent composition must not treat them as synchronising as this prevents the description of asynchronous progress. Additionally, interpretational difficulties can arise if the component idle event names are not unique. Consider the following example where $\hat{\Sigma}(C_0) = \{\{\sigma_0\}\}$ and $\Gamma(C_0) = \{\gamma\}$, and $\Gamma(C_1) = \{\gamma\}$. The asynchronous pairing $\mathcal{A}\Sigma_{\{\sigma_0\},\{\gamma\}}$ yields the event name $\{\sigma_0, \gamma\}$. This event name contains more than one event identifier from the component event name set, that is $\{\sigma_0\} \in \hat{\Sigma}(C_0)$ and $\{\gamma\} \in \hat{\Sigma}(C_0)$. Since one of the component identifiers is the idle event name, transitions can be formed where it appears that a component can progress due to an idle event, that is, the component contributed a non-reflexive idle transition to the concurrent composition.

4.3.5 Concurrent Transition Set $\hat{\Delta}(C'\|C'')$

The *concurrent transition* set, $\hat{\Delta}(C'\|C'')$, of the concurrent composition of the components C' and C'', is defined in Definition 4.10. Concurrent composition will generate transitions that describe asynchronous, simultaneous and synchronous progress of the components.

Asynchronous transitions are formed by pairing transitions from the transition set of one component with implied idle transitions of the other component. Idle transitions, which are formed under concurrent composition, are reflexive, taking the form (Q, Γ, Q). Let such pairings be denoted by $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ and $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$. Both simultaneous (coincidental) and synchronous transitions are formed by the pairings of event names of one component with the event names of the other component. Let such pairings be denoted by $\mathcal{CS}\Delta_{\Delta',\Delta''}$.

Concurrent composition adopts the convention that event names common to components are synchronising, consequently not all pairings of event names are included in the event name set $\hat{\Sigma}(C'||C'')$ (section 4.3.3, page 76). The formation of transitions in the concurrent transition set $\hat{\Delta}(C'||C'')$ must therefore be restricted to the event names of the event name set.

1. $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ pairings.

   The concurrent composition that creates the asynchronous pairings of the C' transitions $\Delta' = (Q', \Sigma', P')$ with implied idle transitions of the form $(Q'', \Gamma(C''), Q'')$ is $(Q' \cup Q'', \Sigma' \cup \Gamma(C''), P' \cup Q'')$, provided that the event name $\Sigma' \in \hat{\Sigma}(C')$ does not contain a component event identifier that is common with any event name $\Sigma'' \in \hat{\Sigma}(C'')$. Any common component event name is treated as synchronous so that the event name $\Sigma' \cup \Gamma(C'')$ will not be in the event name set $\hat{\Sigma}(C'||C'')$. Synchronous event names can be excluded by a test of the form $\Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C'||C'')$.

   Term 4.5 gives the set of $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ asynchronous pairings.

$$\begin{aligned} \{\, (Q' \cup Q'', \Sigma' \cup \Gamma(C''), P' \cup Q'') \mid \\ (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q'' \in \hat{Q}(C'') \wedge \\ \Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C'||C'') \,\} \end{aligned} \tag{4.5}$$

   Observe that this set defines the "horizontal" transitions in the concurrent composition $C_0||C_1$ illustrated in figure 3.3 (page 47).

2. $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$ pairings.

   The derivation of this set of asynchronous pairings is similar to the derivation of $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ asynchronous pairings. Term 4.6 gives the set of $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$ asynchronous pairings.

$$\begin{aligned} \{\, (Q' \cup Q'', \Gamma(C') \cup \Sigma'', Q' \cup P'') \mid \\ Q' \in \hat{Q}(C') \wedge (Q'', \Sigma'', P'') \in \hat{\Delta}(C'') \wedge \\ \Gamma(C') \cup \Sigma'' \in \hat{\Sigma}(C'||C'') \,\} \end{aligned} \tag{4.6}$$

   Observe that this set defines the "vertical" transitions in the concurrent composition $C_0||C_1$ illustrated in figure 3.3 (page 47).

3. $\mathcal{CS}\Delta_{\Delta',\Delta''}$ pairings.

The creation of the simultaneous and synchronous transitions from C' transitions $\Delta' = (Q', \Sigma', P')$ and C'' transitions $\Delta'' = (Q'', \Sigma'', P'')$ generates transitions of the form $(Q' \cup Q'', \Sigma' \cup \Sigma'', P' \cup P'')$, provided that the event name pairing $\Sigma' \cup \Sigma''$ is in the event name set $\hat{\Sigma}(C' \| C'')$. From terms 4.3 (page 78) and 4.4 (page 78) it can be seen that such an event name pairing is required for both simultaneous event name pairings, $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$, and synchronous event name pairings, $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$.

Term 4.7 gives the set of $\mathcal{CS}\Delta_{\Delta',\Delta''}$ simultaneous and synchronous pairings.

$$\begin{aligned}\{\, & (Q' \cup Q'', \Sigma' \cup \Sigma'', P' \cup P'') \mid \\ & (Q', \Sigma', P') \in \hat{\Delta}(C') \ \wedge \ (Q'', \Sigma'', P'') \in \hat{\Delta}(C'') \wedge \\ & \Sigma' \cup \Sigma'' \in \hat{\Sigma}(C' \| C'') \,\}\end{aligned} \tag{4.7}$$

Observe that this set defines the "diagonal" transitions in the concurrent composition $C_0 \| C_1$ illustrated in figures 3.3 (page 47) and 3.4 (page 49).

The concurrent composite transition set, given in Definition 4.10, is the union of the sets of the asynchronous pairings, terms 4.5 and 4.6, and the simultaneous and synchronous pairings, term 4.7.

Definition 4.10 *Concurrent transition set.*

$$\begin{aligned}\hat{\Delta}(C' \| C'') \overset{def}{=} & \ \{\, (Q' \cup Q'', \Sigma' \cup \Gamma(C''), P' \cup Q'') \mid \\ & \qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \ \wedge \ Q'' \in \hat{Q}(C'') \wedge \\ & \qquad \Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C' \| C'') \,\} \\ \bigcup & \ \{\, (Q' \cup Q'', \Gamma(C') \cup \Sigma'', Q' \cup P'') \mid \\ & \qquad Q' \in \hat{Q}(C') \ \wedge \ (Q'', \Sigma'', P'') \in \hat{\Delta}(C'') \wedge \\ & \qquad \Gamma(C') \cup \Sigma'' \in \hat{\Sigma}(C' \| C'') \,\} \\ \bigcup & \ \{\, (Q' \cup Q'', \Sigma' \cup \Sigma'', P' \cup P'') \mid \\ & \qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \ \wedge \ (Q'', \Sigma'', P'') \in \hat{\Delta}(C'') \wedge \\ & \qquad \Sigma' \cup \Sigma'' \in \hat{\Sigma}(C' \| C'') \,\}\end{aligned}$$

Including or excluding transition pairings by reference to the event name set, rather than by reference to the event names in the transitions, avoids the duplication of the terms to determine the three forms of pairings. Moreover, it avoids the following problem. From Definition 3.1, transitions are formed using event names from the event name set, however, there are no constraints on the number of transitions labelled with any event name. Consequently an event name set $\hat{\Sigma}_\Delta$ deduced from the transition set, that is $\hat{\Sigma}_\Delta = \{\Sigma \mid (Q, \Sigma, P) \in \hat{\Delta}(C')\}$, may be a subset of the defined event name set, thus $\hat{\Sigma}_\Delta \subseteq \hat{\Sigma}(C')$. If the event name $\{s\}$ is common to $\hat{\Sigma}(C')$ and $\hat{\Sigma}(C'')$ then the composite event name set will include the event name $\{s\}$ but will exclude any event names that describe asynchrony or simultaneous progression with $\{s\}$. For example, the event names $\{\gamma_0, s\}$ and $\{s, \gamma_1\}$ will be excluded. If only one of the components has a transition defined with $\{s\}$ then, from the event names deduced from the transitions, $\{s\}$ would appear to be asynchronous. Thus transitions would be formed enabling asynchronous progress, for example, transitions would be formed with the event names $\{\gamma_0, s\}$ and $\{s, \gamma_1\}$. Thus an event name set deduced by extraction from the transition set would be a superset of the defined event name set.

A practical consequence of the reference to the event name set is that the absence of a transition with a synchronising event name will result in a composite model that will not progress on that event. This scenario can occur in implemented systems where, for example, a process is waiting on a semaphore that never gets signalled. (Ben-Ari in [7] gives an explanation of semaphores and their use in protecting shared resources from concurrent access.)

4.4 Example of Concurrent Composition of Independent Components

Concurrent composition is illustrated with two examples. The example in this section concurrently composes two components that exhibit only asynchrony. Both components are identical to those used to illustrate merge composition in section 4.2 (page 72). In the example of section 4.5 (page 88) the components have the same structure as in this example, but have a common event name which describes synchronisation.

This example forms the concurrent composition of $C_0 = (\hat{Q}_0, \ddot{Q}_0, \hat{\Sigma}_0, \Gamma_0, \hat{\Delta}_0)$ and $C_1 = (\hat{Q}_1, \ddot{Q}_1, \hat{\Sigma}_1, \Gamma_1, \hat{\Delta}_1)$, as specified below. These Composite Transition Systems were first defined in section 4.2 and are illustrated in figure 4.1 (page 73).

$$\begin{aligned} C_0 &= (\{\{a\},\{b\}\}, \{\{a\}\}, \{\{ab\},\{ba\}\}, \{\gamma_0\}, \\ &\quad \{(\{a\},\{ab\},\{b\}),(\{b\},\{ba\},\{a\})\}) \\ C_1 &= (\{\{e\},\{f\}\}, \{\{e\}\}, \{\{ee\},\{ef\}\}, \{\gamma_1\}, \\ &\quad \{(\{e\},\{ee\},\{e\}),(\{e\},\{ef\},\{f\})\}) \end{aligned}$$

From Definition 4.6, each member of the state set $\hat{Q}(C_0||C_1)$ is formed from the set union of each pairing of states from $\hat{Q}(C_0) = \{\{a\},\{b\}\}$ with $\hat{Q}(C_1) = \{\{e\},\{f\}\}$. Likewise, each member of the initial state set $\ddot{Q}(C_0||C_1)$ is formed from the set union of each pairing of initial states from $\ddot{Q}(C_0) = \{\{a\}\}$ with $\ddot{Q}(C_1) = \{\{e\}\}$. Hence the state set and the initial state set of $C_0||C_1$ are as follows;

$$\begin{aligned} \hat{Q}(C_0||C_1) &= \{\{a,e\},\{b,e\},\{a,f\},\{b,f\}\} \\ \ddot{Q}(C_0||C_1) &= \{\{a,e\}\} \end{aligned}$$

Definition 4.8 generates the following forms of pairings in forming the composite event name set $\hat{\Sigma}(C'||C'')$. In this example there are no common component identifiers, thus all terms of the form $\Upsilon \cap \Sigma = \{\}$ hold true in the pairings of the form $\mathcal{A}\Sigma$ and $\mathcal{C}\Sigma$.

1. $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ describes the asynchronous pairings $\mathcal{A}\Sigma_{\{ab\},\{\gamma_1\}}$ and $\mathcal{A}\Sigma_{\{ba\},\{\gamma_1\}}$. These pairings respectively yield $\{ab,\gamma_1\}$ and $\{ba,\gamma_1\}$.

2. $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$ describes the asynchronous pairings $\mathcal{A}\Sigma_{\{ee\},\{\gamma_0\}}$ and $\mathcal{A}\Sigma_{\{ef\},\{\gamma_0\}}$, respectively yielding $\{\gamma_0, ee\}$ and $\{\gamma_0, ef\}$.

3. $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ describes the simultaneous pairings of $\mathcal{C}\Sigma_{\{ab\},\{ee\}}$, $\mathcal{C}\Sigma_{\{ab\},\{ef\}}$, $\mathcal{C}\Sigma_{\{ba\},\{ee\}}$ and $\mathcal{C}\Sigma_{\{ba\},\{ef\}}$, respectively yielding $\{ab,ee\}$, $\{ab,ef\}$, $\{ba,ee\}$ and $\{ba,ef\}$.

4. $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ gives no synchronous pairings since the conjunction $\Sigma' \cap \Sigma'' \neq \{\}$ does not hold for any pairing of $\Sigma' \in \hat{\Sigma}(C')$ with $\Sigma'' \in \hat{\Sigma}(C'')$. This is expected since there are no common event identifiers.

The concurrent composite event name set $\hat{\Sigma}(C_0||C_1)$ is given by Definition 4.8 as the union of the four forms of event name pairings. The concurrent composite idle event name $\Gamma(C_0||C_1)$ is given by Definition 4.9 as the union of the component idle event names $\{\gamma_0\}$ and $\{\gamma_1\}$. Hence the event name set and the idle event name are as follows;

$$\begin{aligned}
\hat{\Sigma}(C_0||C_1) &= \{\{ab,\gamma_1\},\{ba,\gamma_1\}\} \cup \{\{\gamma_0,ee\},\{\gamma_0,ef\}\} \cup \\
&\quad \{\{ab,ee\},\{ab,ef\},\{ba,ee\},\{ba,ef\}\} \cup \{\} \\
&= \{\{ab,\gamma_1\},\{ba,\gamma_1\},\{\gamma_0,ee\},\{\gamma_0,ef\}, \\
&\quad \{ab,ee\},\{ab,ef\},\{ba,ee\},\{ba,ef\}\} \\
\Gamma(C_0||C_1) &= \{\gamma_0,\gamma_1\}
\end{aligned}$$

The transition set, $\hat{\Delta}(C_0||C_1)$, is defined in Definition 4.10. This definition generates the following forms of pairings. Note that since there are no common event identifiers all asynchronous and simultaneous pairings are formed.

1. $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ yields the following asynchronous transitions.

   $\mathcal{A}\Delta_{(\{a\},\{ab\},\{b\}),(\{e\},\{\gamma_1\},\{e\})}$ yields $(\{a,e\},\{ab,\gamma_1\},\{b,e\})$,

   $\mathcal{A}\Delta_{(\{a\},\{ab\},\{b\}),(\{f\},\{\gamma_1\},\{f\})}$ yields $(\{a,f\},\{ab,\gamma_1\},\{b,f\})$,

   $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{\gamma_1\},\{e\})}$ yields $(\{b,e\},\{ba,\gamma_1\},\{a,e\})$ and

   $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\{f\},\{\gamma_1\},\{f\})}$ yields $(\{b,f\},\{ba,\gamma_1\},\{a,f\})$.

2. $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$ yields the following asynchronous transitions.

   $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(\{a\},\{\gamma_0\},\{a\})}$ yields $(\{a,e\},\{\gamma_0,ee\},\{a,e\})$,

   $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(\{b\},\{\gamma_0\},\{b\})}$ yields $(\{b,e\},\{\gamma_0,ee\},\{b,e\})$,

   $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\{a\},\{\gamma_0\},\{a\})}$ yields $(\{a,e\},\{\gamma_0,ef\},\{a,f\})$ and

   $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\{b\},\{\gamma_0\},\{b\})}$ yields $(\{b,e\},\{\gamma_0,ef\},\{b,f\})$.

3. $\mathcal{CS}\Delta_{\Delta',\Delta''}$ yields the following simultaneous transitions. Since there are no common event identifiers in this example, there are no synchronous events.

   $\mathcal{CS}\Delta_{(\{a\},\{ab\},\{b\}),(\{e\},\{ee\},\{e\})}$ yields $(\{a,e\},\{ab,ee\},\{b,e\})$,

   $\mathcal{CS}\Delta_{(\{a\},\{ab\},\{b\}),(\{e\},\{ef\},\{f\})}$ yields $(\{a,e\},\{ab,ef\},\{b,f\})$,

$\mathcal{CS}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{ee\},\{e\})}$ yields $(\{b,e\},\{ba,ee\},\{a,e\})$ and

$\mathcal{CS}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{ef\},\{f\})}$ yields $(\{b,e\},\{ba,ef\},\{a,f\})$.

The concurrent composite transition set $\hat{\Delta}(C_0||C_1)$ given by Definition 4.10 as the union of the three forms of transition pairings given above. Hence the transition set is as follows;

$$\begin{array}{rcll}
\hat{\Sigma}(C_0||C_1) & = & \{ & (\{a,e\},\{ab,\gamma_1\},\{b,e\}), \quad (\{a,f\},\{ab,\gamma_1\},\{b,f\}), \\
& & & (\{b,e\},\{ba,\gamma_1\},\{a,e\}), \quad (\{b,f\},\{ba,\gamma_1\},\{a,f\})\ \} \\
& \cup & \{ & (\{a,e\},\{\gamma_0,ee\},\{a,e\}), \quad (\{b,e\},\{\gamma_0,ee\},\{b,e\}), \\
& & & (\{a,e\},\{\gamma_0,ef\},\{a,f\}), \quad (\{b,e\},\{\gamma_0,ef\},\{b,f\})\ \} \\
& \cup & \{ & (\{a,e\},\{ab,ee\},\{b,e\}), \quad (\{a,e\},\{ab,ef\},\{b,f\}), \\
& & & (\{b,e\},\{ba,ee\},\{a,e\}), \quad (\{b,e\},\{ba,ef\},\{a,f\})\ \} \\
\\
& = & \{ & (\{a,e\},\{ab,\gamma_1\},\{b,e\}), \quad (\{a,f\},\{ab,\gamma_1\},\{b,f\}), \\
& & & (\{b,e\},\{ba,\gamma_1\},\{a,e\}), \quad (\{b,f\},\{ba,\gamma_1\},\{a,f\}), \\
& & & (\{a,e\},\{\gamma_0,ee\},\{a,e\}), \quad (\{b,e\},\{\gamma_0,ee\},\{b,e\}), \\
& & & (\{a,e\},\{\gamma_0,ef\},\{a,f\}), \quad (\{b,e\},\{\gamma_0,ef\},\{b,f\}), \\
& & & (\{a,e\},\{ab,ee\},\{b,e\}), \quad (\{a,e\},\{ab,ef\},\{b,f\}), \\
& & & (\{b,e\},\{ba,ee\},\{a,e\}), \quad (\{b,e\},\{ba,ef\},\{a,f\})\ \}
\end{array}$$

The concurrent composite, $C_0||C_1$, is illustrated in figure 4.3 and can be compared with the merge composition of the same components illustrated in figure 4.2 (page 73).

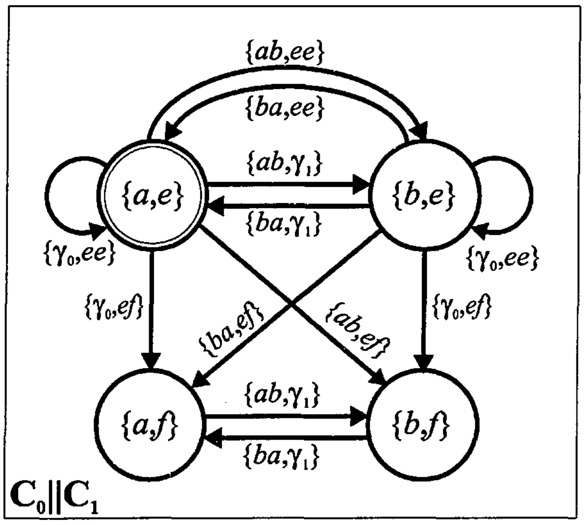


Figure 4.3: Independent $C_0||C_1$

4.5 Example of Concurrent Composition of Dependent Components

In the example of this section the components have the same structure as the components in the example of section 4.4 (page 84), but have a common event name which causes synchronisation. Specifically, the event names in the transitions $(\{a\},\{ab\},\{b\})$ and $(\{e\},\{ef\},\{f\})$ are replaced by the shared event name $\{s\}$, hence C_0 and C_1 are as defined as follows and illustrated in figure 4.4.

$$\begin{aligned}
C_0 &= (\,\{\{a\},\{b\}\},\ \{\{a\}\},\ \{\{s\},\{ba\}\},\ \{\gamma_0\},\\
&\quad\ \{(\{a\},\{s\},\{b\}),(\{b\},\{ba\},\{a\})\}\,)\\
C_1 &= (\,\{\{e\},\{f\}\},\ \{\{e\}\},\ \{\{ee\},\{s\}\},\ \{\gamma_1\},\\
&\quad\ \{(\{e\},\{ee\},\{e\}),(\{e\},\{s\},\{f\})\}\,)
\end{aligned}$$

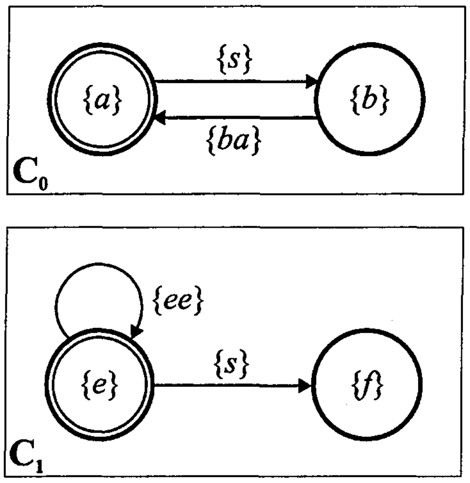


Figure 4.4: C_0 (top) and C_1 (bottom) with shared event $\{s\}$

Since the change of event name does not alter the structure of either C_0 or C_1 then it follows from the previous example that;

$$\begin{aligned}
\hat{Q}(C_0\|C_1) &= \{\{a,e\},\{b,e\},\{a,f\},\{b,f\}\}\\
\ddot{Q}(C_0\|C_1) &= \{\{a,e\}\}\\
\Gamma(C_0\|C_1) &= \{\gamma_0,\gamma_1\}
\end{aligned}$$

Definition 4.8 generates the following forms of pairings in forming the concurrent composite event name set $\hat{\Sigma}(C'||C'')$. In this example some asynchronous and some simultaneous pairings cannot be formed, while some synchronous pairings will be formed.

1. $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ gives the following asynchronous pairings;
   (a) $\mathcal{A}\Sigma_{\{s\},\{\gamma_1\}}$ yields no event name because the event identifier s is common with the event name $\{s\} \in \hat{\Sigma}(C_1)$.
   (b) $\mathcal{A}\Sigma_{\{ba\},\{\gamma_1\}}$ yields the event name $\{ba, \gamma_1\}$ because the event identifier ba is not common with any event name in $\hat{\Sigma}(C_1)$.
2. $\mathcal{A}\Sigma_{\Sigma'',\Gamma'}$ gives the following asynchronous pairings;
   (a) $\mathcal{A}\Sigma_{\{ee\},\{\gamma_0\}}$ yields the event name $\{\gamma_0, ee\}$ because the event identifier ee is not common with any event name in $\hat{\Sigma}(C_0)$.
   (b) $\mathcal{A}\Sigma_{\{s\},\{\gamma_0\}}$ yields no event name because the event identifier s is common with the event name $\{s\} \in \hat{\Sigma}(C_0)$.
3. $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ gives the following simultaneous pairings;
   (a) $\mathcal{C}\Sigma_{\{ba\},\{ee\}}$ yields the event name $\{ba, ee\}$ because the C_0 event identifier ba is not common with any event name of $\hat{\Sigma}(C_1)$, and the C_1 event identifier ee is not common with any event name of $\hat{\Sigma}(C_0)$.
   (b) $\mathcal{C}\Sigma_{\{s\},\{ee\}}$, $\mathcal{C}\Sigma_{\{s\},\{s\}}$ and $\mathcal{C}\Sigma_{\{ba\},\{s\}}$ yield no event names because the event identifier s is common. Note that this $\mathcal{C}\Sigma_{\{s\},\{s\}}$ a synchronous pairing, but $\mathcal{C}\Sigma$ determines simultaneous pairings.
4. $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ gives the following simultaneous pairings;
   (a) $\mathcal{S}\Sigma_{\{s\},\{s\}}$ yields event name $\{s\}$ because the event identifier s is common in this pairing, that is $\{s\} \cap \{s\} = \{s\}$.
   (b) $\mathcal{S}\Sigma_{\{s\},\{ee\}}$, $\mathcal{S}\Sigma_{\{ba\},\{ee\}}$ and $\mathcal{S}\Sigma_{\{ba\},\{s\}}$ yield no event name because there is no common event identifier in these specific pairings, that is $\{s\} \cap \{ee\} = \{\}$, $\{ba\} \cap \{ee\} = \{\}$ and $\{ba\} \cap \{s\} = \{\}$ respectively.

The concurrent composite event name set $\hat{\Sigma}(C_0||C_1)$ given by Definition 4.8 as the union of the sets of the four forms of event name pairings given above. Hence the event name set is as follows;

$$\begin{aligned}\hat{\Sigma}(C_0||C_1) &= \{\{ba,\gamma_1\}\} \cup \{\{\gamma_0, ee\}\} \cup \{\{ba, ee\}\} \cup \{\{s\}\} \\ &= \{\{ba,\gamma_1\}, \{\gamma_0, ee\}, \{ba, ee\}, \{s\}\}\end{aligned}$$

The transition set, $\hat{\Delta}(C_0||C_1)$, is defined in Definition 4.10. This definition generates the following forms of pairings.

1. $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ yields the following asynchronous transitions.
   (a) $\mathcal{A}\Delta_{(\{a\},\{s\},\{b\}),(\{e\},\{\gamma_1\},\{e\})}$ and $\mathcal{A}\Delta_{(\{a\},\{s\},\{b\}),(\{f\},\{\gamma_1\},\{f\})}$ yield no transition because the event name pairing $\{s\} \cup \{\gamma_1\} = \{s, \gamma_1\} \notin \hat{\Sigma}(C_0||C_1)$.
   (b) $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{\gamma_1\},\{e\})}$ and $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\{f\},\{\gamma_1\},\{f\})}$ respectively yield the transitions $(\{b, e\}, \{ba, \gamma_1\}, \{a, e\})$ and $(\{b, f\}, \{ba, \gamma_1\}, \{a, f\})$ because the event name pairing $\{ba\} \cup \{\gamma_1\} = \{ba, \gamma_1\} \in \hat{\Sigma}(C_0||C_1)$.
2. $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$ yields the following asynchronous transitions.
   (a) $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(\{a\},\{\gamma_0\},\{a\})}$ and $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(\{b\},\{\gamma_0\},\{b\})}$ respectively yield the transitions $(\{a, e\}, \{\gamma_0, ee\}, \{a, e\})$ and $(\{b, e\}, \{\gamma_0, ee\}, \{b, e\})$ because the event name pairing $\{\gamma_0\} \cup \{ee\} = \{\gamma_0, ee\} \in \hat{\Sigma}(C_0||C_1)$.
   (b) $\mathcal{A}\Delta_{(\{e\},\{s\},\{e\}),(\{a\},\{\gamma_0\},\{a\})}$ and $\mathcal{A}\Delta_{(\{e\},\{s\},\{e\}),(\{b\},\{\gamma_0\},\{b\})}$ yield no transitions because the event name pairing $\{\gamma_0\} \cup \{s\} = \{\gamma_0, s\} \notin \hat{\Sigma}(C_0||C_1)$.
3. $\mathcal{CS}\Delta_{\Delta',\Delta''}$ yields the following simultaneous and synchronous transitions.
   (a) $\mathcal{CS}\Delta_{(\{a\},\{s\},\{b\}),(\{e\},\{ee\},\{e\})}$ yields no transition because the event name pairing $\{s\} \cup \{ee\} = \{s, ee\} \notin \hat{\Sigma}(C_0||C_1)$.
   (b) $\mathcal{CS}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{ee\},\{e\})}$ yields the transition $(\{b, e\}, \{ba, ee\}, \{a, e\})$ because the event name pairing $\{ba\} \cup \{ee\} = \{ba, ee\} \in \hat{\Sigma}(C_0||C_1)$.

(c) $\mathcal{CS}\Delta_{(\{a\},\{s\},\{b\}),(\{e\},\{s\},\{f\})}$ yields the transition $(\{a,e\},\{s\},\{b,f\})$ because the event name pairing $\{s\} \cup \{s\} = \{s\} \in \hat{\Sigma}(C_0||C_1)$.

(d) $\mathcal{CS}\Delta_{(\{b\},\{ba\},\{a\}),(\{e\},\{s\},\{f\})}$ yields no transition because the event name pairing $\{ba\} \cup \{s\} = \{ba, s\} \notin \hat{\Sigma}(C_0||C_1)$.

The concurrent composite transition set $\hat{\Delta}(C_0||C_1)$ is given by Definition 4.10 as the union of the sets of the three forms of transition pairings given above. Hence the transition set is as follows and $C_0||C_1$ is illustrated in figure 4.5 (*c.f.* figure 4.3 page 87).

$$\begin{array}{rcl}
\hat{\Sigma}(C_0||C_1) & = & \{\ (\{b,e\},\{ba,\gamma_1\},\{a,e\}),\ (\{b,f\},\{ba,\gamma_1\},\{a,f\})\ \} \\
& \cup & \{\ (\{a,e\},\{\gamma_0,ee\},\{a,e\}),\ (\{b,e\},\{\gamma_0,ee\},\{b,e\})\ \} \\
& \cup & \{\ (\{b,e\},\{ba,ee\},\{a,e\}),\ (\{a,e\},\{s\},\{b,f\})\ \} \\
\\
& = & \{\ (\{b,e\},\{ba,\gamma_1\},\{a,e\}),\ (\{b,f\},\{ba,\gamma_1\},\{a,f\}), \\
& & \ \ (\{a,e\},\{\gamma_0,ee\},\{a,e\}),\ (\{b,e\},\{\gamma_0,ee\},\{b,e\}), \\
& & \ \ (\{b,e\},\{ba,ee\},\{a,e\}),\ (\{a,e\},\{s\},\{b,f\})\ \}
\end{array}$$

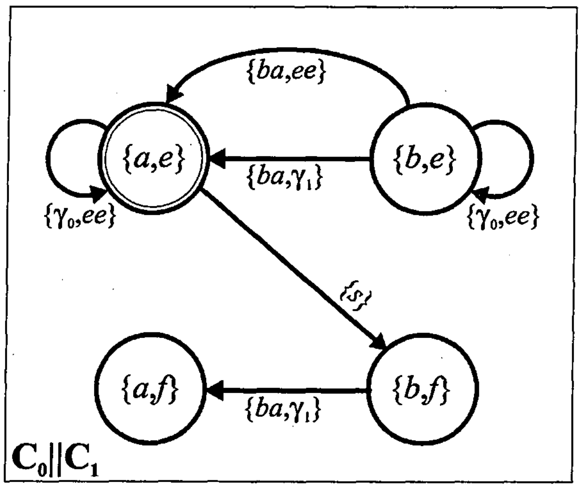


Figure 4.5: $C_0||C_1$ with the shared event $\{s\}$

The interpretation of a common event name as synchronising is the intended consequence of concurrent composition but not the definition of a Composite Transition System. Thus, for example, a transition $(\{a,e\},\{s,ee\},\{b,e\})$ based on an event name $\{s,ee\}$ could be merged with the system illustrated in figure 4.5. This would introduce asynchrony or ambiguity with a shared event name $\{s\}$. For example, the transition $(\{a, e\}, \{s\}, \{b, f\})$ and the merged transition $(\{a, e\}, \{s, ee\}, \{b, e\})$ enables C_0 and C_1 to progress either synchronously for event $\{s\}$, or asynchronously, but simultaneously, with events $\{s\}$ and $\{ee\}$ respectively. This is not denied because in a concurrent composition of the form $(C'||C'')||C'''$, the event name $\{s\}$ may be common only to C' and C'' and not to C'''. This is explored further in Appendix B.2 (page 198).

Observation of figure 4.5 shows that the state $\{b, e\}$ is not reachable. In other words, starting from the initial state $\{a, e\}$ there is no trace that leads to the state $\{b, e\}$. Of course it is possible that $\{b, e\}$ could be an initial state. This shows that concurrent composition (with synchronisation) yields a composite machine that does not take into account the reachability of states. Moreover, it illustrates that the initial state is simply a form of state decoration.

Chapter 5

Extraction

The extraction operator, denoted $\lhd$, generates a Composite Transition System that describes a system component. If $C_{||}$ is taken to be a system specification and C a specification of a prototype component, the extraction operation $C_{||} \lhd C$ generates an *extract* which is a component with a specification that is close to the specification of the prototype C. The specification of the extract can be combined with the specification of another component using concurrent composition to form the system $C_{||}$.

Let a restricted (or modified) form of any Composite Transition System C be denoted $\widetilde{C}$, for example, the restricted form of $C'||C''$ is denoted by $\widetilde{C'||C''}$. Given the system $\widetilde{C'||C''}$, then the extraction operation $(\widetilde{C'||C''}) \lhd C'$ yields the extract $\widetilde{C'}$, which can be taken to be a modified form of C'. Likewise $(\widetilde{C'||C''}) \lhd C''$ yields the extract $\widetilde{C''}$. The components $\widetilde{C'}$ and $\widetilde{C''}$ define the modified behaviour of C' and C'' such that the concurrent composition, $\widetilde{C'}||\widetilde{C''}$, yields a behaviour that is congruent to the restricted system $\widetilde{C'||C''}$. The systems $\widetilde{C'||C''}$ and $\widetilde{C'}||\widetilde{C''}$ are expected to have the same structure, although the interpretation of the events associated with the transitions is likely to be different. Note that where a system has not been restricted, then the extraction operation $(C'||C'') \lhd C'$ will yield C', and the extraction operation $(C'||C'') \lhd C''$ will yield C''.

The remainder of this chapter is organised as follows. The principles of extraction are presented in section 5.1 and the extraction operator is defined in section 5.2 (page 105). An example of extraction applied to a composite system that has not been restricted by system level constraints is presented in section 5.3 (page 124). In section 5.4 (page 130), system level constraints have been applied to an example system and the extraction operation is used to determine the modified components. An analysis of the results is also presented in section 5.4. The mathematical properties of the extraction operator are examined in Appendix B.3 (page 214).

5.1 Principles of Extraction

Extraction proves to be an unexpectedly complicated operation, the complications arising from three significant problems consequent on the definition of the Composite Transition System notation and, more generally, of variants of Labelled Transition System notations.

Definition 3.1 (page 57), allows each event name to label one or more of the explicit transitions of a component. Further, the component idle event name is used to label every implied idle transition of a component. Therefore, the event names and the idle event name do not necessarily identify a specific transition. Concurrent composition forms composite transitions from combinations of the component transitions with the result that some of the composite transitions may be labelled with the same composite event name. In other words, any composite event name can label more than one of the composite transitions. Consequently, there is not a one-to-one relationship between individual transitions of a composite system and the individual transitions of the components of that composition. For an example, consider the system $C_0||C_1$ in figure 3.3 (page 47), in which the composite event name $\{ab, \gamma_1\}$ labels four transitions, although there is only a single transition labelled $\{ab\}$ in C_0. This replication occurs because the component idle event name $\{\gamma_1\}$ labels an implied idle transition on each of the four states $\{e\}$, $\{f\}$, $\{g\}$ and $\{h\}$ of C_1.

Restrictions on the system of components due to resource constraints makes specific transitions invalid and these restrictions are modelled by the removal of the invalid transitions from the composite system. The removal of composite transitions needs to be reflected in modified descriptions of the components, but the required modification is difficult to determine because of the lack of a one-to-one relationship between the individual component transitions and the individual composite transitions. Consider again the system $C_0||C_1$ in figure 3.3 (page 47). The removal of the composite transition from state $\{a, h\}$ to state $\{b, h\}$, labelled with the composite event name $\{ab, \gamma_1\}$, cannot simply lead to the removal of the C_0 transition from state $\{a\}$ to state $\{b\}$ labelled with the event name $\{ab\}$. The consequence would be the failure of concurrent composition to generate the remaining transitions labelled $\{ab, \gamma_1\}$ and the transition labelled $\{ab, fg\}$.

Composite event synchronisation, described in section 5.1.1, is an elaboration of some of the CTS conventions that helps explain the extraction process. The objective is to replace the asynchronous events of the components of a composite system by equivalent sets of synchronous events so the removal of any transition from the composite system eliminates only one transition from each of the components. That is, composite event synchronisation leads to a synchronous representation in which there is a one-to-one relationship between the individual component transitions and the individual composite transitions, thus reducing this first significant problem.

Determining which transitions in the composite require a transition in a particular extract is the second significant problem resulting from the CTS conventions. Composite event synchronisation ensures the concurrent composition of the modified synchronous components will not generate the transitions removed in the design of the modified composite system. Hence, the remaining transitions in the synchronous representations of the components are sufficient to form the restricted composite system and, therefore, form the basis of the transitions in an extract. The solution to this second problem follows from the interpretation and categorisation of each of the (remaining) transitions as either leading to an existing transition of a component of the composition, or leading to a *State Dependent Synchronisation* transition (page 52), or leading to a *Progressive Synchronisation* transition (page 55). This categorisation is presented in section 5.1.2 (page 100).

The third significant problem arises because some transitions in an extract are required because of the absence of other transitions. More specifically, in the extraction operation $\widetilde{C_{||}} \lhd C'$, the transitions of $C_{||}$ that are absent in $\widetilde{C_{||}}$ are not explicitly stated in the description of $\widetilde{C_{||}}$. Instead, the absence of any transition from $\widetilde{C_{||}}$ must be deduced from the presence of related transitions in $\widetilde{C_{||}}$ that would have been formed by the concurrent composition of the components of $C_{||}$. The problem is exacerbated because the system specification $\widetilde{C_{||}}$ does not reveal the components used in its formation. These problems lead to significant complexities in the algebra of the extraction operator.

Although the concept of composite event synchronisation helps explain the principles of extraction, it is, in effect, internal to the extraction operation. Consequently it does not figure explicitly in the algebra of the extraction operator that operates on Composite Transition System descriptions.

5.1.1 Composite Event Synchronisation

Composite event synchronisation represents the components C' and C'' of the composition $C'||C''$ as comprehensively synchronised components denoted D' and D'' respectively. The concurrent composition of D' with D'', denoted $D'||D''$, is required to generate a composite system that describes a behaviour equivalent to that of the composite system $C'||C''$. Additionally, a one-to-one relationship is required between each composite transition of $D'||D''$ and each transition of D' and D''. Hence, for each transition in the composite system $D'||D''$ there will be one transition in D' and one in D''.

Each event name in a CTS may label several transitions, thus an event name can be considered to identify a class of instances of that event name. Let each transition be uniquely renamed. For example, if the event name Σ labels two transitions, then let Σ_0 be the instance in the class identified by Σ that labels one transition, and let Σ_1 be the instance that labels the other transition. Thus, each transition is labelled with a unique event name instance which uniquely identifies the transition.

Similarly, the idle event name Γ of a CTS labels every implied idle transition. Thus the idle event name can be considered to identify a class of instances of that idle event name. Let each instance be uniquely indexed by a component state name. Each idle transition will, therefore, be uniquely labelled because there is only one idle transition per component state. For example, let Γ_Q be the idle event name in the class of idle event names identified by Γ that labels the idle transition on state Q, and let Γ_P be the idle event name in the class identified by Γ that labels the idle transition on state P.

Consider the example illustrated in figure 5.1 (page 98). Each instance of an idle event name Γ_Q is denoted Γ^n_Q, where Q is the instance index and n identifies the component C_n. For example, the C_0 idle event name indexed by the state $\{a\}$ is denoted $\Gamma^0_{\{a\}}$. Similarly, each instance of an event name Σ_i of a component C_n may be denoted Σ^n_i. However, because there is only one instance of each of the C_0 event names A^0 and B^0, and only one instance of the C_1 event name E^1, the instance index will be omitted.

The component C_0 comprises two transitions, one labelled with the event name A^0, and one labelled with the event name B^0. Additionally the state indexed idle event name instances $\Gamma^0_{\{a\}}$ and $\Gamma^0_{\{b\}}$ label the explicit idle transitions on the states $\{a\}$ and $\{b\}$ respectively. The component C_1 comprises a transition labelled with the event name E^1, and the explicit idle transitions are labelled with the state indexed idle event name instances $\Gamma^1_{\{e\}}$ and $\Gamma^1_{\{f\}}$.

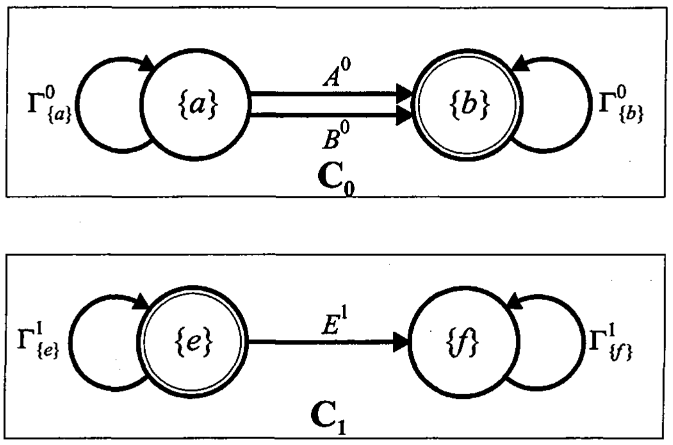


Figure 5.1: C_0 and C_1 with explicit indexed idle transitions.

The event names and transitions in a composition $C'||C''$ are formed from pairings of the component event names and transitions. If every component transition is labelled with a unique event name, then it follows that every composite transition will be labelled with a unique composite event name. For the example components in figure 5.1, the event names of the composition $C_0||C_1$ are $A^0 \cup E^1$, $A^0 \cup \Gamma^1_{\{e\}}$, $A^0 \cup \Gamma^1_{\{f\}}$, $B^0 \cup E^1$, $B^0 \cup \Gamma^1_{\{e\}}$, $B^0 \cup \Gamma^1_{\{f\}}$, $\Gamma^0_{\{a\}} \cup E^1$, $\Gamma^0_{\{b\}} \cup E^1$, $\Gamma^0_{\{a\}} \cup \Gamma^1_{\{e\}}$, $\Gamma^0_{\{a\}} \cup \Gamma^1_{\{f\}}$, $\Gamma^0_{\{b\}} \cup \Gamma^1_{\{e\}}$ and $\Gamma^0_{\{b\}} \cup \Gamma^1_{\{f\}}$. The concurrent composition is shown in figure 5.2 (*cf.* figure 5.9, page 131). Note that the composite reflexive idle transitions are not drawn though it would be useful to do so if the composite system were a component in some larger composition.

Now every component transition Δ is associated with a specific component event name Σ which can be used to define a subset of the composite event name set. Let a component event name Σ define a subset of the composite event name set comprised of just those composite event names that include the component event name Σ. Consider the example of figure 5.2 where the subset defined by the component event name A^0 contains just those composite event names that include A^0, that is $A^0 \cup E^1$, $A^0 \cup \Gamma^1_{\{e\}}$ and $A^0 \cup \Gamma^1_{\{f\}}$. Likewise, the subset defined by the idle event name $\Gamma^1_{\{e\}}$ comprises the composite event names $A^0 \cup \Gamma^1_{\{e\}}$ and $B^0 \cup \Gamma^1_{\{e\}}$.

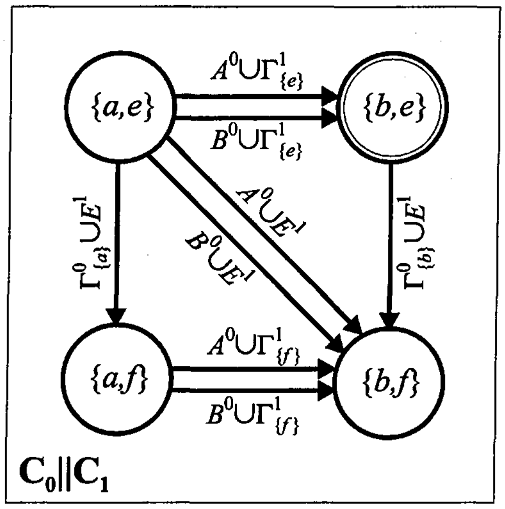

$\{a,e\}$
$\{b,e\}$
$\{a,f\}$
$\{b,f\}$
$A^0 \cup \Gamma^1_{\{e\}}$
$B^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup E^1$
$B^0 \cup E^1$
$\Gamma^0_{\{a\}} \cup E^1$
$\Gamma^0_{\{b\}} \cup E^1$
$A^0 \cup \Gamma^1_{\{f\}}$
$B^0 \cup \Gamma^1_{\{f\}}$
$C_0 || C_1$

Figure 5.2: $C_0 || C_1$ with unique composite events.

Since every component event name identifies a specific component transition, it follows that each composite event name in a subset defined by a component event name Σ identifies the same component transition Δ. Therefore, the component state change labelled by the event name Σ can be replaced by parallel transitions each labelled with one composite event name from the subset of the composite event name set defined by Σ.

Consider the transition labelled A^0 from state $\{a\}$ to state $\{b\}$ in figure 5.1. The event name subset defined by A^0 comprises the composite event names $A^0 \cup E^1$, $A^0 \cup \Gamma^1_{\{e\}}$, $A^0 \cup \Gamma^1_{\{f\}}$, each uniquely identifying a transition in the composite system illustrated in figure 5.2. Thus the C_0 transition labelled A^0 can be replaced by three parallel transitions, one labelled with the composite event name $A^0 \cup E^1$, one with $A^0 \cup \Gamma^1_{\{e\}}$ and one with $A^0 \cup \Gamma^1_{\{f\}}$. In a similar way, the event name subset defined by $\Gamma^1_{\{e\}}$ comprises the composite event names $A^0 \cup \Gamma^1_{\{e\}}$ and $B^0 \cup \Gamma^1_{\{e\}}$. Thus the explicit C_1 idle transition labelled $\Gamma^1_{\{e\}}$ in figure 5.1 can be replaced by two parallel transitions, one labelled with the composite event name $A^0 \cup \Gamma^1_{\{e\}}$, the other with $B^0 \cup \Gamma^1_{\{e\}}$. The other transitions of C_0 and C_1 follow in a similar way. The resulting synchronous components D_0 and D_1 are illustrated in figure 5.3.

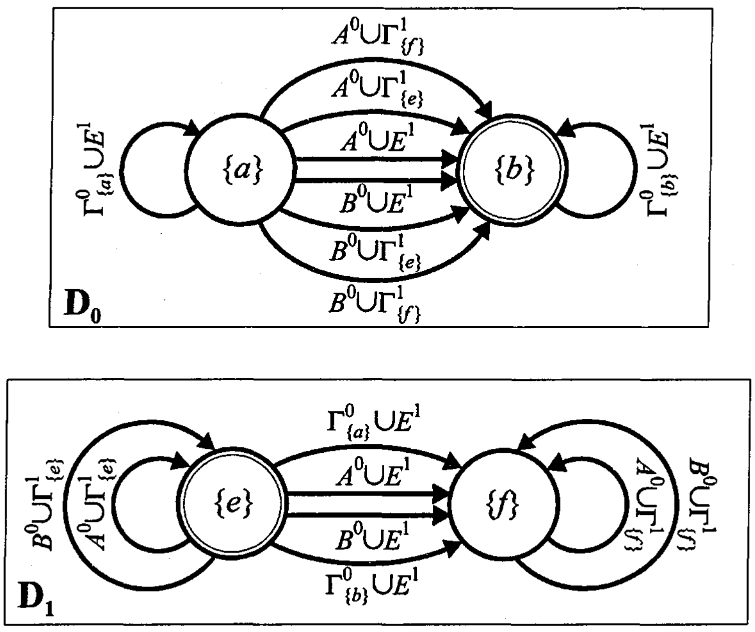

$A^0 \cup \Gamma^1_{\{f\}}$
$A^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup E^1$
$B^0 \cup E^1$
$B^0 \cup \Gamma^1_{\{e\}}$
$B^0 \cup \Gamma^1_{\{f\}}$
$\Gamma^0_{\{a\}} \cup E^1$
$\Gamma^0_{\{b\}} \cup E^1$
$\{a\}$
$\{b\}$
$\mathbf{D_0}$
$\Gamma^0_{\{a\}} \cup E^1$
$A^0 \cup E^1$
$B^0 \cup E^1$
$\Gamma^0_{\{b\}} \cup E^1$
$B^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup \Gamma^1_{\{f\}}$
$B^0 \cup \Gamma^1_{\{f\}}$
$\{e\}$
$\{f\}$
$\mathbf{D_1}$

Figure 5.3: D_0 and D_1 synchronous representations of C_0 and C_1

Composite event synchronisation can be applied to the components of any composite system. The result is the decomposition of each component transition into distinct but synchronous transitions that synchronise with one transition in the other component. As a consequence of the definition of concurrent composition, the concurrent composition of such synchronous transitions generates one transition in the composite system. The removal of a transition from the composite system, therefore, has the effect of removing one transition from each of the components. For example, if the transition labelled $A^0 \cup \Gamma^1_{\{f\}}$ is removed from the composite system of figure 5.2 then the transitions also labelled $A^0 \cup \Gamma^1_{\{f\}}$ must be removed from both the synchronous components in figure 5.3.

5.1.2 Interpretation of the Transitions in a Synchronous Representation

There are three forms of transition in a composite system $C''||C'''$, and these can be distinguished by the form of the composite event name label and thus also in the transition labels in the synchronous representations D' and D'' of the components C' and C''. One form of transition encompasses the original transitions of the component, a second form leads to the *State Dependent Synchronisation* transitions introduced in section 3.1.4 (page 52) and a third form leads to the *Progressive Synchronisation* transitions introduced in section 3.1.4 (page 55).

Original Component Transitions

Transitions of the form $(Q', \Sigma'_m \cup \Gamma''_{Q''}, P')$ describe the progress of D' while the component D'' idles in state Q''. From the conventions on asynchronous progress (section 3.1.1, page 45) and the elaboration of section 5.1.1, an event name of the form $\Sigma'_m \cup \Gamma''_{Q''}$ asserts that if the component event Σ' occurs alone then the component idle event Γ'' will be deemed to have simultaneously occurred, and the composite system will progress. However, the combination of an event name Σ' with the idle event name Γ'' arises as a consequence of concurrent composition. Thus, a transition of the form $(Q', \Sigma'_m \cup \Gamma''_{Q''}, P')$ leads to a transition of the form (Q', Σ', P') in the extract $\widetilde{C_{||}} \lhd C'$, and this is determined by Definition 5.5 (page 110). Observe that any transition of the form (Q', Σ', P') is an explicit transition of the original component C'.

For the example of D_0 illustrated in figure 5.3, both the transitions $(\{a\}, A^0 \cup \Gamma^1_{\{e\}}, \{b\})$ and $(\{a\}, A^0 \cup \Gamma^1_{\{f\}}, \{b\})$ are forms of the C_0 transition $(\{a\}, A^0, \{b\})$, (*cf.* figure 5.1, page 98). Likewise, both the transitions $(\{a\}, B^0 \cup \Gamma^1_{\{e\}}, \{b\})$ and $(\{a\}, B^0 \cup \Gamma^1_{\{f\}}, \{b\})$ are forms of the C_0 transition $(\{a\}, B^0, \{b\})$.

State Dependent Synchronisation Transitions

Transitions of the form $(Q', \Gamma'_{Q'} \cup \Sigma''_n, Q')$ describe the idling of the component D' in state Q' whilst the component D'' progresses by event Σ''_n. The combination of the idle event name Γ' with the event name Σ'' arises as a consequence of the CTS conventions on asynchronous progress (section 3.1.1, page 45). Thus, in an extract of the form $C_{||} \lhd C'$, a transition of the form $(Q', \Gamma' \cup \Sigma'', Q')$ leads to an idle transition of the form (Q', Γ', Q'). Since idle transitions are implied, explicit inclusion in an extract $\widetilde{C_{||}} \lhd C'$ is not required (inclusion is also prohibited by Definition 3.1, page 57).

Now the combination of the idle event name Γ' with an event name Σ'' arises only if Σ'' is determined through a CTS convention on event names to be "asynchronous" to every event of C'. If the condition for asynchrony holds, concurrent composition combines the event Σ'' with every instance of the idle event name $\Gamma'_{Q'}$ of C'. For the example of C_0 illustrated in figure 5.1 (page 98), the C_1 event name E^1 is combined with the C_0 idle event names $\Gamma^0_{\{a\}}$ and $\Gamma^0_{\{b\}}$ to form the composite event names $\Gamma^0_{\{a\}} \cup E^1$ and $\Gamma^0_{\{b\}} \cup E^1$ respectively.

Similarly, when the condition for asynchrony holds, then concurrent composition combines the C'' event name Σ'' with every event name of C' that is determined by the same convention to be "coincidental" to every event of C''. For the example of C_0 illustrated in figure 5.1, the C_1 event name E^1 and C_0 event names A^0 and B^0 are combined to form the composite event names $A^0 \cup E^1$ and $B^0 \cup E^1$. These event combinations each label one transition in each of the synchronous representations D_0 and D_1 which are illustrated in figure 5.3.

The absence of one or more of the transitions labelled with an asynchronous or coincidental combination of any event name Σ'' with the any event name Σ' or Γ' of C' means that the concurrent composition of the extracts $\widetilde{C_{||}} \lhd C'$ and $\widetilde{C_{||}} \lhd C''$ must not generate the absent transitions. This requires the suppression of the absent event combinations, but without additional rules, the asynchronous and coincidental combinations are automatically generated by the concurrent composition operator. The solution is to make the remaining asynchronous combinations synchronous.

Now, every composite event name exposes the set of component events that must occur simultaneously for progress to occur. The idle event name Γ of a component C names an event that implicitly occurs simultaneously with events not named in the event set of C. For the example of figure 5.3 the composite event named $\Gamma^0_{\{a\}} \cup E^1$ occurs when the idle event Γ_0 occurs simultaneously with the C_1 event E^1. Likewise, any composite event name that describes synchronous progress exposes the set of component events that must occur simultaneously. Thus the combination $\Gamma' \cup \Sigma''$ can be considered to be synchronising.

Let those instances of the idle event name $\Gamma'_{Q'}$ in combination with the event name Σ''_n adopt the event name Σ''_n. Thus each transition of the form $(Q', \Gamma'_{Q'} \cup \Sigma''_n, Q')$ becomes a transition of the form (Q', Σ''_n, Q'), which itself is from a class of transitions of the form (Q', Σ'', Q'). These transitions, where the event name Σ'' is now common to the extracts $\widetilde{C_{||}} \lhd C'$ and $\widetilde{C_{||}} \lhd C''$, are the *State Dependent Synchronisation* transitions introduced when an asynchronous or coincidental combination is absent (section 3.1.4, page 52).

If, for example, the asynchronous transition $(\{a\}, \Gamma^0_{\{a\}} \cup E^1, \{a\})$ of D_0 in figure 5.3 was absent, then the transition $(\{b\}, \Gamma^0_{\{b\}} \cup E^1, \{b\})$ leads to the transition $(\{b\}, E^1, \{b\})$ in the extract $\widetilde{C_{||}} \lhd C_0$. If the coincidental transition $(\{a\}, A^0 \cup E^1, \{b\})$ was absent, then both the transitions $(\{a\}, E^1, \{a\})$ and $(\{b\}, E^1, \{b\})$ are contributed to the extract $\widetilde{C_{||}} \lhd C_0$.

Absent asynchronous combinations are determined by Definition 5.8 (page 118) and absent coincidental combinations are determined by Definition 5.9 (page 123).

Progressive Synchronisation Transitions

Transitions of the form $(Q', \Sigma'_m \cup \Sigma''_n, P')$ describe the progress of the component D' by event Σ'_m whilst the component D'' progresses by event Σ''_n. Following the reasoning from the previous section, a transition of the form $(Q', \Sigma'_m \cup \Sigma''_n, P')$ is an instance of a transition of the form $(Q', \Sigma' \cup \Sigma'', P')$. Combinations of an event name Σ' with an event name Σ''

arise only if both the event names are "coincidental". By the same convention, concurrent composition also combines the event name Σ' with every instance of the idle event name $\Gamma'_{Q''}$ of C'', and the event name Σ'' with every instance of the idle event name $\Gamma'_{Q'}$ of C'. These latter two forms are the "asynchronous" event names.

In the previous section on State Dependent Synchronisation, the absence of one or more of the related transitions labelled with an asynchronous event name resulted in synchronisation between the extracts $\widetilde{C_{||}} \lhd C'$ and $\widetilde{C_{||}} \lhd C''$. Concurrent composition of the extracts then generates synchronous combinations in place of the asynchronous combinations, but a consequence of this synchronisation is the suppression of coincidental combinations. In other words, coincidental combinations need also to be explicitly synchronous.

Let the coincidental event name $\Sigma'_m \cup \Sigma''_n$ be replaced with a new unique and explicitly synchronous event name denoted $\mathcal{N}(\Sigma' \cup \Sigma'')_{m,n}$. Therefore, each transition of the form $(Q', \Sigma'_m \cup \Sigma''_n, P')$ in a synchronous representation D' becomes a transition of the form $(Q', \mathcal{N}(\Sigma' \cup \Sigma'')_{m,n}, P')$, which itself is an instance from a class of transitions of the form $(Q', \mathcal{N}(\Sigma' \cup \Sigma''), P')$. These transitions, where the event name $\mathcal{N}(\Sigma' \cup \Sigma'')$ is common to the extracts $\widetilde{C_{||}} \lhd C'$ and $\widetilde{C_{||}} \lhd C''$, are the *Progressive Synchronisation* transitions introduced when an asynchronous combination is absent (section 3.1.4, page 55).

If, for example, the asynchronous transition $(\{a\}, \Gamma^0_{\{a\}} \cup E^1, \{a\})$ of D_0 in figure 5.3 was absent, then the transition $(\{a\}, A^0 \cup E^1, \{b\})$ leads to the transition $(\{a\}, \mathcal{N}(A^0 \cup E^1), \{b\})$ in the extract $\widetilde{C_{||}} \lhd C_0$. Because of the one-to-one relationship between the transitions, the asynchronous transition $(\{e\}, \Gamma^0_{\{a\}} \cup E^1, \{f\})$ of D_1 would also be absent, thus the transition $(\{e\}, A^0 \cup E^1, \{f\})$ leads to the transition $(\{e\}, \mathcal{N}(A^0 \cup E^1), \{f\})$ in the extract $\widetilde{C_{||}} \lhd C_1$.

Absent asynchronous combinations are determined by Definition 5.7 (page 116).

5.2 Extraction Operator

The states, event names and transitions of a composite system $C_{||}$ are formed by the set union of pairings of states, event names and transitions from the components C' and C'', as described in detail in section 4.3 (page 74). The extraction operation $C_{||} \lhd C'$ determines those states, event names and transitions of C' that contributed to the formation of $C_{||}$. A state of C' is said to *exist* in a $C_{||}$ state, if the C' state was used in the formation of a $C_{||}$ state. For example, the C' state $\{a\}$ exists in the $C_{||}$ state $\{a, c\}$, but does not exist in the $C_{||}$ state $\{b, d\}$. Event names are treated in a similar way. A C' transition is said to *exist* in a $C_{||}$ transition if the *from* state, the event name and the *to* state of a C' transition exist, respectively, in the *from* state, the event name and the *to* state of a $C_{||}$ transition. For example, the C' transition $(\{a\}, \{ab\}, \{b\})$ exists in the $C_{||}$ transition $(\{a, f\}, \{ab, fg\}, \{b, g\})$, but does not exist in the $C_{||}$ transition $(\{a, f\}, \{\gamma_0, fg\}, \{a, g\})$.

5.2.1 Extract State Set $\hat{Q}(\widetilde{C_{||}} \lhd C')$

The *extract state set*, $\hat{Q}(\widetilde{C_{||}} \lhd C')$, is defined in Definition 5.1. This definition compares all the states $Q_{||} \in \hat{Q}(\widetilde{C_{||}})$ with states $Q' \in \hat{Q}(C')$ and contributes the state Q' to $\hat{Q}(\widetilde{C_{||}} \lhd C')$ but only if Q' *exists* in $Q_{||}$, that is, if the intersection of Q' and $Q_{||}$ is not empty.

Definition 5.1 *Extract state set.*

$$\hat{Q}(\widetilde{C_{||}} \lhd C') \overset{def}{=} \{\, Q' \mid \exists Q_{||} \bullet (Q_{||} \in \hat{Q}(\widetilde{C_{||}}) \land Q' \in \hat{Q}(C') \land Q_{||} \cap Q' \neq \{\}) \,\}$$

Consider the following example. Let $\hat{Q}(\widetilde{C_{||}}) = \{\{a, c\}\}$ and $\hat{Q}(C') = \{\{a\}, \{b\}\}$. The pairing of $Q_{||} = \{a, c\}$ with $Q' = \{a\}$ gives $Q_{||} \cap Q' = \{a, c\} \cap \{a\} = \{a\}$, thus $\{a\}$ exists in $\{\{a, c\}\}$ and hence $\{a\} \in \hat{Q}(\widetilde{C_{||}} \lhd C')$. However, the pairing of $Q_{||} = \{a, c\}$ with $Q' = \{b\}$ gives $Q_{||} \cap Q' = \{a, c\} \cap \{b\} = \{\}$, thus $\{b\}$ does not exist in $\{\{a, c\}\}$. Thus $\hat{Q}(\widetilde{C_{||}} \lhd C') = \{\{a\}\}$.

When $\widetilde{C_{||}}$ incorporates no modifications it can be replaced with $C_{||}$ and then the simple definition of $\hat{Q}(C_{||} \lhd C') = \hat{Q}(C')$ would suffice. However, as the above example illustrates, the advantage of Definition 5.1 is that it does not generate states in $\hat{Q}(\widetilde{C_{||}} \lhd C')$ that are not in the composite $\widetilde{C_{||}}$ but are in the component C'. This is useful in considering expressions of the form $\hat{Q}(C_{A||B} \lhd C_C)$, that is, the extraction of a machine that is radically different from the components of the system. Observe that the conjunction $Q_{||} \cap Q' \neq \{\}$ has the effect of excluding the *anonymous state*, that is $\{\}$, from the extract state set even if the anonymous state was included in C'. This is discussed in Appendix B.3.

5.2.2 Extract Initial State Set $\ddot{Q}(\widetilde{C_{||}} \lhd C')$

The *extract initial state set*, $\ddot{Q}(\widetilde{C_{||}} \lhd C')$, which is defined in Definition 5.2, follows from Definition 5.1.

Definition 5.2 *Extract initial state set.*

$$\ddot{Q}(\widetilde{C_{||}} \lhd C') \overset{def}{=} \{\dot{Q}' \mid \exists \dot{Q}_{||} \bullet (\dot{Q}_{||} \in \ddot{Q}(\widetilde{C_{||}}) \land \dot{Q}' \in \ddot{Q}(C') \land \dot{Q}_{||} \cap \dot{Q}' \neq \{\})\}$$

The Composite Transition System definition (page 57) defines the initial state set as a subset of the concurrent state set, therefore, the extract initial state set must be a subset of the extract state set, that is, $\ddot{Q}(\widetilde{C_{||}} \lhd C') \subseteq \hat{Q}(\widetilde{C_{||}} \lhd C')$. This holds because $\ddot{Q}(\widetilde{C_{||}}) \subseteq \hat{Q}(\widetilde{C_{||}})$ and $\ddot{Q}(C') \subseteq \hat{Q}(C')$.

5.2.3 Extract Event Name Set $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$

The *extract event name set*, $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$, is defined in Definition 5.3. This definition follows from Definition 5.1. However, the event names of an extract cannot wholly be determined by event name existence since the transitions of an extract (section 5.2.5, page 107), can include *progressive synchronisation* transitions (page 55) and *state dependent synchronisation* transitions (page 52). Specifically, a transition of the extract $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ may include a new event name, or a C'' event. In both cases, the event names exist in neither C' nor $\widetilde{C_{||}}$. Thus the extract event name set $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$ must include event names used in any transition (Q, Σ, P) of the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$. Therefore, the extract transition set must be evaluated before the extract event name set.

Definition 5.3 *Extract event name set.*

$$\begin{aligned}\hat{\Sigma}(\widetilde{C_{||}} \lhd C') \overset{def}{=} & \{\Sigma' \mid \exists \Sigma_{||} \bullet (\Sigma_{||} \in \hat{\Sigma}(\widetilde{C_{||}}) \wedge \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma_{||} \cap \Sigma' \neq \{\})\} \\ \bigcup & \{\Sigma \mid \exists Q, P \bullet (Q, \Sigma, P) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C')\}\end{aligned}$$

5.2.4 Extract Idle Event Name $\Gamma(C_{||} \lhd \widetilde{C_{||}})$

Definition 5.4 defines the extract idle event name as a new unique event name. The extract $\widetilde{C'}$ is a modified form of the operand C' in the extraction operation $\widetilde{C_{||}} \lhd C'$, hence the new name is derived from the idle event name $\Gamma(C')$ of C'.

Definition 5.4 *Extract idle event name.*

$$\Gamma(\widetilde{C_{||}} \lhd C') \overset{def}{=} \mathcal{N}(\Gamma(C'))$$

5.2.5 Extract Transition Set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$

The extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ must include those C' transitions that exist in $\widetilde{C_{||}}$ asynchronous and synchronous transitions. Additionally, the extraction operation $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ must determine and contribute any required *progressive synchronisation* and *state dependent synchronisation* transitions.

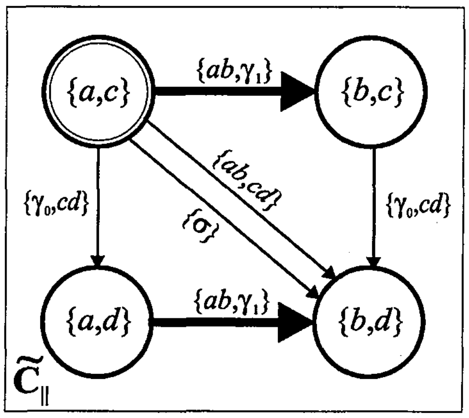
In this section, four different cases of transition generation under the extraction operator are described and then a combined definition for the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ is given in Definition 5.10 (page 123). Each of the cases incorporates a transition existence test, where a transition $(Q', \Sigma', P') \in \hat{\Delta}(C')$ is said to *exist* in a transition $(Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}})$ if the *from* state Q' exists in $Q_{||}$, the event name Σ' exists in $\Sigma_{||}$, and the *to* state P' exists in $P_{||}$.

The following conventions are adopted in this section. First, to aid readability, the terms "vertical", "horizontal" and "diagonal" refer to transitions as drawn in the pictorial representations of a CTS. Second, concurrent composition forms, for example, a concurrent state $Q_{||}$ from the union of Q' and Q'', contributed by C' and C'' respectively. For the extraction operation $\hat{Q}(\widetilde{C_{||}} \lhd C')$ it is necessary to deduce Q'' because the operands of the extraction operator are $\widetilde{C_{||}}$ and C', and not C''. The deduced C'' terms are denoted using calligraphic symbols, hence, $\mathcal{Q}''$, $\mathcal{E}''$ and $\mathcal{I}''$ are the deduced counterparts of Q'', Σ'' and Γ'' respectively.

Extract Asynchronous Transition Set, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$

For a $C_{||}$ transition $\Delta_{||}$ to describe the asynchronous progress of a C' transition Δ', the transition Δ' must have been paired with an implied idle transition of C'' under concurrent composition. Thus, the event of the transition $\Delta_{||}$ will be of the form $\Sigma' \cup \Gamma''$ as a consequence of an event pairing of the form $\mathcal{A}\Sigma_{\Sigma',\Gamma''}$ (page 76).

The extraction of the asynchronous transitions of C' requires a C' transition Δ' to *exist* in a transition $\Delta_{||}$ of $\widetilde{C_{||}}$, provided that the event $\Sigma_{||}$ included in $\Delta_{||}$ is of the form $\Sigma' \cup \Gamma''$. For example, the bold transitions of figure 5.4 illustrate that the C_0 transition $(\{a\}, \{ab\}, \{b\})$ *exists* in the $C_{||}$ asynchronous transitions $(\{a, c\}, \{ab, \gamma_1\}, \{b, c\})$ and $(\{a, d\}, \{ab, \gamma_1\}, \{b, d\})$.



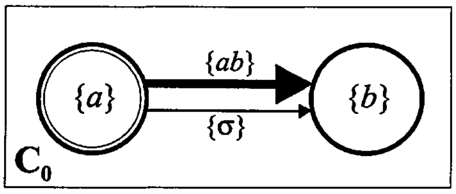


Figure 5.4: Asynchronous extraction of C_0 (right) from $\widetilde{C_{||}}$ (left)

The comparison of the *from* state, event and *to* state of the transitions Δ' and $\Delta_{||}$ take the form of term 5.1.

$$\begin{array}{l}(Q_{||},\Sigma_{||},P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q',\Sigma',P') \in \hat{\Delta}(C') \wedge \\ (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\})\end{array} \tag{5.1}$$

A transition Δ' that exists in a transition $\Delta_{||}$ is contributed to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$ only if C'' contributed the idle event Γ'' to $\Sigma_{||}$. The expression $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$ references $\widetilde{C_{||}}$ and the component C' but not the component C'', specifically it does not reference Γ''. Let the event contributed by C'' to $C_{||}$ be denoted by $\mathcal{E}''$ and let the idle event contributed by C'' to $C_{||}$ be denoted $\mathcal{I}''$. The terms $\mathcal{E}''$ and $\mathcal{I}''$ are evaluated as follows;

1. $\mathcal{E}''$: From Definition 4.8 (page 79), $\Sigma_{||} = \Sigma' \cup \mathcal{E}''$. Using set difference, $\Sigma_{||} - \Sigma' = (\Sigma' \cup \mathcal{E}'') - \Sigma'$ and hence $\Sigma_{||} - \Sigma' = \mathcal{E}''$.

2. $\mathcal{I}''$: From Definition 4.9 (page 80), $\Gamma_{||} = \Gamma' \cup \mathcal{I}''$. Using set difference, $\Gamma_{||} - \Gamma' = (\Gamma' \cup \mathcal{I}'') - \Gamma'$ and hence $\Gamma_{||} - \Gamma' = \mathcal{I}''$.

Therefore, term 5.2 holds true if C'' contributed the idle event Γ''.

$$\exists \mathcal{E}'' = \Sigma_{||} - \Sigma', \mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C') \bullet \mathcal{E}'' = \mathcal{I}'' \tag{5.2}$$

The extract asynchronous transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$, which is defined in Definition 5.5, follows from terms 5.1 and 5.2 and contributes the Δ' transition (Q', Σ', P').

Definition 5.5 *Extract Asynchronous Transition Set* $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$.

$$\begin{array}{lcl}\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A & \overset{def}{=} & \{ (Q', \Sigma', P') \mid \exists\, Q_{||}, \Sigma_{||}, P_{||} \bullet (\\ & & \quad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ & & \quad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\ & & \quad \exists \mathcal{E}'' = \Sigma_{||} - \Sigma',\, \mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C') \bullet \mathcal{E}'' = \mathcal{I}'') \\ & & \}\end{array}$$

Consider the following two cases that can arise with a subsequent concurrent composition of $\widetilde{C_{||}} \lhd C'$ with $\widetilde{C_{||}} \lhd C''$, where a transition Δ' has been contributed to the set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$.

1. In the absence of state dependent synchronisation (page 52), the concurrent composition of $\widetilde{C_{||}} \lhd C'$ with $\widetilde{C_{||}} \lhd C''$ generates asynchronous transitions because of pairings of the form $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')}$ and $\mathcal{A}\Delta_{\Delta'',(Q',\Gamma',Q')}$.

2. In the presence of state dependent synchronisation, the concurrent composition of $\widetilde{C_{||}} \lhd C'$ with $\widetilde{C_{||}} \lhd C''$ generates synchronous transitions as a consequence of pairings of the form $\mathcal{CS}\Delta_{\Delta',\Delta''}$. Asynchronous transitions cannot be generated because the restricted system $\widetilde{C_{||}}$ denies asynchronous progress. Consider again figure 3.8 (page 55). The concurrent composition of the $\widetilde{C_1}$ state dependent synchronisation transition $(\{d\}, \{ab\}, \{d\})$ with the $\widetilde{C_0}$ transition $(\{a\}, \{ab\}, \{b\})$ gives the $\widetilde{C_0} || \widetilde{C_1}$ synchronous transition $(\{a, d\}, \{ab\}, \{b, d\})$ rather than the expected asynchronous transition $(\{a, d\}, \{ab, \gamma_1\}, \{b, d\})$, see figure 3.7 (page 54).

Observe that the absence or presence of state dependent synchronisation has no consequence on the inclusion of a transition Δ' in the extract asynchronous transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$.

Extract Synchronous Transition Set, $\hat{\Delta}(\widetilde{C}_{||} \lhd C')_S$

For a $C_{||}$ transition $\Delta_{||}$ to describe the synchronous progress of a C' transition Δ', the transition Δ' must have been paired, under concurrent composition, with a C'' transition with a shared event. Thus the event of the transition $\Delta_{||}$ will be of the form $\Sigma' \cup \Sigma''$ as a consequence of an event pairing of the form $\mathcal{S}\Sigma_{\Sigma',\Sigma''}$ (page 78).

The extraction of the synchronous transitions of C' requires a C' transition Δ' to *exist* in a transition $\Delta_{||}$ of $\widetilde{C}_{||}$, provided that the event $\Sigma_{||}$ included in $\Delta_{||}$ is of the form $\Sigma' \cup \Sigma''$. For example, the bold transitions of figure 5.5 illustrate the existence of the C_0 transition $(\{a\}, \{\sigma\}, \{b\})$ in the $\widetilde{C}_{||}$ transition $(\{a, c\}, \{\sigma\}, \{b, d\})$.

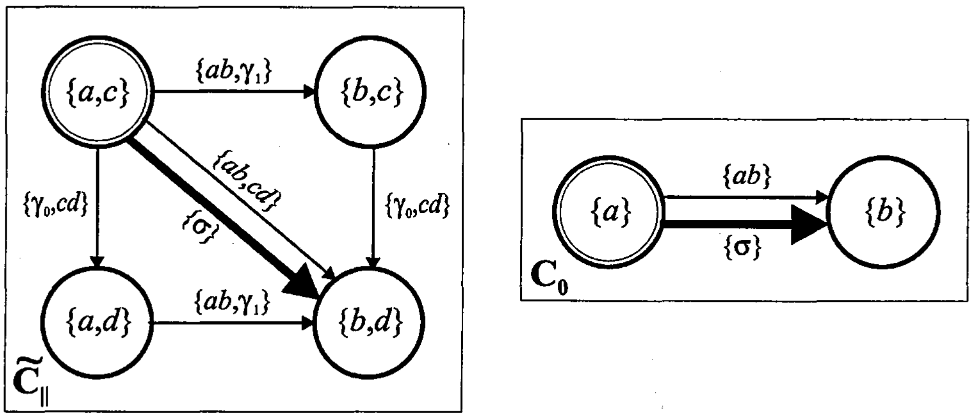


Figure 5.5: Synchronous extraction of C_0 (right) from $\widetilde{C}_{||}$ (left)

The comparison of the *from* state, event and *to* state of the transitions Δ' and $\Delta_{||}$ take the form of term 5.3.

$$\begin{aligned}&(Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C}_{||}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ &(Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\})\end{aligned} \tag{5.3}$$

A transition Δ' that exists in a transition $\Delta_{||}$ can only be contributed to the extract $\hat{\Delta}(\widetilde{C}_{||} \lhd C')_S$ if $\Delta_{||}$ is a synchronous transition. Since the expression $\hat{\Delta}(\widetilde{C}_{||} \lhd C')_S$ only references $\widetilde{C}_{||}$ and the component C', it would seem to follow from the derivation of $\hat{\Delta}(\widetilde{C}_{||} \lhd C')_A$ that a term of the form $\exists \mathcal{E}'' = \Sigma_{||} - \Sigma'$ is required so that a term of the form $\Sigma' = \mathcal{E}''$

can be included to determine a synchronous transition. Instead, term 5.4 will be used to determine if $\Sigma' = \Sigma''$ when $\Sigma_{||}$ was formed and the implications of this will be discussed in section 7.1.1 (page 185).

$$\Sigma_{||} = \Sigma' \tag{5.4}$$

The extract synchronous transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, which is defined in Definition 5.6, follows from terms 5.3 and 5.4 and contributes the Δ' transition (Q', Σ', P').

Definition 5.6 *Extract Synchronous Transition Set* $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$.

$$\begin{array}{lll}\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S & \overset{def}{=} & \{ (Q', \Sigma', P') \mid \exists\, Q_{||}, \Sigma_{||}, P_{||} \bullet (\\ & & \quad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ & & \quad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\ & & \quad \Sigma_{||} = \Sigma') \\ & & \}\end{array}$$

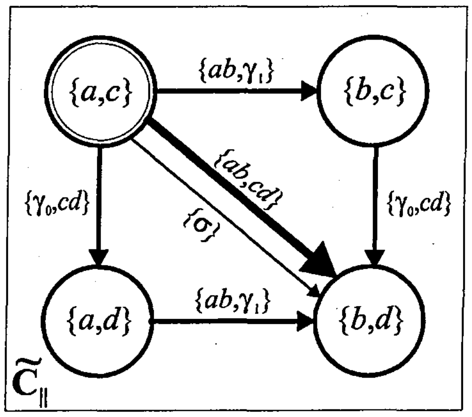
The concurrent composition of the extract $\widetilde{C_{||}} \lhd C'$ with the extract $\widetilde{C_{||}} \lhd C''$ will generate the synchronous transition $\Delta_{||}$. Existing synchronous pairings are not changed by the introduction of progressive or state dependent synchronisation.

Extract Progressive Synchronisation Transition Set, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$

Progressive synchronisation is introduced when a transition $\Delta_{||}$ that describes the simultaneous (coincidental) progress of the components of $C_{||}$ is present in the composite system $C_{||}$, but one or more of the related asynchronous transitions is absent.

For a $C_{||}$ transition $\Delta_{||}$ to describe the simultaneous (coincidental) progress of a C' transition Δ', the transition Δ' must have been paired, under concurrent composition, with a C'' transition without a shared event. Thus the event of the transition $\Delta_{||}$ will be of the form $\Sigma' \cup \Sigma''$, for $\Sigma' \cap \Sigma'' = \{\}$, as a consequence of an event pairing of the form $\mathcal{C}\Sigma_{\Sigma',\Sigma''}$ (page 77).

Determining the need for progressive synchronisation transitions requires a C' transition Δ' to *exist* in a transition $\Delta_{||}$ of $\widetilde{C}_{||}$, provided that the event $\Sigma_{||}$ included in $\Delta_{||}$ is of the form $\Sigma' \cup \Sigma''$. For example, the bold transitions of figure 5.6 illustrate the existence of the C_0 transition $(\{a\}, \{ab\}, \{b\})$, in the $\widetilde{C}_{||}$ transition $(\{a,c\}, \{ab, cd\}, \{b, d\})$.



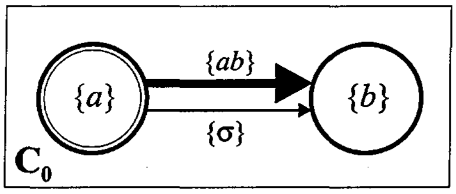


Figure 5.6: Progressive extraction of C_0 (right) from $\widetilde{C}_{||}$ (left)

The comparison of the *from* state, event and *to* state of the transitions Δ' and $\Delta_{||}$ take the form of term 5.5.

$$\begin{array}{l}(Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C}_{||}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\})\end{array} \tag{5.5}$$

Further, it is necessary to determine that a transition Δ' that *exists* in a transition $\Delta_{||}$ describes simultaneous (coincidental) transitions. In other words, asynchronous and synchronous transitions must be ignored. Term 5.2 (page 109) holds true for an asynchronous transition pairing, hence, $\mathcal{E}'' \neq \mathcal{I}''$ will hold true if the transition pairing is not asynchronous. Likewise, term 5.4 (page 112) holds true for a synchronous transition pairing, hence, $\Sigma_{||} \neq \Sigma'$ will hold true if the transition pairing is not synchronous. Hence a term of the form given in term 5.6 determines the simultaneous transitions of $\widetilde{C}_{||}$.

$$\begin{array}{l}(Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C}_{||}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\ \Sigma_{||} \neq \Sigma' \wedge \\ \exists \mathcal{E}'' = \Sigma_{||} - \Sigma', \mathcal{I}'' = \Gamma(\widetilde{C}_{||}) - \Gamma(C') \bullet \mathcal{E}'' \neq \mathcal{I}''\end{array} \tag{5.6}$$

Concurrent composition generates the related asynchronous transitions from pairings of the form $\mathcal{A}\Delta_{\Delta',(S'',\Gamma'',S'')}$ for all $S'' \in \hat{Q}(C'')$ (page 82), and from pairings of the form $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ for all $S' \in \hat{Q}(C')$ (page 82). These asynchronous transitions are, respectively, the horizontal and vertical transitions of $\widetilde{C_{||}}$ in figure 5.6. Recall from figure 3.3 (page 47) that $\hat{Q}(C')$ and $\hat{Q}(C'')$ may contain many states.

Progressive synchronisation, which will synchronise the extract $\widetilde{C_{||}} \lhd C'$ with the extract $\widetilde{C_{||}} \lhd C''$ under concurrent composition, arises only when one or more of the *related* asynchronous transitions is absent. Therefore, it is necessary to determine if a $\mathcal{A}\Delta_{\Delta',(S'',\Gamma'',S'')}$ pairing transition is absent, or if a $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ pairing transition is absent.

1. Absent $\mathcal{A}\Delta_{\Delta',(S'',\Gamma'',S'')}$ pairing transitions.

   For every state $S'' \in \hat{Q}(C'')$, concurrent composition will generate an asynchronous transition of the form $(Q' \cup S'', \Sigma' \cup \Gamma'', P' \cup S'') \in \hat{\Delta}(C_{||})$. Thus a term $\forall \mathcal{S}'' \in \hat{\mathcal{Q}}(\mathcal{C}'') \bullet (Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'') \in \hat{\Delta}(\widetilde{C_{||}})$ would hold true only if all the expected asynchronous transitions exist in $\widetilde{C_{||}}$.

   However, extraction is concerned with the absence of one or more asynchronous transitions. If $p(x)$ is a predicate over x, then by DeMorgan, $\neg\forall x \in X \bullet p(x)$ and $\exists x \in X \bullet \neg p(x)$ are equivalent [52]. Therefore, it follows that a term of the form $\exists \mathcal{S}'' \in \hat{\mathcal{Q}}(\mathcal{C}'') \bullet (Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'') \notin \hat{\Delta}(\widetilde{C_{||}})$ will hold if one or more of the expected asynchronous transitions is absent. The evaluation of $\mathcal{I}''$ is included in term 5.6. The evaluation of $\mathcal{S}''$ requires the evaluation of the state set $\hat{\mathcal{Q}}(\mathcal{C}'')$ rather than a single state. Therefore, $\hat{\mathcal{Q}}(\mathcal{C}'')$ is the set of the set difference of all the pairings of states $S_{||} \in \hat{Q}(\widetilde{C_{||}})$ with the states $S' \in \hat{Q}(C')$, provided S' exists in $S_{||}$.

   Hence, term 5.7 determines if any expected "horizontal" asynchronous transition is absent.

$$\begin{aligned}&\exists \mathcal{S}'' \in \{S_{||} - S' \mid S_{||} \in \hat{Q}(\widetilde{C_{||}}) \wedge S' \in \hat{Q}(C') \wedge S_{||} \cap S' \neq \{\}\} \bullet \\ &\quad (Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'') \notin \hat{\Delta}(\widetilde{C_{||}})\end{aligned} \tag{5.7}$$

2. Absent $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ pairing transitions.

   For every state $S' \in \hat{Q}(C')$, concurrent composition will generate an asynchronous transition $(S' \cup Q'', \Gamma' \cup \Sigma'', S' \cup P'') \in \hat{\Delta}(C_{||})$. It follows from the previous case that a term $\exists S' \in \hat{Q}(C') \bullet (S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}})$ will hold if one or more of the expected asynchronous transitions is absent. The evaluation of $\mathcal{E}''$ is included in term 5.6. The evaluation of $\mathcal{Q}''$ and $\mathcal{P}''$ follow from set difference, hence $\mathcal{Q}'' = Q_{||} - Q'$ and $\mathcal{P}'' = P_{||} - P'$.

   Hence, term 5.8 determines if any expected "vertical" asynchronous transition is absent.

$$\begin{array}{l}\exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{||} - P' \bullet \\ \quad (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}})\end{array} \tag{5.8}$$

Where an asynchronous transition is absent then a transition $(Q', \mathcal{N}(\Sigma' \cup \Sigma''), P')$ is included in the extract $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$. The new event name, $\mathcal{N}(\Sigma' \cup \Sigma'')$, is common to the extract $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$ and the extract $\hat{\Sigma}(\widetilde{C_{||}} \lhd C'')$ to ensure synchronisation under concurrent composition. Hence, the extract progressive synchronisation transition set, which is defined in Definition 5.7, follows from the conjunction of term 5.6, which determines a simultaneous transition, with the disjunction of terms 5.7 and 5.8, which determine absent "horizontal" or "vertical" transitions respectively.

Definition 5.7 *Extract Progressive Synchronisation Transition Set* $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_P$.

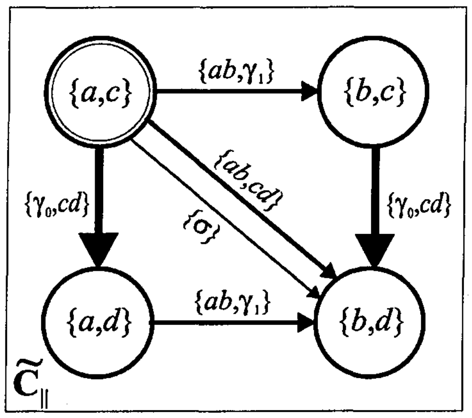
$$
\begin{array}{l}
\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_P \overset{def}{=} \\
\quad \{\, (Q', \mathcal{N}(\Sigma' \cup \mathcal{E}''), P') \mid \exists\, Q_{\|}, \Sigma_{\|}, P_{\|} \bullet (\\
\qquad (Q_{\|}, \Sigma_{\|}, P_{\|}) \in \hat{\Delta}(\widetilde{C_{\|}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
\qquad (Q_{\|} \cap Q' \neq \{\}) \wedge (\Sigma_{\|} \cap \Sigma' \neq \{\}) \wedge (P_{\|} \cap P' \neq \{\}) \wedge \\
\qquad \Sigma_{\|} \neq \Sigma' \wedge \mathcal{E}'' = \Sigma_{\|} - \Sigma' \wedge \\
\qquad \exists \mathcal{I}'' = \Gamma(\widetilde{C_{\|}}) - \Gamma(C') \bullet \\
\qquad\; (\mathcal{E}'' \neq \mathcal{I}'' \wedge \\
\qquad\;\; (\exists \mathcal{S}'' \in \{S_{\|} - S' \mid S_{\|} \in \hat{Q}(\widetilde{C_{\|}}) \wedge S' \in \hat{Q}(C') \wedge S_{\|} \cap S' \neq \{\}\} \bullet \\
\qquad\;\;\; (Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'') \notin \hat{\Delta}(\widetilde{C_{\|}}) \vee \\
\qquad\;\; \exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{\|} - Q',\ \mathcal{P}'' = P_{\|} - P' \bullet \\
\qquad\;\;\; (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{\|}})\,)\,) \\
\qquad)\}
\end{array}
$$

Note that the use of an event of the form $\Sigma' \cup \Sigma''$ rather than $\mathcal{N}(\Sigma' \cup \Sigma'')$ does not give the desired result, for example, if $\{a, b\} = \{a\} \cup \{b\}$ had been used instead of $\mathcal{N}(\{a\} \cup \{b\})$. Let $\hat{\Sigma}_0 = \{\{a\}, \{a, b\}\}$ and $\hat{\Sigma}_1 = \{\{a, b\}\}$, then $\hat{\Sigma}(C_0 \| C_1)$ would incorrectly evaluate as $\{\{a, b\}\}$. Specifically the expected event $\{a, \gamma_1\}$ would be absent because the pairing of $\{a\} \in \hat{\Sigma}_0$ with $\{a, b\} \in \hat{\Sigma}_1$ would be treated as synchronous because of the common component event identifier a.

Extract State Dependent Synchronisation Transition Sets, $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_{DA}$ and $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_{DC}$

The definition of the extract $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_P$, in the previous section, dealt with the introduction progressive synchronisation transitions as a consequence of the denial of asynchronous progress caused by the absence of asynchronous transitions. Likewise, the definition in this section of the extract $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_{DA}$ deals with the introduction of state dependent synchronisation transitions also as a consequence of an absent asynchronous transition. The definition in this section of the extract $\hat{\Delta}(\widetilde{C_{\|}} \lhd C')_{DC}$ also deals with the introduction of state dependent synchronisation transitions but as a consequence of an absent simultaneous (coincidental) transition.

Determining the need for state dependent synchronisation transitions requires a C' implied idle transition of the form (Q', Γ', Q') to *exist* in a transition $\Delta_{||}$ of $\widetilde{C_{||}}$ that describes the asynchronous progress of C'', that is, $\Delta_{||}$ is of the form $(Q' \cup Q'', \Gamma' \cup \Sigma'', Q' \cup P'')$. For example, figure 5.7 illustrates, with the bold directed arcs, the existence of the C_0 implied transitions $(\{a\}, \{\gamma_0\}, \{a\})$ and $(\{b\}, \{\gamma_0\}, \{b\})$, in the $\widetilde{C_{||}}$ transitions $(\{a, c\}, \{\gamma_0, cd\}, \{a, d\})$ and $(\{b, c\}, \{\gamma_0, cd\}, \{b, d\})$ respectively.



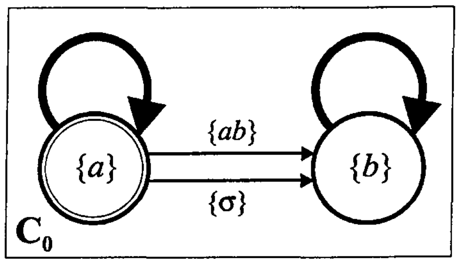


Figure 5.7: State Dependent extraction of C_0 (right) from $\widetilde{C_{||}}$ (left).

The comparison of the *from* state, event and *to* state of the implied idle transitions $(Q', \Gamma(C'), Q')$ of $\hat{Q}(C')$ and the transitions $\Delta_{||}$ of $\widetilde{C_{||}}$ take the form of term 5.9.

$$\begin{aligned}&(Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge Q' \in \hat{\Sigma}(C') \wedge \\ &(Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap Q' \neq \{\})\end{aligned} \tag{5.9}$$

Observe from Definition 3.1 (page 57) that the event $\Sigma_{||}$ cannot be the idle event $\Gamma(\widetilde{C_{||}})$, that is, $\Sigma_{||} \neq \Gamma(C') \cup \Gamma(C'')$. Thus the event Σ'' contributed by C'' to the formation of $\Sigma_{||} = \Gamma(C') \cup \Sigma''$ cannot be the C'' idle event, that is $\Sigma'' \neq \Gamma(C'')$. In other words, it is not necessary to assert that the event contributed by C'' is not the idle event $\Gamma(C'')$. Hence a term of the form given in term 5.9 is sufficient to determine those transitions of $\widetilde{C_{||}}$ that describe the asynchronous progress of C''.

State dependency arises under two conditions and results in the introduction of a reflexive state dependent synchronisation transition of the form $(Q', \mathcal{E}'', Q')$.

1. Absent $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ pairing transitions.

   Term 5.10 determines if any expected "vertical" transition is absent. This case follows exactly from the absent $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ case in the definition of the extract $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$ (page 115).

$$\begin{array}{l} \exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{\underset{\sim}{||}} - P' \bullet \\ (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \end{array} \tag{5.10}$$

   The evaluation of $\mathcal{E}''$ follows from the set difference $\Sigma_{||} - \Gamma(C')$. Observe that the derived event $\mathcal{E}'' \in \hat{\Sigma}(\widetilde{C_{||}} \lhd C')$ and the event $\Sigma'' \in \hat{\Sigma}(\widetilde{C_{||}} \lhd C'')$ are common, that is, $\mathcal{E}'' = \Sigma''$. Therefore, unlike progressive synchronisation which requires the generation of a new common event name in both $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$ and $\hat{\Sigma}(\widetilde{C_{||}} \lhd C'')$, it is not necessary to also determine if any $\mathcal{A}\Delta_{\Delta',(S'',\Gamma'',S'')}$ pairing asynchronous transitions are absent. In other words, it is not necessary to determine if there is also an absent related "horizontal" transition.

   Where an $\mathcal{A}\Delta_{\Delta'',(S',\Gamma',S')}$ asynchronous transition is absent then a transition of the form $(Q', \mathcal{E}'', Q')$ must be included in the transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$. Hence, the extract state dependent synchronisation transition set, which is defined in Definition 5.8, follows from the conjunction of term 5.9, the evaluation of $\mathcal{E}''$ and term 5.10 which determines any absent "vertical" transitions.

   Definition 5.8 *Extract State Dependent Transition Set* $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$.

$$\begin{array}{l} \hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA} \overset{def}{=} \\ \quad \{ (Q', \mathcal{E}'', Q') \mid \exists Q_{||}, \Sigma_{\underset{\sim}{||}}, P_{||} \bullet (\\ \quad\quad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge \\ \quad\quad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap Q' \neq \{\}) \wedge \\ \quad\quad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\ \quad\quad \exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{\underset{\sim}{||}} - Q' \bullet \\ \quad\quad\quad (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \\ \quad\quad) \} \end{array}$$

2. Absent $\mathcal{CS}\Delta_{\Delta',\Delta''}$ pairing transitions.

   Concurrent composition also generates a related simultaneous transition from a $\mathcal{CS}\Delta_{\Delta',\Delta''}$ pairing (page 83), the absence of which introduces state dependent synchronisation to the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$.

Unlike the absence of an expected asynchronous transition, as determined for the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$, a simultaneous transition may be absent because it has been removed, or because it never existed. The latter case arises when the state of either or both of the components is *terminal*, that is, state Q is terminal if there is no transition with a from state of Q. This leads to the observation that the presence or absence of a simultaneous transition in $C'||C''$ depends upon the transitions of both C' and C''.

In the extraction operation $\widetilde{C_{||}} \lhd C'$, the transitions of C' are known, therefore, it is necessary to determine the transitions Δ'' of C''. If a state dependency comparison determines that an implied C' idle transition exists in a $\widetilde{C_{||}}$ "vertical" transition then the from state $\mathcal{Q}''$, the event $\mathcal{E}''$, and the to state $\mathcal{P}''$ of a transition Δ'' can be evaluated. In other words, a non-terminal C'' transition $\Delta'' = (\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$ can be deduced.

Two cases of related simultaneous transition arise. Consider, for example, figure 5.7 (page 117). The simultaneous transition $(\{a, c\}, \{ab, cd\}, \{b, d\})$ is related to the asynchronous (vertical) transition $(\{a, c\}, \{\gamma_0, cd\}, \{a, d\})$ because they have the same *from* state, $\{c\}$, and by the contribution of the C'' transition $(\{c\}, \{cd\}, \{d\})$. The same simultaneous transition is also related to the asynchronous transition $(\{b, c\}, \{\gamma_0, cd\}, \{b, d\})$ because they have the same *to* state, $\{d\}$, and by the contribution of the same C'' transition $(\{c\}, \{cd\}, \{d\})$. Hence;

(a) From a comparison of an implied C' idle transition, (Q', Γ', Q'), with a $\widetilde{C_{||}}$ transition, $(Q_{||}, \Sigma_{||}, P_{||}) = (Q' \cup Q'', \Gamma' \cup \Sigma'', Q' \cup P'')$, the C'' transition $\Delta'' = (\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$ can be deduced, where $\mathcal{Q}'' = Q_{||} - Q'$, $\mathcal{E}'' = \Sigma_{||} - \Sigma'$ and $\mathcal{P}'' = P_{||} - Q'$. The pairing of Δ'' with each of the transitions $(Q', \Sigma', P') \in \hat{\Delta}(C')$ gives a simultaneous transition $\mathcal{CS}\Delta_{\Delta',\Delta''}$. Hence, term 5.11 will hold true if an expected simultaneous transition is absent.

$$(Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \tag{5.11}$$

In other words, the deduced transition Δ'', with a *from* state $\mathcal{Q}''$, is paired with every C' transition with a *from* state Q' to form each possible related simultaneous transition that would, by concurrent composition, exist in the transition set $\hat{\Delta}(C_{||})$, but which may not necessarily exist in the transition set $\hat{\Delta}(\widetilde{C_{||}})$.

(b) From a comparison of an implied C' idle transition of the form (P', Γ', P') with a $\widetilde{C_{||}}$ transition of the form $(Q_{||}, \Sigma_{||}, P_{||}) = (P' \cup Q'', \Gamma' \cup \Sigma'', P' \cup P'')$, the C'' transition $\Delta'' = (\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$ can be deduced, where $\mathcal{Q}'' = Q_{||} - P'$, $\mathcal{E}'' = \Sigma_{||} - \Sigma'$ and $\mathcal{P}'' = P_{||} - P'$.

The pairing of the deduced transition Δ'' with each of the C' transitions $(Q', \Sigma', P') \in \hat{\Delta}(C')$ gives a simultaneous transition $\mathcal{CS}\Delta_{\Delta', \Delta''}$. Hence, term 5.12 will hold true if an expected simultaneous transition is absent.

$$(Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \tag{5.12}$$

In other words, the deduced transition Δ'', with a *to* state $\mathcal{P}''$, is paired with every C' transition with a *to* state P' to form each possible related simultaneous transition that would, by concurrent composition, exist in the transition set $\hat{\Delta}(C_{||})$, but which may not necessarily exist in the transition set $\hat{\Delta}(\widetilde{C_{||}})$.

Pairing with C' simultaneous transitions is required because, under concurrent composition, all simultaneous pairings will generate the same "vertical" transition. Figure 5.8 illustrates two components and their concurrent composition $C_0 || C_1$. The composition is illustrated as the merge of the concurrent composition of the C_0 transition $(\{a\}, \{ab\}, \{b\})$ with the C_1 transition $(\{e\}, \{ef\}, \{f\})$ and the concurrent composition of the C_0 transition $(\{a\}, \{ac\}, \{c\})$ also with the C_1 transition $(\{e\}, \{ef\}, \{f\})$. This example shows that both of these pairings generate the same "vertical" transition $(\{a, e\}, \{\gamma_0, ef\}, \{a, f\})$ in which the C' implied idle transition $(\{a\}, \{\gamma_0\}, \{a\})$ exists.

Pairing with C' synchronous transitions must be excluded. Concurrent composition will not generate a transition from a pairing of a C' synchronous transition with a C'' asynchronous transition, that is, with the deduced $\mathcal{C}''$ asynchronous transition $(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$. Term 5.4 (page 112) asserts that $\Sigma_{||} = \Sigma'$, where $\Sigma_{||} \in \hat{\Sigma}(C_{||})$, is sufficient to determine a synchronous pairing in the formation of $\Sigma_{||}$. In this case it is necessary to determine that the event Σ' is not an event of $\widetilde{C_{||}}$, that is $\Sigma' \neq \Sigma_{||}$ for any $\Sigma_{||}$. This leads to term 5.13.

$$\Sigma' \notin \hat{\Sigma}(\widetilde{C_{||}}) \tag{5.13}$$

Where a simultaneous transition is absent for case 2a (page 119), a transition of the form $(Q', \mathcal{E}'', Q')$ must be included in the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$. This contribution is the first term of the set union in Definition 5.9. The contribution follows from term 5.9, the evaluation of terms $\mathcal{Q}''$, $\mathcal{E}''$ and $\mathcal{P}''$, and the conjunction of term 5.11 with $(Q', \Sigma', P') \in \hat{Q}(C')$ and term 5.13 which constrains the combinations of $Q' \in \hat{Q}(C')$, $\Sigma' \in \hat{\Sigma}(C')$ and $P' \in \hat{Q}(C')$ to the simultaneous (and not synchronous) transitions of $\hat{\Delta}(C')$.

Likewise, where a simultaneous transition is absent for case 2b (page 120), a transition of the form $(P', \mathcal{E}'', P')$ must be included in the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$. This contribution, which is the second term of the set union in Definition 5.9, differs from the first term in that P' and Q' have been exchanged.

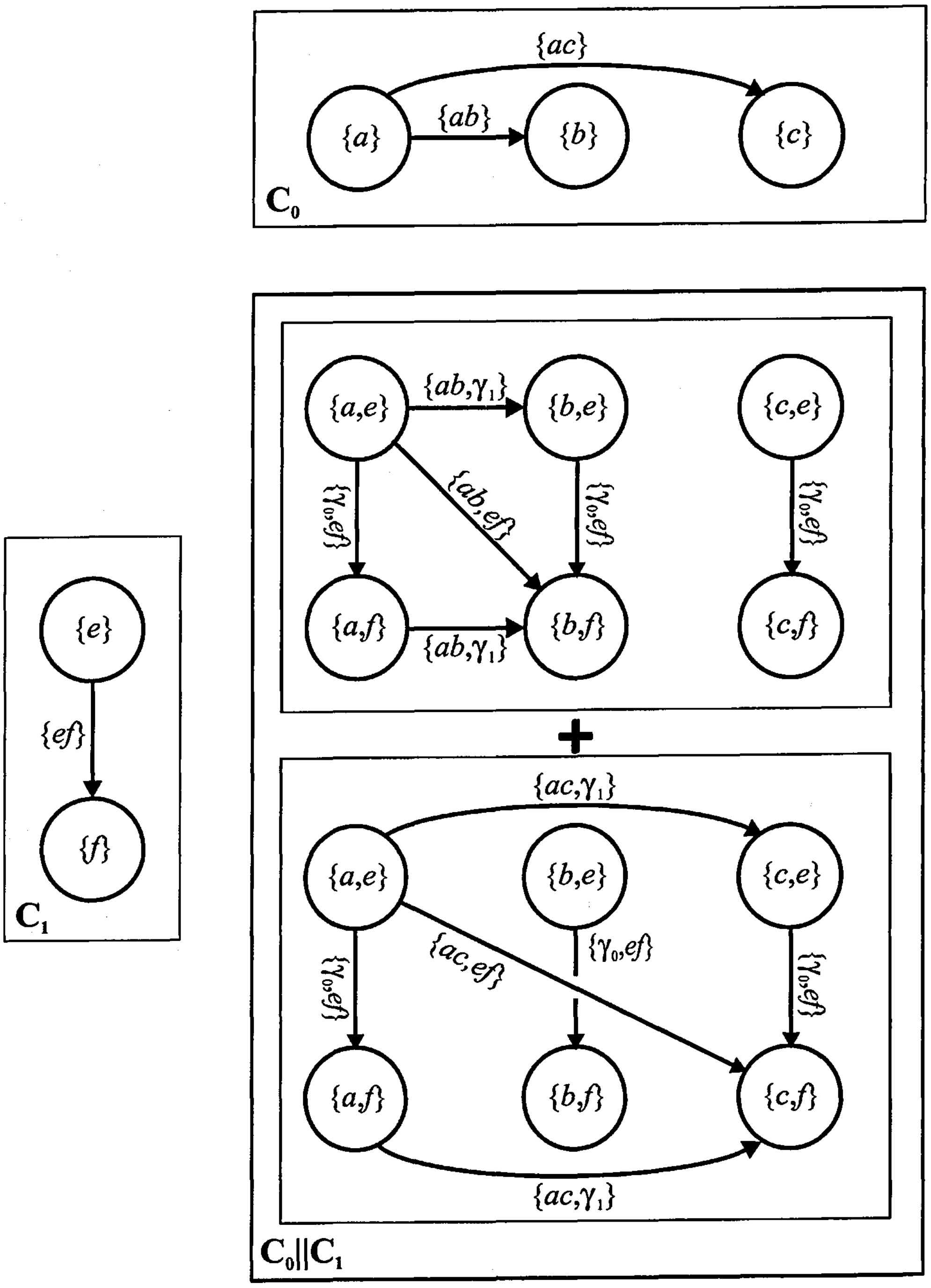


Figure 5.8: Duplication of related transitions under concurrent composition.

Definition 5.9 *Extract State Dependent Transition Set* $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$.

$$
\begin{array}{l}
\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC} \overset{def}{=} \\
\quad \{ (Q', \mathcal{E}'', Q') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge \\
\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap Q' \neq \{\}) \wedge \\
\qquad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\
\qquad \exists P' \in \hat{Q}(C'),\ \Sigma' \in \hat{\Sigma}(C'), \\
\qquad\quad \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{||} - Q' \bullet \\
\qquad\quad ((Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \wedge \\
\qquad\quad\ (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
\qquad\quad\ \Sigma' \notin \hat{\Sigma}(\widetilde{C_{||}})) \\
\qquad)\} \\
\quad \bigcup \\
\quad \{ (P', \mathcal{E}'', P') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge P' \in \hat{Q}(C') \wedge \\
\qquad (Q_{||} \cap P' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\
\qquad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\
\qquad \exists Q' \in \hat{Q}(C'),\ \Sigma' \in \hat{\Sigma}(C'), \\
\qquad\quad \mathcal{Q}'' = Q_{||} - P',\ \mathcal{P}'' = P_{||} - P' \bullet \\
\qquad\quad ((Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \wedge \\
\qquad\quad\ (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
\qquad\quad\ \Sigma' \notin \hat{\Sigma}(\widetilde{C_{||}})) \\
\qquad)\}
\end{array}
$$

Extract Transition Set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$

The terms $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$ and $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$ contribute transitions to the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ given in Definition 5.10.

Definition 5.10 *Extract Transition Set*

$$
\begin{array}{lcl}
\hat{\Delta}(\widetilde{C_{||}} \lhd C') & \overset{def}{=} & \hat{\Delta}(\widetilde{C_{||}} \lhd C')_A \\
& \bigcup & \hat{\Delta}(\widetilde{C_{||}} \lhd C')_S \\
& \bigcup & \hat{\Delta}(\widetilde{C_{||}} \lhd C')_P \\
& \bigcup & \hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA} \\
& \bigcup & \hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}
\end{array}
$$

5.3 Example of Extraction from an Unrestricted System

This example illustrates the application of the extraction operator on the unrestricted composite machine $C_0||C_1$. As an unrestricted machine, the extract $C_{||} \lhd C_n$ and the component C_n should be the same.

Consider the system $C_0||C_1$, illustrated in figure 4.5 (page 91), formed from the concurrent composition of C_0 and C_1, illustrated in figure 4.4 (page 88). Observe that one of the components has a reflexive non-idle transition, illustrating that the so called "diagonal", "vertical" and "horizontal" transitions still exist. For reference, the definition of $C_0||C_1$ is repeated here;

$$\begin{array}{rcl}
\hat{Q}(C_0||C_1) & = & \{\{a,e\},\{b,e\},\{a,f\},\{b,f\}\} \\
\ddot{Q}(C_0||C_1) & = & \{\{a,e\}\} \\
\hat{\Sigma}(C_0||C_1) & = & \{\{ba,\gamma_1\},\{\gamma_0,ee\},\{ba,ee\},\{s\}\} \\
\Gamma(C_0||C_1) & = & \{\gamma_0,\gamma_1\} \\
\hat{\Delta}(C_0||C_1) & = & \{(\{b,e\},\{ba,\gamma_1\},\{a,e\}),(\{b,f\},\{ba,\gamma_1\},\{a,f\}), \\
 & & (\{a,e\},\{\gamma_0,ee\},\{a,e\}),(\{b,e\},\{\gamma_0,ee\},\{b,e\}), \\
 & & (\{b,e\},\{ba,ee\},\{a,e\}),(\{a,e\},\{s\},\{b,f\})\}
\end{array}$$

5.3.1 Extraction of C_0 from $C_0||C_1$

Let the extraction operation $\widetilde{C_0} = C_0||C_1 \lhd C_0$ be written as $\widetilde{C_0} = C_{||} \lhd C_0$, where C_0 is given below. Carefully note that for this extraction operation, any C' term in any of the definitions of the extraction operator refers to C_0, and any derived C'' term infers C_1.

$$\begin{array}{rcl}
\hat{Q}(C_0) & = & \{\{a\},\{b\}\} \\
\ddot{Q}(C_0) & = & \{\{a\}\} \\
\hat{\Sigma}(C_0) & = & \{\{s\},\{ba\}\} \\
\Gamma(C_0) & = & \{\gamma_0\} \\
\hat{\Delta}(C_0) & = & \{(\{a\},\{s\},\{b\}),(\{b\},\{ba\},\{a\})\}
\end{array}$$

From Definition 5.1 (page 105), each state of $\hat{Q}(C_{||} \lhd C_0)$ must exist in both $\hat{Q}(C_{||})$ and $\hat{Q}(C_0)$. Hence, the pairing of $\{a,e\} \in \hat{Q}(C_{||})$ with $\{a\} \in \hat{Q}(C_0)$ gives $\{a,e\} \cap \{a\} = \{a\}$, thus the conjunction $Q_{||} \cap Q' \neq \{\}$ holds true and $\{a\}$ is included in $\hat{Q}(C_{||} \lhd C_0)$. Conversely, the pairing of $\{a, e\}$ with $\{b\}$ gives the empty set $\{\}$, thus the conjunction $Q_{||} \cap Q' \neq \{\}$ does not hold and hence $\{b\}$ is not included in $\hat{Q}(C_{||} \lhd C_0)$. Evaluation of all pairings in accordance with Definition 5.1 is required to determine the state set. Likewise, the initial state set follows from Definition 5.2 (page 106). Hence;

$$\begin{aligned}\hat{Q}(C_{||} \lhd C_0) &= \{\{a\}, \{b\}\} \\ \ddot{Q}(C_{||} \lhd C_0) &= \{\{a\}\}\end{aligned}$$

From Definition 5.3 (page 107), each event name of $\hat{\Sigma}(C_{||} \lhd C_0)$ must exist in both $\hat{\Sigma}(C_{||})$ and $\hat{\Sigma}(C_0)$, or may be a synchronising event resulting from the extraction. In this example, the system $C_{||}$ has not been restricted and therefore extraction does not generate any synchronising events, thus the set $\hat{\Sigma}(C_{||} \lhd C_0)$ is formed only from the existence of $\hat{\Sigma}(C_0)$ event names in $\hat{\Sigma}(C_{||})$ events. Hence;

$$\hat{\Sigma}(C_{||} \lhd C_0) = \{\{s\}, \{ba\}\}$$

From Definition 5.4 (page 107), the extract idle event name $\Gamma(C_{||} \lhd C_0)$ is a new idle event name $\mathcal{N}(\{\gamma_0\})$. For simplicity, $\{\gamma_0\}$ will be used as the system $C_{||}$ has not been restricted. Hence;

$$\Gamma(C_{||} \lhd C_0) = \{\gamma_0\}$$

From Definition 5.10 (page 123), the transition set is formed from the union of the contributions of $\hat{\Delta}(C_{||} \lhd C')_A$, $\hat{\Delta}(C_{||} \lhd C')_S$, $\hat{\Delta}(C_{||} \lhd C')_P$, $\hat{\Delta}(C_{||} \lhd C')_{DA}$ and $\hat{\Delta}(C_{||} \lhd C')_{DC}$.

1. $\hat{\Delta}(C_{||} \lhd C_0)_A$

   Of all the possible transition pairings, table 5.1 lists only those where a transition $\hat{\Delta}(C_0)$ exists in a $\Delta_{||}$ transition. However, only those pairings where $\mathcal{E}'' = \mathcal{I}''$ contribute the transition $\hat{\Delta}(C_0)$ to $\hat{\Delta}(C_{||} \lhd C')_A$. The contributing pairings are marked in the contribution (C) column. In this example, only the transition $(\{b\}, \{ba\}, \{a\})$ contributes to $\hat{\Delta}(C_{||} \lhd C')_A$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_0)$	$\mathcal{E}''$	$\mathcal{I}''$	C
$(\{b,e\},\{ba,\gamma_1\},\{a,e\})$		$\{\gamma_1\}$		$\star$
$(\{b,f\},\{ba,\gamma_1\},\{a,f\})$	$(\{b\},\{ba\},\{a\})$	$\{\gamma_1\}$	$\{\gamma_1\}$	$\star$
$(\{b,e\},\{ba,ee\},\{a,e\})$		$\{ee\}$		
$(\{a,e\},\{s\},\{b,f\})$	$(\{a\},\{s\},\{b\})$	$\{\}$	$\{\gamma_1\}$	

Table 5.1: Existent and contributing pairings of $\hat{\Delta}(C_{||} \lhd C_0)_A$

2. $\hat{\Delta}(C_{||} \lhd C_0)_S$

Table 5.2 lists those transitions $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} = \Sigma'$ contribute the transition $\hat{\Delta}(C_0)$ to $\hat{\Delta}(C_{||} \lhd C')_S$. In this example, only the transition $(\{a\}, \{s\}, \{b\})$ is contributed to $\hat{\Delta}(C_{||} \lhd C')_S$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_0)$	$\Sigma_{\|\|}$	Σ'	C
$(\{b,e\},\{ba,\gamma_1\},\{a,e\})$		$\{ba,\gamma_1\}$		
$(\{b,f\},\{ba,\gamma_1\},\{a,f\})$	$(\{b\},\{ba\},\{a\})$	$\{ba,\gamma_1\}$	$\{ba\}$	
$(\{b,e\},\{ba,ee\},\{a,e\})$		$\{ba,ee\}$		
$(\{a,e\},\{s\},\{b,f\})$	$(\{a\},\{s\},\{b\})$	$\{s\}$	$\{s\}$	$\star$

Table 5.2: Existent and contributing pairings of $\hat{\Delta}(C_{||} \lhd C_0)_S$

3. $\hat{\Delta}(C_{||} \lhd C_0)_P$

Table 5.3 lists those transitions $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} \neq \Sigma'$ and $\mathcal{E}'' \neq \mathcal{I}''$ lead to term 5.6 holding true, and these pairings are marked in the E (existence) column. In this example, term 5.6 holds true for the third pairing because $\{ba, ee\} \neq \{ba\}$ and $\{ee\} \neq \{\gamma_1\}$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_0)$	$\Sigma_{\|\|}$	Σ'	$\mathcal{E}''$	$\mathcal{I}''$	E
$(\{b,e\},\{ba,\gamma_1\},\{a,e\})$		$\{ba,\gamma_1\}$		$\{\gamma_1\}$		
$(\{b,f\},\{ba,\gamma_1\},\{a,f\})$	$(\{b\},\{ba\},\{a\})$	$\{ba,\gamma_1\}$	$\{ba\}$	$\{\gamma_1\}$	$\{\gamma_1\}$	
$(\{b,e\},\{ba,ee\},\{a,e\})$		$\{ba,ee\}$		$\{ee\}$		$\dagger$
$(\{a,e\},\{s\},\{b,f\})$	$(\{a\},\{s\},\{b\})$	$\{s\}$	$\{s\}$	$\{\}$	$\{\gamma_1\}$	

Table 5.3: Existent pairings of $\hat{\Delta}(C_{||} \lhd C_0)_P$

From the existence of the transition $(\{b\}, \{ba\}, \{a\})$ in the coincidental transition $(\{b, e\}, \{ba, ee\}, \{a, e\})$ it is necessary to determine the absence of any related asynchronous transitions. For this pairing the following terms may be deduced by set difference of the corresponding $C_{||}$ and C_0 terms, hence $\mathcal{Q}'' = \{e\}$, $\mathcal{E}'' = \{ee\}$, $\mathcal{P}'' = \{e\}$ and $\mathcal{I}'' = \{\gamma_1\}$.

(a) Absent $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ transitions: Term 5.7 (page 114) forms transitions of the form $(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$, where $\mathcal{S}'' \in \hat{\mathcal{Q}}(\mathcal{C}'')$, to determine absence from the transition set $\hat{\Delta}(C_{||})$. The evaluation of $\hat{\mathcal{Q}}(\mathcal{C}'')$ is formed by pairing, for example, the pairing of $\{a, e\} \in \hat{Q}(C_{||})$ with $\{a\} \in \hat{Q}(C_0)$ contributes the state $\{e\}$ to $\hat{\mathcal{Q}}(\mathcal{C}'')$ by $\{a, e\} - \{a\}$ and because the predicate $\{a, e\} \cap \{a\} \neq \{\}$ holds true. However, the pairing of $\{a, e\} \in \hat{Q}(C_{||})$ with $\{b\} \in \hat{Q}(C_0)$ does not contribute $\{a, e\}$ by $\{a, e\} - \{b\}$ because the predicate $\{a, e\} \cap \{b\} \neq \{\}$ does not hold. The evaluation of all the pairings yields $\mathcal{S}'' \in \{\{e\}, \{f\}\}$.

Table 5.4 lists the pairings and the transition formed for each pairing. Absent transitions are marked in the absence (A) column. Since none of the formed transitions is absent from $\hat{\Delta}(C_{||})$, progressive synchronisation is not introduced.

(Q', Σ', P')	$(\mathcal{S}'', \mathcal{I}'', \mathcal{S}'')$	$(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$	A
$(\{b\}, \{ba\}, \{a\})$	$(\{e\}, \{\gamma_1\}, \{e\})$	$(\{b, e\}, \{ba, \gamma_1\}, \{a, e\})$	
	$(\{f\}, \{\gamma_1\}, \{f\})$	$(\{b, f\}, \{ba, \gamma_1\}, \{a, f\})$	

Table 5.4: Absent $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ pairings for $\hat{\Delta}(C_{||} \lhd C_0)_P$

(b) Absent $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(S',\Gamma',S')}$ transitions: Term 5.8 (page 115) forms transitions of the form $(S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'')$, where $S' \in \hat{Q}(C')$, to determine absence from the transition set $\hat{\Delta}(C_{||})$. Table 5.5 lists the pairings and the transition formed for each pairing. Since none of the formed transitions is absent from $\hat{\Delta}(C_{||})$, progressive synchronisation is not introduced.

(S',Γ',S')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(S'\cup\mathcal{Q}'',\Gamma'\cup\mathcal{E}'',S'\cup\mathcal{P}'')$	A
$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ee\},\{e\})$	$(\{a,e\},\{\gamma_0,ee\},\{a,e\})$	
$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ee\},\{b,e\})$	

Table 5.5: Absent $\mathcal{A}\Delta_{(\{e\},\{ee\},\{e\}),(S',\Gamma',S')}$ pairings for $\hat{\Delta}(C_{||} \lhd C_0)_P$

4. $\hat{\Delta}(C_{||} \lhd C_0)_{DA}$

Table 5.6 lists those implied idle transitions (Q',Γ',Q') of $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition, and lists the derived terms $\mathcal{Q}''$, $\mathcal{E}''$ and $\mathcal{P}''$. Recall that there will be an implied idle transition for every state $Q' \in \hat{Q}(C_0)$.

(Q',Γ',Q')	$\Delta_{\|\|}$	$\mathcal{Q}''$	$\mathcal{E}''$	$\mathcal{P}''$
$(\{a\},\{\gamma_0\},\{a\})$	$(\{a,e\},\{\gamma_0,ee\},\{a,e\})$	$\{e\}$	$\{ee\}$	$\{e\}$
$(\{b\},\{\gamma_0\},\{b\})$	$(\{b,e\},\{\gamma_0,ee\},\{b,e\})$			

Table 5.6: Derivation of Δ'' transitions for $\hat{\Delta}(C_{||} \lhd C_0)_{DA}$

Table 5.7 lists the deduced C'' transitions $(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$, the implied idle transitions (S',Γ',S') and the formed transition for each pairing. Recall that there will be an implied idle transition for every state $S' \in \hat{Q}(C_0)$. Since none of the formed transitions is absent from $\hat{\Delta}(C_{||})$, state dependent synchronisation is not required.

Q'	(S',Γ',S')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(S'\cup\mathcal{Q}'',\Gamma'\cup\mathcal{E}'',S'\cup\mathcal{P}'')$	A
$\{a\}$	$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ee\},\{e\})$	$(\{a,e\},\{\gamma_0,ee\},\{a,e\})$	
	$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ee\},\{b,e\})$	
$\{b\}$	$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ee\},\{e\})$	$(\{a,e\},\{\gamma_0,ee\},\{a,e\})$	
	$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ee\},\{b,e\})$	

Table 5.7: $\mathcal{A}\Delta_{\Delta_{||},(S',\Gamma',S')}$ pairings for $\hat{\Delta}(C_{||} \lhd C_0)_{DA}$

5. $\hat{\Delta}(C_{||} \lhd C_0)_{DC}$

Table 5.6 showed that the implied idle transitions $(\{a\},\{\gamma_0\},\{a\})$ and $(\{b\},\{\gamma_0\},\{b\})$ of C' determine the existence of the Δ'' transition $(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'') = (\{e\},\{ee\},\{e\})$. The existence of the related simultaneous transitions must be determined, that is, pairings of the transition $(\{e\},\{ee\},\{e\})$ with C' transitions with a *from* state or *to* state of $\{a\}$ or $\{b\}$.

(a) Related *from* state simultaneous transitions: Table 5.8 lists the deduced C'' transitions $(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$, and the C_0 transitions (Q', Σ', P') where the *from* state Q' is either $\{a\}$ or $\{b\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(C_{||})$. In this example, the formed transition is not absent from $\hat{\Delta}(C_{||})$, thus state dependent synchronisation is not required.

Q'	(Q', Σ', P')	$(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$	$(Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'')$	A
$\{a\}$	$(\{a\}, \{s\}, \{b\})$	$(\{e\}, \{ee\}, \{e\})$	Note: $\Sigma' \notin \hat{\Sigma}(C_{\|\|})$ is false	
$\{b\}$	$(\{b\}, \{ba\}, \{a\})$		$(\{b, e\}, \{ba, ee\}, \{a, e\})$	

Table 5.8: Related *from* state transitions for $\hat{\Delta}(C_{||} \lhd C_0)_{DC}$

(b) Related *to* state simultaneous transitions: Table 5.9 lists the deduced C'' transitions $(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$, and the C_0 transitions (Q', Σ', P') where the *to* state P' is either $\{a\}$ or $\{b\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(C_{||})$. In this example, the formed transition is not absent from $\hat{\Delta}(C_{||})$, thus a state dependent synchronisation transition is not required.

P'	(Q', Σ', P')	$(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$	$(Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'')$	A
$\{a\}$	$(\{b\}, \{ba\}, \{a\})$	$(\{e\}, \{ee\}, \{e\})$	$(\{b, e\}, \{ba, ee\}, \{a, e\})$	
$\{b\}$	$(\{a\}, \{s\}, \{b\})$		Note: $\Sigma' \notin \hat{\Sigma}(C_{\|\|})$ is false	

Table 5.9: Related *to* state transitions for $\hat{\Delta}(C_{||} \lhd C_0)_{DC}$

From cases 1 and 2, the transition set $\hat{\Delta}(C_{||} \lhd C_0)$ is as follows.

$$\hat{\Delta}(C_{||} \lhd C_0) = \{(\{a\}, \{s\}, \{b\}), (\{b\}, \{ba\}, \{a\})\}$$

Finally, the complete extraction $\widetilde{C_0} = C_0||C_1 \lhd C_0$ can be written as follows. Comparison of $\widetilde{C_0}$ and C_0 shows that they describe the same machine. Since $C_0||C_1$ was not restricted, this is the expected result.

$$\begin{aligned}\widetilde{C_0} &= (\hat{Q}(C_{||} \lhd C_0), \ddot{Q}(C_{||} \lhd C_0), \hat{\Sigma}(C_{||} \lhd C_0), \Gamma(C_{||} \lhd C_0), \hat{\Delta}(C_{||} \lhd C_0)) \\ &= (\{\{a\},\{b\}\},\{\{a\}\}, \\ &\quad \{\{s\},\{ab\}\},\{\gamma_0\}, \\ &\quad \{(\{a\},\{s\},\{b\}),(\{b\},\{ba\},\{a\})\}\})\end{aligned}$$

5.3.2 Extraction of C_1 from $C_0||C_1$

The extraction $\widetilde{C_1} = C_0||C_1 \lhd C_1$ follows the same process as the extraction $C_0||C_1 \lhd C_0$. Since $C_0||C_1$ was not restricted, it is expected that $\widetilde{C_1} = C_1$.

$$\begin{aligned}\widetilde{C_1} &= (\hat{Q}(C_{||} \lhd C_1), \ddot{Q}(C_{||} \lhd C_1), \hat{\Sigma}(C_{||} \lhd C_1), \Gamma(C_{||} \lhd C_1), \hat{\Delta}(C_{||} \lhd C_1)) \\ &= (\{\{e\},\{f\}\},\{\{e\}\}, \\ &\quad \{\{s\},\{ee\}\},\{\gamma_1\}, \\ &\quad \{(\{e\},\{s\},\{f\}),(\{e\},\{ee\},\{e\})\}\})\end{aligned}$$

5.4 Example of Extraction from a Restricted System

This second example illustrates extraction from a restricted composite machine $\widetilde{C_0||C_1}$ and thus the introduction of progressive and state dependent synchronisation. As a restricted machine, the extraction $\widetilde{C_{||}} \lhd C_n$ should give $\widetilde{C_n}$. Since there is no prior knowledge of the extracted machines, an analysis of the results of extraction is presented in section 5.4.3 (page 142).

Consider the example system illustrated in figure 5.9 and defined below. Note that the dotted transitions indicate those removed from $C_0||C_1$ to form $\widetilde{C_0||C_1}$ and, therefore, they are not considered in the extraction of C_0 and C_1.

$$\begin{aligned}\hat{Q}(\widetilde{C_0||C_1}) &= \{\{a,e\},\{b,e\},\{a,f\},\{b,f\}\} \\ \ddot{Q}(\widetilde{C_0||C_1}) &= \{\{b,e\}\} \\ \hat{\Sigma}(\widetilde{C_0||C_1}) &= \{\{ab,\gamma_1\},\{cd,\gamma_1\},\{\gamma_0,ef\},\{ab,ef\},\{cd,ef\}\} \\ \Gamma(\widetilde{C_0||C_1}) &= \{\gamma_0,\gamma_1\} \\ \hat{\Delta}(\widetilde{C_0||C_1}) &= \{(\{a,e\},\{ab,\gamma_1\},\{b,e\}),(\{a,e\},\{cd,\gamma_1\},\{b,e\}), \\ &\quad (\{a,e\},\{ab,ef\},\{b,f\}), \\ &\quad (\{a,e\},\{\gamma_0,ef\},\{a,f\}),(\{b,e\},\{\gamma_0,ef\},\{b,f\}), \\ &\quad (\{a,f\},\{cd,\gamma_1\},\{b,f\})\}\end{aligned}$$

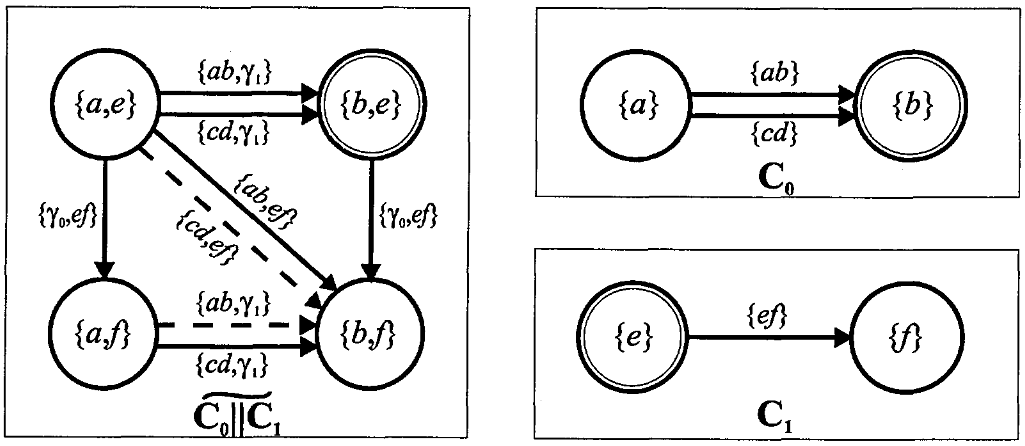


Figure 5.9: $\widetilde{C_0\|C_1}$ (left), C_0 (top right) and C_1 (bottom right)

Two transitions have been removed from $C_0\|C_1$ to illustrate, i) the absence of simultaneous progression, and ii) that there is no ambiguity if any two or more transitions share the same *from* and *to* states. Although this example is quite simple, it demonstrates the difficulty in determining the required interaction between two components to give the (desired) restricted behaviour and how the CTS notation provides an effective solution.

5.4.1 Extraction of C_0 from $\widetilde{C_0\|C_1}$

Let the extraction $\widetilde{C_0} = \widetilde{C_0\|C_1} \lhd C_0$ be written as $\widetilde{C_0} = \widetilde{C_{\|}} \lhd C_0$, where C_0 is given below. Carefully note that for this extraction, any C' term in any of the definitions of extraction refers to C_0, and any derived C'' term infers C_1.

$$\begin{aligned}
\hat{Q}(C_0) &= \{\{a\},\{b\}\}\\
\ddot{Q}(C_0) &= \{\{b\}\}\\
\hat{\Sigma}(C_0) &= \{\{ab\},\{cd\}\}\\
\Gamma(C_0) &= \{\gamma_0\}\\
\hat{\Delta}(C_0) &= \{(\{a\},\{ab\},\{b\}),(\{a\},\{cd\},\{b\})\}
\end{aligned}$$

Evaluation of the state set and initial state set follow from the example of section 5.3 (page 124). The idle event name is determined by $\mathcal{N}(\{\gamma_0\})$, and in this case $\{\tilde{\gamma}_0\}$ will be used.

Hence;

$$\begin{aligned}
\hat{Q}(\widetilde{C_{||}} \lhd C_0) &= \{\{a\},\{b\}\}\\
\ddot{Q}(\widetilde{C_{||}} \lhd C_0) &= \{\{b\}\}\\
\Gamma(\widetilde{C_{||}} \lhd C_0) &= \{\widetilde{\gamma_0}\}
\end{aligned}$$

Unlike the unrestricted example of section 5.3 (page 124), this example generates progressive and state dependent synchronising transitions and, therefore, shared event names. By Definition 5.3 (page 107), the evaluated event name set, which depends upon the transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)$, evaluates as $\hat{\Sigma}(\widetilde{C_{||}} \lhd C_0) = \{\{ab\},\{cd\},\{ef\},\{abef\}\}$. The event name $\{abef\}$ is contributed by the transition $(\{a\},\{abef\},\{b\}) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$ (page 133), and $\{ef\}$ is contributed by the transitions $(\{a\},\{ef\},\{a\}) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DC}$ and $(\{b\},\{ef\},\{b\}) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DC}$ (page 136).

From Definition 5.10 (page 123), the transition set is formed from the union of the contributions of $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$ and $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$.

1. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_A$

   Table 5.10 lists those transitions $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\mathcal{E}'' = \mathcal{I}''$ contribute the transition $\hat{\Delta}(C_0)$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$. The contributing pairings are marked in the contribution (C) column. In this example, the first and last two pairings contribute the transitions $(\{a\},\{ab\},\{b\})$ and $(\{a\},\{cd\},\{b\})$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$.

$\Delta_{||}$	$\hat{\Delta}(C_0)$	$\mathcal{E}''$	$\mathcal{I}''$	C
$(\{a,e\},\{ab,\gamma_1\},\{b,e\})$	$(\{a\},\{ab\},\{b\})$	$\{\gamma_1\}$	$\{\gamma_1\}$	$\star$
$(\{a,e\},\{ab,ef\},\{b,f\})$		$\{ef\}$		
$(\{a,e\},\{cd,\gamma_1\},\{b,e\})$	$(\{a\},\{cd\},\{b\})$	$\{\gamma_1\}$	$\{\gamma_1\}$	$\star$
$(\{a,f\},\{cd,\gamma_1\},\{b,f\})$				$\star$

Table 5.10: Existent and contributing pairings of $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_A$

2. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_S$

Table 5.11 lists those transitions $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} = \Sigma'$ contribute the transition $\hat{\Delta}(C_0)$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$. In this example there are no synchronous pairings, thus no transitions are contributed to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_0)$	$\Sigma_{\|\|}$	Σ'	C
$(\{a,e\},\{ab,\gamma_1\},\{b,e\})$	$(\{a\},\{ab\},\{b\})$	$\{ab,\gamma_1\}$	$\{ab\}$	
$(\{a,e\},\{ab,ef\},\{b,f\})$		$\{ab,ef\}$		
$(\{a,e\},\{cd,\gamma_1\},\{b,e\})$	$(\{a\},\{cd\},\{b\})$	$\{cd,\gamma_1\}$	$\{cd\}$	
$(\{a,f\},\{cd,\gamma_1\},\{b,f\})$				

Table 5.11: Existent and contributing pairings of $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_S$

3. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$

Table 5.12 lists those transitions $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} \neq \Sigma'$ and $\mathcal{E}'' \neq \mathcal{I}''$ lead to term 5.6 holding true, and these pairings are marked in the E (existence) column. In this example, term 5.6 holds true for the second pairing because $\{ab, ef\} \neq \{ab\}$ and $\{ef\} \neq \{\gamma_1\}$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_0)$	$\Sigma_{\|\|}$	Σ'	$\mathcal{E}''$	$\mathcal{I}''$	E
$(\{a,e\},\{ab,\gamma_1\},\{b,e\})$	$(\{a\},\{ab\},\{b\})$	$\{ab,\gamma_1\}$	$\{ab\}$	$\{\gamma_1\}$	$\{\gamma_1\}$	
$(\{a,e\},\{ab,ef\},\{b,f\})$		$\{ab,ef\}$		$\{ef\}$		†
$(\{a,e\},\{cd,\gamma_1\},\{b,e\})$	$(\{a\},\{cd\},\{b\})$	$\{cd,\gamma_1\}$	$\{cd\}$	$\{\gamma_1\}$	$\{\gamma_1\}$	
$(\{a,f\},\{cd,\gamma_1\},\{b,f\})$						

Table 5.12: Existent pairings of $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$

From the existence of the transition $(\{a\},\{ab\},\{b\})$ in the coincidental transition $(\{a,e\},\{ab,ef\},\{b,f\})$ it is necessary to determine the absence of any related asynchronous transitions. For this pairing the following terms may be deduced by set difference of the corresponding $\widetilde{C_{||}}$ and C_0 terms, hence $\mathcal{Q}'' = \{e\}$, $\mathcal{E}'' = \{ef\}$, $\mathcal{P}'' = \{f\}$ and $\mathcal{I}'' = \{\gamma_1\}$.

(a) Absent $\mathcal{A}\Delta_{(\{a\},\{ab\},\{b\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ transitions: Term 5.7 (page 114) gives $\hat{\mathcal{Q}}(\mathcal{C}'') = \{\{e\},\{f\}\}$, and forms transitions of the form $(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$, where $\mathcal{S}'' \in \hat{\mathcal{Q}}(\mathcal{C}'')$, to determine absence from the transition set $\hat{\Delta}(\widetilde{C_{||}})$. Table 5.13 lists the pairings and the transition formed for each pairing. Absent transitions are marked in the absence (A) column.

The pairing marked $\star$ is absent from the transition set $\hat{\Delta}(\widetilde{C_{||}})$ and progressive synchronisation must be introduced. Thus the transition $(\{a\}, \mathcal{N}(\{ab\} \cup \{ef\}), \{b\})$ must be contributed to the transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$. Let $\mathcal{N}(\{ab\} \cup \{ef\}) = \{abef\}$, hence the contributed transition will be $(\{a\},\{abef\},\{b\})$.

(Q',Σ',P')	$(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')$	$(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$	A
$(\{a\},\{ab\},\{b\})$	$(\{e\},\{\gamma_1\},\{e\})$	$(\{a,e\},\{ab,\gamma_1\},\{b,e\})$	
	$(\{f\},\{\gamma_1\},\{f\})$	$(\{a,f\},\{ab,\gamma_1\},\{b,f\})$	$\star$

Table 5.13: Absent $\mathcal{A}\Delta_{(\{b\},\{ba\},\{a\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ pairings for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$

(b) Absent $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\mathcal{S}',\Gamma',\mathcal{S}')}$ transitions: Term 5.8 (page 115) forms transitions of the form $(S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'')$, where $S' \in \hat{Q}(C')$, to determine absence from the transition set $\hat{\Delta}(\widetilde{C_{||}})$. Table 5.14 lists the pairings and the transition formed for each pairing. Since none of the formed transitions is absent from $\hat{\Delta}(\widetilde{C_{||}})$, progressive synchronisation is not introduced.

(S',Γ',S')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'')$	A
$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ef\},\{f\})$	$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$	
$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$	

Table 5.14: Absent $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\mathcal{S}',\Gamma',\mathcal{S}')}$ pairings for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_P$

4. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DA}$

Table 5.15 lists those implied idle transitions (Q',Γ',Q') of $\hat{\Delta}(C_0)$ that exist in a $\Delta_{||}$ transition, and lists the derived terms $\mathcal{Q}''$, $\mathcal{E}''$ and $\mathcal{P}''$. Recall that there will be an implied idle transition for every state $Q' \in \hat{Q}(C_0)$.

(Q',Γ',Q')	$\Delta_{\|}$	Q''	$\mathcal{E}''$	$\mathcal{P}''$
$(\{a\},\{\gamma_0\},\{a\})$	$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$	$\{e\}$	$\{ef\}$	$\{f\}$
$(\{b\},\{\gamma_0\},\{b\})$	$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$			

Table 5.15: Derivation of Δ'' transitions for $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_0)_{DA}$

Table 5.16 lists the deduced C'' transitions $(Q'',\mathcal{E}'',\mathcal{P}'')$, the implied idle transitions (S',Γ',S') and the formed transition for each pairing. Recall that there will be an implied idle transition for every state $S' \in \hat{Q}(C_0)$. Since none of the formed transitions is absent from $\hat{\Delta}(C_{\|})$, state dependent synchronisation is not required.

Q'	(S',Γ',S')	$(Q'',\mathcal{E}'',\mathcal{P}'')$	$(S' \cup Q'',\Gamma' \cup \mathcal{E}'',S' \cup \mathcal{P}'')$	A
$\{a\}$	$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ef\},\{f\})$	$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$	
	$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$	
$\{b\}$	$(\{a\},\{\gamma_0\},\{a\})$	$(\{e\},\{ef\},\{f\})$	$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$	
	$(\{b\},\{\gamma_0\},\{b\})$		$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$	

Table 5.16: $\mathcal{A}\Delta_{\Delta_{\|},(S',\Gamma',S')}$ pairings for $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_0)_{DA}$

5. $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_0)_{DC}$

Table 5.15 showed that the idle transitions $(\{a\},\{\gamma_0\},\{a\})$ and $(\{b\},\{\gamma_0\},\{b\})$ of C_0 determine the existence of the Δ'' transition $(Q'',\mathcal{E}'',\mathcal{P}'') = (\{e\},\{ef\},\{f\})$. The existence of the related simultaneous transitions must be determined, that is, pairings of the transition $(\{e\},\{ef\},\{f\})$ with C_0 transitions with a *from* state or *to* state of $\{a\}$ or $\{b\}$.

(a) Related *from* state simultaneous transitions: Table 5.17 lists the deduced C'' transitions $(Q'',\mathcal{E}'',\mathcal{P}'')$, and the C_0 transitions (Q',Σ',P') where the *from* state Q' is either $\{a\}$ or $\{b\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(\widetilde{C_{\|}})$. Note that there are no C_0 transitions with a *from* state of $\{b\}$. In this example, the formed transition $(\{a,e\},\{cd,ef\},\{b,f\})$ is absent from $\hat{\Delta}(\widetilde{C_{\|}})$, thus the state dependent transition $(Q',\mathcal{E}'',Q') = (\{a\},\{ef\},\{a\})$ is contributed to $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_0)_{DC}$.

Q'	(Q',Σ',P')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(Q'\cup\mathcal{Q}'',\Sigma'\cup\mathcal{E}'',P'\cup\mathcal{P}'')$	A
$\{a\}$	$(\{a\},\{ab\},\{b\})$	$(\{e\},\{ef\},\{f\})$	$(\{a,e\},\{ab,ef\},\{b,f\})$	
	$(\{a\},\{cd\},\{b\})$		$(\{a,e\},\{cd,ef\},\{b,f\})$	⋆
$\{b\}$				

Table 5.17: Related *from* state transitions for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DC}$

(b) Related *to* state simultaneous transitions: Table 5.18 lists the deduced $\mathcal{C}''$ transitions $(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$, and the C_0 transitions (Q',Σ',P') where the *to* state P' is either $\{a\}$ or $\{b\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(\widetilde{C_{||}})$. Note that there are no C_0 transitions with a *to* state of $\{a\}$. In this example, the formed transition $(\{a,e\},\{cd,ef\},\{b,f\})$ is absent from $\hat{\Delta}(\widetilde{C_{||}})$, thus the state dependent transition $(P',\mathcal{E}'',P') = (\{b\},\{ef\},\{b\})$ is contributed to $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DC}$.

P'	(Q',Σ',P')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(Q'\cup\mathcal{Q}'',\Sigma'\cup\mathcal{E}'',P'\cup\mathcal{P}'')$	A
$\{a\}$				
$\{b\}$	$(\{a\},\{ab\},\{b\})$	$(\{e\},\{ef\},\{f\})$	$(\{a,e\},\{ab,ef\},\{b,f\})$	
	$(\{a\},\{cd\},\{b\})$		$(\{a,e\},\{cd,ef\},\{b,f\})$	⋆

Table 5.18: Related *to* state transitions for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)_{DC}$

From cases 1, 3a, 5a and 5b the transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C_0)$ is as follows.

$$\begin{aligned}\hat{\Delta}(\widetilde{C_{||}} \lhd C_0) \;=\; \{\, & (\{a\},\{ab\},\{b\}),\ (\{a\},\{cd\},\{b\}),\\ & (\{a\},\{abef\},\{b\}),\\ & (\{a\},\{ef\},\{a\}),\ (\{b\},\{ef\},\{b\})\,\}\end{aligned}$$

Finally, the complete extraction $\widetilde{C_0} = \widetilde{C_0||C_1} \lhd C_0$, which is illustrated in figure 5.10 (page 143), can be written as follows;

$$\begin{aligned}\widetilde{C_0} &= (\hat{Q}(\widetilde{C_{||}} \lhd C_0), \ddot{Q}(\widetilde{C_{||}} \lhd C_0), \hat{\Sigma}(\widetilde{C_{||}} \lhd C_0), \Gamma(\widetilde{C_{||}} \lhd C_0), \hat{\Delta}(\widetilde{C_{||}} \lhd C_0))\\ &= (\ \{\{a\},\{b\}\},\\ &\quad \{\{b\}\},\\ &\quad \{\{ab\},\{cd\},\{abef\},\{ef\}\},\\ &\quad \{\widetilde{\gamma_0}\},\\ &\quad \{(\{a\},\{ab\},\{b\}),\ (\{a\},\{cd\},\{b\}),\\ &\quad\ (\{a\},\{abef\},\{b\}),\\ &\quad\ (\{a\},\{ef\},\{a\}),\ (\{b\},\{ef\},\{b\})\}\\ &\ \ )\end{aligned}$$

5.4.2 Extraction of C_1 from $\widetilde{C_0 \| C_1}$

Let the extraction $\widetilde{C_1} = \widetilde{C_0 \| C_1} \lhd C_1$ be written as $\widetilde{C_1} = \widetilde{C_{\|}} \lhd C_1$, where C_1 is given below. Carefully note that for the extraction $\widetilde{C_{\|}} \lhd C_1$, any C' term in any of the definitions of extraction refers to C_1, and any derived C'' term infers C_0.

$$\begin{array}{lcl}
\hat{Q}(C_1) & = & \{\{e\},\{f\}\} \\
\ddot{Q}(C_1) & = & \{\{e\}\} \\
\hat{\Sigma}(C_1) & = & \{\{ef\}\} \\
\Gamma(C_1) & = & \{\gamma_1\} \\
\hat{\Delta}(C_1) & = & \{(\{e\},\{ef\},\{f\})\}
\end{array}$$

Evaluation of the state set and initial state set follow from the unrestricted example of section 5.3 (page 124). The idle event name is determined by $\mathcal{N}(\{\gamma_1\})$, and in this case $\{\widetilde{\gamma_1}\}$ will be used. Hence;

$$\begin{array}{lcl}
\hat{Q}(\widetilde{C_{\|}} \lhd C_1) & = & \{\{e\},\{f\}\} \\
\ddot{Q}(\widetilde{C_{\|}} \lhd C_1) & = & \{\{e\}\} \\
\Gamma(\widetilde{C_{\|}} \lhd C_1) & = & \{\widetilde{\gamma_1}\}
\end{array}$$

Unlike the unrestricted example of section 5.3, this example generates progressive and state dependent synchronising transitions and, therefore, shared event names. By Definition 5.3 (page 107), the evaluated event name set, which depends upon the transition set $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)$, evaluates as;

$$\hat{\Sigma}(\widetilde{C_{\|}} \lhd C_1) \;=\; \{\{ab\},\{cd\},\{ef\},\{abef\}\}$$

The event name $\{ef\}$ is a consequence of the event name set existence test. However, the event name $\{abef\}$ is contributed by the transition $(\{e\},\{abef\},\{f\}) \in \hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_P$ by case 3b (page 139). The event name $\{ab\}$ is contributed by the transition $(\{e\},\{ab\},\{e\}) \in \hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DA}$ (page 140). Finally, the event name $\{cd\}$ is contributed by the transitions $(\{e\},\{cd\},\{e\}) \in \hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DC}$ and $(\{f\},\{cd\},\{f\}) \in \hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DC}$ (page 141).

From Definition 5.10 (page 123), the transition set is formed from the union of the contributions of $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$ and $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$.

1. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_A$

   Table 5.19 lists those transitions $\hat{\Delta}(C_1)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\mathcal{E}'' = \mathcal{I}''$ contribute the transition $\hat{\Delta}(C_1)$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$. In this example, the last two pairings both contribute the transition $(\{e\}, \{ef\}, \{f\})$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_1)$	$\mathcal{E}''$	$\mathcal{I}''$	C
$(\{a,e\},\{ab,ef\},\{b,f\})$	$(\{e\},\{ef\},\{f\})$	$\{ab\}$	$\{\gamma_0\}$	
$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$		$\{\gamma_0\}$		$\star$
$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$		$\{\gamma_0\}$		$\star$

Table 5.19: Existent and contributing pairings of $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_A$

2. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_S$

   Table 5.20 lists those transitions $\hat{\Delta}(C_1)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} = \Sigma'$ contribute the transition $\hat{\Delta}(C_1)$ to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$. In this example there are no synchronous pairings, thus no transitions are contributed to $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$.

$\Delta_{\|\|}$	$\hat{\Delta}(C_1)$	$\Sigma_{\|\|}$	Σ'	C
$(\{a,e\},\{ab,ef\},\{b,f\})$	$(\{e\},\{ef\},\{f\})$	$\{ab,ef\}$	$\{ef\}$	
$(\{a,e\},\{\gamma_0,ef\},\{a,f\})$		$\{\gamma_0,ef\}$		
$(\{b,e\},\{\gamma_0,ef\},\{b,f\})$		$\{\gamma_0,ef\}$		

Table 5.20: Existent and contributing pairings of $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_S$

3. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_P$

   Table 5.21 lists those transitions $\hat{\Delta}(C_1)$ that exist in a $\Delta_{||}$ transition. However, only those pairings where $\Sigma_{||} \neq \Sigma'$ and $\mathcal{E}'' \neq \mathcal{I}''$ lead to term 5.6 holding true, and these pairings are marked in the E (existence) column. In this example, Term 5.6 holds true for the first pairing because $\{ab, ef\} \neq \{ef\}$ and $\{ab\} \neq \{\gamma_0\}$.

$\Delta_{\|}$	$\hat{\Delta}(C_1)$	$\Sigma_{\|}$	Σ'	$\mathcal{E}''$	$\mathcal{I}''$	E
$(\{a, e\}, \{ab, ef\}, \{b, f\})$	$(\{e\}, \{ef\}, \{f\})$	$\{ab, ef\}$	$\{ef\}$	$\{ab\}$	$\{\gamma_0\}$	†
$(\{a, e\}, \{\gamma_0, ef\}, \{a, f\})$		$\{\gamma_0, ef\}$		$\{\gamma_0\}$		
$(\{b, e\}, \{\gamma_0, ef\}, \{b, f\})$						

Table 5.21: Existent pairings of $\hat{\Delta}(\widetilde{C}_{\|} \lhd C_1)_P$

From the existence of the transition $(\{e\}, \{ef\}, \{f\})$ in the simultaneous transition $(\{a, e\}, \{ab, ef\}, \{b, f\})$, it is necessary to determine the absence of any related asynchronous transitions. For this pairing the following terms may be deduced by set difference of the corresponding $\widetilde{C}_{\|}$ and C_1 terms, hence $\mathcal{Q}'' = \{a\}$, $\mathcal{E}'' = \{ab\}$, $\mathcal{P}'' = \{b\}$ and $\mathcal{I}'' = \{\gamma_0\}$.

(a) Absent $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ transitions: Term 5.7 (page 114) gives $\hat{\mathcal{Q}}(\mathcal{C}'') = \{\{a\}, \{b\}\}$, and forms transitions of the form $(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$, where $\mathcal{S}'' \in \hat{\mathcal{Q}}(\mathcal{C}'')$, to determine absence from the transition set $\hat{\Delta}(\widetilde{C}_{\|})$. Table 5.22 lists the pairings and the transition formed for each pairing. Since none of the formed transitions is absent from $\hat{\Delta}(\widetilde{C}_{\|})$, progressive synchronisation is not introduced.

(Q', Σ', P')	$(\mathcal{S}'', \mathcal{I}'', \mathcal{S}'')$	$(Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'')$	A
$(\{e\}, \{ef\}, \{f\})$	$(\{a\}, \{\gamma_0\}, \{a\})$	$(\{a, e\}, \{\gamma_0, ef\}, \{a, f\})$	
	$(\{b\}, \{\gamma_0\}, \{b\})$	$(\{b, e\}, \{\gamma_0, ef\}, \{b, f\})$	

Table 5.22: Absent $\mathcal{A}\Delta_{(\{e\},\{ef\},\{f\}),(\mathcal{S}'',\mathcal{I}'',\mathcal{S}'')}$ pairings for $\hat{\Delta}(\widetilde{C}_{\|} \lhd C_1)_P$

(b) Absent $\mathcal{A}\Delta_{(\{a\},\{ab\},\{b\}),(S',\Gamma',S')}$ transitions: Term 5.8 (page 115) forms transitions of the form $(S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'')$, where $S' \in \hat{\mathcal{Q}}(C')$, to determine absence from the transition set $\hat{\Delta}(\widetilde{C}_{\|})$. Table 5.23 lists the pairings and the transition formed for each pairing. The pairing marked $\star$ is absent from the transition set $\hat{\Delta}(\widetilde{C}_{\|})$ and progressive synchronisation must be introduced. Thus a transition of the form $(\{e\}, \mathcal{N}(\{ab\} \cup \{ef\}), \{f\})$ must be contributed to the transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_P$. Let $\mathcal{N}(\{ab\} \cup \{ef\}) = \{abef\}$, hence the contributed transition will be $(\{e\}, \{abef\}, \{f\})$.

(S', Γ', S')	$(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$	$(S' \cup \mathcal{Q}'', \Gamma' \cup \mathcal{E}'', S' \cup \mathcal{P}'')$	A
$(\{e\}, \{\gamma_1\}, \{e\})$	$(\{a\}, \{ab\}, \{b\})$	$(\{a, e\}, \{ab, \gamma_1\}, \{b, e\})$	
$(\{f\}, \{\gamma_1\}, \{f\})$		$(\{a, f\}, \{ab, \gamma_1\}, \{b, f\})$	$\star$

Table 5.23: Absent $\mathcal{A}\Delta_{(\{a\},\{ab\},\{b\}),(S',\Gamma',S')}$ pairings for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_P$

4. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_{DA}$

Table 5.24 lists those implied idle transitions (Q', Γ', Q') of $\hat{\Delta}(C_1)$ that exist in a $\Delta_{||}$ transition, and lists the derived terms $\mathcal{Q}''$, $\mathcal{E}''$ and $\mathcal{P}''$. Recall that there will be an implied idle transition for every state $Q' \in \hat{Q}(C_1)$.

(Q', Γ', Q')	$\Delta_{\parallel}$	$\mathcal{Q}''$	$\mathcal{E}''$	$\mathcal{P}''$
$(\{e\}, \{\gamma_1\}, \{e\})$	$(\{a, e\}, \{ab, \gamma_1\}, \{b, e\})$	$\{a\}$	$\{ab\}$	$\{b\}$
	$(\{a, e\}, \{cd, \gamma_1\}, \{b, e\})$	$\{a\}$	$\{cd\}$	$\{b\}$
$(\{f\}, \{\gamma_1\}, \{f\})$	$(\{a, f\}, \{cd, \gamma_1\}, \{b, f\})$	$\{a\}$	$\{ab\}$	$\{b\}$

Table 5.24: Derivation of Δ'' transitions for $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_{DA}$

Table 5.25 lists the deduced $\mathcal{C}''$ transitions $(\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$, the implied idle transitions (S', Γ', S') and the formed transition for each pairing. Recall that there will be an implied idle transition for every state $S' \in \hat{Q}(C_1)$. In this example, the formed transition $(\{a, f\}, \{ab, \gamma_1\}, \{b, f\})$ is absent from $\hat{\Delta}(\widetilde{C_{||}})$, thus the state dependent transition $(Q', \mathcal{E}'', Q') = (\{e\}, \{ab\}, \{e\})$ is contributed to $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_{DA}$.

5. $\hat{\Delta}(\widetilde{C_{||}} \lhd C_1)_{DC}$

Table 5.24 showed that the idle transitions $(\{e\}, \{\gamma_1\}, \{e\})$ and $(\{f\}, \{\gamma_1\}, \{f\})$ of C_1 determine the existence of the $\Delta'' = (\mathcal{Q}'', \mathcal{E}'', \mathcal{P}'')$ transitions $(\{a\}, \{ab\}, \{b\})$ and $(\{a\}, \{cd\}, \{b\})$. The existence of the related simultaneous transitions must be

Q'	(S',Γ',S')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(S'\cup\mathcal{Q}'',\Gamma'\cup\mathcal{E}'',S'\cup\mathcal{P}'')$	A
$\{e\}$	$(\{e\},\{\gamma_1\},\{e\})$	$(\{a\},\{ab\},\{b\})$	$(\{a,e\},\{ab,\gamma_1\},\{b,e\})$	
	$(\{f\},\{\gamma_1\},\{f\})$		$(\{a,f\},\{ab,\gamma_1\},\{b,f\})$	$\star$
	$(\{e\},\{\gamma_1\},\{e\})$	$(\{a\},\{cd\},\{b\})$	$(\{a,e\},\{cd,\gamma_1\},\{b,e\})$	
	$(\{f\},\{\gamma_1\},\{f\})$		$(\{a,f\},\{cd,\gamma_1\},\{b,f\})$	
$\{f\}$	$(\{e\},\{\gamma_1\},\{e\})$	$(\{a\},\{cd\},\{b\})$	$(\{a,e\},\{cd,\gamma_1\},\{b,e\})$	
	$(\{f\},\{\gamma_1\},\{f\})$		$(\{a,f\},\{cd,\gamma_1\},\{b,f\})$	

Table 5.25: $\mathcal{A}\Delta_{\Delta_{\|},(S',\Gamma',S')}$ pairings for $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DA}$

determined, that is, pairings of the transitions $(\{a\},\{ab\},\{b\})$, $(\{a\},\{cd\},\{b\})$ with C_1 transitions with a *from* state or *to* state of $\{e\}$ or $\{f\}$.

(a) Related *from* state simultaneous transitions: Table 5.26 lists the deduced C'' transitions $(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$, and the C_1 transitions (Q',Σ',P') where the *from* state Q' is either $\{e\}$ or $\{f\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(\widetilde{C_{\|}})$. Note there are no C_1 transitions with a *from* state of $\{f\}$. In this example, the formed transition $(\{a,e\},\{cd,ef\},\{b,f\})$ is absent from $\hat{\Delta}(\widetilde{C_{\|}})$, therefore, the state dependent transition $(Q',\mathcal{E}'',Q') = (\{e\},\{cd\},\{e\})$ is contributed to $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DC}$.

Q'	(Q',Σ',P')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(Q'\cup\mathcal{Q}'',\Sigma'\cup\mathcal{E}'',P'\cup\mathcal{P}'')$	A
$\{e\}$	$(\{e\},\{ef\},\{f\})$	$(\{a\},\{ab\},\{b\})$	$(\{a,e\},\{ab,ef\},\{b,f\})$	
		$(\{a\},\{cd\},\{b\})$	$(\{a,e\},\{cd,ef\},\{b,f\})$	$\star$
$\{f\}$				

Table 5.26: Related *from* state transitions for $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DC}$

(b) Related *to* state simultaneous transitions: Table 5.27 lists the deduced C'' transitions $(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$, and the C_1 transitions (Q',Σ',P') where the *to* state P' is either $\{e\}$ or $\{f\}$. Each pairing is then formed, provided that the event Σ' is not synchronous, to determine its absence from $\hat{\Delta}(\widetilde{C_{\|}})$. Note there are no C_1 transitions with a *to* state of $\{e\}$. In this example, the formed transition $(\{a,e\},\{cd,ef\},\{b,f\})$ is absent from $\hat{\Delta}(\widetilde{C_{\|}})$, therefore, the state dependent transition $(P',\mathcal{E}'',P') = (\{f\},\{cd\},\{f\})$ is contributed to $\hat{\Delta}(\widetilde{C_{\|}} \lhd C_1)_{DC}$.

P'	(Q',Σ',P')	$(\mathcal{Q}'',\mathcal{E}'',\mathcal{P}'')$	$(Q'\cup\mathcal{Q}'',\Sigma'\cup\mathcal{E}'',P'\cup\mathcal{P}'')$	A
$\{e\}$				
$\{f\}$	$(\{e\},\{ef\},\{f\})$	$(\{a\},\{ab\},\{b\})$	$(\{a,e\},\{ab,ef\},\{b,f\})$	
		$(\{a\},\{cd\},\{b\})$	$(\{a,e\},\{cd,ef\},\{b,f\})$	$\star$

Table 5.27: Related *to* state transitions for $\hat{\Delta}(\widetilde{C}_{||} \lhd C_1)_{DC}$

From cases 1, 3b, 4, 5a and 5b the transition set $\hat{\Delta}(\widetilde{C}_{||} \lhd C_1)$ is as follows.

$$\begin{aligned}\hat{\Delta}(\widetilde{C}_{||} \lhd C_1) \;=\; \{ & (\{e\},\{ef\},\{f\}),\\ & (\{e\},\{abef\},\{f\}),\\ & (\{e\},\{ab\},\{e\}),\\ & (\{e\},\{cd\},\{e\}),\ (\{f\},\{cd\},\{f\}) \}\end{aligned}$$

Finally, the complete extraction $\widetilde{C_1} = \widetilde{C_0||C_1} \lhd C_1$, which is illustrated in figure 5.10 (page 143), can be written as follows;

$$\begin{aligned}\widetilde{C_1} &= (\hat{Q}(\widetilde{C}_{||} \lhd C_1), \check{Q}(\widetilde{C}_{||} \lhd C_1), \hat{\Sigma}(\widetilde{C}_{||} \lhd C_1), \Gamma(\widetilde{C}_{||} \lhd C_1), \hat{\Delta}(\widetilde{C}_{||} \lhd C_1))\\ &= (\ \{\{e\},\{f\}\},\\ &\qquad \{\{e\}\},\\ &\qquad \{\{ab\},\{cd\},\{ef\},\{abef\}\},\\ &\qquad \{\tilde{\gamma}_1\},\\ &\qquad \{(\{e\},\{ef\},\{f\}),\\ &\qquad\ (\{e\},\{abef\},\{f\}),\\ &\qquad\ (\{e\},\{ab\},\{e\}),\\ &\qquad\ (\{e\},\{cd\},\{e\}),\ (\{f\},\{cd\},\{f\})\}\ )\end{aligned}$$

5.4.3 Comparison of $\widetilde{C_0||C_1}$ and $\widetilde{C_0}||\widetilde{C_1}$

The extracted machines $\widetilde{C_0}$, evaluated in section 5.4.1, and $\widetilde{C_1}$, evaluated in section 5.4.2, are illustrated in figure 5.10. In this particular example both extracts are fully synchronous, that is, every event is common to both extracts. From the four event names, the five transitions in each extract can be interpreted as follows;

1. $\{ab\}$: $\widetilde{C_0}$ can only progress from state $\{a\}$ to $\{b\}$ when $\widetilde{C_1}$ remains in state $\{e\}$. This combination gives the transition $(\{a,e\},\{ab\},\{b,e\})$ in the composition $\widetilde{C_0}||\widetilde{C_1}$ illustrated in figure 5.11.

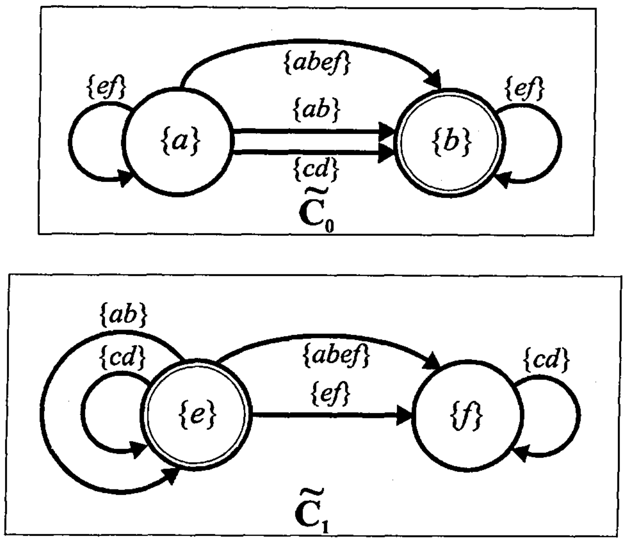


Figure 5.10: $\widetilde{C_0} = \widetilde{C_0||C_1} \lhd C_0$ (top) and $\widetilde{C_1} = \widetilde{C_0||C_1} \lhd C_1$ (bottom)

2. $\{cd\}$: $\widetilde{C_0}$ can only progress from state $\{a\}$ to $\{b\}$ when $\widetilde{C_1}$ remains in state $\{e\}$ or remains in state $\{f\}$. These two combinations give the transitions $(\{a,e\},\{cd\},\{b,e\})$ and $(\{a,f\},\{cd\},\{b,f\})$ in the composition $\widetilde{C_0}||\widetilde{C_1}$.

3. $\{abef\}$: $\widetilde{C_0}$ progress from state $\{a\}$ to $\{b\}$ synchronously with the progress of $\widetilde{C_1}$ from state $\{e\}$ to $\{f\}$. This combination gives the transition $(\{a,e\},\{abef\},\{b,f\})$ in the composition $\widetilde{C_0}||\widetilde{C_1}$.

4. $\{ef\}$: $\widetilde{C_1}$ can only progress from state $\{e\}$ to $\{f\}$ when $\widetilde{C_0}$ remains in state $\{a\}$ or remains in state $\{b\}$. These two combinations give the transitions $(\{a,e\},\{ef\},\{a,f\})$ and $(\{b,e\},\{ef\},\{b,f\})$ in the composition $\widetilde{C_0}||\widetilde{C_1}$.

Concurrent composition of the extracts $\widetilde{C_0}$ and $\widetilde{C_1}$ leads to the system $\widetilde{C_0}||\widetilde{C_1}$, which, along with the restricted system $\widetilde{C_0||C_1}$, is illustrated in figure 5.11. Observe that these two composite systems have the same structure, but they are not equal. Since asynchrony has been denied through the removal of some transitions, some synchronisation is expected. In particular, the asynchronous composite event names have been replaced by synchronous composite event names as follows; $\{ab\}$ replaces $\{ab,\gamma_1\}$, $\{cd\}$ replaces $\{cd,\gamma_1\}$ and $\{ef\}$ replaces $\{\gamma_0, ef\}$.

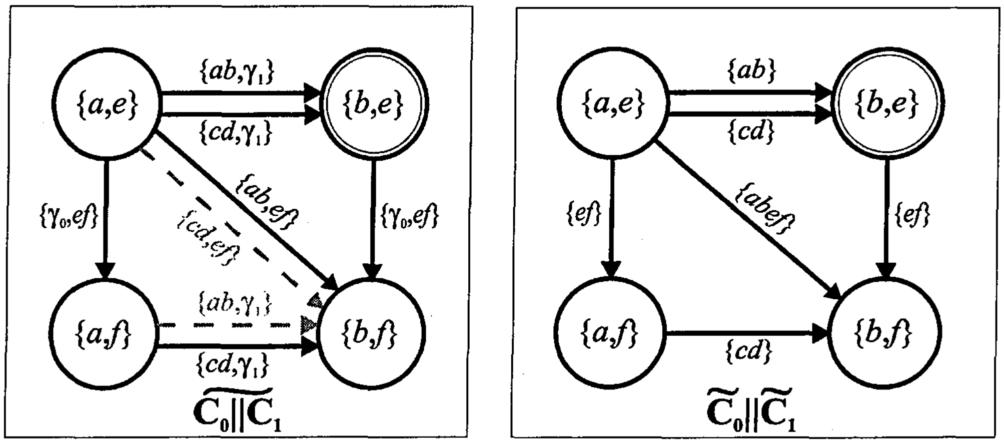


Figure 5.11: $\widetilde{C_0||C_1}$ (left), $\widetilde{C_0}||\widetilde{C_1}$ (right)

Any composite system reveals a set of events to which the components of that composition will react. In this example the events are $\{ab\}$, $\{cd\}$ and $\{ef\}$. However, the composition $\widetilde{C_0}||\widetilde{C_1}$, reveals not only the events $\{ab\}$, $\{cd\}$ and $\{ef\}$ but also the new event $\{abef\}$ which arises from progressive synchronisation. From the convention on precedence of progressive synchronisation (page 56), the event name $\{abef\}$, which is derived from $\mathcal{N}(\{ab\} \cup \{ef\})$, asserts that if the events $\{ab\}$ and $\{ef\}$ occur simultaneously then the system $\widetilde{C_0}||\widetilde{C_1}$ will progress from state from state $\{a, e\}$ to $\{b, f\}$ by event $\{abef\}$. Specifically, the system will not react to just the event $\{ab\}$ or just the event $\{ef\}$ through some non-deterministic choice.

The system designer now has to determine if the extracts $\widetilde{C_0}$ and $\widetilde{C_1}$ can be implemented and if the system defined by $\widetilde{C_0}||\widetilde{C_1}$ meets the requirements.

In this example, the original components C_0 and C_1, illustrated in figure 5.9 (page 131), describe the same behaviour as the example components used to illustrate the principles of extraction in section 5.1.1, *cf.* figure 5.1 (page 98). Therefore, the remaining transitions in the synchronous representations D_0 and D_1, illustrated in figure 5.3 (page 100), can be used to confirm the extracts $\widetilde{C_0}$ and $\widetilde{C_1}$, which are illustrated in figure 5.10.

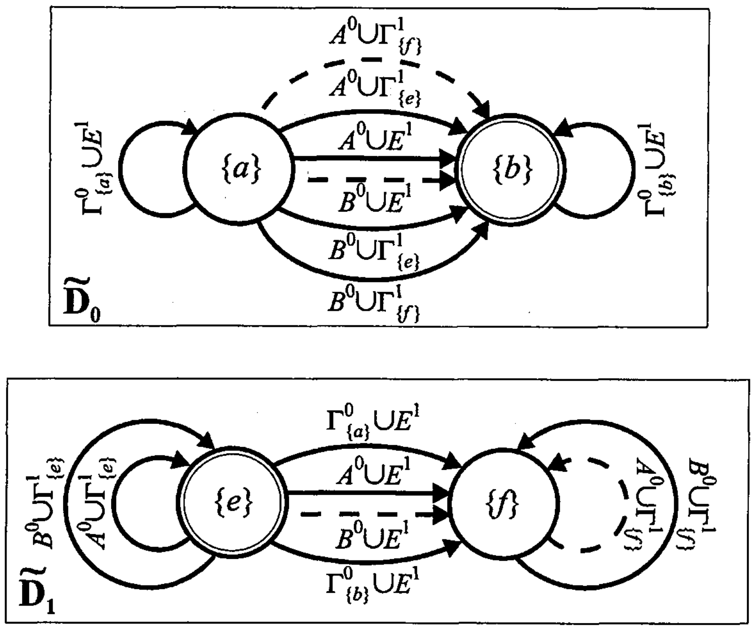

$A^0 \cup \Gamma^1_{\{f\}}$
$A^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup E^1$
$B^0 \cup E^1$
$B^0 \cup \Gamma^1_{\{e\}}$
$B^0 \cup \Gamma^1_{\{f\}}$
$\Gamma^0_{\{a\}} \cup E^1$
$\Gamma^0_{\{b\}} \cup E^1$
$\{a\}$
$\{b\}$
$\widetilde{\mathbf{D}}_0$
$\Gamma^0_{\{a\}} \cup E^1$
$A^0 \cup E^1$
$B^0 \cup E^1$
$\Gamma^0_{\{b\}} \cup E^1$
$B^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup \Gamma^1_{\{e\}}$
$A^0 \cup \Gamma^1_{\{f\}}$
$B^0 \cup \Gamma^1_{\{f\}}$
$\{e\}$
$\{f\}$
$\widetilde{\mathbf{D}}_1$

Figure 5.12: $\widetilde{D_0}$ (top) and $\widetilde{D_1}$ (bottom)

Figure 5.12 replicates figure 5.3, but the dotted arcs labelled $A^0 \cup \Gamma^1_{\{f\}}$ and $B^0 \cup E^1$ indicate removed transitions. Let the event name A^0 represent the event name $\{ab\}$, B^0 represent $\{cd\}$, E^1 represent $\{ef\}$ and $\Gamma^1_{\{f\}}$ represent the C_1 idle event name applied to the C_1 state $\{f\}$. Hence, the dotted arcs represent the removed transition labelled $\{cd, ef\}$ and the removed transition labelled $\{ab, \gamma_1\}$ between state $\{a, f\}$ and $\{b, f\}$ in the system $\widetilde{C_0 \| C_1}$ in figure 5.9 (page 131). Table 5.28 confirms the relationship between each $\widetilde{D_0}$ and $\widetilde{D_1}$ transition and each extract transition $\hat{\Delta}(\widetilde{C_0})$ and $\hat{\Delta}(\widetilde{C_1})$. Note that the "type" columns indicate if the extract transition is an original (O), state dependent (S), or progressive (P) transition.

$\hat{\Delta}(\widetilde{D_0})$, $\hat{\Delta}(\widetilde{D_1})$	$\hat{\Delta}(\widetilde{C_0})$	type	$\hat{\Delta}(\widetilde{C_1})$	type
$A^0 \cup E^1$	$(\{a\}, \{abef\}, \{b\})$	P	$(\{e\}, \{abef\}, \{f\})$	P
$A^0 \cup \Gamma^1_{\{e\}}$	$(\{a\}, \{ab\}, \{b\})$	O	$(\{e\}, \{ab\}, \{e\})$	S
$B^0 \cup \Gamma^1_{\{e\}}$	$(\{a\}, \{cd\}, \{b\})$	O	$(\{e\}, \{cd\}, \{e\})$	S
$B^0 \cup \Gamma^1_{\{f\}}$			$(\{f\}, \{cd\}, \{f\})$	S
$\Gamma^0_{\{a\}} \cup E^1$	$(\{a\}, \{ef\}, \{a\})$	S	$(\{e\}, \{ef\}, \{f\})$	O
$\Gamma^0_{\{b\}} \cup E^1$	$(\{b\}, \{ef\}, \{b\})$	S		

Table 5.28: Relationship between $\hat{\Delta}(\widetilde{D_0})$ and $\hat{\Delta}(\widetilde{C_0})$, and $\hat{\Delta}(\widetilde{D_1})$ and $\hat{\Delta}(\widetilde{C_1})$

5.5 Summary of Extraction

The extraction operator correctly generates an extract from a system which comprises two concurrent components, therefore, the extraction operator can be used to derive the required specification of the components.

Now, any extraction $\widetilde{C_{||}} \lhd C'$ defines how $\widetilde{C'}$ interacts with an unidentified component (or composite system), say X, where $\widetilde{C_{||}} = X||\widetilde{C'}$. This raises two important questions given a system of three or more concurrent components.

First, the expression $\widetilde{C'||C''||C'''} \lhd C'$ determines how the extract $\widetilde{C'}$ interacts with the system $\widetilde{C''||C'''}$. Likewise, the expression $\widetilde{C'||C''||C'''} \lhd (C''||C''')$ also determines how the extract $\widetilde{C''||C'''}$ interacts with the component $\widetilde{C'}$. It is not clear whether or not these two expressions yield the same result. Further analysis of the mathematical properties of the extraction operator is required.

Second, the expression $\widetilde{C'||C''||C'''} \lhd C'$ (as stated above) determines how the extract $\widetilde{C'}$ interacts with the system $\widetilde{C''||C'''}$, more significantly, the expression does not determine how the extract $\widetilde{C'}$ interacts with $\widetilde{C''}$ and $\widetilde{C'''}$ as separate entities. It is not clear if the concurrent compositions $\widetilde{C'}||\widetilde{C''}||\widetilde{C'''}$ and $\widetilde{C'||C''||C'''}$ are congruent. Again, further analysis of the mathematical properties of the extraction operator is required.

Dealing with synchronous transitions in extraction reveals a limitation in the current definition of the extraction operator and a deficiency in the CTS notation. Consider term 5.4 (page 112) which determines a synchronous event name in the components of a composite system by the test $\Sigma_{||} = \Sigma'$. Where the event name $\Sigma_{||}$ is formed under concurrent composition from the synchronous event names Σ' and Σ'', then $\Sigma' = \Sigma''$. This correctly contributes a synchronous transition to an extract. Now, where $\Sigma_{||}$ is formed from the component event names Σ', Σ'' and Σ''' then the method of detecting the presence of a C' transition synchronous with, say a C'' transition in a composite system $C'\|C''\|C'''$ is not always sufficient. Only when $\Sigma' = \Sigma'' = \Sigma'''$ will the test $\Sigma_{\|} = \Sigma'$ hold in the extraction $(C'\|C''\|C''') \lhd C'$.

Chapter 6

Example of an Application

This chapter illustrates a method of modelling resource contention using the Composite Transition System notation and the operators introduced in Chapter 4 and Chapter 5. A simple, application has been chosen because the method is applied using the algebra of the operators rather than computer based tools. Note that there is no objective to draw any specific conclusion about the application.

Each of the component processes of the system is specified in isolation, consequently, no consideration is given to any required co-ordination with the other components of the system. Such an approach simplifies the design of each component, but can lead to resource contention, that is, violation of the permitted behaviour of the shared resources. To prevent resource contention, the concurrent system formed from the components must be restricted just to the permitted behaviour of the shared resource. Component extraction based on the resource restricted system is then used to derive component specifications that are consistent with the restricted system and are devoid of resource contention.

Section 6.1 (page 151) describes a data acquisition system encountered in an commercial application. The component processes are modelled by Composite Transition Systems where the states describe the *use* and *non-use* of resources shared by the processes. Often only a single non-use state is required between any two use states. It will be seen in the following analysis that process models that reflect resource use and non-use can result in a small number of states and transitions. Minimising the number of component states and transitions minimises the number of states and transitions when all the possible combinations are created through concurrent composition, a phenomenon often referred to as *state space explosion.*

Section 6.2 (page 157) defines the permitted behaviour of the resources shared between the component processes of the system. To avoid introducing another notation, the resource models will be presented using the conventions of the CTS notation though they will not be subjected to any of the CTS operators.

The method adopted in this chapter takes the following steps;

1. Component Specification: Models of the component processes are derived from the description given in section 6.1. Let such component models be denoted U and V.
2. System Composition: The concurrent composition $U\|V$ is formed.
3. Application of Resource Constraints: Restriction of the composite system $U\|V$ according to the permitted behaviour of a shared resource, denoted R, is achieved as follows;
   (a) The states of $U\|V$ are mapped to the states of the resource model R. Consider the example illustrated in figure 6.1 (page 150) in which the resource states are shaded and the resource transitions are the bold directed arcs with open arrow heads. Thus, for this example, the $U\|V$ states $\{u_0, v_0\}$ and $\{u_1, v_0\}$ map to the resource state $\{r\}$, the $U\|V$ state $\{u_0, v_1\}$ maps to the resource state $\{s\}$, and the $U\|V$ state $\{u_1, v_1\}$ maps to the resource state $\{t\}$.
   (b) The *from* and *to* states of the transitions of the resource model R are replaced by the mapped states of $U\|V$. For the previous example, the resource transition $(\{s\}, \{\ldots\}, \{t\})$ would become $(\{u_0, v_1\}, \{\ldots\}, \{u_1, v_1\})$. Where a resource

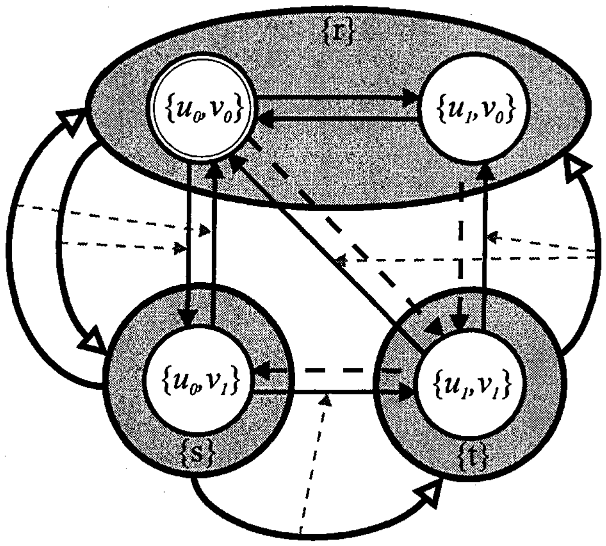


Figure 6.1: Mapping resource states and transitions to a composite system

state does not map to any state of $U\|V$, then that resource state is not permitted and any transitions *to* or *from* that state must be removed. For example, if resource state $\{r\}$ did not map to any $U\|V$ state, then any resource transition $(\{r\},\{\ldots\},\{\ldots\})$ must be removed. This step leads to a new resource model that is defined in terms of the states of $U\|V$. Let the transitions of this new model be denoted $\hat{\Delta}_R$.

(c) The transitions of the resource model $\hat{\Delta}_R$ define the permitted transitions between the states of $U\|V$. The $U\|V$ transition $(\{u_0,v_0\},\{\ldots\},\{u_0,v_1\})$, for example, is allowed in $\widetilde{U\|V}$ only because the resource model $\hat{\Delta}_R$ includes a transition from state $\{u_0,v_0\}$ to state $\{u_0,v_1\}$. Thus restricting the transitions of $\hat{\Delta}(U\|V)$ by $\hat{\Delta}_R$ takes the following form and determines the restricted transition set $\hat{\Delta}(\widetilde{U\|V})$. Hence;

$$\begin{aligned}\hat{\Delta}(\widetilde{U\|V}) \;=\; & \{(Q,\Sigma,P) \mid (Q,\Sigma,P) \in \hat{\Delta}(U\|V) \wedge \\ & \exists \tau \bullet (Q,\{\tau\},P) \in \hat{\Delta}_R\}\end{aligned} \tag{6.1}$$

4. Extraction: The required components $\tilde{U}$ and $\tilde{V}$ are evaluated by $\widetilde{U\|V} \lhd U$ and $\widetilde{U\|V} \lhd V$.

5. Verification: The concurrent composition of the extracts $\tilde{U}$ and $\tilde{V}$ is performed in order to verify that the system meets the requirements.

For brevity, only the result of each of the steps in the analysis will be presented unless any specific observations are made.

6.1 System Description

The example application is the radar data acquisition system illustrated in figure 6.2 as a block diagram.

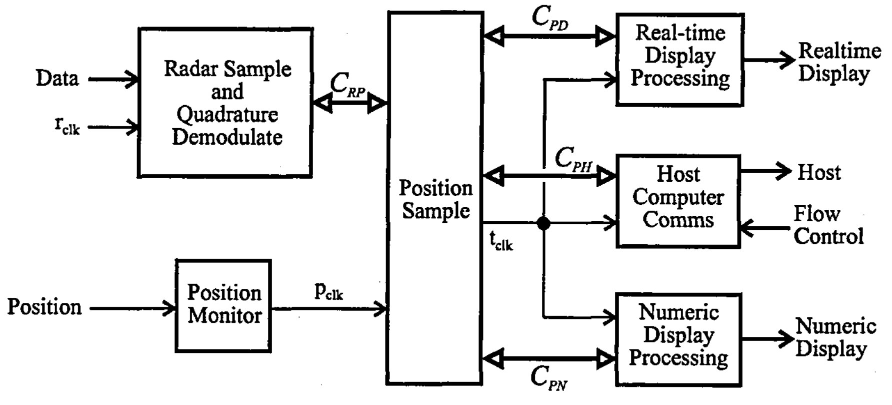


Figure 6.2: Acquisition System Block Diagram

In response to the clock signal r_{clk}, the *Radar Sample and Quadrature Demodulate* process, denoted R, samples anti-alias filtered quadrature multiplexed analogue signals via an analogue to digital converter. No samples may be missed as this will compromise the quadrature de-multiplexing of the sampled data (see section 6.1.1). The sample clock r_{clk} is periodic and of frequency, f_r. The execution time required to process the sampled data is assumed never to exceed the period of the highest possible sample clock frequency.

A sampling signal p_{clk} is generated by the *Position Monitor* which continuously monitors the position of the radar using data from a positional transducer. In response to the p_{clk} signal, the de-multiplexed data are sampled by the *Position Sample* process, denoted P. Note that the frequency of the p_{clk} depends upon the position sampling interval and the velocity of the radar. The maximum position sampling frequency f_p is known, and is much lower than the radar data sampling frequency, hence $f_p \ll f_r$.

The asynchrony of the r_{clk} and p_{clk} sampling clocks makes simultaneous access a possibility. From the sampling frequency inequality $f_p \ll f_r$, it can be deduced that the time available to process the radar data is much less than that for the position sampling. Hence, in the event of simultaneity, preference should be given to data sampling. Thus, notations based upon a non-deterministic choice in the event of simultaneity are not appropriate.

Further, as $f_p \ll f_r$, not all the radar data samples are position sampled. This means that only the newest radar data samples must be position sampled and any old data must be ignored, therefore, communication via the resource C_{RP} must be asynchronous and buffered. Hence, a model based on synchronous communication channels (such as those explored in Chapter 2), is not applicable, and the buffering cannot follow the first-in-first-out policy often assumed for such communication.

Communication of sampled data between processes R and P is via the shared resource denoted C_{RP}. The application of C_{RP} resource constraints to the processes R and P is presented in section 6.3 (page 160).

Position sampled data output by process P are required by process D which outputs derived data via a digital to analogue converter; none of the data may be lost. Communication between P and D is via the resource denoted C_{PD} and the application of C_{PD} resource constraints to the processes P and D is presented in section 6.4 (page 170).

The analysis of the interaction via the shared resources C_{RP} and C_{PD} is sufficient to illustrate the use of the notation for modelling resource level constraints and the derivation of the required process behaviours $\widetilde{R}$, $\widetilde{P}$ and $\widetilde{D}$. Interaction via the shared resources C_{PH} and C_{PN} is not presented.

6.1.1 Data Sampling and Quadrature De-multiplexing, R

In the actual system there are multiple anti-alias filtered input signals. The r_{clk} sampling signal simultaneously sets a sample–and–hold circuit on each input to the *hold* state and indicates that sampling should be performed. Use of sample–and–hold circuits ensures that the multiple inputs are all captured simultaneously as this is a requirement of the application. A single Analogue–to–Digital converter is used in turn to sample each of the multiple inputs before the sample–and–hold circuits are returned to the *sample* state. This introduces some latency in sampling the analogue signal. Provided $t_l + t_s \leq \frac{1}{f_r}$, where t_l is the sampling latency, t_s is the sampling time (from start to finish), and f_r is the data sampling clock frequency, then no r_{clk} signal will be missed.

The input, $s(t)$, is a signal in which $a(t)$ and $b(t)$ have been *quadrature multiplexed* [54, 93]. Quadrature multiplexing enables two signals to be transmitted simultaneously over a single linear transmission channel and is achieved by multiplying $a(t)$ and $b(t)$ by the carriers $\cos(\omega_c t)$ and $\sin(\omega_c t)$ respectively, and summing the result. Hence, $s(t) = a(t).\cos(\omega_c t) + b(t).\sin(\omega_c t)$. De-multiplexing requires the signal $s(t)$ to be multiplied by $\cos(\omega_c t)$ and $\sin(\omega_c t)$ and then low-pass filtered (the multiplication also generates terms at $2\omega_c t$). Any phase difference between the multiplexing and de-multiplexing carriers results in interference between $a(t)$ and $b(t)$ [93]. For example, an error of $\frac{\pi}{2}$ will interchange $a(t)$ and $b(t)$.

In this application, $a(t)$ and $b(t)$ represent quadrature components where the amplitude and phase change are important. Both $a(t)$ and $b(t)$ are very low frequency signals and, without quadrature multiplexing, two matched low-pass filters would be required for each channel. Quadrature multiplexing means that a single analogue band-pass filter centred at $\omega_c t$ is required. In the actual application, there are multiple inputs and matched band-pass filters are easier to build than low-pass filters.

The data sampling clock r_{clk} is coincident with carrier phases of $\frac{n\pi}{2}$, for $n = 0, 1, 2, 3, \ldots$ Hence there are 4 clocks per cycle of the carriers. The sampled quadrature multiplexed signal coincident with each clock, s_n, can be written as follows, and an algorithm is used to compute a value for a_n and b_n for $n = 0, 1, 2, 3, \ldots$.

$$s_n = a_n . \cos(\frac{n\pi}{2}) + b_n . \sin(\frac{n\pi}{2})$$

Note that the clock r_{clk} does not indicate if the carrier phase is 0, $\frac{\pi}{2}$, π, or $\frac{3\pi}{2}$. This impacts the absolute phase but not the amplitude of the signal represented by the quadrature components a_n and b_n. This phase impact is actually not important because the application requirement is to measure phase change. However, if m consecutive sample clocks were missed then the phase will step by $\frac{m\pi}{2}$ and so give erroneous phase change measurements, unless m happens to be a multiple of 4. Hence, no r_{clk} samples must be missed.

Figure 6.3 (page 155) is a simplified flow chart for the data sampling process R, where r_{clk} is the sample clock, r_{a+ib} is the quadrature multiplexed signal from which r_a and r_b are the obtained by demodulation. In another form of implementation it would be possible to move the demodulation into the position sampling process so that demodulation is only performed on each position sample. This would require the sampling process to deliver r_{a+ib} and n, where n increments, modulo 4, on each r_{clk}. Since $f_p \ll f_r$, the overall processor usage might be expected to be reduced.

Observe that the communication of the demodulated data between process R and the position sample process P has been separated into two steps. This has been done because the design choice was a shared memory interface where the possibility of simultaneity of r_{clk} and p_{clk} can lead to interleaved read and write operations on r_a and r_b. Failure to prevent such interleaving may result in the data set taken in response to one position clock comprising some undefined mix of current and previous radar data.

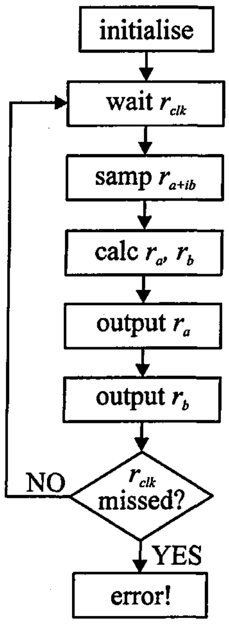


Figure 6.3: Data Sampling Process Flow Chart

6.1.2 Position Sampling, P

At each required position, the position sampling process must read the data provided by the demodulation process R and make this available to its recipients. At each sample position, the amplitude and phase represented by $a(p, t)$ and $b(p, t)$ are required. From this data, phase change is calculated as the phase at the current sample position minus the phase at the last sample position (modulo π). Therefore, if a position sampling point is missed then errors in the phase change measurement can result.

Figure 6.4 is a flow chart for the position sampling process P, where p_{clk} is the position sampling clock, and r_a and r_b are the demodulated sample data. These data are output and the recipients signalled with the clock t_{clk}. In the actual system, the position sampling process annotated the sample data with position information. The real-time display and host processes, D and H, must not miss the data suggesting synchronous communication based on t_{clk}. The numeric display process N can loose data but, because

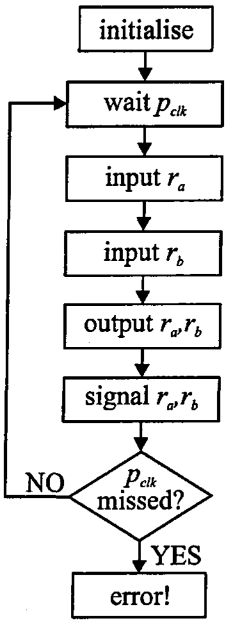


Figure 6.4: Position Sampling Flow Chart

it calculates phase change, then either phase change data must be provided, or process N must guarantee to collect data from two consecutive position samples.

6.1.3 Real-time Display, D

Figure 6.5 (page 157) is a flow chart for the real-time display process D (the flow chart for process H is identical). The data are read on each and every t_{clk}, processed and output. For process D, the output is to a digital–to–analogue converter which, for the purpose of this analysis, can be considered to be a write operation to a hardware register. In this way some timing relationship to the position sampling process P is maintained. For the process H, the data would be buffered to ensure efficient communication to the host.

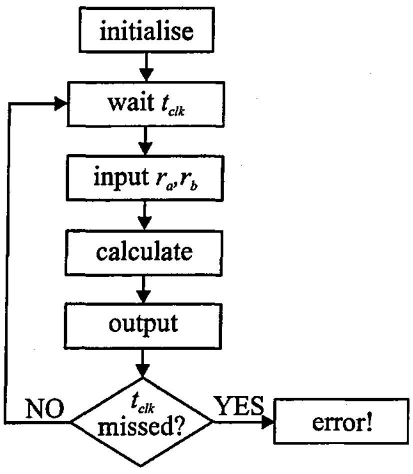


Figure 6.5: Real-time Display Flow Chart

6.2 Shared Resource Models

This section presents models of the shared resources C_{RP} and C_{PD}. These models will be used in the derivation of a restricted CTS model in sections 6.3 (page 160) and 6.4 (page 170).

6.2.1 C_{RP} Communication

Figure 6.6 (page 158) is a model of the communication resource C_{RP} which is the interface between the processes R and P. Input and output accesses must not be interwoven, however, the ordering of input and output operations is arbitrary as a consequence of the requirement for asynchronous access and the recognition that not all input data will be used. The states of C_{RP} are interpreted as follows;

1. State $\{f\}$, the initial state, indicates that the resource is *free*, that is, C_{RP} contains no new data and is not being accessed.
2. Output of r_a, denoted by the state $\{o_a\}$, must always be followed by the output of r_b, denoted by the state $\{o_b\}$. Meeting the requirement of no interwoven access

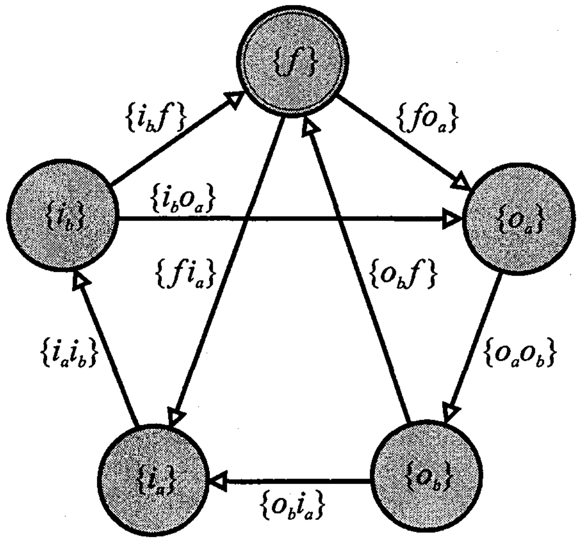


Figure 6.6: C_{RP} Resource Model

means that the state $\{o_b\}$ is the only possible *to* state of state $\{o_a\}$. Likewise, the input of r_b, denoted the state $\{i_b\}$, is the only possible *to state* of the state $\{i_a\}$ which denotes the input of r_a.

3. The transition $(\{o_b\}, \{o_b i_a\}, \{i_a\})$ represents $\{o_b\}$ completing and $\{i_a\}$ commencing simultaneously. This simultaneity can be either a coincidence or as a consequence of synchronisation.

4. The transition $(\{o_b\}, \{o_b f\}, \{f\})$ represents the case where $\{o_b\}$ completes but $\{i_a\}$ is not yet ready to commence. Observe that this return to state $\{f\}$ allows further $\{o_a\}$ and $\{o_b\}$ operations before any input operations and meets the requirement that not all input data are read.

5. The transition $(\{f\}, \{f i_a\}, \{i_a\})$ allows input operations to be performed except during output operations. This transition also means that input operations can be performed after $n = 0, 1, 2, \ldots$ output operations.

6. Once r_a and r_b have been input, there are two possibilities. First, the transition $(\{i_b\}, \{i_b o_a\}, \{o_a\})$ represents $\{i_b\}$ completing and $\{o_a\}$ commencing immediately. Second, the transition $(\{i_b\}, \{i_b f\}, \{f\})$ represents the case where $\{i_b\}$ completes, but $\{o_a\}$ is not yet ready to commence.

6.2.2 C_{PD} Communication

Figure 6.7 is a model of the communication resource between processes P and D. A strict output—then—input ordering is required, however, there is no requirement for the input operation to immediately follow the output operation.

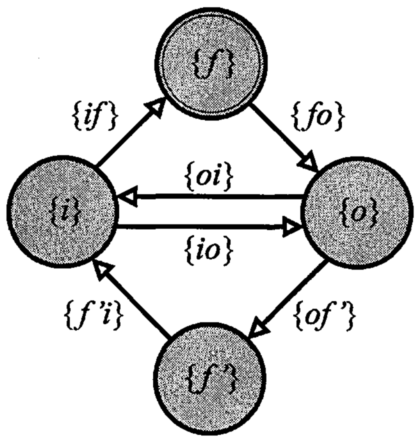


Figure 6.7: C_{PD} Resource Model

The states of C_{PD} are interpreted as follows;

1. State $\{f\}$, the initial state, indicates that the resource is *free*, that is C_{PD} contains either no valid data or valid data have already been input. From the requirement for an output—then—input ordering, the only possible *to* state is $\{o\}$.
2. Following an output, an input, $\{i\}$, must be performed. This can be either immediately reached by transition $(\{o\}, \{oi\}, \{i\})$, or later by transition $(\{o\}, \{of'\}, \{f'\})$ to the state $\{f'\}$, followed by transition $(\{f'\}, \{f'i\}, \{i\})$ to state $\{i\}$.
3. The state $\{f'\}$ indicates that the resource contains valid data but these data are awaiting an input operation.
4. Following an input, another output can be performed. This output can be immediate, by transition $(\{i\}, \{io\}, \{o\})$, or later by transition $(\{i\}, \{if\}, \{f\})$ to state $\{f\}$ followed by transition $(\{f\}, \{fo\}, \{o\})$ to state $\{o\}$.

6.3 Interaction between R and P due to resource C_{RP}

This section determines the required interaction between processes R and P to meet the constraints of the shared resource C_{RP}. Hence the modified components $\widetilde{R}$ and $\widetilde{P}$ are derived using the expressions $\widetilde{R} = (\widetilde{R\|P}) \lhd R$ and $\widetilde{P} = (\widetilde{R\|P}) \lhd P$. The method follows the five steps given on page 149.

6.3.1 Specification of Components R and P

The states of process R inferred from the flow chart of figure 6.3 (page 155) include waiting for r_{clk}, sampling r_{a+ib}, calculating r_a and r_b and checking if r_{clk} has been missed. These operations do not use the shared resource C_{RP} and will be abstracted to form a single state $\{r_w\}$. The asynchronous execution of the processes R and P means that consecutive access to the a and b elements of the shared resource C_{RP} must be guaranteed. Therefore, it is important that the output of r_a and r_b is modelled as separate states. A process R resource C_{RP} use model is illustrated in figure 6.8.

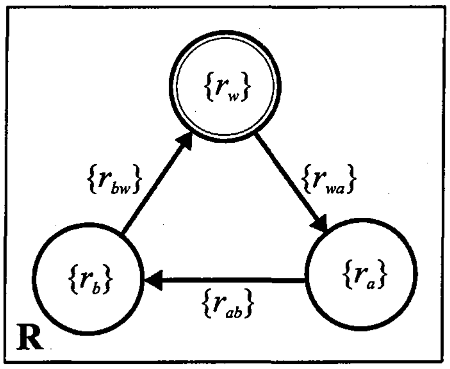


Figure 6.8: Process R resource C_{RP} use model

The states of process P inferred from the flow chart of figure 6.4 (page 156) include waiting for p_{clk}, delivery of r_a and r_b, and checking if p_{clk} has been missed. These operations do not access the shared resource C_{RP} and will be abstracted to form a single state $\{p_w\}$. The input of r_a and r_b does require access to the resource C_{RP} and this will be modelled by the states $\{p_a\}$ and $\{p_b\}$. A process P resource C_{RP} use model is illustrated in figure 6.9.

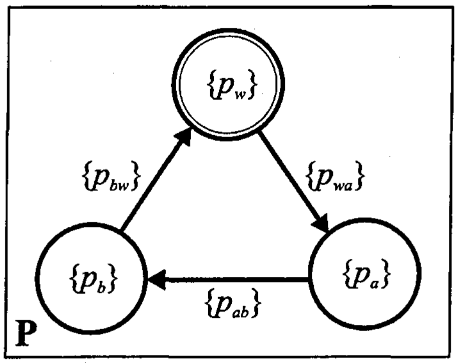


Figure 6.9: Process P resource C_{RP} use model

6.3.2 Composition of System $R\|P$

The composition $R\|P$ leads to the following CTS, where γ_R and γ_P are the idle event name identifiers for R and P respectively. The composite system $R\|P$ is not illustrated.

$$
\begin{aligned}
\hat{Q}(R\|P) &= \{\{r_w,p_w\},\{r_w,p_a\},\{r_w,p_b\},\{r_a,p_w\},\{r_a,p_a\},\{r_a,p_b\},\\
&\quad \{r_b,p_w\},\{r_b,p_a\},\{r_b,p_b\}\}\\
\ddot{Q}(R\|P) &= \{\{r_w,p_w\}\}\\
\hat{\Sigma}(R\|P) &= \{\{r_{wa},\gamma_P\},\{r_{ab},\gamma_P\},\{r_{bw},\gamma_P\},\{\gamma_R,p_{wa}\},\{\gamma_R,p_{ab}\},\{\gamma_R,p_{bw}\},\\
&\quad \{r_{wa},p_{wa}\},\{r_{wa},p_{ab}\},\{r_{wa},p_{bw}\},\{r_{ab},p_{wa}\},\{r_{ab},p_{ab}\},\{r_{ab},p_{bw}\},\\
&\quad \{r_{bw},p_{wa}\},\{r_{bw},p_{ab}\},\{r_{bw},p_{bw}\}\}\\
\Gamma(R\|P) &= \{\gamma_R,\gamma_P\}\\
\hat{\Delta}(R\|P) &= \{(\{r_a,p_a\},\{\gamma_R,p_{ab}\},\{r_a,p_b\}),(\{r_a,p_a\},\{r_{ab},\gamma_P\},\{r_b,p_a\}),\\
&\quad (\{r_a,p_a\},\{r_{ab},p_{ab}\},\{r_b,p_b\}),(\{r_a,p_b\},\{\gamma_R,p_{bw}\},\{r_a,p_w\}),\\
&\quad (\{r_a,p_b\},\{r_{ab},\gamma_P\},\{r_b,p_b\}),(\{r_a,p_b\},\{r_{ab},p_{bw}\},\{r_b,p_w\}),\\
&\quad (\{r_a,p_w\},\{\gamma_R,p_{wa}\},\{r_a,p_a\}),(\{r_a,p_w\},\{r_{ab},\gamma_P\},\{r_b,p_w\}),\\
&\quad (\{r_a,p_w\},\{r_{ab},p_{wa}\},\{r_b,p_a\}),(\{r_b,p_a\},\{\gamma_R,p_{ab}\},\{r_b,p_b\}),\\
&\quad (\{r_b,p_a\},\{r_{bw},\gamma_P\},\{r_w,p_a\}),(\{r_b,p_a\},\{r_{bw},p_{ab}\},\{r_w,p_b\}),\\
&\quad (\{r_b,p_b\},\{\gamma_R,p_{bw}\},\{r_b,p_w\}),(\{r_b,p_b\},\{r_{bw},\gamma_P\},\{r_w,p_b\}),\\
&\quad (\{r_b,p_b\},\{r_{bw},p_{bw}\},\{r_w,p_w\}),(\{r_b,p_w\},\{\gamma_R,p_{wa}\},\{r_b,p_a\}),\\
&\quad (\{r_b,p_w\},\{r_{bw},\gamma_P\},\{r_w,p_w\}),(\{r_b,p_w\},\{r_{bw},p_{wa}\},\{r_w,p_a\}),\\
&\quad (\{r_w,p_a\},\{\gamma_R,p_{ab}\},\{r_w,p_b\}),(\{r_w,p_a\},\{r_{wa},\gamma_P\},\{r_a,p_a\}),\\
&\quad (\{r_w,p_a\},\{r_{wa},p_{ab}\},\{r_a,p_b\}),(\{r_w,p_b\},\{\gamma_R,p_{bw}\},\{r_w,p_w\}),\\
&\quad (\{r_w,p_b\},\{r_{wa},\gamma_P\},\{r_a,p_b\}),(\{r_w,p_b\},\{r_{wa},p_{bw}\},\{r_a,p_w\}),\\
&\quad (\{r_w,p_w\},\{\gamma_R,p_{wa}\},\{r_w,p_a\}),(\{r_w,p_w\},\{r_{wa},\gamma_P\},\{r_a,p_w\}),\\
&\quad (\{r_w,p_w\},\{r_{wa},p_{wa}\},\{r_a,p_a\})\}
\end{aligned}
$$

6.3.3 Application of C_{RP} Resource Constraints to give $\widetilde{R||P}$

Restriction of the behaviour of $R||P$ by the shared resource C_{RP} requires analysis of the behaviour of the resource and the relationship between the resource transitions and the transitions $\hat{\Delta}(R||P)$ of $R||P$. From figure 6.6 (page 158), the C_{RP} states can be mapped to the states of $R||P$ as follows.

1. State $\{f\}$ indicates that C_{RP} is free, that is, neither process R nor P are accessing the resource. Hence, process R must be in the state $\{r_w\}$ and process P must be in state $\{p_w\}$. Hence $\{f\}$ maps to $\{r_w, p_w\}$.
2. State $\{o_a\}$ indicates that C_{RP} is being accessed for the output of the value r_a, thus process R must be in state $\{r_a\}$. Further, process P must not be accessing C_{RP}, thus P must be in state $\{p_w\}$. Hence $\{o_a\}$ maps to $\{r_a, p_w\}$.
3. State $\{o_b\}$ follows in a similar way to state $\{o_a\}$, hence $\{o_b\}$ maps to $\{r_b, p_w\}$.
4. State $\{i_a\}$ indicates that C_{RP} is being accessed for the input of the value r_a, thus process P must be in state $\{p_a\}$. Further, process R must not be accessing C_{RP}, hence process R must be in state $\{r_w\}$. Hence $\{i_a\}$ maps to $\{r_w, p_a\}$.
5. State $\{i_b\}$ follows in a similar way to state $\{i_a\}$, hence $\{o_b\}$ maps to $\{r_w, p_b\}$.

From the state mappings, the restricted form of $R||P$ can be determined from the transition relationship. In other words, transitions between the states of the shared resource C_{RP} determine the permitted transitions between the states of $R||P$. The transitions of C_{RP}, as illustrated in figure 6.6 (page 158), are as follows;

$$\begin{aligned}\hat{\Delta}(C_{RP}) = \{ & (\{f\},\{fo_a\},\{o_a\}),\ (\{o_a\},\{o_ao_b\},\{o_b\}),\ (\{o_b\},\{o_bi_a\},\{i_a\}),\\ & (\{i_a\},\{i_ai_b\},\{i_b\}),\ (\{i_b\},\{i_bf\},\{f\}),\ (\{f\},\{fi_a\},\{i_a\}),\\ & (\{o_b\},\{o_bf\},\{f\}),\ (\{i_b\},\{i_bo_a\},\{o_a\})\}\end{aligned}$$

Figure 6.10 illustrates the relationship between the states of C_{RP} and $R||P$, and the permitted transitions of $\hat{\Delta}(C_{RP})$. The outer circles are the states of the resource C_{RP} and the directed arcs are the transitions of the resource C_{RP}. The inner circles are the states of $R||P$ but, for clarity, the transitions of $R||P$ have been omitted. Note that the C_{RP} transitions only define possible transitions of $R||P$, in other words, there may be zero, one or many $R||P$ transitions. For example, the C_{RP} transition from $\{f\}$ to $\{i_a\}$ permits $R||P$ to progress from state $\{r_w, p_w\}$ to state $\{r_w, p_a\}$. Conversely, the absence of a C_{RP} transition from $\{i_b\}$ to $\{i_a\}$ would deny the progress of $R||P$ from state $\{r_w, p_b\}$ to $\{r_w, p_a\}$ if there was such an $R||P$ transition.

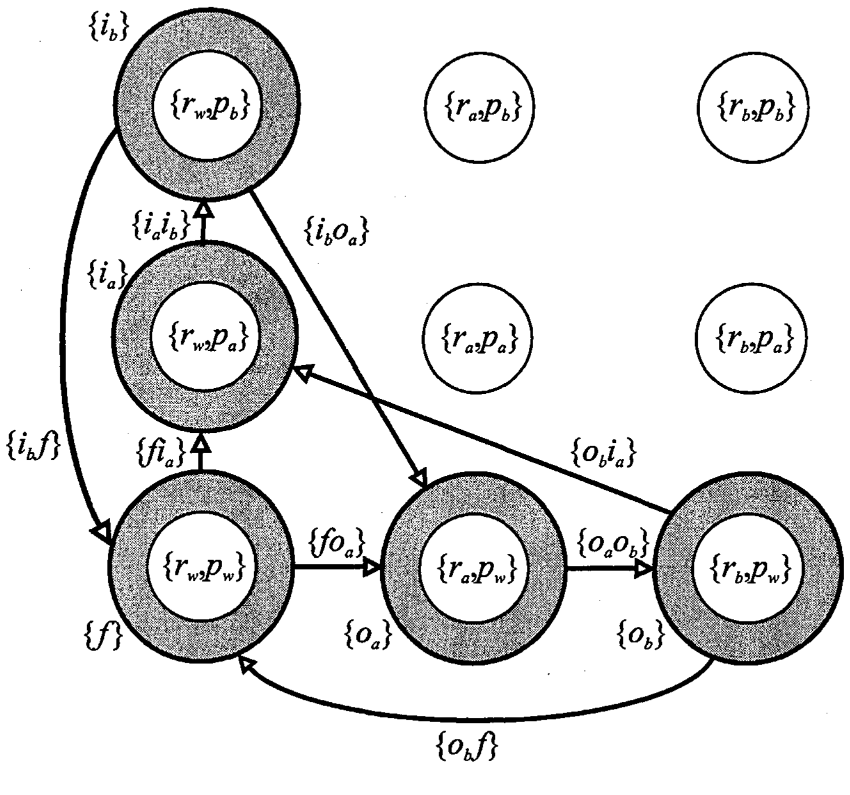


Figure 6.10: C_{RP} permitted states and transitions of $R||P$

By substitution of the mapped states of $\hat{Q}(C_{RP})$, a transition set $\hat{\Delta}_{C_{RP}}$ can be deduced. For example, consider the transition, $(\{f\}, \{fo_a\}, \{o_a\}) \in \hat{\Delta}(C_{RP})$. The state $\{r_w, p_w\}$ can be substituted for state $\{f\}$. Likewise, the state $\{r_a, p_w\}$ can be substituted for state $\{o_a\}$. This is the first transition in $\hat{\Delta}_{C_{RP}}$, the complete set is as follows;

$$\begin{aligned}\hat{\Delta}_{C_{RP}} = \{&(\{r_w, p_w\}, \{fo_a\}, \{r_a, p_w\}), (\{r_a, p_w\}, \{o_a o_b\}, \{r_b, p_w\}),\\ &(\{r_b, p_w\}, \{o_b f\}, \{r_w, p_w\}), (\{r_w, p_w\}, \{fi_a\}, \{r_w, p_a\}),\\ &(\{r_w, p_a\}, \{i_a i_b\}, \{r_w, p_b\}), (\{r_w, p_b\}, \{i_b f\}, \{r_w, p_w\}),\\ &(\{r_w, p_b\}, \{i_b o_a\}, \{r_a, p_w\}), (\{r_b, p_w\}, \{o_b i_a\}, \{r_w, p_a\})\}\end{aligned}$$

The set $\hat{\Delta}_{C_{RP}}$ defines those transitions between the states of $\hat{Q}(R||P)$ that are permitted by the states and transitions of the shared resource C_{RP}. In other words, the structure represented by the set $\hat{\Delta}_{C_{RP}}$ defines the extent of the permitted behaviour and must be applied to $R||P$ to form $\widetilde{R||P}$. Specifically, the transition set $\hat{\Delta}(\widetilde{R||P})$ must only contain transitions that are permitted by the set $\hat{\Delta}_{C_{RP}}$. Hence the transition set $\hat{\Delta}(\widetilde{R||P})$ can be stated and evaluated as follows. Figure 6.11 illustrates $\widetilde{R||P}$.

$$\begin{aligned}\hat{\Delta}(\widetilde{R||P}) &= \{(Q, \Sigma, P) | (Q, \Sigma, P) \in \hat{\Delta}(R||P) \wedge \exists\tau \bullet (Q, \{\tau\}, P) \in \hat{\Delta}_{C_{RP}}\}\\ &= \{(\{r_w, p_w\}, \{r_{wa}, \gamma_P\}, \{r_a, p_w\}), (\{r_a, p_w\}, \{r_{ab}, \gamma_P\}, \{r_b, p_w\}),\\ &\quad (\{r_b, p_w\}, \{r_{bw}, \gamma_P\}, \{r_w, p_w\}), (\{r_w, p_w\}, \{\gamma_R, p_{wa}\}, \{r_w, p_a\}),\\ &\quad (\{r_w, p_a\}, \{\gamma_R, p_{ab}\}, \{r_w, p_b\}), (\{r_w, p_b\}, \{\gamma_R, p_{bw}\}, \{r_w, p_w\}),\\ &\quad (\{r_w, p_b\}, \{r_{wa}, p_{bw}\}, \{r_a, p_w\}), (\{r_b, p_w\}, \{r_{bw}, p_{wa}\}, \{r_w, p_a\})\}\end{aligned}$$

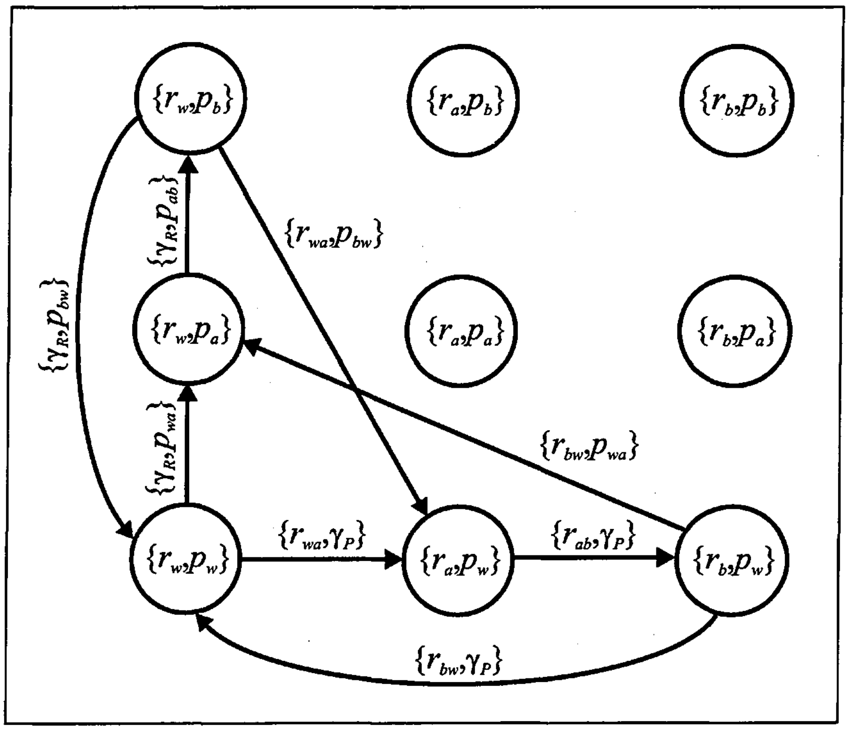


Figure 6.11: $\widetilde{R||P}$

Finally, observe that $\widetilde{R\|P}$ can be defined as follows, noting that only the transition set has been restricted;

$$\widetilde{R\|P} = (\hat{Q}(R\|P), \ddot{Q}(R\|P), \hat{\Sigma}(R\|P), \Gamma(R\|P), \hat{\Delta}(\widetilde{R\|P}))$$

6.3.4 Extraction to derive $\widetilde{R}$ and $\widetilde{P}$

Evaluation of $\widetilde{R} = \widetilde{R\|P} \lhd R$

In the extraction $\widetilde{R} = \widetilde{R\|P} \lhd R$ the state set $\hat{Q}(\widetilde{R})$, the initial state set $\ddot{Q}(\widetilde{R})$, and the idle event name $\Gamma(\widetilde{R})$ evaluate as follows.

$$\begin{aligned}
\hat{Q}(\widetilde{R}) &= \{\{r_w\}, \{r_a\}, \{r_b\}\} \\
\ddot{Q}(\widetilde{R}) &= \{\{r_w\}\} \\
\Gamma(\widetilde{R}) &= \{\widetilde{\gamma_R}\}
\end{aligned}$$

From Definition 5.3 (page 107), the event name set includes terms from the transition set. Therefore the transition set must be evaluated prior to the evaluation of the event name set.

1. $\hat{\Delta}((\widetilde{R\|P}) \lhd R)_A$: All the asynchronous transitions of $\hat{\Delta}(R)$ exist in the transitions of $\hat{\Delta}(\widetilde{R\|P})$, hence;

$$\begin{aligned}
\hat{\Delta}((\widetilde{R\|P}) \lhd R)_A = \{&(\{r_w\}, \{r_{wa}\}, \{r_a\}), (\{r_a\}, \{r_{ab}\}, \{r_b\}), \\
&(\{r_b\}, \{r_{bw}\}, \{r_w\})\}
\end{aligned}$$

2. $\hat{\Delta}((\widetilde{R\|P}) \lhd R)_S$: There are no synchronous transitions so there is no contribution to the transition set.

3. $\hat{\Delta}((\widetilde{R\|P}) \lhd R)_P$: Progressive synchronisation arises because some of the expected "horizontal" and "vertical" transitions related to the $(\{r_w, p_b\}, \{r_{wa}, p_{bw}\}, \{r_a, p_w\})$ "diagonal" transition, formed from $(\{r_w\}, \{r_{wa}\}, \{r_a\})$, do not exist in $\hat{\Delta}(\widetilde{R\|P})$. The required progressive synchronisation transition will use the new synchronising event

name $\{wabw\} = \mathcal{N}(\{r_{wa}\} \cup \{p_{bw}\})$. Similarly, the $(\{r_b, p_w\}, \{r_{bw}, p_{wa}\}, \{r_w, p_a\})$ "diagonal" transition, formed from $(\{r_b\}, \{r_{bw}\}, \{r_w\})$, must become progressive, in this case the new synchronising event name will be $\{bwwa\} = \mathcal{N}(\{r_{bw}\} \cup \{p_{wa}\})$, hence;

$$\hat{\Delta}((\widetilde{R \| P}) \lhd R)_P \;=\; \{(\{r_w\}, \{wabw\}, \{r_a\}), (\{r_b\}, \{bwwa\}, \{r_w\})\}$$

4. $\hat{\Delta}((\widetilde{R \| P}) \lhd R)_D$: State dependent synchronisation arises because some of the expected asynchronous transitions are absent from $\hat{\Delta}(\widetilde{R \| P})$. For example the "vertical" transition from state $\{r_w, p_w\}$ to state $\{r_w, p_a\}$ by event name $\{\gamma_R, p_{wa}\}$ exists, but the related "vertical" transition from state $\{r_a, p_w\}$ to state $\{r_a, p_a\}$ by event name $\{\gamma_R, p_{wa}\}$ is absent, hence the state dependent synchronisation transition $(\{r_w\}, \{p_{wa}\}, \{r_w\})$ is required. Likewise, the expected transitions from $\{r_a, p_a\}$ to $\{r_a, p_b\}$ by event name $\{\gamma_R, p_{ab}\}$ and from $\{r_a, p_b\}$ to $\{r_a, p_w\}$ by event name $\{\gamma_R, p_{bw}\}$ are also absent. Hence;

$$\begin{aligned}\hat{\Delta}((\widetilde{R \| P}) \lhd R)_D \;=\; &\{(\{r_w\}, \{p_{wa}\}, \{r_w\}), (\{r_w\}, \{p_{ab}\}, \{r_w\}),\\ &(\{r_w\}, \{p_{bw}\}, \{r_w\})\}\end{aligned}$$

Recall that state dependent synchronisation also arises from absent *simultaneous* transitions (Definition 5.9, page 123). This cause of state dependent synchronisation also occurs in this extraction and contributes the same set of transitions.

Finally, the complete extract transition set can be written as follows, and the event name set evaluated;

$$\begin{aligned}\hat{\Delta}((\widetilde{R \| P}) \lhd R) \;=\; &\{(\{r_w\}, \{r_{wa}\}, \{r_a\}),\ (\{r_a\}, \{r_{ab}\}, \{r_b\}),\\ &(\{r_b\}, \{r_{bw}\}, \{r_w\}),\ (\{r_w\}, \{wabw\}, \{r_a\}),\\ &(\{r_b\}, \{bwwa\}, \{r_w\}),\ (\{r_w\}, \{p_{wa}\}, \{r_w\}),\\ &(\{r_w\}, \{p_{ab}\}, \{r_w\}),\ (\{r_w\}, \{p_{bw}\}, \{r_w\})\}\end{aligned}$$

$$\begin{aligned}\hat{\Sigma}((\widetilde{R \| P}) \lhd R) \;=\; &\{\{r_{wa}\}, \{r_{ab}\}, \{r_{bw}\},\\ &\{wabw\}, \{bwwa\},\\ &\{p_{wa}\}, \{p_{ab}\}, \{p_{bw}\}\}\end{aligned}$$

Figure 6.12 illustrates the extracted component $\widetilde{R}$. The three reflexive state dependent transitions on state $\{r_w\}$ require $\widetilde{R}$ to stay in this state if component $\widetilde{P}$ progresses by events $\{p_{wa}\}$, $\{p_{ab}\}$ or $\{p_{bw}\}$. In this application this synchronisation prevents component $\widetilde{R}$ from accessing the shared resource C_{RP} if component $\widetilde{P}$ is accessing the resource.

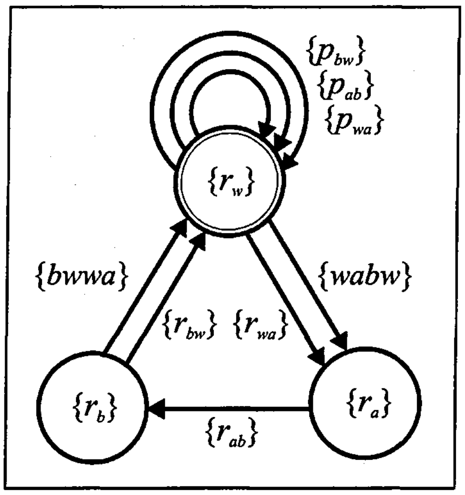


Figure 6.12: $\widetilde{R} = (\widetilde{R\|P}) \lhd R$

The progressive synchronisation transition from state $\{r_w\}$ to state $\{r_a\}$ by event name $\{wabw\}$ allows component $\widetilde{R}$ to start accessing the resource C_{RP}, if component $\widetilde{P}$ simultaneously stops accessing the resource by simultaneously progressing from state $\{p_b\}$ to state $\{p_w\}$. Similarly, the progressive synchronisation transition from state $\{r_b\}$ to state $\{r_w\}$ by event name $\{bwwa\}$ allows component $\widetilde{P}$ to start accessing the resource C_{RP} by progressing from state $\{p_w\}$ to state $\{p_a\}$, if component $\widetilde{R}$ simultaneously stops accessing the resource. The extracted component $\widetilde{P}$ is described in the next section.

Evaluation of $\widetilde{P} = \widetilde{R\|P} \lhd P$

In the analysis of the use of resource C_{RP}, component P has the same structure as component R and because of the symmetry in the restricted machine $\widetilde{R\|P}$, the extraction $\widetilde{P} = \widetilde{R\|P} \lhd P$ follows in a similar way to the extraction of $\widetilde{R}$.

The state set, the initial state set, and the idle event name evaluate as follows.

$$\begin{aligned}\hat{Q}(\widetilde{P}) &= \{\{p_w\},\{p_a\},\{p_b\}\}\\ \ddot{Q}(\widetilde{P}) &= \{\{p_w\}\}\\ \Gamma(\widetilde{P}) &= \{\widetilde{\gamma_P}\}\end{aligned}$$

The transition set also follows in a similar way, but with the following notable differences. First, because some of the transitions related to the $(\{r_w,p_b\},\{r_{wa},p_{bw}\},\{r_a,p_w\})$ and $(\{r_b,p_w\},\{r_{bw},p_{wa}\},\{r_w,p_a\})$ "diagonal" transitions do not exist in $\hat{\Delta}(\widetilde{R\|P})$, progressive synchronisation is required. The new event names are $\{wabw\} = \mathcal{N}(\{r_{wa}\} \cup \{p_{bw}\})$ and $\{bwwa\} = \mathcal{N}(\{r_{bw}\} \cup \{p_{wa}\})$ respectively. Hence;

$$\hat{\Delta}((\widetilde{R\|P}) \lhd P)_P = \{(\{p_b\},\{wabw\},\{p_w\}),(\{p_w\},\{bwwa\},\{p_a\})\}$$

Second, state dependent synchronisation arises because the expected "horizontal" asynchronous transitions $(\{r_w,p_a\},\{r_{wa},\gamma_P\},\{r_a,p_a\})$, $(\{r_a,p_a\},\{r_{ab},\gamma_P\},\{r_b,p_a\})$ and $(\{r_b,p_a\},\{r_{bw},\gamma_P\},\{r_w,p_a\})$ are absent. Hence;

$$\begin{aligned}\hat{\Delta}((\widetilde{R\|P}) \lhd P)_D = \{&(\{p_w\},\{r_{wa}\},\{p_w\}),(\{p_w\},\{r_{ab}\},\{p_w\}),\\ &(\{p_w\},\{r_{bw}\},\{p_w\})\}\end{aligned}$$

Finally, the complete extraction transition set can be written as follows, and the event name set evaluated;

$$\begin{aligned}\hat{\Delta}((\widetilde{R\|P}) \lhd P) &= \{(\{p_w\},\{p_{wa}\},\{p_a\}),(\{p_a\},\{p_{ab}\},\{p_b\}),\\ &\quad (\{p_b\},\{p_{bw}\},\{p_w\}),(\{p_w\},\{bwwa\},\{p_a\}),\\ &\quad (\{p_b\},\{wabw\},\{p_w\}),(\{p_w\},\{r_{wa}\},\{p_w\}),\\ &\quad (\{p_w\},\{r_{ab}\},\{p_w\}),(\{p_w\},\{r_{bw}\},\{p_w\})\}\\ \hat{\Sigma}((\widetilde{R\|P}) \lhd P) &= \{\{p_{wa}\},\{p_{ab}\},\{p_{bw}\},\\ &\quad \{bwwa\},\{wabw\},\\ &\quad \{r_{wa}\},\{r_{ab}\},\{r_{bw}\}\}\end{aligned}$$

Figure 6.13 illustrates the extracted component $\widetilde{P}$. The three reflexive state dependent transitions on state $\{p_w\}$ require $\widetilde{P}$ to stay in this state if component $\widetilde{R}$ progresses by events $\{r_{wa}\}$, $\{r_{ab}\}$ or $\{r_{bw}\}$. In this application this synchronisation prevents component $\widetilde{P}$ from accessing the shared resource C_{RP} if component $\widetilde{R}$ is accessing the resource.

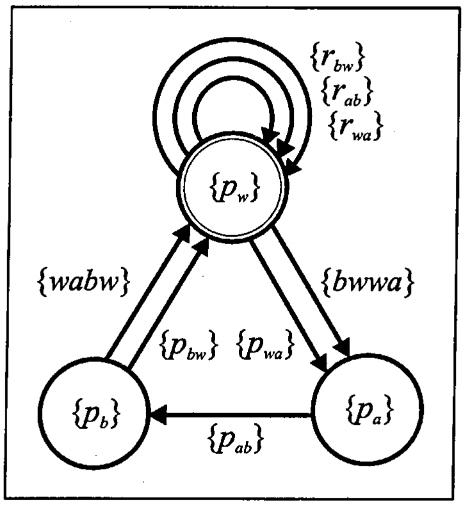


Figure 6.13: $\widetilde{P} = \widetilde{(R||P)} \lhd P$

The progressive synchronisation transition from state $\{p_w\}$ to state $\{p_a\}$ by event name $\{bwwa\}$ allows component $\widetilde{P}$ to start accessing the resource C_{RP}, if component $\widetilde{R}$ simultaneously stops accessing the resource by simultaneously progressing from state $\{r_b\}$ to state $\{r_w\}$. Similarly, the progressive synchronisation transition from state $\{p_b\}$ to state $\{p_w\}$ by event name $\{wabw\}$ allows component $\widetilde{R}$ to start accessing the resource C_{RP} by progressing from state $\{r_w\}$ to state $\{r_a\}$, if component $\widetilde{P}$ simultaneously stops accessing the resource.

6.3.5 Verification of $\widetilde{R}||\widetilde{P}$

Figure 6.14 illustrates the concurrent composition $\widetilde{R}||\widetilde{P}$ of the extracted components. In this example, all the event names of the components are shared and hence all the transitions are synchronous. Comparison with figure 6.11 (page 164) shows that the behaviour of the composite $\widetilde{R}||\widetilde{P}$ does not violate the intended behaviour of the shared resource C_{RP}.

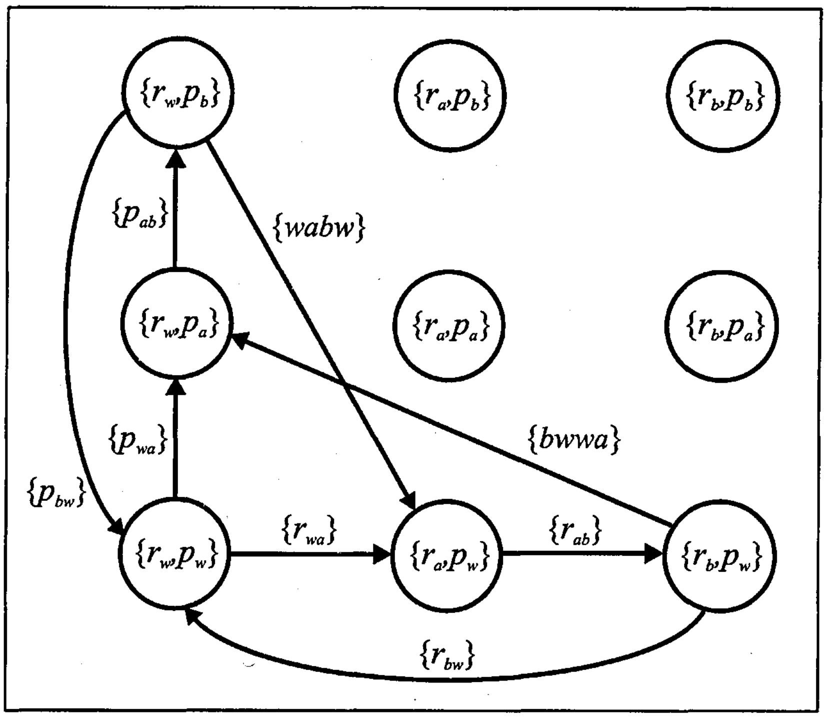


Figure 6.14: $\widetilde{R}\|\widetilde{P}$

6.4 Interaction between P and D due to resource C_{PD}

This section presents a detailed analysis of the expressions $\widetilde{P} = (\widetilde{P\|D}) \lhd P$ and $\widetilde{D} = (\widetilde{P\|D}) \lhd D$, where $\widetilde{P\|D}$ is $P\|D$ constrained by the resource C_{PD} as defined in figure 6.7 (page 159). The method follows the five steps given on page 149.

6.4.1 Specification of Components P and D

Processes P and D use the shared resource C_{PD}. Therefore the use of this resource must be described in the states and transitions in the CTS model of P (and D). This results in a model of P that is different to the model of P presented in section 6.3.1 (page 160).

The states of process P inferred from figure 6.4 (page 156) include waiting for p_{clk}, the input of r_a and r_b and checking if a p_{clk} has been missed. These operations do not use the shared resource C_{PD} and will be modelled by the single state $\{p_w\}$. The operation of data output will be modelled by a state $\{p_o\}$ and the act of generating the t_{clk} signal will be modelled by a synchronous transition. Due to the synchronisation, the output of the data will be modelled by a single state rather than the two states used in the model of P in section 6.3.1 (page 160). Process P (and process D) is illustrated in figure 6.15.

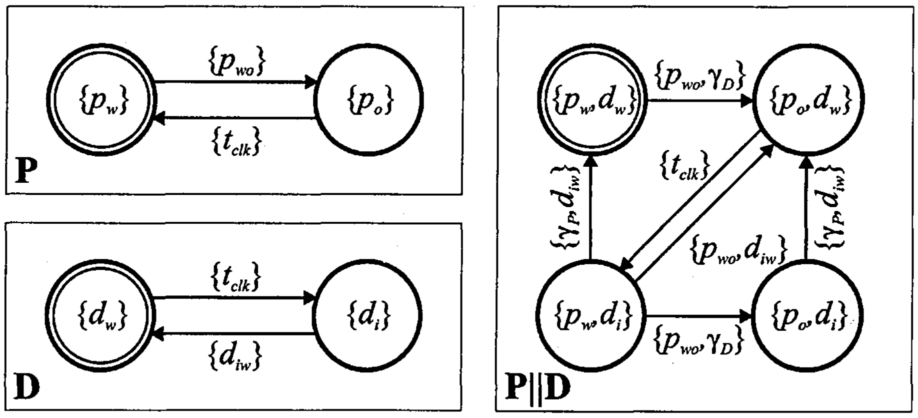


Figure 6.15: Processes P, D and $P||D$ resource C_{PD} use model

Process D uses the shared resource C_{PD} and this must be described in the states and transitions of the Composite Transition System model of D for the analysis of $P||D$.

The states of process D inferred from figure 6.5 (page 157) include waiting for t_{clk}, the calculation and output of data, and the check for a missed t_{clk}. These operations do not use the shared resource C_{PD} and will be modelled by the single state $\{d_w\}$. As a consequence of the synchronisation of the processes P and D, the input of r_a and r_b will also be modelled as a single state, $\{d_i\}$. Process D is illustrated in figure 6.15.

6.4.2 Composition of System $P||D$

The concurrent composition $P||D$, which is illustrated in figure 6.15, leads to the following Composite Transition System, where γ_P and γ_D are the idle transitions for P and D respectively. Observe that from state $\{p_o, d_w\}$ the system can only progress as a consequence of t_{clk} synchronisation.

$$\begin{aligned}
\hat{Q}(P\|D) &= \{\{p_w, d_w\}, \{p_o, d_w\}, \{p_w, d_i\}, \{p_o, d_i\}\} \\
\ddot{Q}(P\|D) &= \{\{p_w, d_w\}\}
\end{aligned}$$

$$\begin{aligned}
\hat{\Sigma}(P\|D) &= \{\{p_{wo}, \gamma_D\}, \{p_{wo}, d_{iw}\}, \{\gamma_P, d_{iw}\}, \{t_{clk}\}\} \\
\Gamma(P\|D) &= \{\gamma_P, \gamma_D\}
\end{aligned}$$

$$\begin{aligned}
\hat{\Delta}(P\|D) = \{&(\{p_w, d_w\}, \{p_{wo}, \gamma_D\}, \{p_o, d_w\}), (\{p_w, d_i\}, \{p_{wo}, \gamma_D\}, \{p_o, d_i\}), \\
&(\{p_w, d_i\}, \{\gamma_P, d_{iw}\}, \{p_w, d_w\}), (\{p_o, d_i\}, \{\gamma_P, d_{iw}\}, \{p_o, d_w\}), \\
&(\{p_w, d_i\}, \{p_{wo}, d_{iw}\}, \{p_o, d_w\}), (\{p_o, d_w\}, \{t_{clk}\}, \{p_w, d_i\})\}
\end{aligned}$$

6.4.3 Application of C_{PD} Resource Constraints to give $\widetilde{P\|D}$

Restriction of the behaviour of $P\|D$ by the shared resource C_{PD} requires the evaluation of the relationship between the resource transitions and the transitions of $\hat{\Delta}(P\|D)$. From figure 6.7 (page 159) the C_{PD} states can be mapped to the states of $P\|D$ as follows.

1. State $\{f\}$ indicates that C_{PD} is free, that is, neither process P nor D are accessing the resource, and either no data have been written or the written data have been read. Hence, process P must be in the state $\{p_w\}$ and process D must be in state $\{d_w\}$. Hence $\{f\}$ maps to $\{p_w, d_w\}$.

2. State $\{o\}$ indicates that C_{PD} is being accessed for the output of data. Hence process P must be in state p_o. Further, process D must not be accessing C_{PD}, hence process D must be in state $\{d_w\}$. Hence $\{o\}$ maps to $\{p_o, d_w\}$.

3. State $\{f'\}$ indicates that C_{PD} is not being accessed, but data has been written which has not been read. This state causes some difficulty. For process P, the state $\{p_w\}$ indicates that data has been written, assuming the state $\{p_o\}$ has been reached at least once. For process D, the state $\{d_w\}$ indicates that data has not yet been read. However, if the state $\{f'\}$ were mapped to the composite state $\{p_w, d_w\}$ then the requirement for a strict write–read sequence is not met as a write–write sequence is forced as a consequence of the transition from $\{f\}$ to $\{o\}$. In other words, there is no transition from $\{f\}$ to $\{i\}$. However, such a transition would incorrectly allow both

write–write and read–read sequences. In this example, the use of synchronisation between P and D and the C_{PD} transition from state $\{o\}$ to state $\{i\}$ leads to the decision to not map $\{f'\}$.

4. State $\{i\}$ indicates that C_{PD} is being accessed for the input of data. Hence process D must be in state $\{d_i\}$. Further, process P must not be accessing C_{PD}, hence process P must be in state $\{p_w\}$. Hence $\{i\}$ maps to $\{p_w, d_i\}$.

From the state mappings, the restricted form of $P\|D$ can be evaluated. The transitions of C_{PD}, as illustrated in figure 6.7 (page 159), are as follows. Figure 6.16 (page 174) illustrates this state mapping and the transitions of $\hat{\Delta}(C_{PD})$. For clarity, the transitions of $P\|D$ have been omitted.

$$\begin{aligned}\hat{\Delta}(C_{PD}) &= \{(\{f\},\{fo\},\{o\}),(\{o\},\{of'\},\{f'\}),(\{o\},\{oi\},\{i\}),\\ &\quad (\{f'\},\{f'i\},\{i\}),(\{i\},\{io\},\{o\}),(\{i\},\{if\},\{f\})\}\end{aligned}$$

By substitution of the mapped states of $\hat{Q}(C_{PD})$, the transition set $\hat{\Delta}_{C_{PD}}$ can be determined, hence;

$$\begin{aligned}\hat{\Delta}_{C_{PD}} &= \{(\{p_w,d_w\},\{fo\},\{p_o,d_w\}),(\{p_o,d_w\},\{oi\},\{p_w,d_i\}),\\ &\quad (\{p_w,d_i\},\{io\},\{p_o,d_w\}),(\{p_w,d_i\},\{if\},\{p_w,d_w\})\}\end{aligned}$$

The set $\hat{\Delta}_{C_{PD}}$ defines the transitions between the states of $\hat{Q}(P\|D)$ that are permitted by the states and transitions of the shared resource C_{PD}. Specifically, the *from* state set of $\hat{\Delta}_{C_{PD}}$ defines the permitted *from* states of $\widetilde{P\|D}$. Likewise, the *to* state set of $\hat{\Delta}_{C_{PD}}$ defines the permitted *to* states of $\widetilde{P\|D}$. Hence the transition set $\hat{\Delta}(\widetilde{P\|D})$ evaluates as follows, and $\widetilde{P\|D}$ is illustrated in figure 6.17 (page 174).

$$\begin{aligned}\hat{\Delta}(\widetilde{P\|D}) &= \{(Q,\Sigma,P) | (Q,\Sigma,P) \in \hat{\Delta}(P\|D) \wedge \exists\tau \bullet (Q,\{\tau\},P) \in \hat{\Delta}_{C_{PD}}\}\\ &= \{(\{p_w,d_w\},\{p_{wo},\gamma_D\},\{p_o,d_w\}),(\{p_w,d_i\},\{\gamma_P,d_{iw}\},\{p_w,d_w\}),\\ &\quad (\{p_w,d_i\},\{p_{wo},d_{iw}\},\{p_o,d_w\}),(\{p_o,d_w\},\{t_{clk}\},\{p_w,d_i\})\}\end{aligned}$$

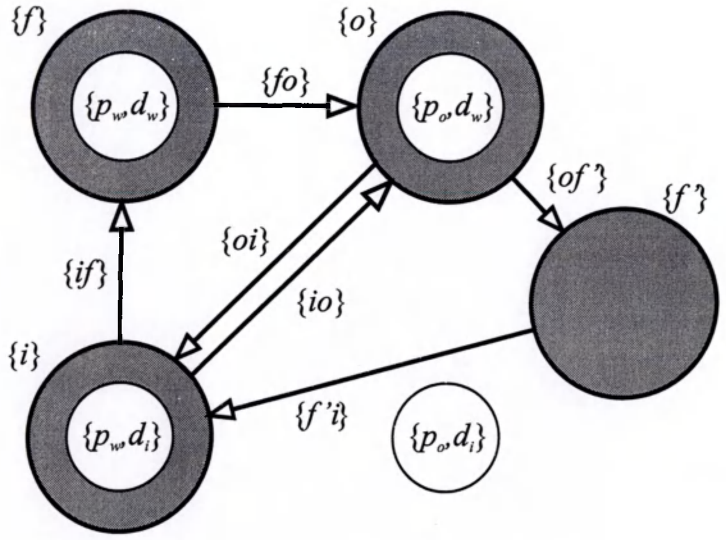


Figure 6.16: C_{PD} permitted states and transitions of $P||D$

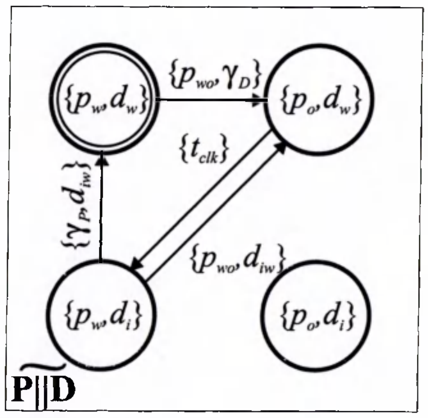


Figure 6.17: $\widetilde{P||D}$

Finally, observe that $\widetilde{P||D}$ can be defined as follows, noting that only the transition set has been restricted;

$$\widetilde{P||D} \quad = \quad (\hat{Q}(P||D), \ddot{Q}(P||D), \hat{\Sigma}(P||D), \Gamma(P||D), \hat{\Delta}(\widetilde{P||D}))$$

6.4.4 Extraction to derive $\widetilde{P}$ and $\widetilde{D}$

Evaluation of $\widetilde{P} = \widetilde{P \| D} \lhd P$

In the extraction $\widetilde{P} = \widetilde{P \| D} \lhd P$ the state set $\hat{Q}(\widetilde{P})$, the initial state set $\ddot{Q}(\widetilde{P})$, and the idle event name $\Gamma(\widetilde{P})$ evaluate as follows.

$$\begin{aligned}
\hat{Q}(\widetilde{P}) &= \{\{p_w\}, \{p_o\}\} \\
\ddot{Q}(\widetilde{P}) &= \{\{p_w\}\} \\
\Gamma(\widetilde{P}) &= \{\gamma_{\widetilde{P}}\}
\end{aligned}$$

From Definition 5.3 (page 107), the event name set will include terms from the transition set. Therefore the transition set must be evaluated first.

1. $\hat{\Delta}((\widetilde{P \| D}) \lhd P)_A$: The asynchronous transition $(\{p_w\}, \{p_{wo}\}, \{p_o\})$ of $\hat{\Delta}(P)$ exists in the transitions of $\hat{\Delta}(\widetilde{P \| D})$, hence;

$$\hat{\Delta}((\widetilde{P \| D}) \lhd P)_A = \{(\{p_w\}, \{p_{wo}\}, \{p_o\})\}$$

2. $\hat{\Delta}((\widetilde{P \| D}) \lhd P)_S$: The synchronous $(\{p_o\}, \{t_{clk}\}, \{p_w\})$ of $\hat{\Delta}(P)$ exists in the transition $(\{p_o, d_w\}, \{t_{clk}\}, \{p_w, d_i\})$ of $\hat{\Delta}(\widetilde{P \| D})$, hence;

$$\hat{\Delta}((\widetilde{P \| D}) \lhd P)_S = \{(\{p_o\}, \{t_{clk}\}, \{p_w\})\}$$

3. $\hat{\Delta}((\widetilde{P \| D}) \lhd P)_P$: Progressive synchronisation arises because some of the expected "horizontal" and "vertical" transitions related to the $(\{p_w, d_i\}, \{p_{wo}, d_{iw}\}, \{p_o, d_w\})$ "diagonal" transition, formed from $(\{p_w\}, \{p_{wo}\}, \{p_o\})$, do not exist in $\hat{\Delta}(\widetilde{P \| D})$. The required progressive synchronisation transition will use the new synchronising event name $\{woiw\} = \mathcal{N}(\{p_{wo}\} \cup \{d_{iw}\})$. Hence;

$$\hat{\Delta}((\widetilde{P \| D}) \lhd P)_P = \{(\{p_w\}, \{woiw\}, \{p_o\})\}$$

4. $\hat{\Delta}((\widetilde{P\|D}) \lhd P)_D$: State dependent synchronisation because some of the expected asynchronous transitions are absent from $\hat{\Delta}(\widetilde{P\|D})$. Specifically, the "vertical" transition from state $\{p_w, d_i\}$ to state $\{p_w, d_w\}$ by the event name $\{\gamma_P, d_{iw}\}$ exists, but the related transition from state $\{p_o, d_i\}$ to state $\{p_o, d_w\}$ by the event name $\{\gamma_P, d_{iw}\}$ is absent. Hence the state dependent synchronisation transition $(\{p_w\}, \{d_{iw}\}, \{p_w\})$ is required.

$$\hat{\Delta}((\widetilde{P\|D}) \lhd P)_D = \{(\{p_w\}, \{d_{iw}\}, \{p_w\})\}$$

State dependent synchronisation also arises from absent *simultaneous* transitions (Definition 5.9, page 123). This cause of state dependent synchronisation does not occur in this extraction.

Finally, the complete extraction transition set can be written as follows, and the event name set evaluated;

$$\begin{aligned}\hat{\Delta}((\widetilde{P\|D}) \lhd P) &= \{(\{p_w\}, \{p_{wo}\}, \{p_o\}), (\{p_o\}, \{t_{clk}\}, \{p_w\}),\\ &\quad (\{p_w\}, \{woiw\}, \{p_o\}), (\{p_w\}, \{d_{iw}\}, \{p_w\})\}\\ \hat{\Sigma}((\widetilde{P\|D}) \lhd P) &= \{\{p_{wo}\}, \{t_{clk}\}, \{woiw\}, \{d_{iw}\}\}\end{aligned}$$

Figure 6.18 illustrates the extracted component $\widetilde{P}$. The reflexive state dependent transition on state $\{p_w\}$ requires $\widetilde{P}$ to stay in this state if component $\widetilde{D}$ progresses by the event $\{d_{iw}\}$. In this application, this synchronisation prevents component $\widetilde{P}$ from accessing the shared resource C_{PD} if component $\widetilde{D}$ is not accessing the resource following the previous t_{clk} synchronisation. In other words, synchronisation on t_{clk} is not sufficient to ensure that the shared resource is accessed correctly. This situation is often called a *race hazard.*

The progressive synchronisation transition from state $\{p_w\}$ to state $\{p_o\}$ by event name $\{woiw\}$ allows component $\widetilde{P}$ to start accessing the resource C_{PD}, if component $\widetilde{D}$

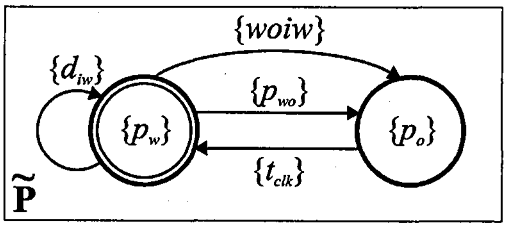


Figure 6.18: $\widetilde{P} = (\widetilde{P\|D}) \lhd P$

simultaneously stops accessing the resource by simultaneously progressing from state $\{d_i\}$ to state $\{d_w\}$. Elimination of the transition from $\{i\}$ to $\{o\}$ by $\{io\}$ in the behaviour of the resource C_{RP} would remove this synchronising transition.

Evaluation of $\widetilde{D} = \widetilde{P\|D} \lhd D$

In this application, component D has the same structure as component P and because of the symmetry in the restricted machine $\widetilde{P\|D}$, the extraction $\widetilde{D} = \widetilde{P\|D} \lhd D$ follows in a similar way to the extraction of $\widetilde{P}$.

The state set, the initial state set, and the idle event name evaluate as follows.

$$\begin{aligned}\hat{Q}(\widetilde{D}) &= \{\{d_w\}, \{d_i\}\}\\ \ddot{Q}(\widetilde{D}) &= \{\{d_w\}\}\\ \Gamma(\widetilde{D}) &= \{\gamma_{\widetilde{D}}\}\end{aligned}$$

The transition set also follows in a similar way, but with following notable differences. First, because some of the transitions related to the $(\{p_w, d_i\}, \{p_{wo}, d_{iw}\}, \{p_o, d_w\})$ "diagonal" transition do not exist in $\hat{\Delta}(\widetilde{P\|D})$, progressive synchronisation is required. The new event name is $\{woiw\} = \mathcal{N}(\{p_{wo}\} \cup \{d_{iw}\})$. Hence;

$$\hat{\Delta}((\widetilde{P\|D}) \lhd D)_P = \{(\{d_i\}, \{woiw\}, \{d_w\})\}$$

Second, state dependent synchronisation arises because the expected "horizontal" asynchronous transition $(\{p_w, d_i\}, \{p_{wo}, \gamma_D\}, \{p_o, d_i\})$ is absent. Hence;

$$\hat{\Delta}((\widetilde{P\|D}) \lhd D)_D = \{(\{d_w\}, \{p_{wo}\}, \{d_w\})\}$$

Finally, the complete extraction transition set can be written as follows, and the event name set evaluated;

$$\begin{aligned}
\hat{\Delta}((\widetilde{P\|D}) \lhd D) &= \{(\{d_i\},\{d_{iw}\},\{d_w\}),(\{d_w\},\{t_{clk}\},\{d_i\}),\\
&\quad (\{d_i\},\{woiw\},\{d_w\}),(\{d_w\},\{p_{wo}\},\{d_w\})\}\\
\hat{\Sigma}((\widetilde{P\|D}) \lhd D) &= \{\{d_{iw}\},\{t_{clk}\},\{woiw\},\{p_{wo}\}\}
\end{aligned}$$

Figure 6.19 illustrates the extracted component $\widetilde{D}$. The reflexive state dependent transition on state $\{d_w\}$ introduces synchronisation that enables component $\widetilde{P}$ to commence accessing the shared resource C_{PD} if component $\widetilde{D}$ is not accessing the resource.

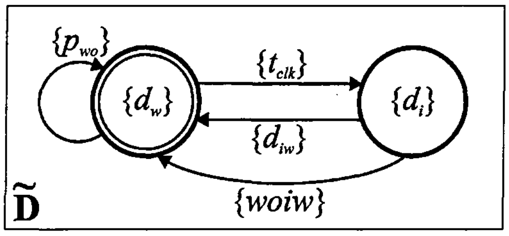


Figure 6.19: $\widetilde{D} = (\widetilde{P\|D}) \lhd D$

The progressive synchronisation transition from state $\{d_i\}$ to state $\{d_w\}$ by event name $\{woiw\}$ allows component $\widetilde{P}$ to start accessing the resource C_{PD}, if component $\widetilde{D}$ simultaneously stops accessing the resource by simultaneously progressing from state $\{d_i\}$ to state $\{d_w\}$.

6.4.5 Verification of $\widetilde{P}\|\widetilde{D}$

Figure 6.20 illustrates the concurrent composition $\widetilde{P}\|\widetilde{D}$ of the extracted components. In this example, all the event names of the components are shared and hence all the transitions are synchronous. Comparison with figure 6.17 (page 174) shows that the behaviour of the composite $\widetilde{P}\|\widetilde{D}$ does not violate the intended behaviour of the shared resource C_{PD}.

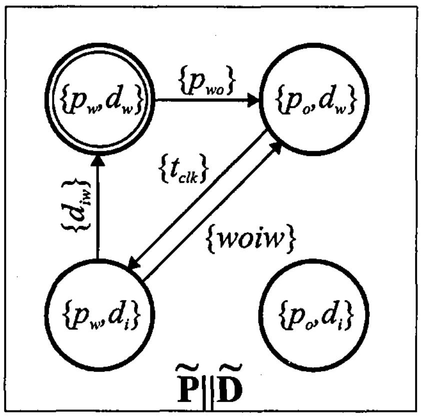


Figure 6.20: $\widetilde{P}||\widetilde{D}$

Chapter 7

Summary, Discussion and Conclusion

Concurrency is a common design choice for real-time systems but co-ordinated use of the system-level resources shared between the concurrent components is essential. Failure to co-ordinate correctly may lead to violation of the required use of a resource — resource contention — and a system that does not function as the designer intended. Consequently, an implementation may not always meet its specification and the indeterminacy of concurrent systems can make the cause of the problem difficult to determine [63].

Many notations for specifying concurrent systems exist. Typically, these notations have a rich syntax designed for specifying the behaviour of sequential components. Concurrent composition of the components though is often limited to some assertion that the components execute concurrently and interact through the interfaces exposed at the component level. The result is a concurrent system specified by the behaviour of its components that is inadequate for dealing with system-level resource contention and this renders many of the existing notations inappropriate.

The Composite Transition System notation developed in this thesis provides ways to derive components that respect the system-level resource constraints abstracted away during the design process.

7.1 Summary of the CTS Notation

Descriptions of the behaviour of concurrent systems require constructs to describe concurrency, asynchrony, simultaneity and synchronicity; constructs that are absent in Labelled Transition Systems. Despite this, the development of a notation with similarities to Labelled Transition Systems seemed a natural choice because they are well understood, especially by engineers developing embedded systems, and are embodied in standards such as the Unified Modelling Language [2, 24], LOTOS [41] and Z [86]. The Composite Transition System diagrams in this thesis are recognisable as state transition diagrams and section 3.3 (page 65) showed that translation from an LTS into a CTS is straightforward.

The Composite Transition System notation achieves the required refinement of Labelled Transition Systems by labelling each state and event name with a set of identifiers (rather than an unstructured identifier) and by the explicit definition of the semantics of internal idle events. Each system state describes the contemporaneous state of existence of the system components. Similarly, each system event name describes the component events that must occur simultaneously for the system to progress. Synchronicity is a form of simultaneity which is distinguished through a convention on component event names, and asynchrony is transformed into a form of simultaneity by the idle events. This specific use of idle events is a significant difference to not only Labelled Transition Systems, but also notations which, for example, use internal events to represent, perhaps, a minimal delay, or the internalisation of some interaction.

Operators that act on Composite Transition Systems have been defined to enable the modelling resource contention. The operational steps are component specification, concurrent composition, restriction and extraction.

The concurrent composition operator takes CTS descriptions of the system components and generates a CTS that describes the behaviour of the concurrent composition of the components. Conversely, no operator has been defined to restrict the behaviour of a concurrent system as a consequence of the application of system-level resource constraints. Analysis of each specific application is required and, therefore, the algebra of a restriction operator is not likely to be generic.

The method for applying resource constraints adopted in the applied example of Chapter 6 was to map the permitted states of a shared resource to the states of the composite system to derive a transition "mask", for an example see $\hat{\Delta}_R$ in term 6.1 (page 150). Other forms of restriction may be required, for example, when two processes share a single processor and cannot simultaneously progress, then simultaneous transitions must be prohibited. Or, perhaps, there are prohibitions on specific transitions between states.

A system model incorporating resource constraints is not sufficient to ensure that an implementation will meet the system specification. It is also necessary to determine the behaviour of the required components and their interaction which is only revealed at the system-level. The extraction operator can be used to generate the CTS descriptions of the required components. Verification that the restricted system and the extracted components meet the system requirements must be performed by the system designer. Extraction is a significant contribution of the CTS notation because it is crucial to the verification that an implementation will meet its specification.

The CTS notation was applied in Chapter 6 to an example system as a demonstration of a method of modelling resource contention. The example illustrated that the use of shared resources can be modelled with few states and transitions, preventing an unmanageable state-space and transition-space. A system model was formed by concurrent composition of the components. A "mask" defined the system states and changes in system state permitted by a system-level resource and the extraction operator was applied to determine the required modified behaviour of the components. Verification was performed by comparing the restricted system and the concurrent composition of the extracted components.

Manual calculation of concurrent composition is relatively straightforward but can become error prone as the number of states and transitions increases. Manual calculation of extraction is very difficult, even for simple systems. Without software tools the complexity of the computations would be an impediment to the adoption of the notation by system designers. Therefore, the operators have been comprehensively defined and, although it was not the aim of this work to design and build a software package, some use of the Mathematica [98] tool was sufficient to demonstrate the feasibility of generating software tools.

7.1.1 Operators

The CTS notation can describe asynchronous, simultaneous and synchronous progress. However, making the distinction introduces significant complexity into the algebra of the concurrent composition and extraction operators. Moreover, this algebraic complexity limits the mathematical properties of these operators. The mathematical properties are explored in Appendix B (page 196) and summarised in table B.2 (page 219).

Merge Composition

Merge composition is fundamental to the "superposition" of machines (Chapter 3) and the algebra of the operator proves to be quite straightforward, largely because it is unnecessary to distinguish synchronous progress from simultaneous or asynchronous progress.

Merge composition necessitates a set of initial states. Each initial state is a decoration identifying a *possible* starting point. The notion of an *actual* initial state is useful in considering state reachability, that is, the ability of a machine to reach certain states and to execute transitions from that state; a consequence of this is discussed in section 7.2 (page 186).

Concurrent Composition

Concurrent composition, in contrast to merge composition, generates both synchronous and asynchronous forms of progress and this introduces algebraic complexity to the operator. The complexity, which arises from determining if an event name is synchronous, means that the operator is only *associative* under certain conditions, while the property of *distribution over merge* is denied. These limitations restrict the expressions that can be written over Composite Transition Systems.

If, instead, only asynchronous progress were to be described, then the tests for synchronous event names would be unnecessary and all combinations of component event names and transitions would be formed. The concurrent composition operator would then be associative and would distribute over merge. But the absence of any form of description of synchronous progress would render the notation useless because synchronisation is a common technique for co-ordinating concurrent processes. Indeed, this is why real-time operating systems have synchronisation primitives such as semaphores and mutexes. Further, the absence of a description of synchronisation would prohibit the definition of the extraction operator, without which, the objective of modelling resource contention could not be met.

Similarly, if only synchronous progress were to be described, then term 4.4 (page 78) is sufficient to define concurrent composition. But only those event names synchronous to every component of a composition and which label at least one transition in every component would yield a transition in the composite system. Such strong synchronisation would significantly restrict the applicability of the notation.

Despite the complexity and limitations to the mathematical properties, the concurrent operator generates a concurrent system which incorporates state concurrency and asynchronous, simultaneous and synchronous progress.

Extraction

Development of the extraction operator proved to be difficult in two respects. The first difficulty was to determine the synchronising transitions required to suppress the concurrent composition of transitions absent from a restricted system. Various techniques based upon additional "intermediate" states were explored. However these still required synchronisation and altered the structure of a restricted system. Moreover, the concurrent composition of the extracts and the restricted system would differ significantly unless additional complexity was added to the algebra of concurrent composition.

The second development difficulty was to determine the conditions for the introduction of the synchronising transitions. Much of this difficulty arises because an expression of the form $\widetilde{C_{||}} \lhd C'$ specifies neither the components of $C_{||}$ nor the transitions of $C_{||}$ that are absent in $\widetilde{C_{||}}$. If $C_{||}$ is the composition of C' with some (unknown) component, say X, then the extraction operator attempts to determine X such that $C'||X$, that is $C_{||}$, can be formed and compared with $\widetilde{C_{||}}$.

Now, simplifications could be made if the extraction operation was defined as a function, perhaps of the form $\lhd(C', C'', \widetilde{C'||C''}, C''')$. Such a function reads as the extraction of C''' (which may or may not be the same as C' or C'') from $\widetilde{C'||C''}$, where the operands of the unmodified concurrent composition $C'||C''$ are explicitly given. Such an approach was not adopted because a design objective was to be able to determine an extract where the components of the composite system may not be known. Additionally, binary operators lead more naturally to algebraic expressions, for example $((\widetilde{C_0||C_1} + C_3) \lhd C_1)||(C_4 + C_5)$, and hence the notation could form a machine algebra.

The extraction operator correctly generates an extract from a system which comprises two concurrent components. Interpretational difficulties arise when a composite system comprises three or more components and this is discussed in section 5.5 (page 146).

7.2 Discussion

Structure and Execution

Translation of asynchrony into a form of simultaneity reveals that a CTS is a model of a sequential machine. Therefore, progress (execution) of the system can be non-deterministic if two or more system events occur simultaneously (this should not be confused with the simultaneous occurrence of any two or more component events).

Section 3.1.1 (page 45) revealed that a composite event name that incorporates a component idle event name leads to apparent simultaneity of system events. This apparent simultaneity, an artifact of the notation, was suppressed with a precedence rule (page 46), hence the notation does not introduce non-deterministic execution. More significantly, the need for the precedence rule identifies a distinction between "executional" and "structural" determinism. The dis-ambiguation property [20] only ensures that the structure of a system is deterministic.

This distinction between structure and execution is often overlooked and this is a consequence of the misuse of the nomenclature of events. In the Composite Transition System notation, the term *event name* identifies the name of an event to which a system will react. These event names label transitions which describe a change of state if the named event occurs. Within the structure of a CTS (or indeed any Labelled Transition System) an event name may label more than one transition. The labelling of a transition in a CTS with an event name makes no assertion about how many times, if any, the named event will occur. However, each time an event occurs then the event name can be appended to the *trace* of events (page 65) that have occurred. Thus, the labelling of a transition with an event name relates to the "structure" of a CTS, but the occurrence of an event relates to its "execution".

This distinction between structure and execution is important because the identification of resource contention involves the derivation of the required structure of a component rather than the proof of any specific properties of the execution of a component. The CTS notation and its operators have been defined to provide structural analysis rather than the executional analysis that is often the design objective of existing notations.

One further observation arising from this distinction is that while the execution of a process may be expected to be infinite, the process specification – its structure – is almost certainly finite. In many of the notations that are mathematically motivated, this distinction can be significant; Aczel in [1], however, also noted the distinction and uses the *Anti-Foundation Axiom* with non-well-founded sets as a basis for describing the finite structure of a Labelled Transition System but its infinite execution.

Unreachable States

Removal of unreachable states simplifies a component and any concurrent system formed from a component. Unreachable states are most obvious in the structure of a CTS. With the exception of an initial state, any state that is not the *to* state of any transition cannot by be reached. The ability of a CTS to reach a state by execution is less obvious and depends upon the *actual* initial state and the analysis of all possible execution traces.

The removal of any transitions from unreachable states in a composite system, might lead the extraction operator to introduce synchronisation that might not otherwise be required. Recall that synchronisation is introduced by the absence of certain transitions and not whether they are reachable. In other words, the extract operator acts only on the structure of the component. Hence, retaining any unreachable states and transitions that will not or cannot be executed might result in fewer synchronisation transitions. But arguably the removal of transitions creating the need for additional synchronisation makes for a more robust system.

Abstraction

System complexity leads designers to partition systems into idealised entities each of a manageable complexity. Abstraction and idealisation introduced in this partitioning has the effect of hiding aspects of the system complexity deemed irrelevant to the specific analysis being performed. In a CTS the abstraction is manifest in the states, the events to which the system may react, and the state changes defined by the transitions.

A state represents "something" — perhaps some resource consuming activity, reaction to events or even inactivity — but that "something" is not revealed. Consequently, a state is commonly understood to represent some form of indivisibility. In a Composite Transition System every state of a component is likely to be subsumed into a number of composite states. For example, the component state $\{e\}$ in figure 3.2 (page 45) is subsumed in the composite states $\{a, e\}$ and $\{b, e\}$. Consequently, a composite state change does not necessarily mean a change in component state. Therefore, any divisibility apparent at the system level does not necessarily mean divisibility at the component level. In other words, any familiarity with Labelled Transition Systems may lead to misinterpretation of a Composite Transition System.

A system state change in which a component does not change state is a consequence of component idle event names and the implied reflexive idle transitions. Component idle transitions enable other components in a composition to progress, specifically, the other components progress by their defined transitions. More generally, the abstraction represented by a state is assumed to be completely independent of any implicit idle transitions and any explicit transitions of other components. Consequently, internalising a common event through abstraction in a component state may make the event asynchronous according to the conventions of concurrent composition. Thus different abstractions of the behaviour of a component may significantly alter the structure and execution of a composite system formed from that component.

Computational Model

A commonly held interpretation of Labelled Transition Systems is that a transition between states is instantaneous, or at least indivisible, and that a state represents some form of activity that consumes processor resource (which, given finite processing performance, cannot be instantaneous). Interpretations of this form raise the question about the underlying computational model. For systems that exhibit *true* concurrency, the *maximum parallelism* computational model is often assumed and this implies one processor resource for each process. The CTS concurrent composition operator is a *true* concurrency operator.

However, each state is an abstraction for something that is not defined by the constructs of the Composite Transition System notation (or indeed any form of state machine notation). A composite state defines the contemporaneous state of existence of the components of the composite system, but does not necessarily describe executional concurrency. Consequently, the required processing resource depends upon the specific interpretation of each component state and this interpretation cannot be gleaned from the constructs included in the CTS notation.

Since it is the interpretation of each state in a Composite Transition System model that defines the required processing resource, the notion of an underlying computational model is actually irrelevant. The objective of the requirement on page 42 that the notation should not assume any specific computational model really requires that the notation itself should not impose any constraints on the behaviour of a composite system. Thus an implementation may range from one processor per process through to a single processor for all the processes. A processor is a shared resource for which the processes compete, therefore, the resulting resource contention should be explicitly modelled. Note that the definition of an *interleaved* concurrency operator which generates a composite system that does not include simultaneous progress, would be feasible.

Anti-events

The method adopted in Chapter 6 (page 148) to derive a restricted system was to apply a "mask" to remove certain transitions, and this can be thought of as the "inverse" of merge composition. Extraction from the restricted system then introduces synchronisation transitions. Observe that restriction removes transitions from the structure of a system, yet the extraction operator, in general, adds transitions to the structure of an extract that is a required component of that restricted system.

An alternative approach is to form a transition that removes transitions under merge and concurrent composition. Consider the concept of *anti-event names* and *anti-transitions.* Any event name Σ could have a counterpart anti-event name, perhaps denoted $\overline{\Sigma}$. The occurrence of an event Σ would not cause any transition labelled with the anti-event name $\overline{\Sigma}$ to progress. In effect, a transition labelled with an anti-event name is the same as an absent transition. Thus, an anti-transition is a transition that is a constituent part of the structure of a component but not of the execution of a component.

Restricting the behaviour of a system $C_{||}$ to form the system $\widetilde{C_{||}}$ would require the conversion of the event name label on those prohibited transitions to the corresponding anti-event name label. Thus the "absent" transitions in the system $\widetilde{C_{||}}$ would be exposed and this might lead to a simpler extraction operator and alternative techniques for the introduction of synchronisation. However, much of the complication of extraction comes from determining the other component of a composition and determining which transitions exist and which do not. If anti-events were used then a similar algorithm would likely be required but the determination would be based on the presence of transitions labelled with the related anti-event name rather than the absence of related transitions.

Algebraic Structures

The concept of anti-events can be extended to the other terms in the definition of a Composite Transition System. Since merge composition is commutative and associative (Appendix B.1.1 and B.1.2), then the existence of a unique merge composition identity CTS denoted U^+, such that $C + U^+ = U^+ + C = C$, and the existence of an inverse $\overline{C}$ of a CTS C, such that $C + \overline{C} = \overline{C} + C = U^+$, would give the merge operator the property of a *group* [6]. Indeed, a merge identity and the inverse of a CTS can be defined using the anonymous states, events, and so on, introduced in Chapter 3. However, the definition of each term of the merge composition $C' + C''$ becomes more complex, for example;

$$\begin{aligned}\hat{\Sigma}(C' + C'') \overset{def}{=} \; & \{\Sigma | (\Sigma \in \hat{\Sigma}(C') \wedge \overline{\Sigma} \notin \hat{\Sigma}(C'')) \vee (\Sigma \in \hat{\Sigma}(C'') \wedge \overline{\Sigma} \notin \hat{\Sigma}(C'))\} \cup \\ & \{\overline{\Sigma} | (\Sigma \in \hat{\Sigma}(C') \wedge \overline{\Sigma} \notin \hat{\Sigma}(C'')) \vee (\Sigma \in \hat{\Sigma}(C'') \wedge \overline{\Sigma} \notin \hat{\Sigma}(C'))\}\end{aligned}$$

A similar approach can be taken with the concurrent composition operator, but not without significant complexity.

The advantage of relating the operators of the Composite Transition System notation to well-known algebraic structures such as a *group* or a *ring* is that well-established theorems can then be applied to determine different properties of the systems. Now, to have the property of a *ring*, an "addition" operator (merge) and a "multiplication" operator (concurrent composition) are required. But, both operators need to have the properties of a *group* and concurrent composition must distribute over merge. Appendix B.2.3 (page 209) shows that the law of distribution only holds for specific conditions. Therefore, the advantages of algebraic structures cannot be exploited without changes to the algebra of the operators.

7.3 Conclusion

The concept of resource contention, its impact on the interaction of independently specified processes, and the importance of modelling resource contention were examined in detail in this thesis. Some existing notations were reviewed, but their applicability to process specification made them less than appropriate for the purpose of modelling resource contention. This motivated the development of the Composite Transition System notation which enables a system designer to describe the behaviour of sequential components and calculate the behaviour of their concurrent composition. The concurrent system can then be constrained by the application of system-level resources and the specifications of the required components can be computed.

However, five significant areas for further work have been identified.

The first area concerns the interpretation of a Composite Transition System and its translation into an implementation. Interpretation can be difficult for all but the simplest of systems and the synchronisation introduced by the extraction operator exacerbates the problem. Understanding how to interpret a CTS requires further investigation and the application of the notation to further case studies.

The second area concerns the minimisation of synchronisation. One possible technique that should be investigated is to extend the convention on common event names to include specific states, thus tightening the criteria for determining synchronisation during concurrent composition. However, such a change would likely require significant re-definition of the notation and the algebra of the operators.

The third area concerns the impact to an extract as a consequence of removing transitions that will not or cannot execute in a restricted system. Detecting such transitions is the subject of reachability analysis, however, techniques that are more sophisticated than a simple "mask" that determines the structure of the extract without regard for the execution of the system could lead to useful simplification.

The fourth area concerns the development of "remainder" operator. For any expression $C_{||} \lhd C'$, the algebra of the extraction operator determines with what C' would have been combined to form $C_{||}$, in other words, extraction determines the "other" component of $C_{||}$. This suggest that a "remainder" operator should be defined to determine the "other" component. This is of use in an expression of the form $(C_A||C_B) \lhd C_C$, which determines $\widetilde{C_C}$. The "remainder" operator would generate $\widetilde{C_D}$, such that $\widetilde{C_D}||\widetilde{C_D} = C_A||C_B$. The application of this proposed operator is the study of alternative components of a system.

The final area also concerns the extraction of components not used in the formation of a restricted system. The motivation is to reason about alternative sets of component processes. In other words, there may be other component specifications that would meet the system requirements but with less resource contention and, therefore, less interaction between the components. One approach might be to evaluate partitioning algorithms that arise in the domain of graph theory. Similarly, theorems based on algebraic structures could be investigated if solutions can be found to the limitations of the mathematical properties of the operators.

Appendix A

Definitions

A.1 Merge Composition

$$\begin{array}{rcl}
\hat{Q}(C'+C'') & \overset{def}{=} & \hat{Q}(C') \cup \hat{Q}(C'') \\
\ddot{Q}(C'+C'') & \overset{def}{=} & \ddot{Q}(C') \cup \ddot{Q}(C'') \\
\hat{\Sigma}(C'+C'') & \overset{def}{=} & \hat{\Sigma}(C') \cup \hat{\Sigma}(C'') \\
\Gamma(C'+C'') & \overset{def}{=} & \mathcal{N}(\Gamma(C') \cup \Gamma(C'')) \\
\hat{\Delta}(C'+C'') & \overset{def}{=} & \hat{\Delta}(C') \cup \hat{\Delta}(C'')
\end{array}$$

A.2 Concurrent Composition

$$\begin{array}{rcl}
\hat{Q}(C'\|C'') & \overset{def}{=} & \{Q' \cup Q'' | Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'')\} \\
\ddot{Q}(C'\|C'') & \overset{def}{=} & \{\dot{Q}' \cup \dot{Q}'' | \dot{Q}' \in \ddot{Q}(C') \wedge \dot{Q}'' \in \ddot{Q}(C'')\} \\
\hat{\Sigma}(C'\|C'') & \overset{def}{=} & \{\Sigma' \cup \Gamma(C'') | \Sigma' \in \hat{\Sigma}(C') \ \wedge\ \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}\ \} \bigcup \\
 & & \{\Gamma(C') \cup \Sigma'' | \Sigma'' \in \hat{\Sigma}(C'') \ \wedge\ \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}\ \} \bigcup \\
 & & \{\Sigma' \cup \Sigma'' | \Sigma' \in \hat{\Sigma}(C') \ \wedge\ \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \wedge \\
 & & \qquad\qquad \Sigma'' \in \hat{\Sigma}(C'') \ \wedge\ \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}\ \} \bigcup \\
 & & \{\Sigma' \cup \Sigma'' | \Sigma' \in \hat{\Sigma}(C') \ \wedge\ \Sigma'' \in \hat{\Sigma}(C'') \ \wedge\ \Sigma' \cap \Sigma'' \neq \{\}\ \} \\
\Gamma(C'\|C'') & \overset{def}{=} & \Gamma(C') \cup \Gamma(C'') \\
\hat{\Delta}(C'\|C'') & \overset{def}{=} & \{(Q' \cup Q'', \Sigma' \cup \Gamma(C''), P' \cup Q'') | (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
 & & \qquad Q'' \in \hat{\Sigma}(C'') \wedge \Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C'\|C'')\} \bigcup \\
 & & \{(Q' \cup Q'', \Gamma(C') \cup \Sigma'', Q' \cup P'') | Q' \in \hat{\Sigma}(C'') \wedge \\
 & & \qquad (Q'', \Sigma'', P') \in \hat{\Delta}(C'') \wedge \Gamma(C') \cup \Sigma'' \in \hat{\Sigma}(C'\|C'')\ \} \bigcup \\
 & & \{(Q' \cup Q'', \Sigma' \cup \Sigma'', P' \cup P'') | (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
 & & \qquad (Q'', \Sigma'', P'') \in \hat{\Delta}(C'') \wedge \Sigma' \cup \Sigma'' \in \hat{\Sigma}(C'\|C'')\}
\end{array}$$

A.3 Extraction Operator

$$
\begin{aligned}
\hat{Q}(\widetilde{C_{||}} \lhd C') &\overset{def}{=} \{ Q' \mid \exists Q_{||} \bullet (Q_{||} \in \hat{Q}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge Q_{||} \cap Q' \neq \{\})\} \\
\ddot{Q}(\widetilde{C_{||}} \lhd C') &\overset{def}{=} \{ \dot{Q}' \mid \exists \dot{Q}_{||} \bullet (\dot{Q}_{||} \in \ddot{Q}(\widetilde{C_{||}}) \wedge \dot{Q}' \in \ddot{Q}(C') \wedge \dot{Q}_{||} \cap \dot{Q}' \neq \{\})\} \\
\hat{\Sigma}(\widetilde{C_{||}} \lhd C') &\overset{def}{=} \{ \Sigma' \mid \exists \Sigma_{||} \bullet (\Sigma_{||} \in \hat{\Sigma}(\widetilde{C_{||}}) \wedge \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma_{||} \cap \Sigma' \neq \{\})\} \bigcup \\
&\quad\ \ \{ \Sigma \mid \exists Q, P \bullet (Q, \Sigma, P) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C') \} \\
\Gamma(\widetilde{C_{||}} \lhd C') &\overset{def}{=} \mathcal{N}(\Gamma(C')) \\
\hat{\Delta}(\widetilde{C_{||}} \lhd C') &\overset{def}{=} \hat{\Delta}(\widetilde{C_{||}} \lhd C')_A \bigcup \hat{\Delta}(\widetilde{C_{||}} \lhd C')_S \bigcup \hat{\Delta}(\widetilde{C_{||}} \lhd C')_P \bigcup \\
&\quad\ \ \hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA} \cup \hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC} \\
&\overset{def}{=} \{ (Q', \Sigma', P') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
&\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\
&\qquad \exists \mathcal{E}'' = \Sigma_{||} - \Sigma', \mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C') \bullet \mathcal{E}'' = \mathcal{I}'') \} \bigcup \\
&\quad\ \ \{ (Q', \Sigma', P') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
&\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\
&\qquad \Sigma_{||} = \Sigma')\} \bigcup \\
&\quad\ \ \{ (Q', \mathcal{N}(\Sigma' \cup \mathcal{E}''), P') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
&\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\
&\qquad \Sigma_{||} \neq \Sigma' \wedge \mathcal{E}'' = \Sigma_{||} - \Sigma' \wedge \\
&\qquad \exists \mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C') \bullet (\mathcal{E}'' \neq \mathcal{I}'' \wedge \\
&\qquad\quad (\exists \mathcal{S}'' \in \{ S_{||} - S' \mid S_{||} \in \hat{Q}(\widetilde{C_{||}}) \wedge S' \in \hat{Q}(C') \wedge S_{||} \cap S' \neq \{\}\} \bullet \\
&\qquad\qquad (Q' \cup \mathcal{S}'', \Sigma' \cup \mathcal{I}'', P' \cup \mathcal{S}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \vee \\
&\qquad\quad\ \exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{||} - P' \bullet \\
&\qquad\qquad (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}})))) \} \bigcup \\
&\quad\ \ \{ (Q', \mathcal{E}'', Q') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge \\
&\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap Q' \neq \{\}) \wedge \\
&\qquad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\
&\qquad \exists S' \in \hat{Q}(C'),\ \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{||} - Q' \bullet \\
&\qquad\ \ (S' \cup \mathcal{Q}'', \Gamma(C') \cup \mathcal{E}'', S' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}})) \} \bigcup \\
&\quad\ \ \{ (Q', \mathcal{E}'', Q') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge \\
&\qquad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap Q' \neq \{\}) \wedge \\
&\qquad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\
&\qquad \exists P' \in \hat{Q}(C'),\ \Sigma' \in \hat{\Sigma}(C'), \mathcal{Q}'' = Q_{||} - Q',\ \mathcal{P}'' = P_{||} - Q' \bullet (\\
&\qquad\quad (Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
&\qquad\quad \Sigma' \notin \hat{\Sigma}(\widetilde{C_{||}})) \} \bigcup \\
&\quad\ \ \{ (P', \mathcal{E}'', P') \mid \exists Q_{||}, \Sigma_{||}, P_{||} \bullet (\\
&\qquad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge P' \in \hat{Q}(C') \wedge \\
&\qquad (Q_{||} \cap P' \neq \{\}) \wedge (\Sigma_{||} \cap \Gamma(C') \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\
&\qquad \mathcal{E}'' = \Sigma_{||} - \Gamma(C') \wedge \\
&\qquad \exists Q' \in \hat{Q}(C'),\ \Sigma' \in \hat{\Sigma}(C'),\ \mathcal{Q}'' = Q_{||} - P',\ \mathcal{P}'' = P_{||} - P' \bullet (\\
&\qquad\quad (Q' \cup \mathcal{Q}'', \Sigma' \cup \mathcal{E}'', P' \cup \mathcal{P}'') \notin \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\
&\qquad\quad \Sigma' \notin \hat{\Sigma}(\widetilde{C_{||}}))) \}
\end{aligned}
$$

Appendix B

Mathematical Properties

This appendix examines the mathematical properties of the merge, concurrent composition and extract operators defined in Chapters 4 and 5. Although not explicitly used in this thesis, the merge composition operator defined in section 4.1 is extremely important because it forms the basis of the "superposition" of machines introduced in Chapter 3 and, therefore, the basis of the concurrent composition and extraction operators.

Every composite transition system C comprises a set of states, a set of initial states, a set of event names and a set of transitions. Any set can be written as the union of the members of that set, in other words, if $A = \{a, b, c, \ldots\}$ then A can be written as $A = \{a\} \cup \{b\} \cup \{c\} \cup \ldots$. Since the terms of merge composition are defined by set union, it follows that any system C can be expressed as the merge composition of machines, that is $A = \{a\} + \{b\} + \{c\} + \ldots$, provided that merge composition is commutative and associative in the same way as set union.

Merge composition, concurrent composition and extraction have been defined with set theoretic operators, the logical operators of conjunction and disjunction, and universal and existential quantification. The laws of these operators are well defined [6, 30, 52, 71].

B.1 Merge Composition

The section proves the commutative and associative properties of merge composition, defined in section 4.1 (page 69).

B.1.1 Commutative Property of Merge

Law B.1 states that the merge composition operator is commutative, in other words, the order of the operands is irrelevant.

Law B.1 $C' + C'' = C'' + C'$

Proof follows from the proof that the merge composition of every term of a Composite Transition System is commutative according to the laws of set union, that is, $A \cup B = B \cup A$.

1. From Definition 4.1 (page 69), $\hat{Q}(C' + C'') \overset{def}{=} \hat{Q}(C') \cup \hat{Q}(C'')$, which, by commutivity of set union and Definition 4.1, can be written as $\hat{Q}(C'') \cup \hat{Q}(C') \overset{def}{=} \hat{Q}(C'' + C')$. Hence, $\hat{Q}(C' + C'') = \hat{Q}(C'' + C')$.
2. $\ddot{Q}(C' + C'') = \ddot{Q}(C'' + C')$ follows from the same reasoning.
3. $\hat{\Sigma}(C' + C'') = \hat{\Sigma}(C'' + C')$ follows from the same reasoning.
4. From Definition 4.4 (page 71), $\Gamma(C' + C'') \overset{def}{=} \mathcal{N}(\Gamma(C') \cup \Gamma(C''))$. Since set union is commutative, it can be defined that $\mathcal{N}(\Gamma(C') \cup \Gamma(C''))$ and $\mathcal{N}(\Gamma(C'') \cup \Gamma(C'))$ will generate the same new event name. Hence $\Gamma(C' + C'') = \Gamma(C'' + C')$.
5. $\hat{\Delta}(C' + C'') = \hat{\Delta}(C'' + C')$ follows from the same reasoning as item 1 above.

Since the merge composition of every term of $C' + C''$ is commutative, it follows that merge composition is commutative, thus $C' + C'' = C'' + C'$.

B.1.2 Associative Law of Merge

Law B.2 states that the merge composition operator is associative. This law and law B.1 justify the notational convenience of $C_0 + C_1 + \ldots + C_{n-1}$, where there is no defined order of composition, and provides the basis of the "superposition" of Composite Transition Systems.

Law B.2 $(C' + C'') + C''' = C' + (C'' + C''')$

Proof follows from the proof that the merge composition of every term of a CTS is associative according to the laws of set union, that is, $(A \cup B) \cup C = A \cup (B \cup C)$.

1. From Definition 4.1 (page 69) and by substitution, $\hat{Q}((C'+C'')+C''') \overset{def}{=} (\hat{Q}(C') \cup \hat{Q}(C'')) \cup \hat{Q}(C''')$ which, by associativity of set union and Definition 4.1, becomes $\hat{Q}(C') \cup (\hat{Q}(C'') \cup \hat{Q}(C''')) \overset{def}{=} \hat{Q}(C'+(C''+C'''))$. Hence, $\hat{Q}((C'+C'')+C''') = \hat{Q}(C'+(C''+C'''))$.

2. $\ddot{Q}((C'+C'')+C''') = \ddot{Q}(C'+(C''+C'''))$ follows from the same reasoning.

3. $\hat{\Sigma}((C'+C'')+C''') = \hat{\Sigma}(C'+(C''+C'''))$ follows from the same reasoning.

4. From Definition 4.4 (page 71) and by substitution, $\Gamma((C'+C'')+C''') \overset{def}{=} \mathcal{N}((\Gamma(C') \cup \Gamma(C'')) \cup \Gamma(C'''))$. Since $(\Gamma(C') \cup \Gamma(C'')) \cup \Gamma(C''')$ is associative, it can be defined that $\mathcal{N}((\Gamma(C') \cup \Gamma(C'')) \cup \Gamma(C'''))$ and $\mathcal{N}(\Gamma(C') \cup (\Gamma(C'') \cup \Gamma(C''')))$ will generate the same event name. Hence $\Gamma((C'+C'')+C''') = \Gamma(C'+(C''+C''))$.

5. $\hat{\Delta}((C'+C'')+C''') = \hat{\Delta}(C'+(C''+C'''))$ follows from the same reasoning as item 1 above.

Since the merge composition of every term of $(C'+C'')+C'''$ is associative, it follows that merge composition is associative, thus $(C'+C'')+C''' = C'+(C''+C''')$.

B.2 Concurrent Composition

This section proves the commutative property of concurrent composition and examines the associative and distributive properties. Concurrent composition is defined in section 4.3 (page 74).

B.2.1 Commutative Law of Concurrent Composition

Law B.3 states that the current composition operator is commutative, in other words, the order of the operands is irrelevant.

Law B.3 $C'||C'' = C''||C'$

Proof follows from the proof that the concurrent composition of every term of a Composite Transition System is commutative according to the commutative laws of set union, that is, $A \cup B = B \cup A$, and the commutative laws of conjunction, that is $a \wedge b = b \wedge a$.

1. $\hat{Q}(C'||C'') = \hat{Q}(C''||C')$ follows from Definition 4.6 (page 75),
$$\begin{aligned}\hat{Q}(C'||C'') &\overset{def}{=} \{Q' \cup Q'' | Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'')\}\\ &= \{Q'' \cup Q' | Q'' \in \hat{Q}(C'') \wedge Q' \in \hat{Q}(C')\}\\ &= \hat{Q}(C''||C')\end{aligned}$$

2. $\ddot{Q}(C'||C'') = \ddot{Q}(C''||C')$ follows from the same reasoning.

3. The event name set $\hat{\Sigma}(C'||C'')$ is defined in Definition 4.8 (page 79) as the set union of term 4.1 through to term 4.4. Consider term 4.1 repeated below;
$$\{\, \Sigma' \cup \Gamma(C'') \mid \Sigma' \in \hat{\Sigma}(C') \wedge \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \,\}$$
Set union and intersection are commutative, the scope of the universal quantification includes an equality which is commutative, and conjunction is also commutative. Hence, by induction, term 4.1 is commutative. For similar reasons, the event names formed by terms 4.2, 4.3 and 4.4 are also commutative. The four commutative terms are combined by set union, which is commutative and associative, hence it follows that $\hat{\Sigma}(C'||C'') = \hat{\Sigma}(C''||C')$.

4. From Definition 4.9, $\Gamma(C'||C'') \overset{def}{=} \Gamma(C') \cup \Gamma(C'')$. Since set union is commutative it follows that, $\Gamma(C'||C'') = \Gamma(C''||C')$.

5. The transition set $\hat{\Delta}(C'||C'')$ is defined in Definition 4.10 (page 83) as the set union of terms 4.5, 4.6 and 4.7. Consider term 4.5 repeated below;
$$\begin{aligned}\{\, &(Q' \cup Q'', \Sigma' \cup \Gamma(C''), P' \cup Q'') \mid \\ &\quad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q'' \in \hat{\Sigma}(C'') \wedge \Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C'||C'') \,\}\end{aligned}$$
The *from* state, event name and *to* state of the formed transition are commutative because they are formed by set union. Additionally, the term $\Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C'||C'')$ is commutative by item 3 above, and conjunction is commutative and associative. Hence, term 4.5 is commutative. For similar reasons, the transitions formed by terms 4.6 and 4.7 are also commutative. Since all three contributions to the transition set $\hat{\Delta}(C'||C'')$ are commutative and the contributions are combined by set union, which is commutative and associative, it follows that $\hat{\Delta}(C'||C'') = \hat{\Delta}(C''||C')$.

Since the concurrent composition of every term of $C'||C''$ is commutative it follows that concurrent composition is commutative, thus $C'||C'' = C''||C'$.

B.2.2 Associative Law of Concurrent Composition

The associative properties of concurrent composition follow from the substitution for all terms of a Composite Transition System except for the event name set $\hat{\Sigma}((C'||C'')||C''')$

and the transition set $\hat{\Delta}((C'\|C'')\|C'')$. For the event name set, substitution is complex because of the universally quantified terms used to determine that an event name pairing should be subject to asynchronous composition, and because of the number of substitutions required in each of the four terms defined in Definition 4.8 (page 79). For the transition set, substitution is complex because the number of substitutions and because the definition explicitly includes the event name set.

Let those event names of C' that are asynchronous to *all* the event names of C'', be denoted $\mathcal{A}\Sigma_{\Sigma'}$, and let those event names of C' that are synchronous with an event name of C'' be denoted $\mathcal{S}\Sigma_{\Sigma'}$. In this way each and every event name in an event name set is categorised as either an asynchronous event name or a synchronous event name. Similarly, let those event names of C'' that are asynchronous to *all* the event names of C', be denoted $\mathcal{A}\Sigma_{\Sigma''}$, and let those event names of C'' that are synchronous to C' be denoted $\mathcal{S}\Sigma_{\Sigma''}$. Therefore, in general, the event name sets $\hat{\Sigma}(C')$, $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C'\|C'')$ can be written as follows;

$$\begin{aligned}
\hat{\Sigma}(C') &= \{\mathcal{A}\Sigma_{A'}, \mathcal{A}\Sigma_{B'}, \ldots\} + \{\mathcal{S}\Sigma_{M'}, \mathcal{S}\Sigma_{N'}, \ldots\} \\
&= \{\mathcal{A}\Sigma_{A'}\} + \{\mathcal{A}\Sigma_{B'}\} + \ldots + \{\mathcal{S}\Sigma_{M'}\} + \{\mathcal{S}\Sigma_{N'}\} + \ldots \\
\hat{\Sigma}(C'') &= \{\mathcal{A}\Sigma_{A''}, \mathcal{A}\Sigma_{B''}, \ldots\} + \{\mathcal{S}\Sigma_{M''}, \mathcal{S}\Sigma_{N''}, \ldots\} \\
&= \{\mathcal{A}\Sigma_{A''}\} + \{\mathcal{A}\Sigma_{B''}\} + \ldots + \{\mathcal{S}\Sigma_{M''}\} + \{\mathcal{S}\Sigma_{N''}\} + \ldots
\end{aligned}$$

$$\begin{aligned}
\hat{\Sigma}(C'\|C'') &= (\{\mathcal{A}\Sigma_{A'}\} + \{\mathcal{A}\Sigma_{B'}\} + \ldots + \{\mathcal{S}\Sigma_{M'}\} + \{\mathcal{S}\Sigma_{N'}\} + \ldots)\| \\
&\quad (\{\mathcal{A}\Sigma_{A''}\} + \{\mathcal{A}\Sigma_{B''}\} + \ldots + \{\mathcal{S}\Sigma_{M''}\} + \{\mathcal{S}\Sigma_{N''}\} + \ldots)
\end{aligned}$$

Recall from Chapter 4 that concurrent composition forms all event name pairings and tests each pairing for asynchrony or synchrony such that an asynchronous event name or synchronous event name can be contributed to the composite event name set. Recall also that any set can be defined by the set union of each of the members of that set and, further, that set union can be written as merge composition. Thus, the event name set $\hat{\Sigma}(C'\|C'')$, which can be written as in term B.1, is the "superposition" of each individual event name pairing.

$$\begin{aligned}
\hat{\Sigma}(C'\|C'') &= (\{\mathcal{A}\Sigma_{A'}\}\|_{\mathcal{A}}\{\mathcal{A}\Sigma_{A''}\}) + (\{\mathcal{A}\Sigma_{A'}\}\|_{\mathcal{A}}\{\mathcal{A}\Sigma_{B''}\}) + \ldots \\
&\quad +(\{\mathcal{A}\Sigma_{B'}\}\|_{\mathcal{A}}\{\mathcal{A}\Sigma_{A''}\}) + (\{\mathcal{A}\Sigma_{B'}\}\|_{\mathcal{A}}\{\mathcal{A}\Sigma_{B''}\}) + \ldots \\
&\quad +(\{\mathcal{S}\Sigma_{M'}\}\|_{\mathcal{S}}\{\mathcal{S}\Sigma_{M''}\}) + (\{\mathcal{S}\Sigma_{N'}\}\|_{\mathcal{S}}\{\mathcal{S}\Sigma_{N''}\}) + \ldots
\end{aligned} \quad (B.1)$$

Observe that $\|_{\mathcal{A}}$ forms all pairings of asynchronous event names, but $\|_{\mathcal{S}}$ forms a synchronous event name from a pairing of $\{\mathcal{S}\Sigma_{M'}\}$ with $\{\mathcal{S}\Sigma_{M''}\}$ only if there is at least one common component identifier in the event names M' and M''. Thus, $\|_{\mathcal{A}}$ distributes over $+$ in accordance with *ring* theory [6], set union over set intersection, conjunction over disjunction, and multiplication over addition [6, 52, 71]. However, $\|_{\mathcal{S}}$ does not distribute in this way.

In the definition of the concurrent event name set, section 4.3.3 (page 76), terms 4.1, 4.2 and 4.3 generate asynchronous and simultaneous event names, that is, the event names generated by the operator $\|_{\mathcal{A}}$. Term 4.4 (page 78) generates synchronous event names, that is, the event names generated by the operator $\|_{\mathcal{S}}$. Therefore, by Definition 4.8 (page 79), the operators $\|_{\mathcal{A}}$ and $\|_{\mathcal{S}}$ are defined as follows, and hence $\hat{\Sigma}(C'\|C'') = \hat{\Sigma}(C')\|_{\mathcal{A}}\hat{\Sigma}(C'') \bigcup \hat{\Sigma}(C')\|_{\mathcal{S}}\hat{\Sigma}(C'')$.

$$\begin{array}{rcll}
\hat{\Sigma}(C')\|_{\mathcal{A}}\hat{\Sigma}(C'') & \overset{def}{=} & \{\ \Sigma' \cup \Gamma(C'') & \mid\ \Sigma' \in \hat{\Sigma}(C') \wedge \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}\ \} \\
& \bigcup & \{\ \Gamma(C') \cup \Sigma'' & \mid\ \Sigma'' \in \hat{\Sigma}(C'') \wedge \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}\ \} \\
& \bigcup & \{\ \Sigma' \cup \Sigma'' & \mid\ \Sigma' \in \hat{\Sigma}(C') \wedge \forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\} \wedge \\
& & & \quad \Sigma'' \in \hat{\Sigma}(C'') \wedge \forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}\ \}
\end{array}$$

$$\begin{array}{rcll}
\hat{\Sigma}(C')\|_{\mathcal{S}}\hat{\Sigma}(C'') & \overset{def}{=} & \{\ \Sigma' \cup \Sigma'' & \mid\ \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma'' \in \hat{\Sigma}(C'') \wedge \\
& & & \quad \Sigma' \cap \Sigma'' \neq \{\}\ \}
\end{array}$$

Each synchronous event name pairing will contribute an event name of the form $\Sigma' \cup \Sigma''$ provided that $\Sigma' \cap \Sigma'' \neq \{\}$. Each asynchronous event name pairing, may contribute some or all of the following event names;

1. $\Sigma' \cup \Gamma(C'')$ is contributed only if the event name Σ' has nothing in common with any event name of C'', that is, the test $\forall \Upsilon \in \hat{\Sigma}(C'') \bullet \Sigma' \cap \Upsilon = \{\}$ in term 4.1 (page 77) holds true. In other words, Σ' is asynchronous to every event name of $\hat{\Sigma}(C'')$.

2. $\Gamma(C') \cup \Sigma''$ is contributed only if the event name Σ'' has nothing in common with any event name of C', that is, the test $\forall \Upsilon \in \hat{\Sigma}(C') \bullet \Sigma'' \cap \Upsilon = \{\}$ in term 4.2 (page 77) holds true. In other words, Σ'' is asynchronous to every event name of $\hat{\Sigma}(C')$.

3. $\Sigma' \cup \Sigma''$ is contributed only if both the event names $\Sigma' \cup \Gamma(C'')$ and $\Gamma(C') \cup \Sigma''$ are contributed. This is a consequence of term 4.3 (page 78).

This treatment of event names is sufficient for a proof of Law B.4 that states that concurrent composition operator is associative, in other words, the order of composition is irrelevant. Further, this law justifies the notational convenience of $C_0\| \ldots \|C_{n-1}$, where, there is no defined order of composition.

Law B.4 $(C'\|C'')\|C''' = C'\|(C''\|C''')$

Proof follows from the proof that the concurrent composition of every term of a Composite Transition System is associative.

1. From Definition 4.6 (page 75) and by substitution the state set $\hat{Q}((C'\|C'')\|C''')$ is $\{(Q' \cup Q'') \cup Q''' \mid (Q' \cup Q'') \in \hat{Q}(C'\|C'') \wedge Q''' \in \hat{Q}(C''')\}$. The states $(Q' \cup Q'') \in \hat{Q}(C'\|C'')$ are given by Definition 4.6, hence the state set $\hat{Q}((C'\|C'')\|C''')$ can be written as $\{(Q' \cup Q'') \cup Q''' \mid (Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'')) \wedge Q''' \in \hat{Q}(C''')\}$. Noting that conjunction is associative, that is $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, the state set can be written as $\{Q' \cup (Q'' \cup Q''') \mid Q' \in \hat{Q}(C') \wedge (Q'' \in \hat{Q}(C'') \wedge Q''' \in \hat{Q}(C'''))\}$, which, by Definition 4.6, can be written as $\{Q' \cup (Q'' \cup Q''') \mid Q' \in \hat{Q}(C') \wedge (Q'' \cup Q''') \in \hat{Q}(C''\|C''')\}$. Hence, $\hat{Q}((C'\|C'')\|C''') = \hat{Q}(C'\|(C''\|C'''))$.

2. $\ddot{Q}((C'\|C'')\|C''') = \ddot{Q}(C'\|(C''\|C'''))$ follows from the same reasoning.

3. To prove that $\hat{\Sigma}((C'\|C'')\|C''') = \hat{\Sigma}(C'\|(C''\|C''')$, the following four cases of substitution must all be associative. By the principle of "superposition", it is sufficient to consider the composition of individual event names.

   (a) $(\{\Sigma'\}\|_{\mathcal{A}}\{\Sigma''\})\|_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}\|_{\mathcal{A}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$, where the event name sets $\hat{\Sigma}(C')$, $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C''')$ have no component event name identifers in common.

      i. $\{\Sigma'\}\|_{\mathcal{A}}\{\Sigma''\}$ contributes the event names $\{\Sigma' \cup \Gamma''\} + \{\Gamma' \cup \Sigma''\} + \{\Sigma' \cup \Sigma''\}$.

      ii. By substitution and distribution $(\{\Sigma'\}\|_{\mathcal{A}}\{\Sigma''\})\|_{\mathcal{A}}\{\Sigma'''\}$ can be written as;

$$\begin{aligned}&(\{\Sigma' \cup \Gamma''\} + \{\Gamma' \cup \Sigma''\} + \{\Sigma' \cup \Sigma''\})\|_{\mathcal{A}}\{\Sigma'''\}\\ &= (\{\Sigma' \cup \Gamma''\}\|_{\mathcal{A}}\{\Sigma'''\}) + (\{\Gamma' \cup \Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\}) + (\{\Sigma' \cup \Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})\end{aligned}$$

      and contributes the following event names, note that $\{(\Gamma' \cup \Gamma'') \cup \Sigma'''\}$ is contributed three times;

$$\begin{aligned}&\{(\Sigma' \cup \Gamma'') \cup \Gamma'''\} \cup \{(\Gamma' \cup \Gamma'') \cup \Sigma'''\} \cup \{(\Sigma' \cup \Gamma'') \cup \Sigma''\}\\ &\cup \{(\Gamma' \cup \Sigma'') \cup \Gamma'''\} \cup \{(\Gamma' \cup \Gamma'') \cup \Sigma'''\} \cup \{(\Gamma' \cup \Sigma'') \cup \Sigma''\}\\ &\cup \{(\Sigma' \cup \Sigma'') \cup \Gamma'''\} \cup \{(\Gamma' \cup \Gamma'') \cup \Sigma'''\} \cup \{(\Sigma' \cup \Sigma'') \cup \Sigma''\}\end{aligned}$$

      which, by associativity of set union, can be written as;

$$\begin{aligned}&\{\Sigma' \cup (\Gamma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Gamma'' \cup \Sigma''')\}\\ &\cup \{\Gamma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Gamma' \cup (\Sigma'' \cup \Sigma''')\}\\ &\cup \{\Sigma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}\end{aligned}$$

      iii. Two of the $\{\Gamma' \cup (\Gamma'' \cup \Sigma''')\}$ terms can be deleted. However, in a composition of the form $\{\Sigma'\}\|_{\mathcal{A}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$, the term $\{\Sigma' \cup (\Gamma'' \cup \Gamma''')\}$ would be repeated three times. Hence, with some re-ordering, the contributed event names can be written as;

$$\begin{aligned}&\{\Sigma' \cup (\Gamma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Gamma''')\}\\ &\cup \{\Sigma' \cup (\Gamma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Gamma'' \cup \Sigma''')\}\\ &\cup \{\Sigma' \cup (\Gamma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Sigma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}\end{aligned}$$

iv. The contributed event names would also be contributed by;

$$(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma'' \cup \Gamma'''\}) + (\{\Sigma'\}||_{\mathcal{A}}\{\Gamma'' \cup \Sigma'''\}) + (\{\Sigma'\}||_{\mathcal{A}}(\{\Sigma'' \cup \Sigma'''\})$$
$$= \{\Sigma'\}||_{\mathcal{A}}(\{\Sigma'' \cup \Gamma'''\} + \{\Gamma'' \cup \Sigma'''\} + \{\Sigma'' \cup \Sigma'''\})$$

and the terms $\{\Sigma'' \cup \Gamma'''\} + \{\Gamma'' \cup \Sigma'''\} + \{\Sigma'' \cup \Sigma'''\}$ would also be contributed by $\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\}$, hence the contributed event names would also be generated by a composition of the form $\{\Sigma'\}||_{\mathcal{A}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$.

Hence $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{A}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$.

(b) $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$, where only the event names Σ' and Σ''' have some component event name identifier in common. Hence, Σ' must synchronise with Σ''', but Σ'' is asynchronous to both C' and C'''.

i. $\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\}$ contributes the event names $\{\Sigma' \cup \Gamma''\} + \{\Gamma' \cup \Sigma''\} + \{\Sigma' \cup \Sigma''\}$.

ii. By substitution and distribution $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\}$ can be written as $(\{\Sigma' \cup \Gamma''\}||_{\mathcal{S}}\{\Sigma'''\}) + (\{\Gamma' \cup \Sigma''\}||_{\mathcal{A}}\{\Sigma'''\}) + (\{\Sigma' \cup \Sigma''\}||_{\mathcal{S}}\{\Sigma'''\})$, where the expected $\{\Gamma' \cup \Sigma''\}||_{\mathcal{S}}\{\Sigma'''\}$ composition has become asynchronous because both Γ' and Σ'' are asynchronous to Σ'''.

The two synchronous compositions contribute the event names $(\Sigma' \cup \Gamma'') \cup \Sigma'''$ and $(\Sigma' \cup \Sigma'') \cup \Sigma'''$. The asynchronous composition contributes the event name $(\Gamma' \cup \Sigma'') \cup \Gamma'''$ because the event name $(\Gamma' \cup \Sigma'')$ has nothing in common with any of the event names of $\hat{\Sigma}(C''')$ (item 1, page 201). However, an event name $(\Gamma' \cup \Gamma'') \cup \Sigma'''$ is not contributed because the event name Σ''' includes a component event name which is common with some event name in $\hat{\Sigma}(C'||C'')$, specifically, $(\Sigma' \cup \Gamma'')$ (item 2, page 201). Hence, the event names contributed by $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\}$ are;
$\{(\Sigma' \cup \Gamma'') \cup \Sigma'''\} \cup \{(\Gamma' \cup \Sigma'') \cup \Gamma'''\} \cup \{(\Sigma' \cup \Sigma'') \cup \Sigma'''\}$
which, by associativity of set union, can be written as;
$\{\Gamma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Sigma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}$

iii. The event names $\Sigma'' \cup \Gamma'''$, $\Gamma'' \cup \Sigma'''$ and $\Sigma'' \cup \Sigma'''$ would also be contributed by $\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\}$ if the event name sets $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C''')$ have no component event name identifiers in common. Consider the evaluation of $\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$, that is, $\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma'' \cup \Gamma'''\} + \{\Gamma'' \cup \Sigma'''\} + \{\Sigma'' \cup \Sigma'''\})$ which can be written as follows;
$(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma'' \cup \Gamma'''\}) + (\{\Sigma'\}||_{\mathcal{S}}\{\Gamma'' \cup \Sigma'''\}) + (\{\Sigma'\}||_{\mathcal{S}}\{\Sigma'' \cup \Sigma''\})$
where the expected composition $\{\Sigma'\}||_{\mathcal{S}}\{\Sigma'' \cup \Gamma'''\}$ has become an asynchronous composition because both Σ'' and Γ''' are asynchronous to Σ'. However, as an asynchronous composition only the event name $\Gamma' \cup (\Sigma'' \cup$

$\Gamma''')$ is contributed because the event name $\Sigma'' \cup \Gamma'''$ has nothing in common with any event name of $\hat{\Sigma}(C')$ (item 2, page 201). Conversely, an event name $\Sigma' \cup (\Gamma'' \cup \Gamma''')$ is not contributed because the event name Σ' has something in common with the event names of $\hat{\Sigma}(C''\|C''')$, for example, the event name $\Sigma'' \cup \Sigma'''$ (item 1, page 201). The synchronous compositions contribute $\Sigma' \cup (\Gamma'' \cup \Sigma''')$ and $\Sigma' \cup (\Sigma'' \cup \Sigma''')$. Hence the event names contributed by $\{\Sigma'\}\|_{\mathcal{S}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$ are as follows;

$$\{\Gamma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Sigma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}$$

These event names are the same as item 3(b)ii above, and, likewise, require Σ'' to be asynchronous to both Σ' and Σ'''.

Hence $(\{\Sigma'\}\|_{\mathcal{A}}\{\Sigma''\})\|_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}\|_{\mathcal{S}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$. In this analysis of associativity, observe that the asynchronous composition $\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\}$ contributed event names of the form $\Sigma'' \cup \Gamma'''$, $\Gamma'' \cup \Sigma'''$ and $\Sigma'' \cup \Sigma'''$. Therefore, not only are the specific event names Σ'' and Σ''' required to be asynchronous, but Σ'' cannot be synchronous with any event name of C''' and Σ''' cannot be synchronous with any event name of C''.

(c) $(\{\Sigma'\}\|_{\mathcal{S}}\{\Sigma''\})\|_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}\|_{\mathcal{S}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$, where only the event names Σ' and Σ'' have some component event name identifier in common. Hence, Σ' must synchronise with Σ'', but Σ''' is asynchronous to both C' and C''.

i. $\{\Sigma'\}\|_{\mathcal{S}}\{\Sigma''\}$ contributes $\{\Sigma' \cup \Sigma''\}$, hence $(\{\Sigma'\}\|_{\mathcal{S}}\{\Sigma''\})\|_{\mathcal{A}}\{\Sigma'''\}$ can be written as $\{\Sigma' \cup \Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\}$ which contributes the event names;

$$\begin{aligned}&\{(\Sigma' \cup \Sigma'') \cup \Gamma'''\} + \{(\Gamma' \cup \Gamma'') \cup \Sigma'''\} + \{(\Sigma' \cup \Sigma'') \cup \Sigma'''\}\\ &= \{\Sigma' \cup (\Sigma'' \cup \Gamma''')\} + \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} + \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}\end{aligned}$$

ii. The event names $\Sigma'' \cup \Gamma'''$, $\Gamma'' \cup \Sigma'''$ and $\Sigma'' \cup \Sigma'''$ would also be contributed by $\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\}$ if the event name sets $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C''')$ have no component event name identifiers in common. Consider the evaluation of $\{\Sigma'\}\|_{\mathcal{S}}(\{\Sigma''\}\|_{\mathcal{A}}\{\Sigma'''\})$, that is, $\{\Sigma'\}\|_{\mathcal{S}}(\{\Sigma'' \cup \Gamma'''\} + \{\Gamma'' \cup \Sigma'''\} + \{\Sigma'' \cup \Sigma'''\})$ which can be written as follows;

$$(\{\Sigma'\}\|_{\mathcal{S}}\{\Sigma'' \cup \Gamma'''\}) + (\{\Sigma'\}\|_{\mathcal{A}}\{\Gamma'' \cup \Sigma'''\}) + (\{\Sigma'\}\|_{\mathcal{S}}\{\Sigma'' \cup \Sigma'''\})$$

where the expected term $\{\Sigma'\}\|_{\mathcal{S}}\{\Gamma'' \cup \Sigma'''\}$ has become an asynchronous composition because both Γ'' and Σ''' are asynchronous to Σ'. However, as an asynchronous composition only the event name $\Gamma' \cup (\Gamma'' \cup \Sigma''')$ is contributed because the event name $\Gamma'' \cup \Sigma'''$ has nothing in common with any event name of $\hat{\Sigma}(C')$ (item 2, page 201). Conversely, an event name $\Sigma' \cup (\Gamma'' \cup \Sigma''')$ is not contributed because the event name Σ' has something in common with the event names of $\hat{\Sigma}(C''||C''')$, for example, the event name $\Sigma'' \cup \Sigma'''$ (item 1, page 201). The synchronous compositions contribute $\Sigma' \cup (\Gamma'' \cup \Sigma''')$ and $\Sigma' \cup (\Sigma'' \cup \Sigma''')$. Hence the event names contributed by $\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$ are as follows;

$$\{\Sigma' \cup (\Sigma'' \cup \Gamma''')\} \cup \{\Gamma' \cup (\Gamma'' \cup \Sigma''')\} \cup \{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}$$

These event names are the same as item 3(c)i, above, and, likewise, require Σ''' to be asynchronous to both Σ' and Σ''.

Hence $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$. In a similar way to case 3b (page 203), C''' must be asynchronous to both C' and C''.

(d) $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{S}}\{\Sigma'''\})$, where the event names Σ', Σ'' and Σ''' have some component event name identifier in common, that is, $\Sigma' \cap \Sigma'' \cap \Sigma''' \neq \{\}$.

i. $\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\}$ contributes $\{\Sigma' \cup \Sigma''\}$.

ii. By substitution, $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\}$ becomes $(\{\Sigma' \cup \Sigma''\})||_{\mathcal{S}}\{\Sigma'''\}$, which contributes $\{(\Sigma' \cup \Sigma'') \cup \Sigma'''\}$.

iii. By associativity of set union, the contribution $\{(\Sigma' \cup \Sigma'') \cup \Sigma'''\}$ can be written as $\{\Sigma' \cup (\Sigma'' \cup \Sigma''')\}$, which would be contributed by;
$\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma'' \cup \Sigma'''\})$

iv. Since Σ'' synchronises with Σ''', the term $\{\Sigma'' \cup \Sigma'''\}$ would be contributed by $\{\Sigma''\}||_{\mathcal{S}}\{\Sigma'''\}$. Hence, $\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma'' \cup \Sigma'''\})$ can be written as;
$\{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{S}}\{\Sigma'''\})$

Hence $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{S}}\{\Sigma'''\})$.

Since all four combinations of asynchronous and synchronous composition are associative, it follows that $\hat{\Sigma}((C'||C'')||C''') = \hat{\Sigma}(C'||(C''||C''')$, however, the following conditions arose;

(a) There is no synchronisation between any of the components. Proof of this follows from case 3a, the proof of $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{A}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$.

(b) There is synchronisation between only two of the components. Proof of this follows from case 3b, the proof of $(\{\Sigma'\}||_{\mathcal{A}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$, and from case 3c, the proof of $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{A}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{A}}\{\Sigma'''\})$. Note that this condition makes case 3d, the proof of $(\{\Sigma'\}||_{\mathcal{S}}\{\Sigma''\})||_{\mathcal{S}}\{\Sigma'''\} = \{\Sigma'\}||_{\mathcal{S}}(\{\Sigma''\}||_{\mathcal{S}}\{\Sigma'''\})$, irrelevant.

4. By substitution and associativity of set union, proof that the composition of the idle event name is associative is as follows;

$$\begin{aligned}\Gamma((C'\|C'')\|C''') &\overset{def}{=} \Gamma(C'\|C'') \cup \Gamma(C''') \overset{def}{=} (\Gamma(C') \cup \Gamma(C'')) \cup \Gamma(C''') \\ &= \Gamma(C') \cup (\Gamma(C'') \cup \Gamma(C''')) \\ &= \Gamma(C'\|(C''\|C'''))\end{aligned}$$

5. The formation of the transition set $\hat{\Delta}((C'\|C'')\|C''')$ requires the substitution of the transition set $\hat{\Delta}(C'\|C'')$ for the C_l term in the formation of the transition set $\hat{\Delta}(C_l\|C''')$. The transition sets $\hat{\Delta}(C'\|C'')$ and $\hat{\Delta}(C_l\|C''')$ are defined as the union of terms 4.5, 4.6 and 4.7 (page 82 *et.seq.*); consider the substitutions required in each of these terms.

   (a) Term 4.5, repeated below in terms of C_l and C''', gives the set of $\mathcal{A}\Delta_{\Delta,(Q''',\Gamma''',Q''')}$ asynchronous transitions.

   $$\begin{aligned}&\{ (Q \cup Q''', \Sigma \cup \Gamma''', P \cup Q''') \mid \\ &\qquad (Q, \Sigma, P) \in \hat{\Delta}(C_l) \wedge Q''' \in \hat{Q}(C''') \wedge \Sigma \cup \Gamma''' \in \hat{\Sigma}(C_l\|C''') \}\end{aligned}$$

   The C_l contribution is a transition (Q, Σ, P) that under the substitution of C_l terms is a $\hat{\Delta}(C'\|C'')$ transition, itself formed by the union of terms 4.5, 4.6 and 4.7, thus there are three substitutions to perform. For brevity, only the substitution of term 4.5 is described in detail as the substitutions of term 4.6 and 4.7 follow in a similar way.

      i. Term 4.5 generates transitions of the form $(Q' \cup Q'', \Sigma' \cup \Gamma'', P' \cup Q'')$. Substituting $Q' \cup Q''$ for Q, $\Sigma' \cup \Gamma''$ for Σ, and $P' \cup Q''$ for P gives;

      $$\begin{aligned}&\{ ((Q' \cup Q'') \cup Q''', (\Sigma' \cup \Gamma'') \cup \Gamma''', (P' \cup Q'') \cup Q''') \mid \\ &\qquad (Q' \cup Q'', \Sigma' \cup \Gamma'', P' \cup Q'') \in \hat{\Delta}(C'\|C'') \wedge Q''' \in \hat{Q}(C''') \wedge \\ &\qquad (\Sigma' \cup \Gamma'') \cup \Gamma''' \in \hat{\Sigma}((C'\|C'')\|C''') \}\end{aligned}$$

      ii. Term 4.5 forms the substituted transition, $(Q' \cup Q'', \Sigma' \cup \Gamma'', P' \cup Q'')$, when $(Q', \Sigma', P') \in \hat{\Delta}(C')$, $Q'' \in \hat{Q}(C'')$, and $\Sigma' \cup \Gamma'' \in \hat{\Sigma}(C'\|C'')$. Hence;

      $$\begin{aligned}&\{ ((Q' \cup Q'') \cup Q''', (\Sigma' \cup \Gamma'') \cup \Gamma''', (P' \cup Q'') \cup Q''') \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q'' \in \hat{\Sigma}(C'') \wedge Q''' \in \hat{Q}(C''') \wedge \\ &\qquad (\Sigma' \cup \Gamma'') \cup \Gamma''' \in \hat{\Sigma}((C'\|C'')\|C''') \}\end{aligned}$$

      which, by associativity of the concurrent composition of event name sets and associativity of set union, yields;

      $$\begin{aligned}&\{ (Q' \cup (Q'' \cup Q'''), \Sigma' \cup (\Gamma'' \cup \Gamma'''), P' \cup (Q'' \cup Q''')) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q'' \in \hat{\Sigma}(C'') \wedge Q''' \in \hat{Q}(C''') \wedge \\ &\qquad \Sigma' \cup (\Gamma'' \cup \Gamma''') \in \hat{\Sigma}(C'\|(C''\|C''')) \}\end{aligned}$$

      iii. A transition of the form $(Q'' \cup Q''', \Gamma'' \cup \Gamma''', Q'' \cup Q''')$ is an implied idle transition of $C''\|C'''$. By Definition 4.6 (page 75), the state $Q'' \cup Q'''$

is a state of $\hat{Q}(C''\|C''')$, which is formed by a conjunction of the form $Q'' \in \hat{Q}(C'') \wedge Q''' \in \hat{Q}(C''')$. Hence;

$$\begin{aligned}\{\, &(Q' \cup (Q'' \cup Q'''), \Sigma' \cup (\Gamma'' \cup \Gamma'''), P' \cup (Q'' \cup Q''')) \mid \\ &\quad (Q', \Sigma', P') \in \hat{\Delta}(C') \;\wedge\; (Q'' \cup Q''') \in \hat{Q}(C''\|C''') \;\wedge \\ &\quad \Sigma' \cup (\Gamma'' \cup \Gamma''') \in \hat{\Sigma}(C'\|(C''\|C''')) \,\}\end{aligned}$$

iv. Substituting Q for $Q'' \cup Q'''$, Γ for $\Gamma'' \cup \Gamma'''$ and C_r for $C''\|C'''$ gives the following, which is of the same form as term 4.5.

$$\begin{aligned}\{\, &(Q' \cup Q, \Sigma' \cup \Gamma, P' \cup Q) \mid \\ &\quad (Q', \Sigma', P') \in \hat{\Delta}(C') \;\wedge\; Q \in \hat{Q}(C_r) \;\wedge\; \Sigma' \cup \Gamma \in \hat{\Sigma}(C'\|C_r) \,\}\end{aligned}$$

The other two substitutions in a term 4.5 transition are as follows. Substitution of term 4.6 gives $((Q' \cup Q'') \cup Q''', (\Gamma' \cup \Sigma'') \cup \Gamma''', (Q' \cup P'') \cup Q''')$, which, by set associativity, is $(Q' \cup (Q'' \cup Q'''), \Gamma' \cup (\Sigma'' \cup \Gamma'''), Q' \cup (P'' \cup Q'''))$. The latter form is a $C''\|C'''$ term 4.5 transition, that is $(Q'' \cup Q''', \Sigma'' \cup \Gamma''', P'' \cup Q''')$, substituted in a term 4.6 transition, that is $(Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P)$.

Substitution of term 4.7 gives $((Q' \cup Q'') \cup Q''', (\Sigma' \cup \Sigma'') \cup \Gamma''', (P' \cup P'') \cup Q''')$, which, by set associativity, is $(Q' \cup (Q'' \cup Q'''), \Sigma' \cup (\Sigma'' \cup \Gamma'''), P' \cup (P'' \cup Q'''))$. The latter transition is also of the form of a $C''\|C'''$ term 4.5 transition, but substituted in a term 4.7 transition, that is $(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P)$.

(b) Term 4.6, repeated below in terms of C_l and C''', gives the set of $\mathcal{A}\Delta_{\Delta''',(Q,\Gamma,Q)}$ asynchronous transitions.

$$\begin{aligned}\{\, &(Q \cup Q''', \Gamma \cup \Sigma''', Q \cup P''') \mid \\ &\quad Q \in \hat{Q}(C_l) \;\wedge\; (Q''', \Sigma''', P''') \in \hat{\Delta}(C''') \;\wedge\; \Gamma \cup \Sigma''' \in \hat{\Sigma}(C_l\|C''') \,\}\end{aligned}$$

The C_l contribution is the implied transition (Q, Γ, Q), thus the implied idle transition $(Q' \cup Q'', \Gamma' \cup \Gamma'', Q' \cup Q'')$ is the only applicable substitution.

i. Substituting $Q' \cup Q''$ for Q, and $\Gamma' \cup \Gamma''$ for Γ gives;

$$\begin{aligned}\{\, &((Q' \cup Q'') \cup Q''', (\Gamma' \cup \Gamma'') \cup \Sigma''', (Q' \cup Q'') \cup P''')) \mid \\ &\quad (Q' \cup Q'') \in \hat{Q}(C'\|C'') \;\wedge\; (Q''', \Sigma''', P''') \in \hat{\Delta}(C''') \;\wedge \\ &\quad (\Gamma' \cup \Gamma'') \cup \Sigma''' \in \hat{\Sigma}((C'\|C'')\|C''') \,\}\end{aligned}$$

ii. By Definition 4.6 (page 75), the state $\hat{Q}(C'\|C'')$ is formed by a conjunction of the form $Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'')$. Hence;

$$\begin{aligned}\{\, &((Q' \cup Q'') \cup Q''', (\Gamma' \cup \Gamma'') \cup \Sigma''', (Q' \cup Q'') \cup P''') \mid \\ &\quad Q' \in \hat{Q}(C') \;\wedge\; Q'' \in \hat{Q}(C'') \;\wedge\; (Q''', \Sigma''', P''') \in \hat{\Delta}(C''') \;\wedge \\ &\quad (\Gamma' \cup \Gamma'') \cup \Sigma''' \in \hat{\Sigma}((C'\|C'')\|C''') \,\}\end{aligned}$$

which, by associativity of the concurrent composition of event name sets and associativity of set union, yields;

$$\begin{aligned}\{\, & Q' \cup (Q'' \cup Q'''), \Gamma' \cup (\Gamma'' \cup \Sigma'''), Q' \cup (Q'' \cup P''')) \mid \\ & Q' \in \hat{Q}(C') \wedge Q'' \in \hat{Q}(C'') \wedge (Q''', \Sigma''', P''') \in \hat{\Delta}(C''') \wedge \\ & \Gamma' \cup (\Gamma'' \cup \Sigma''') \in \hat{\Sigma}(C' \| (C'' \| C''')) \,\}\end{aligned}$$

iii. Term 4.6 contributes a transition of the form $(Q'' \cup Q''', \Gamma'' \cup \Sigma''', Q'' \cup P''')$ to the transition set $\hat{\Delta}(C'' \| C''')$ when $Q'' \in \hat{Q}(C'')$, $(Q''', \Sigma''', P''') \in \hat{\Delta}(C''')$, and the event name $\Gamma'' \cup \Sigma'''$ is in the event name set $\hat{\Sigma}(C'' \| C''')$. Hence;

$$\begin{aligned}\{\, & Q' \cup (Q'' \cup Q'''), \Gamma' \cup (\Gamma'' \cup \Sigma'''), Q' \cup (Q'' \cup P''')) \mid \\ & Q' \in \hat{Q}(C') \wedge (Q'' \cup Q''', \Gamma'' \cup \Sigma''', Q'' \cup P''') \in \hat{\Delta}(C'' \| C''') \wedge \\ & \Gamma' \cup (\Gamma'' \cup \Sigma''') \in \hat{\Sigma}(C' \| (C'' \| C''')) \,\}\end{aligned}$$

iv. The transition $(Q'' \cup Q''', \Gamma'' \cup \Sigma''', Q'' \cup P''')$ represents progress of $C'' \| C'''$. Therefore, substituting Q for $Q'' \cup Q'''$, Σ for $\Gamma'' \cup \Sigma'''$, P for $Q'' \cup P'''$ and C_r for $C'' \| C'''$ gives the following, which is of the same form as term 4.6;

$$\begin{aligned}\{\, & Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P) \mid \\ & Q' \in \hat{Q}(C') \wedge (Q, \Sigma, P) \in \hat{Q}(C_r) \wedge \Gamma' \cup \Sigma \in \hat{\Sigma}(C' \| C_r) \,\}\end{aligned}$$

(c) Term 4.7, repeated below in terms of C_l and C''', gives the set of $\mathcal{CS}\Delta_{\Delta,\Delta'''}$ simultaneous and synchronous transitions.

$$\begin{aligned}\{\, & (Q \cup Q''', \Sigma \cup \Sigma''', P \cup P''') \mid \\ & (Q, \Sigma, P) \in \hat{\Delta}(C_l) \wedge (Q''', \Sigma''', P''') \in \hat{\Delta}(C''') \wedge \Sigma \cup \Sigma''' \in \hat{\Sigma}(C_l \| C''') \,\}\end{aligned}$$

Here, the C_l contribution is a transition (Q, Σ, P) that under a substitution of $C' \| C''$ for C_l terms is a transition of $\hat{\Delta}(C' \| C'')$ which is formed by the union of terms 4.5, 4.6 and 4.7. Thus there are three substitutions to perform, however, these follow in a similar way to those presented for case 5a (page 206, *et.seq.*). Substitution of term 4.5 gives $((Q' \cup Q'') \cup Q''', (\Sigma' \cup \Gamma'') \cup \Sigma''', (P' \cup Q'') \cup P''')$, which, by set associativity, is $(Q' \cup (Q'' \cup Q'''), \Sigma' \cup (\Gamma'' \cup \Sigma'''), P' \cup (Q'' \cup P'''))$. The latter form is a $C'' \| C'''$ term 4.6 transition, that is $(Q'' \cup Q''', \Gamma'' \cup \Sigma''', Q'' \cup P''')$, substituted in a term 4.7 transition, that is $(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P)$.
Substitution of term 4.6 gives $((Q' \cup Q'') \cup Q''', (\Gamma' \cup \Sigma'') \cup \Sigma''', (Q' \cup P'') \cup P''')$, which, by set associativity, is $(Q' \cup (Q'' \cup Q'''), \Gamma' \cup (\Sigma'' \cup \Sigma'''), Q' \cup (P'' \cup P'''))$. The latter form is a $C'' \| C'''$ term 4.7 transition, that is $(Q'' \cup Q''', \Sigma'' \cup \Sigma''', P'' \cup P''')$, substituted in a term 4.6 transition, that is $(Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P)$.
Substitution of term 4.7 gives $((Q' \cup Q'') \cup Q''', (\Sigma' \cup \Sigma'') \cup \Sigma''', (P' \cup P'') \cup P''')$, which, by set associativity, is $(Q' \cup (Q'' \cup Q'''), \Sigma' \cup (\Sigma'' \cup \Sigma'''), P' \cup (P'' \cup P'''))$. The

latter form is a $C''\|C'''$ term 4.7 transition, that is $(Q'' \cup Q''', \Sigma'' \cup \Sigma''', P'' \cup P''')$, substituted in a term 4.7 transition, that is $(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P)$.

Associativity is proven if the definition of the transition set $\hat{\Delta}((C'\|C'')\|C''')$ generates the same transitions as the transition set $\hat{\Delta}(C'\|(C''\|C'''))$, itself formed by a substitution of the transition set $\hat{\Delta}(C''\|C''')$ for the C_r term in the transition set defined by $\hat{\Delta}(C'\|C_r)$. Since all combinations of transition substitution are associative, subject to the associativity constraints of the event name set, it follows that $\hat{\Delta}((C'\|C'')\|C''') = \hat{\Delta}(C'\|(C''\|C'''))$.

Table B.1 summarises the cases of transition substitution based upon the event names because it is the evaluation of the event name set that determines the formation of asynchronous or synchronous event names. The figures in parenthesis refer to the defining terms in section 4.3.5 (page 81).

case	$\hat{\Sigma}(C_l\|C''')$	$C_l = \hat{\Sigma}(C'\|C'')$	Composite	$C_r = \hat{\Sigma}(C''\|C''')$	$\hat{\Sigma}(C'\|C_r)$
(5a)	Σ, Γ''' (4.5)	Σ', Γ'' (4.5)	$\Sigma', \Gamma'', \Gamma'''$	Γ'', Γ''' (idle)	Σ', Γ (4.5)
		Γ', Σ'' (4.6)	$\Gamma', \Sigma'', \Gamma'''$	Σ'', Γ''' (4.5)	Γ', Σ (4.6)
		Σ', Σ'' (4.7)	$\Sigma', \Sigma'', \Gamma'''$	Σ'', Γ''' (4.5)	Σ', Σ (4.7)
(5b)	Γ, Σ''' (4.6)	Γ', Γ'' (idle)	$\Gamma', \Gamma'', \Sigma'''$	Γ'', Σ''' (4.6)	Γ', Σ (4.6)
(5c)	Σ, Σ''' (4.7)	Σ', Γ'' (4.5)	$\Sigma', \Gamma'', \Sigma'''$	Γ'', Σ''' (4.6)	Σ', Σ (4.7)
		Γ', Σ'' (4.6)	$\Gamma', \Sigma'', \Sigma'''$	Σ'', Σ''' (4.7)	Γ', Σ (4.6)
		Σ', Σ'' (4.7)	$\Sigma', \Sigma'', \Sigma'''$	Σ'', Σ''' (4.7)	Σ', Σ (4.7)

Table B.1: Associativity of concurrent composition of transitions based on event names.

B.2.3 Distributive Law of Concurrent Composition

Law B.5 states that current composition distributes over merge composition, however, it will be shown that distribution leads to congruence rather than equality and only if the components are asynchronous, that is, there are no event names common to the components.

Law B.5 $C'\|(C'' + C''') \equiv (C'\|C'') + (C'\|C''')$

Proof follows from the proof that the concurrent composition of every term of a Composite Transition System is distributive.

1. Proof that $\hat{Q}(C'||(C''+C''')) = \hat{Q}(C'||C'') + \hat{Q}(C'||C''')$, is as follows;

   (a) From Definition 4.6 (page 75), the state set $\hat{Q}(C'||(C''+C'''))$ can be written as $\{Q' \cup Q \,|\, Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C''+C''')\}$. From Definition 4.1 (page 69), the state set $\hat{Q}(C''+C''')$ is $\hat{Q}(C'') \cup \hat{Q}(C''')$. Therefore, by substitution, the state set $\hat{Q}(C'||(C''+C'''))$ becomes $\{Q' \cup Q \,|\, Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C'') \cup \hat{Q}(C''')\}$.

   (b) Set union is defined as $A \cup B = \{x | x \in A \vee x \in B\}$ [52], therefore, the term $Q \in \hat{Q}(C'') \cup \hat{Q}(C''')$ can be expressed as $Q \in \hat{Q}(C'') \vee Q \in \hat{Q}(C''')$. Hence, $\hat{Q}(C'||(C''+C''')) = \{Q' \cup Q | Q' \in \hat{Q}(C') \wedge (Q \in \hat{Q}(C'') \vee Q \in \hat{Q}(C'''))\}$, which, by distribution of conjunction over disjunction, that is, $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$, yields $\{Q' \cup Q | (Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C'')) \vee (Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C'''))\}$.

   (c) The conjunction $Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C'')$ defines the state set of $\hat{Q}(C'||C'')$, and $Q' \in \hat{Q}(C') \wedge Q \in \hat{Q}(C''')$ defines the state set of $\hat{Q}(C'||C''')$. Hence, the state set can be written as;

   $\{Q' \cup Q \,|\, Q' \cup Q \in \hat{Q}(C'||C'') \vee Q' \cup Q \in \hat{Q}(C'||C''')\}$

   which, from the definition of set union, can be written as;

   $\hat{Q}(C'||C'') \cup \hat{Q}(C'||C''')$

   (d) From Definition 4.1 (page 69) $\hat{Q}(C'||C'') \cup \hat{Q}(C'||C''')$ defines the merge composite state set $\hat{Q}(C'||C'') + \hat{Q}(C'||C''')$.

   Hence $\hat{Q}(C'||(C''+C''')) = \hat{Q}(C'||C'') + \hat{Q}(C'||C''')$.

2. $\ddot{Q}(C'||(C''+C''')) = \ddot{Q}(C'||C'') + \ddot{Q}(C'||C''')$ follows from the same reasoning.

3. Analysis of the distributive properties of the concurrent composition of the event name set follows from the notation introduced in section B.2.2 (page 199) which asserted that for the expression $\hat{\Sigma}(C'||C'')$, concurrent composition distributes over merge composition of each of the event names of C''. Thus for $\hat{\Sigma}(C'||C)$ written in terms of C' and C;

$$\begin{aligned}\hat{\Sigma}(C'||C) &= (\{\mathcal{A}\Sigma_{A'}\} + \{\mathcal{A}\Sigma_{B'}\} + \ldots + \{\mathcal{S}\Sigma_{M'}\} + \{\mathcal{S}\Sigma_{N'}\} + \ldots)|| \\ &\qquad (\{\mathcal{A}\Sigma_{A}\} + \{\mathcal{A}\Sigma_{B}\} + \ldots + \{\mathcal{S}\Sigma_{M}\} + \{\mathcal{S}\Sigma_{N}\} + \ldots) \\ &= (\{\mathcal{A}\Sigma_{A'}\}||\{\mathcal{A}\Sigma_{A}\}) + (\{\mathcal{A}\Sigma_{A'}\}||\{\mathcal{A}\Sigma_{B}\}) + \ldots \\ &\quad +(\{\mathcal{A}\Sigma_{B'}\}||\{\mathcal{A}\Sigma_{A}\}) + (\{\mathcal{A}\Sigma_{B'}\}||\{\mathcal{A}\Sigma_{B}\}) + \ldots \\ &\qquad +(\{\mathcal{S}\Sigma_{M'}\}||\{\mathcal{S}\Sigma_{M}\}) + (\{\mathcal{S}\Sigma_{N'}\}||\{\mathcal{S}\Sigma_{N}\}) + \ldots\end{aligned}$$

which, by distribution, may also be written as;

$$\begin{aligned}\hat{\Sigma}(C'||C) &= (\{\mathcal{A}\Sigma_{A'}\}||(\{\mathcal{A}\Sigma_{A}\} + \{\mathcal{A}\Sigma_{B}\} + \ldots)) + \\ &\quad (\{\mathcal{A}\Sigma_{B'}\}||(\{\mathcal{A}\Sigma_{A}\} + \{\mathcal{A}\Sigma_{B}\} + \ldots)) + \\ &\quad (\{\mathcal{S}\Sigma_{M'}\}||(\{\mathcal{S}\Sigma_{M}\} + \ldots)) + \\ &\qquad (\{\mathcal{S}\Sigma_{N'}\}||(\{\mathcal{S}\Sigma_{N}\} + \ldots)) + \ldots\end{aligned}$$

Now consider the composition denoted $C'||C^+$, where C^+ denotes the merge composition $C'' + C'''$. Let $\hat{\Sigma}(C'') = \{A\}$ and $\hat{\Sigma}(C''') = \{B\}$ and, therefore, $\hat{\Sigma}(C^+) = \{A\} + \{B\}$. Let $\Gamma(C^+) = \mathcal{N}(\Gamma(C'') \cup \Gamma(C'''))$.

(a) Consider the composition $\{\Sigma'\}||_{\mathcal{A}}(\{A\}+\{B\})$, where the event name sets $\hat{\Sigma}(C')$ and $\hat{\Sigma}(C^+)$ have no component event name identifers in common.

From term B.1 (page 200), the composition $\{\Sigma'\}||_{\mathcal{A}}(\{A\}+\{B\})$ can be written as $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma^+_A\})+(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma^+_B\})$, where $\mathcal{A}\Sigma^+_A$ and $\mathcal{A}\Sigma^+_B$ denotes that the (asynchronous) event names A and B are drawn from the event name set $\hat{\Sigma}(C^+)$. Consequently, the composition $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma^+_A\})$ can determine that the event name Σ' is asynchronous to A and to B. Likewise, the composition $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma^+_B\})$ can determine that Σ' is asynchronous to A and B.

Asynchronous event names contributed by this form of composition include the idle event name $\Gamma(C')$ and $\Gamma(C^+)$, both components of the idle event name of $\Gamma(C'||(C'' + C'''))$, see case 4a (page 212).

(b) Now consider the composition $(\{\Sigma'\}||_{\mathcal{A}}\{A\}) + (\{\Sigma'\}||_{\mathcal{A}}\{B\})$, where the event name sets $\hat{\Sigma}(C')$, $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C''')$ have no component event name identifers in common.

From term B.1 (page 200), the composition $(\{\Sigma'\}||_{\mathcal{A}}\{A\}) + (\{\Sigma'\}||_{\mathcal{A}}\{B\})$ can be written as $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma''_A\}) + (\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma'''_B\})$, where $\mathcal{A}\Sigma''_A$ and $\mathcal{A}\Sigma'''_B$ denote that the (asynchronous) event names A and B are drawn from the event name sets $\hat{\Sigma}(C'')$ and $\hat{\Sigma}(C''')$ respectively.

Unlike case 3a above, the composition $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma''_A\})$ can only determine if the event name Σ' is asynchronous to the event names of the event name set $\hat{\Sigma}(C'')$, which includes the event name A but not B. Therefore, Σ' will be combined with A as an asynchronous pairing, even though Σ' may be synchronous with the event name B. Similarly, the composition $(\{\mathcal{A}\Sigma_{\Sigma'}\}||_{\mathcal{A}}\{\mathcal{A}\Sigma'''_B\})$ can only determine if Σ' is asynchronous to the event names of $\hat{\Sigma}(C''')$, which includes the event name B but not A. Therefore, Σ' will be combined with B as an asynchronous pairing, even though Σ' may be synchronous with the event name A.

Asynchronous event names contributed by $C'||C''$ include the idle event names $\Gamma(C')$ and $\Gamma(C'')$, while $C'||C'''$ contributions include $\Gamma(C')$ and $\Gamma(C''')$. However, the idle event name $(C'||C'') + (C''||C''')$ is $\mathcal{N}(\Gamma(C') \cup \Gamma(C'') \cup \Gamma(C'''))$, see case 4b (page 212).

The differences between the composition $\{\Sigma'\}||_{\mathcal{A}}(\{A\} + \{B\})$ and the "distributed" form $(\{\Sigma'\}||_{\mathcal{A}}\{A\}) + (\{\Sigma'\}||_{\mathcal{A}}\{B\})$ are the event name set used in the test for asynchrony and, therefore, which event names are contributed, and the idle event names. Specifically, the distributed form may generate asynchronous event names when the non-distributed form would not because of synchronisation and different idle event names are used.

Consequently, $\hat{\Sigma}(C'\|(C''+C'''))$ and $\hat{\Sigma}(C'\|C'')+\hat{\Sigma}(C'\|C''')$ can be said to be congruent only for asynchronous components and because the idle event names in the distributed form are incorrect.

4. Concurrent composition does not distribute over merge for the evaluation of the idle event name, that is $\Gamma(C'\|(C''+C''')) \neq \Gamma((C'\|C'')+(C''\|C'''))$. This arises because the idle event names of the components C'' and C''', respectively $\Gamma(C'')$ and $\Gamma(C''')$, must be different by Definition 3.1 (page 57).

   (a) Consider the evaluation of $\Gamma(C'\|(C''+C'''))$. From Definition 4.4 (page 71) the idle event name of $C''+C'''$ is $\mathcal{N}(\Gamma(C'') \cup \Gamma(C'''))$. From Definition 4.9 (page 80) and by substitution, the idle event name of $\Gamma(C'\|(C''+C'''))$ is $\Gamma(C') \cup \mathcal{N}(\Gamma(C'') \cup \Gamma(C'''))$.

   (b) Now consider the evaluation of $\Gamma((C'\|C'')+(C''\|C'''))$. From Definition 4.9 the idle event names of $C'\|C''$ and $C''\|C'''$ are $\Gamma(C') \cup \Gamma(C'')$ and $\Gamma(C'') \cup \Gamma(C''')$ respectively. From Definition 4.4 and substitution, the idle event name of $(C'\|C'')+(C''\|C''')$ is $\mathcal{N}(\Gamma(C') \cup \Gamma(C'') \cup \Gamma(C'''))$.

5. Analysis of the distributive properties of the concurrent composition of the transition set $\hat{\Delta}(C'\|(C''+C'''))$ follows from substitution of the transition set $\hat{\Delta}(C''+C''')$ for the C term of the transition set $\hat{\Delta}(C'\|C)$. However, the definition of the concurrent composite transition set explicitly includes the concurrent composite event name set, the distributive properties of which were determined in case 3, above, to be limited.

   The definition of the concurrent composite transition set comprises the set union of terms 4.5, 4.6 and 4.7. For brevity, only the substitution in term 4.5 is described as the other substitutions follow in a similar way.

   (a) Term 4.5, repeated below in terms of C' and C, gives the set of $\mathcal{A}\Delta_{\Delta',(Q,\Gamma,Q)}$ asynchronous transitions.

   $$\{\, (Q' \cup Q, \Sigma' \cup \Gamma, P' \cup Q) \mid (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C) \wedge \Sigma' \cup \Gamma \in \hat{\Sigma}(C'\|C) \,\}$$

      i. The C contribution is an implied transition (Q, Γ, Q) that under substitution is an implied idle transition of a $C''+C'''$, denoted C^+. Hence;

      $$\{\, (Q' \cup Q, \Sigma' \cup \Gamma(C^+), P' \cup Q) \mid (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C^+) \wedge \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C'\|C^+) \,\}$$

ii. The state set $\hat{Q}(C^+)$, that is $\hat{Q}(C''+C''')$, is, by Definition 4.1 (page 69), $\hat{Q}(C'') \cup \hat{Q}(C''')$. Thus, $Q \in \hat{Q}(C^+)$ becomes $Q \in \hat{Q}(C'') \cup \hat{Q}(C''')$, which, from the definition of set union, see item 1b (page 210), can be written $Q \in \hat{Q}(C'') \vee Q \in \hat{Q}(C''')$. Hence;

$$\begin{aligned}&\{ (Q' \cup Q, \Sigma' \cup \Gamma(C^+), P' \cup Q) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge (Q \in \hat{Q}(C'') \vee Q \in \hat{Q}(C''')) \wedge \\ &\qquad \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C' \| C^+) \}\end{aligned}$$

which, by distribution of conjunction over disjunction, yields;

$$\begin{aligned}&\{ (Q' \cup Q, \Sigma' \cup \Gamma(C^+), P' \cup Q) \mid \\ &\qquad ((Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C'') \wedge \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C' \| C^+)) \vee \\ &\qquad ((Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C''') \wedge \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C' \| C^+))\}\end{aligned}$$

iii. From the definition of set union, disjunction can be written as set union;

$$\begin{aligned}&\{ (Q' \cup Q, \Sigma' \cup \Gamma(C^+), P' \cup Q) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C'') \wedge \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C' \| C^+)\} \\ &\bigcup \{ (Q' \cup Q, \Sigma' \cup \Gamma(C^+), P' \cup Q) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C''') \wedge \Sigma' \cup \Gamma(C^+) \in \hat{\Sigma}(C' \| C^+)\}\end{aligned}$$

iv. Now consider term 4.5 in terms of $(C' \| C'') + (C' \| C''')$, which gives the merge composition of the set of $\mathcal{A}\Delta_{\Delta',(Q'',\Gamma'',Q'')} \cup \mathcal{A}\Delta_{\Delta',(Q''',\Gamma''',Q''')}$ asynchronous transitions.

$$\begin{aligned}&\{ (Q' \cup Q, \Sigma' \cup \Gamma(C''), P' \cup Q) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C'') \wedge \Sigma' \cup \Gamma(C'') \in \hat{\Sigma}(C' \| C'') \} \\ &\bigcup \{ (Q' \cup Q, \Sigma' \cup \Gamma(C'''), P' \cup Q) \mid \\ &\qquad (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge Q \in \hat{Q}(C''') \wedge \Sigma' \cup \Gamma(C''') \in \hat{\Sigma}(C' \| C''') \}\end{aligned}$$

With the exception of the idle event names, the compositions $\hat{\Delta}(C' \| (C''+C'''))$ and $\hat{\Delta}(C' \| C'') + \hat{\Delta}(C' \| C''')$ are the same. Thus, the compositions can be said to be congruent as a consequence of the event name set, in other words, only for asynchronous components and because the idle event names in the distributed form are incorrect.

(b) Substitution in term 4.6 follows in a similar way, and, for the composition $\hat{\Delta}(C' \| (C''+C'''))$, yields;

$$\begin{aligned}&\{ (Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P) \mid \\ &\qquad Q' \in \hat{Q}(C') \wedge (Q, \Sigma, P) \in \hat{\Delta}(C'') \wedge \Gamma' \cup \Sigma \in \hat{\Sigma}(C' \| C^+)\} \\ &\bigcup \{ (Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P) \mid \\ &\qquad Q' \in \hat{Q}(C') \wedge (Q, \Sigma, P) \in \hat{\Delta}(C''') \wedge \Gamma' \cup \Sigma \in \hat{\Sigma}(C' \| C^+)\}\end{aligned}$$

and, for the composition $\hat{\Delta}(C' \| C'') + \hat{\Delta}(C' \| C''')$, yields;

$$\begin{aligned}&\{\,(Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P) \mid \\ &\qquad Q' \in \hat{Q}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C'') \wedge \Gamma' \cup \Sigma \in \hat{\Sigma}(C'\|C'')\} \\ &\bigcup\{\,(Q' \cup Q, \Gamma' \cup \Sigma, Q' \cup P) \mid \\ &\qquad Q' \in \hat{Q}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C''') \wedge \Gamma' \cup \Sigma \in \hat{\Sigma}(C'\|C''')\}\end{aligned}$$

(c) Substitution in term 4.7 follows in a similar way, and, for the composition $\hat{\Delta}(C'\|(C''+C'''))$, yields;

$$\begin{aligned}&\{\,(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P) \mid \\ &\qquad (Q',\Sigma',P') \in \hat{\Delta}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C'') \wedge \Sigma' \cup \Sigma \in \hat{\Sigma}(C'\|C^+)\} \\ &\bigcup\{\,(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P) \mid \\ &\qquad (Q',\Sigma,P') \in \hat{\Delta}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C''') \wedge \Sigma' \cup \Sigma \in \hat{\Sigma}(C'\|C^+)\}\end{aligned}$$

and, for the composition $\hat{\Delta}(C'\|C'') + \hat{\Delta}(C'\|C''')$, yields;

$$\begin{aligned}&\{\,(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P) \mid \\ &\qquad (Q',\Sigma',P') \in \hat{\Delta}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C'') \wedge \Sigma' \cup \Sigma \in \hat{\Sigma}(C'\|C'')\} \\ &\bigcup\{\,(Q' \cup Q, \Sigma' \cup \Sigma, P' \cup P) \mid \\ &\qquad (Q',\Sigma',P') \in \hat{Q}(C') \wedge (Q,\Sigma,P) \in \hat{\Delta}(C''') \wedge \Sigma' \cup \Sigma \in \hat{\Sigma}(C'\|C''')\}\end{aligned}$$

All three terms in the concurrent composition of the transition set depend upon the event name set, otherwise concurrent composition distributes over merge. In other words, distribution in the concurrent composition of transitions is limited only by the distribution in the concurrent composition of the event name set.

In general, concurrent composition does not distribute over merge composition. Congruence can be found only when the components are asynchronous.

B.3 Extraction

This section briefly examines the mathematical properties of the extraction operator which is defined in section 5.2 (page 105).

B.3.1 Commutivity

Consider the extract state set given in Definition 5.1 and repeated below;

$$\hat{Q}(\widetilde{C_{\|}} \lhd C') \stackrel{def}{=} \{\, Q' \mid \exists Q_{\|} \bullet \ (Q_{\|} \in \hat{Q}(\widetilde{C_{\|}}) \ \wedge \ Q' \in \hat{Q}(C') \ \wedge \ Q_{\|} \cap Q' \neq \{\}) \,\}$$

The extract state set comprises the elements Q' drawn from the state set of the right hand operand, $\hat{Q}(C')$, subject to the conjunction $Q_{\|} \cap Q' \neq \{\}$ which determines the *existence* of Q' in $Q_{||}$. Hence the state set of the expression $\hat{Q}(C' \lhd \widetilde{C_{||}})$ would also comprise elements from the right hand operand, that is, $Q_{||}$ drawn from $\hat{Q}(\widetilde{C_{||}})$. Except when $C_{||} = C'$, $\hat{Q}(\widetilde{C_{||}} \lhd C')$ will not yield the same set as $\hat{Q}(C' \lhd \widetilde{C_{||}})$, that is, this definition is not commutative. However, if the contribution of Q' is changed to $Q_{||} \cap Q'$, which is commutative, then the formation of the state set $\hat{Q}(\widetilde{C_{||}} \lhd C')$ can be re-written in the following commutative form;

$$\hat{Q}(\widetilde{C_{||}} \lhd C') \stackrel{def}{=} \{ Q_{||} \cap Q' \mid Q_{||} \in \hat{Q}(\widetilde{C_{||}}) \wedge Q' \in \hat{Q}(C') \wedge Q_{||} \cap Q' \neq \{\} \} \tag{B.2}$$

A similar modification can be made to the initial state set $\ddot{Q}(\widetilde{C_{||}} \lhd C')$, given by Definition 5.2 (page 106), and the event set $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$, given by Definition 5.3 (page 107). The definition of initial state set would then be commutative. However, the event set, given below in the modified form, would not be commutative because the transition extraction operation which generates $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$ will be seen later not to be commutative.

$$\begin{array}{rcl} \hat{\Sigma}(\widetilde{C_{||}} \lhd C') & \stackrel{def}{=} & \{ \Sigma_{||} \cap \Sigma' \mid \Sigma_{||} \in \hat{\Sigma}(\widetilde{C_{||}}) \wedge \Sigma' \in \hat{\Sigma}(C') \wedge \Sigma_{||} \cap \Sigma' \neq \{\} \} \\ & \bigcup & \{ \Sigma \mid \exists Q, P \bullet (Q, \Sigma, P) \in \hat{\Delta}(\widetilde{C_{||}} \lhd C') \} \end{array}$$

The extract idle event $\mathcal{N}(\Gamma(C'))$, from Definition 5.4 (page 107), is not commutative. However, as a new idle event, the definition could be changed to be a new idle event based on some commutative operation on $C_{||}$ and C', for example, $\mathcal{N}(\Gamma(C_{||}) \cup \Gamma(C'))$.

Now consider each of the contributions to the extract transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')$, given by Definition 5.10 (page 123).

1. The extract asynchronous transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$, repeated below, is defined in Definition 5.5 (page 110).

$$\begin{array}{rcl} \hat{\Delta}(\widetilde{C_{||}} \lhd C')_A & \stackrel{def}{=} & \{ (Q', \Sigma', P') \mid \exists\, Q_{||}, \Sigma_{||}, P_{||} \bullet (\\ & & \quad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ & & \quad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\ & & \quad \exists \mathcal{E}'' = \Sigma_{||} - \Sigma', \mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C') \bullet \mathcal{E}'' = \mathcal{I}'') \\ & & \} \end{array}$$

In a similar way to the state set, the existence tests for the *from* state Q', the event Σ' and the *to* state P', allows the formation of the transition (Q', Σ', P') to be replaced by the commutative form $(Q_{||} \cap Q', \Sigma_{||} \cap \Sigma', P_{||} \cap P')$. Despite this, the set difference operator is not commutative, thus the terms $\mathcal{E}'' = \Sigma_{||} - \Sigma'$ and $\mathcal{I}'' = \Gamma(\widetilde{C_{||}}) - \Gamma(C')$ are not commutative. Hence the operations $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_A$ and $\hat{\Delta}(C' \lhd \widetilde{C_{||}})_A$ will not generate the same contribution.

2. The extract synchronous transition set $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, which is defined in Definition 5.6 (page 112), can be made commutative by the formation of a transition of the form $(Q_{||} \cap Q', \Sigma_{||} \cap \Sigma', P_{||} \cap P')$, hence the definition would become;

$$\begin{array}{lcl} \hat{\Delta}(\widetilde{C_{||}} \lhd C')_S & \overset{def}{=} & \{ (Q_{||} \cap Q', \Sigma_{||} \cap \Sigma', P_{||} \cap P') \mid \\ & & \quad (Q_{||}, \Sigma_{||}, P_{||}) \in \hat{\Delta}(\widetilde{C_{||}}) \wedge (Q', \Sigma', P') \in \hat{\Delta}(C') \wedge \\ & & \quad (Q_{||} \cap Q' \neq \{\}) \wedge (\Sigma_{||} \cap \Sigma' \neq \{\}) \wedge (P_{||} \cap P' \neq \{\}) \wedge \\ & & \quad \Sigma_{||} = \Sigma') \\ & & \} \end{array}$$

3. The remaining three terms, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_P$ (page 116), $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DA}$ (page 118) and $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_{DC}$ (page 123) follow from case 1, above, and are not commutative.

The extraction operation is not commutative. Even with the modified definition of the terms of a Composite Transition System, commutivity is found only in the very limited case where the extract synchronous transition set is the only contribution to the extract transition set.

B.3.2 Associativity

Consider the revised term, term B.2, for the extract state set, but written in terms of the operands C and D', and in terms of the operands D and E.

$$\begin{array}{lcl} \hat{Q}(C \lhd D') & \overset{def}{=} & \{ Q_C \cap Q_{D'} \mid Q_C \in \hat{Q}(C) \wedge Q_{D'} \in \hat{Q}(D') \wedge Q_C \cap Q_{D'} \neq \{\} \} \\ \hat{Q}(D \lhd E) & \overset{def}{=} & \{ Q_D \cap Q_E \mid Q_D \in \hat{Q}(D) \wedge Q_E \in \hat{Q}(E) \wedge Q_D \cap Q_E \neq \{\} \} \end{array}$$

Substituting the extraction $D \lhd E$ for the D' term in the extraction $\hat{Q}(C \lhd D')$ gives the extraction $\hat{Q}(C \lhd (D \lhd E))$ which can be written as follows;

$$\begin{array}{l} \{ Q_C \cap (Q_D \cap Q_E) \mid Q_C \in \hat{Q}(C) \wedge \\ \quad Q_D \cap Q_E \in \{ Q_D \cap Q_E \mid Q_D \in \hat{Q}(D) \wedge Q_E \in \hat{Q}(E) \wedge Q_D \cap Q_E \neq \{\} \} \wedge \\ \quad Q_C \cap (Q_D \cap Q_E) \neq \{\} \} \end{array}$$

and which can be simplified to give;

$$\begin{array}{l} \{ Q_C \cap (Q_D \cap Q_E) \mid Q_C \in \hat{Q}(C) \wedge \\ \quad Q_D \in \hat{Q}(D) \wedge Q_E \in \hat{Q}(E) \wedge Q_D \cap Q_E \neq \{\} \wedge \\ \quad Q_C \cap (Q_D \cap Q_E) \neq \{\} \} \end{array}$$

Observe that the term $Q_D \cap Q_E \neq \{\}$ must hold true if the term $Q_C \cap (Q_D \cap Q_E) \neq \{\}$ is to hold true. Therefore, the term $Q_D \cap Q_E \neq \{\}$ is redundant and the extraction

$\hat{Q}(C \lhd (D \lhd E))$ can be written as follows. Similar steps for the extraction $\hat{Q}((C \lhd D) \lhd E)$ also lead to the following expression. Hence, the extract state set as defined by term B.2 (page 215) is associative.

$$\{ Q_C \cap Q_D \cap Q_E \mid Q_C \in \hat{Q}(C) \wedge Q_D \in \hat{Q}(D) \wedge Q_E \in \hat{Q}(E) \wedge \\ Q_C \cap Q_D \cap Q_E \neq \{\} \}$$

A similar approach to the definition of the extract initial state $\ddot{Q}(\widetilde{C_{||}} \lhd C')$ also leads to an associative operation. However, a similar approach to the event set $\hat{\Sigma}(\widetilde{C_{||}} \lhd C')$ does not lead to associativity because the event set depends upon the transition extraction operation, see Definition 5.3 (page 107), which will be seen later not to be associative.

The extract idle event $\mathcal{N}(\Gamma(C'))$ is not associative, see Definition 5.4 (page 107). However, the definition could be changed to follow that proposed in Appendix B.3.1, and elaborated to ensure that $\mathcal{N}(C \cup \mathcal{N}(D \cup E))$ generates the same new idle event name as $\mathcal{N}(\mathcal{N}(C \cup D) \cup E)$.

Appendix B.3.1 showed that with the exception of the extract synchronous transition set, $\hat{\Delta}(\widetilde{C_{||}} \lhd C')_S$, all other terms of the extract transition set cannot be commutative because set difference is not commutative. Similarly, the same terms of the extract transition set cannot be associative because set difference is not associative. Any set B can be defined by $B = \{x | x \in B\}$. The *complement* of any set B is denoted B' and is defined as $B' = \{x | x \notin B\}$, thus $A - B \overset{def}{=} A \cap B'$ [6, 52]. Hence, $X - (Y - Z)$ can be written as $X \cap (Y \cap Z')'$ which, by DeMorgan, can be written as $X \cap (Y' \cup Z'')$, and, by the complement laws, can be written as $X \cap (Y' \cup Z)$. Finally, by distribution, $X \cap (Y' \cup Z) = (X \cap Y') \cup (X \cap Z)$. Similar steps on the expression $(X - Y) - Z$ lead to $(X \cap Y') \cup (X \cap Z')$. Since $(X \cap Y') \cup (X \cap Z) \neq (X \cap Y') \cup (X \cap Z')$, set difference is not associative, and therefore extraction cannot be associative. The stated laws on the complement of a set can be found in [6] and [52].

B.3.3 Distribution

Distribution of extraction over merge is briefly discussed in this section. Consider the expression $(C + D) \lhd E = (C \lhd E) + (D \lhd E)$.

Where E incorporates state and event identifiers only present in C (for example) then $(C + D) \lhd E$ is congruent with $(C \lhd E)$. The extraction operation $D \lhd E$ generates a component comprising an empty state set, an empty initial state set, an empty event name set, and an empty transition set. The idle event name for $D \lhd E$ will take on the form $\mathcal{N}(\Gamma(E))$. Therefore, $D \lhd E = (\{\},\{\},\{\},\mathcal{N}(\Gamma(E)),\{\})$. The merge composition $(C \lhd E) + (D \lhd E)$ can be written as follows and, term by term, the merge composition can be formed;

$$\begin{array}{rl} & (\,\hat{Q}(C \lhd E),\, \ddot{Q}(C \lhd E),\, \hat{\Sigma}(C \lhd E),\, \Gamma(C \lhd E),\, \hat{\Delta}(C \lhd E)\,) + \\ & \quad (\,\{\},\, \{\},\, \{\},\, \mathcal{N}(\Gamma(E)),\, \{\}\,) \\ = & (\,\hat{Q}(C \lhd E) + \{\},\, \ddot{Q}(C \lhd E) + \{\}, \\ & \quad \hat{\Sigma}(C \lhd E) + \{\},\, \Gamma(C \lhd E) + \mathcal{N}(\Gamma(E)), \\ & \quad\quad \hat{\Delta}(C \lhd E) + \{\}\,) \\ = & (\,\hat{Q}(C \lhd E),\, \ddot{Q}(C \lhd E),\, \hat{\Sigma}(C \lhd E),\, \mathcal{N}(\Gamma(C \lhd E) \cup \Gamma(E)),\, \hat{\Delta}(C \lhd E)\,) \\ \equiv & (\,\hat{Q}(C \lhd E),\, \ddot{Q}(C \lhd E),\, \hat{\Sigma}(C \lhd E),\, \Gamma(C \lhd E),\, \hat{\Delta}(C \lhd E)\,) \end{array}$$

Now consider the case where E incorporates state and event identifiers present in both C and D. Distribution cannot be assumed because extraction depends upon the existence or absence of related transitions which may be contributed by C or D. In other words, the extraction $C \lhd E$ might lead to synchronisation because a transition in $C + D$ may be determined to be "absent" in considering only C, that is, the "absent" transition is contributed by D. Thus $(C + D) \lhd E \not\equiv (C \lhd E) + (D \lhd E)$, and hence distribution is denied.

If the component E can be partitioned such $E = E_C + E_D$, where E_C incorporates state and event identifiers related only to C, and E_D incorporates state and event identifiers related only to D, then $(C+D) \lhd (E_C+E_D)$ should be equivalent to $(C \lhd E_C) + (D \lhd E_D)$. More generally, any component C can be partitioned such that $C = C^0 + C^1 + \ldots + C^n$. Likewise, any concurrent composite $C_{||}$ can be partitioned into components combined by merge composition, that is, $C_{||} = C^0_{||} + C^1_{||} + \ldots + C^n_{||}$, where the partition $C^i_{||}$ comprises all those elements of $C_{||}$ related to the component partition C^i. Note that a composite partition $C^i_{||}$ may comprise elements of other partitions, $C^j_{||}$. Thus $C_{||} \lhd C$ can be written as follows and distributes as follows, although this form of distribution is not in accordance with *ring theory* [6];

$$\begin{array}{rcl} C_{||} \lhd C & = & (C^0_{||} + C^1_{||} + \ldots + C^n_{||}) \lhd (C^0 + C^1 + \ldots + C^n) \\ & = & (C^0_{||} \lhd C^0) + (C^1_{||} \lhd C^1) + \ldots + (C^n_{||} \lhd C^n) \end{array}$$

B.4 Mathematical Properties Summary

The mathematical properties of the merge composition, concurrent composition and extraction operators are summarised in table B.2. Idempotent laws, that is $C + C$, $C \Vert C$, and $C \lhd C$ have not been investigated, however the expected results are also summarised in table B.2.

Idempotent	$C+C=C$	
	$C \Vert C$	← Note 1
	$C \lhd C = C$	
Commutativity	$C'+C''=C''+C'$	
	$C' \Vert C'' = C'' \Vert C'$	
	$C_{\Vert} \lhd C' \neq C' \lhd C_{\Vert}$	
Associativity	$(C'+C'')+C'''=C'+(C''+C''')$	
	$(C' \Vert C'') \Vert C''' = C' \Vert (C'' \Vert C''')$	← Note 2
	$(C_{\Vert} \lhd C') \lhd C'' \neq C_{\Vert} \lhd (C' \lhd C'')$	
Distribution	$C' \Vert (C''+C''') \equiv (C' \Vert C'') + (C'' \Vert C''')$	← Note 3
	$(C_{\Vert} \lhd C') \lhd C'' \neq C_{\Vert} \lhd (C' \lhd C'')$	

Table B.2: Summary of Mathematical Properties.

Notes:

1. States common to the operands are denied, see page 74, hence $C \Vert C$ is a prohibited expression.
2. Only if there is synchronisation between no more than two of the operands.
3. Congruence can be found only when the components are asynchronous, otherwise distribution does not hold.

Bibliography

[1] Aczel, P. *Non-Well-Founded Sets.* CSLI lecture Notes, Number 14, ISBN 0-937073-22-9.

[2] Alhir S. *UML in a Nutshell.* O'Reilly, ISBN 1-56592-448-7, 1998.

[3] Alur R, Dill D. *Automata For Modelling Real-Time Systems.* Proceedings 17th International Colloquium on Automata, Languages and Programming 1990. Springer Verlag Lecture Notes in Computer Science 443, 1990, pages 322-335.

[4] Alur R, Henziger T. *Logics and Models of Real Time: A Survey.* Springer Verlag Lecture Notes in Computer Science 600, 1992, pages 74-106.

[5] ANSI/MIL-STD 1815A. *Reference Manual for the Ada Programming Language.* January 1983.

[6] Ayres F. *Modern Abstract Algebra.* McGraw-Hill 1965, ISBN 0-07-002655-6.

[7] Ben-Ari M. *Principles of Concurrent and Distributed Programming.* Prentice Hall International Series on Computer Science, 1990, ISBN 0-13-711821-X

[8] Benjamin M. *A Message Passing System. An example of combining CSP and Z.* 4th Z Users Workshop (ed Nichols JE), Workshops in Computing, Springer-Verlag 1989, pages 221-228.

[9] Booch G. *Software Engineering with Ada.* The Benjamin/Cummings Series in Ada and Software Engineering, 2nd Edition 1986, ISBN 0-8053-0604-8

[10] Burns A. *Scheduling hard real-time systems: a review.* Software Engineering Journal, May 1991, pages 116-128.

[11] Burns A, Wellings AJ. *Specifying an Ada tasking run-time support system.* Ada User, Volume 12, Number 4, December 1991, ISSN 0268-652X.

[12] Burns A, Wellings AJ. *Real-time Systems and Programming Languages.* Addison-Wesley User, 1997, ISBN 0-201-40365-X.

[13] Cattani GL, Sassone V. *Higher Dimensional Transition Systems.* IEEE Proceedings of the 11th Logic In Computer Science symposium, 27-30 July 1996, pages 55-62.

[14] Chartrand G, Oellerman OR. *Applied and Algorithmic Graph Theory.* McGraw-Hill 1993, ISBN 0-07-557101-3.

[15] Chen L. *An Interleaving Model for Real-Time Systems.* Logical Foundations of Computer Science. Springer Verlag Lecture Notes in Computer Science 620, 1992, pages 81-92.

[16] Cleaveland R, Henessy M. *Priorities in Process Algebras.* IEEE Proceedings of the 3rd Logic In Computer Science symposium 1988, pages 193-202.

[17] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. *Deriving Petri Nets from Finite Transitions Systems.* IEEE Transactions on Computers, Volume 47, Number 8, August 1998, pages 859-882.

[18] Davies J, Scheider S. *An Introduction to Timed CSP.* Technical Monograph PRG-75, Oxford University Computing Laboratory, Programming Research Group, 1989, ISBN 0-902928-57-0.

[19] Djordjevic GLJ, Tosic MB. *A Compile-Time Scheduling Heuristic for Multiprocessor Architectures.* The Computer Journal, Volume 39, Number 8, pages 663-674.

[20] Droste M, *Concurrency, Automata and Domains.* Proceedings 17th International Colloquium on Automata, Languages and Programming 1990. Springer Verlag Lecture Notes in Computer Science 443, 1990, pages 195-208.

[21] Duke R, Smith G. *Temporal Logic and Z Specifications.* The Australian Computer Journal, Volume 21, Number 2, May 1990, pages 62-66.

[22] Duri S, Buy U, Devarapalli R, Shatz SM. *Application and Experimental Evaluation of State Space Reduction Methods for Deadlock Analysis in Ada.* ACM Transactions on Software Engineering and Methodology, Volume 3, Number 4, October 1994, pages 340-380.

[23] Fidge CJ. *A Formal Definition of Priority in CSP.* ACM Transactions on Programming Languages and Systems, Volume 15, Number 4, September 1993, pages 681-705.

[24] Fowler M, Scott K. *UML Distilled.* Addison-Wesley, 1997, ISBN 0-201-32563-2.

[25] Gerber R, Lee I. *Communicating Shared Resources: A Model for Distributed Real-Time Systems.* Proceedings IEEE Real-Time Systems Symposium, December 1989, pages 67-78.

[26] Gerber R, Lee I. *CCSR: A Calculus for Communicating Shared Resources.* Proceedings CONCUR '90, Springer Verlag Lecture Notes in Computer Science 458, 1990, pages 263-276.

[27] Gerber R, Lee I. *A Proof System for Communicating Shared Resources.* Proceedings of the 11th Real-Time Systems Symposium, 5-7 December 1990, pages 288-299.

[28] Gerber R, Lee I. *Specification and Analysis of resource-Bound Real-Time Systems.* Real Time:Theory in Practice, 1991. Springer Verlag Lecture Notes in Computer Science 600, 1992, pages 371-396.

[29] Gibbons A. *Algorithmic Graph Theory.* Cambridge University Press, 1994, ISBN 0-521-28881-9.

[30] Green JA. *Sets and Groups.* Rourledge & Kegan Paul, 1978, ISBN 0-7100-4356-2.

[31] Gries D. *Compiler Construction for Digital Computers.* Wiley International Edition, 1971, ISBN 0-471-32771-9.

[32] Hale R. *Temporal Logic Programming.* Temporal Logics and their Applications. Academic Press Limited, 1987, ISBN 0-12-274060-2, pages 81-120.

[33] Hendry DC. *Heterogeneous Petri net methodology for the design of complex controllers.* IEE Proceedings on Computer Digital Technology, Volume 141, Number 5, September 1994, pages 293-297.

[34] Henziger TA, Manna Z, Pneuli A. *Timed Transition Systems.* Real Time: Theory in Practice, 1991. Springer Verlag Lecture Notes in Computer Science 600, 1992, pages 226-251.

[35] Hesham ER, Hesham HA, Lewis T. *Task Scheduling in Multiprocessing Systems.* IEEE Computer, Volume 28, Number 12, December 1995, pages 27-37.

[36] Hoare CAR. *Communicating Sequential Processes.* Communications of the ACM, Volume 21, Number 8, August 1978, pages 666-677.

[37] Hoare CAR. *Communicating Sequential Processes.* Prentice Hall International Series on Computer Science, 1985, ISBN 0-13-153289-8

[38] Hopcroft JE, Ullman JD. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979, ISBN 0-201-02988-X.

[39] Ince DC. *An Introduction to Discrete Mathematics, Formal System Specification, and Z.* Oxford Applied Mathematics and Computing Series, 1992.

[40] INMOS Limited. occam2 *Reference Manual.* Prentice Hall International Series in Computer Science, 1988.

[41] ISO 8807:1989. *Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour.* International Standards Organisation, 1989.

[42] Jackel M. *ADA Concurrency Specified by Graph Grammars.* Graph Theoretic Concepts in Computer Science, International Workshop 1996. Springer Verlag Lecture Notes in Computer Science 246, 1996, pages 41-57.

[43] Jain R, Somalwar K, Werth J, Browne JC. *Heuristics for Scheduling I/O Operations.* IEEE Transactions on Parallel and Distributed Systems, Volume 8, Number 3, March 1997, pages 310-320.

[44] Juan E, Tsai J, Murata T, Zhou Y. *Reduction Methods for Real-Time Systems Using Delay Time Petri Nets.* IEEE Transactions on Software Engineering, Volume 27, Number 5, May 2001, pages 422-427.

[45] Khendek F, Bochmann GV. *Merging Behaviour Specifications.* Formal Methods in System Design, Volume 6, 1995, pages 259-293.

[46] Kröger F. *Temporal Logic of Programs.* EATCS Monographs on Theoretical Computer Science, Volume 8, Springer-Verlag, 1987, ISBN 0-387-17030-8.

[47] Kurki-Suonio R. *Real Time: Further Misconceptions (or Half-Truths).* IEEE Computer, June 1994, pages 71-76.

[48] Lamport L. *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, Volume 21, Number 7, July 1978, pages 558-565.

[49] Lamport L. *Temporal Logic of Actions.* ACM Transactions on Programming Languages and Systems, Volume 16, Number 3, May 1994, pages 872-923.

[50] Laplante PA. *Real-Time Systems Design and Analysis (second edition).* IEEE Press, 1997, ISBN 0-7803-3400-0.

[51] Lee I, Brémond-Grégiore P, Gerber R. *A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems.* Proceedings IEEE, Volume 82, Number 1, January 1994, pages 158-171.

[52] Lipschutz S. *Set Theory and Related Topics.* McGraw-Hill 1964, ISBN 0-07-037986-6.

[53] Lowe G. *Probabilities and Priorities in Timed CSP.* PhD Thesis, St. Hugh's College, Oxford. 1993.

[54] Lynn PA. *An Introduction to the Analysis and Processing of Signals.* MacMillan, 1980, ISBN 0-330-14353-1.

[55] Magee J, Kramer J. *Concurrency, State Models and Java programs.* John Wiley & Sons, 1999, ISBN 0-471-98710-7.

[56] Mathai J, Goswani A. *What's 'Real' about Real-time Systems.* IEEE, 1988, pages 78-85.

[57] Milner R. *Communication and Concurrency.* Prentice Hall International Series on Computer Science, 1989, ISBN 0-13-115007-3.

[58] Milner R. *The Polyadic π-Calculus: a Tutorial.* Laboratory for Foundations of Computer Science, Computer Science Department, University of Edinburgh, October 1991.

[59] Moszkowski B. *Executing temporal logic programs.* Cambridge University Press, 1987, ISBN 0-521-31099-7.

[60] Olariu S, Zomaya AY. *A Time- and Cost-Optimal Algorithm for Interlocking Sets - With Applications.* IEEE Transactions on Parallel and Distributed Systems, October 1996, pages 1009-1025.

[61] Ostroff JS. *Temporal Logic for Real-Time Systems.* Research Studies Press, 1989, ISBN 0 86380 086 6 (Wiley Inc. 0471 92402 4).

[62] Papelis YE, Casavant TL A. *Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets with Transition Enabling Functions.* IEEE Transactions on Software Engineering, Volume 18, Number 3, March 1992, pages 252-261.

[63] Peng W, Purushothaman S. *Analysis of Communicating Processes for Non-progress.* Proceedings of the 9th International Conference on Distributed Computing Systems, 5-9 June 1989, pages 280-287.

[64] Pengelly AD. *The Application of Graph Theory to the Synthesis of Protocol Converters via the Interface Equation.* Ph.D. Thesis, Open University, 1995.

[65] Pengelly AD, Ince DC. *Quotient Machines, the Interface Equation and Protocol Conversion.* The Computer Journal, Volume 43, Number 1, January 2000, pages 24-39.

[66] Peterson JL. *Petri Nets.* Computing Surveys, Volume 9, Number 3, September 1977, pages 223-252.

[67] Peterson JL, *Petri Net Theory theory and the Modelling of Systems.* Prentice-Hall, 1981.

[68] Peyravian M, Lea CT. *An Algorithmic Method for Protocol Conversion.* Proceedings of the 12th International Conference on Distributed Computing Systems, 9-12 June 1992, pages 323-335.

[69] Pilling M, Burns A, Raymond K. *Formal specifications and proofs of inheritance protocols for real-time scheduling.* IEE Software Engineering Journal, September 1990, pages 263-279.

[70] Pountain D. *A tutorial introduction to OCCAM programming.* INMOS Ltd., 1987.

[71] Quine WVO. *Elementary Logic.* Harvard University Press, 1980, ISBN 0-674-24451-6.

[72] Ramamritham R and Stankovic JA. *Scheduling Algorithms and Operating Systems Support for Real-Time Systems.* IEEE Proceedings, Volume 82, Number 1, January 1994, pages 55-67.

[73] Rao NSV. *On Parallel Algorithms for Single-Fault Diagnosis in Fault Propagation Graph Systems.* IEEE Transactions on Parallel and Distributed Systems, Volume 7, Number 12, December 1996, pages 1217-1223.

[74] Reed D. *Examples of Using Timed-CSP for modelling Hardware.* Faculty of Technology, System Architecture Group, Open University report SAG/1990/RR5/DJR.

[75] Reed D. *Using Cadiz for Specifying and Verifying Hardware.* Faculty of Technology, System Architecture Group, Open University report SAG/1992/RR22/DJR.

[76] Reed GM, Roscoe AW. *A Timed Model for Communicating Sequential Processes.* Springer-Verlag Lecture Notes in Computer Science 226, 1986, pages 314-323.

[77] Schwan K, Zhou Hongyi. *Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads.* IEEE Transactions on Software Engineering, Volume 18, Number 8, August 1992, pages 736-748.

[78] Sekerinski E, Sere K. *A Theory of Prioritizing Computation.* The Computer Journal, Volume 39, Number 8, 1996, pages 701-712.

[79] Selvakumur S, Siva Ram Murthy C. *Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors.* IEEE Transactions on Parallel and Distributed Systems, Volume 5, Number 3, March 1994, pages 328-336.

[80] Shatz SM, Tu S, Murata T, Duri S. *An Application of Petri Net Reduction for Ada Task Deadlock Analysis.* IEEE Transactions on Parallel and Distributed Systems, Volume 7, Number 12, December 1996, pages 1307-1322.

[81] Shaw AC. *Communicating Real-Time State Machines.* IEEE Transactions on Software Engineering, September 1992, pages 805-816.

[82] Shin KG. *Real-Time Computing: A New Discipline of Computer Science and Engineering.* Proceedings of the IEEE, Volume 82, Number 1, January 1994, pages 6-23.

[83] Sih GC, Lee EA. *A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures.* IEEE Transactions on Parallel and Distributed Systems, Volume 4, Number 2, February 1993, pages 175-187.

[84] Sisto R, Valenzano A. *Mapping Petri Nets with Inhibitor Arcs onto Basic LOTOS Behaviour Expressions.* IEEE Transactions on Computers, Volume 6, Number 12, December 1995, pages 1341-1370.

[85] Smart RJ. *A Generalised Description of Ordered Events and its Application to the Prediction of Resource Constraints.* Open University research degree paper, 1996.

[86] Spivey JM. *The Z notation: A Reference Manual.* Prentice Hall International Series on Computer Science, 1989.

[87] Spivey JM. *Specifying a Real-Time Kernel.* IEEE Software, September 1990, pages 21-28.

[88] Spivey JM. *An introduction to Z and formal specifications.* IEE Software Engineering Journal, Volume 4, Number 1, January 1991, pages 40-50.

[89] Stankovic JA, Ramamritham K. *The Design of the Spring Kernel.* Proceedings of the IEEE Real-Time Systems Symposium, December 1987, pages 146-155.

[90] Stankovic JA. *A Serious Problem for Next-Generation Systems.* IEEE Computer, October 1988, pages 10-19.

[91] Stark EW. *Concurrent Transition Systems.* Theoretical Computer Science, Number 64, 1989, pages 221-269.

[92] Stoyenko AD and Baker TP. *Real-Time Schedulability-Analyzable Mechanisms in Ada9X.* IEEE Proceedings, Volume 82, Number 1, January 1994, pages 95-107.

[93] Stremler FG. *Introduction to Communication Systems.* Addison Wesley, 1976, ISBN 0-201-07244.

[94] Turski WM. *Time Considered Irrelevant for Real-Time Systems.* Bit, Volume 82, Number 3, 1988, pages 473-486.

[95] Varea M, Al-Hashimi B. *Dual Transition Petri Net based Modelling Technique for Embedded Systems Specification.* Design, Automation and Test in Europe 2001. IEEE Conference and Exhibition Proceedings, March 2001, pages 566-571.

[96] Wedde H. *A Graph-Theoretic Approach for Designing Fair Distributed Resource Scheduling Algorithms.* Graph Theoretic Concepts in Computer Science, International Workshop 1996. Springer Verlag Lecture Notes in Computer Science 246, 1996, pages 204-226.

[97] Wolf WH. *Hardware-Software Co-Design for Embedded Systems.* Proceedings of the IEEE, Volume 82, Number 7, July 1994, pages 967-989.

[98] Wolfram S. *The Mathematica Book.* Cambridge University Press, 1996, ISBN 0-521-58889-8.

[99] Wu MY. *On Runtime Parallel Scheduling for Processor Load Balancing.* IEEE Transactions on Parallel and Distributed Systems, February 1997, pages 173-186.

[100] Young S. *Real Time Languages: Design and Development.* Ellis Horwood, 1992.