TIME CAUSALITY AND CONCURRENCY

DAVID MURPHY
Time, Causality, and Concurrency

A thesis submitted for the degree of Doctor of Philosophy

For Elly

David V. J. Murphy
Department of Electrical and Electronic Engineering,
University of Surrey

© David V. J. Murphy, 18th December 1989
Prelude – Abstract

There is a considerable practical and theoretical interest in real-timed concurrent systems. Previous attempts at building theories of such systems have imposed time post hoc onto an extant model: we will attempt to start from scratch.

The fundamental things of interest to us will be occurrences of events. Each of these occurrences will have an associated duration. Hence, each occurrence of an event will have a start and a finish. Causality will be modelled by giving a partial order to these starts and finishes, while timing will be modelled by associating each of them with a real number. The causal part of the model will thus be like Winskel’s event structures, (especially once we add conflict), while the timed part will associate an interval of the reals with each occurrence of an event; the model is therefore termed the interval event structure model.

Various categories are defined whose objects are interval event structures and whose morphisms indicate when one interval event structure can simulate another. Operations such as nondeterministic and parallel composition of interval event structures are defined as limits in these categories.

A semantics is given to the model by considering when bets placed on an interval event structure about its subsequent behaviour may and must win.

The interval event structure model suggest a natural timed process algebra, interval process algebra. This algebra is developed and given both denotational, axiomatic and operational semantics. The denotational and axiomatic semantics are proved equivalent. Some consideration is given to the relationship between various ‘truly concurrent’ operational techniques, our own included.

Finally, notions of event refinement and abstraction for interval event structures are proposed.
Contents

Prelude — Abstract

0 Introduction: Motivations and Justifications

0.0 A short intellectual history of concurrency ................................................... 7
0.1 A taxonomy of untimed models ................................................................. 10
0.2 Timed concurrency theories .................................................................... 23
0.3 On the nature of a possible concurrency theory ............................................ 27
0.4 A model proposed ......................................................................................... 29

Interlude — Personal Remarks

1 Definitions: Linear and Branching Interval Event Structures

1.0 Linear and Branching Time .......................................................................... 33
1.1 Structures in Time ....................................................................................... 40
1.2 Orders on Labelled Events ............................................................................ 47
1.3 Primitive Concurrency .................................................................................. 48
1.4 Specification and Inhibition ............................................................................ 49
1.5 Safety and j–morphism ............................................................................... 57
1.6 Observing interval event structures .............................................................. 59

Interlude — Category Theory and Computer Science

2 Operations on Structures: Synchronisation and Morphism

2.0 Prefixing ........................................................................................................ 67
2.1 Morphisms .................................................................................................... 69
2.2 The disjoint sum ............................................................................................. 72
2.3 Sequential composition ............................................................................... 74
### 2.4 Synchronisation ................................................................. 77
### 2.5 The free product ............................................................... 79
### 2.6 Synchronising structures .................................................. 85
### 2.7 Parallel composition ......................................................... 88
### 2.8 Coequalisers and Initial objects ........................................ 91
### 2.9 Concluding remarks and reformulations ............................ 93

**Interlude — Categories of computations**

### 3 Meanings of Structures: Semantics and Interpretation

#### 3.0 Observing interval event structures again .......................... 103
#### 3.1 Betting on transitions .................................................... 107
#### 3.2 Characterising structures using bets ................................. 114
#### 3.3 Causal bets .................................................................... 125
#### 3.4 Bets and equivalence ...................................................... 130
#### 3.5 An introduction to metric space semantics ....................... 131
#### 3.6 Metric spaces and bets ................................................... 137

**Interlude — Categories of behaviours**

### 4 An algebraic entelech: Interval Process Algebra

#### 4.0 IPA — A timed process algebra ...................................... 151
#### 4.1 An introduction to interval process algebra ....................... 153
#### 4.2 The axiomatic semantics of interval process algebra .......... 167
#### 4.3 The denotational semantics of interval process algebra ....... 169
#### 4.4 The operational semantics of interval process algebra ........ 173
#### 4.5 Aside — Asynchronous Operational Models ..................... 182
#### 4.6 The relationship between the semantics of IPA ................. 188

**Interlude — External choice and Recursion**
5 Manipulating Structures: Abstraction and Refinement

5.0 Event refinement ................................................................. 205
5.1 Substructure abstraction ..................................................... 209

Postlude — Conclusions and Further Work

Bibliography
0
Introduction:
Motivations and Justifications

"Light thickens, and the crow
Makes wing to th'rocky wood.
Good things of day begin to droop and drowse,
While night's black agents to their prey do rouse.
Thou marvell'st at my words; but hold thee still.
Things bad begun make strong themselves by ill
So prithee go with me.

Shakespeare

This thesis is about a model of concurrency incorporating time and causality. Concurrency theories with these concerns are not new, but there are various ways to build such a theory, and the one adopted here is, to the best of my knowledge, unique.

We shall be concerned with putting time on an equal footing with causality, rather than merely adding it to an extant model. This is because the order on time is just as important as the (causal) order on happenings, and we wish to investigate the interaction of these two orders. In particular, we want to discover new insights about concurrency and nondeterminism through our study of time.

The thesis is composed of chapters separated by interludes; this is the first chapter; it is introductory, being concerned with the general problem area addressed in this thesis. This chapter is also meant to give some necessary background, so that the ideas motivating the model announced above can be understood. It is divided into five main sections. In the first, the general problem area addressed is explained, and its history elucidated. We shall be concerned with concurrency theories and, in particular, with the rôle of time and causality in concurrency theories. The second section provides a taxonomy of currently-popular untimed concurrency theories, and a survey of methods. This section is rather long, as there is much work to examine. The third section does a similar job for timed concurrency theories, and describes why timing is useful.

The fourth section is more philosophical; it describes the ontological position taken here on time, concurrency and causality. In the final section, some decisions are made about the kind of theory that will be developed in later chapters, and the origin of these decisions is explained. In the interlude which separates this chapter for the next my acknowledgements are made and the structure of the rest of the thesis is explained.
0.0 A short intellectual history of concurrency

Here we will give a brief and highly personal outline of some developments in concurrency theory: concurrency theory started, in computer science at least, as an attempt to provide a formal technique for reasoning about distributed computer systems. The earliest work recognised today, such as [Petri 1962] and [Kahn 1974], concentrated on describing the flow of information around a distributed system and, hence, the relationship between happenings in the system. This 'dataflow' approach turned out to be very rich, — witness the bulk of the Petri net literature and the subtle problems associated with Kahn processes [Brock & Ackerman 1983].

The ultimate aim of this research could be seen as the description of the behaviour of distributed systems; all behavioural features should be correctly described, no spurious features should be introduced, and it should be possible to reason about behaviours. A central strand in this work is the treatment of causality; in a distributed system it is important to know what causes what, since this tells us what must be accessible to what, and hence gives information about acceptable implementations. Clearly not all happenings are causally related, so we can represent causality as a poset; elements of the poset are happenings, and $a \leq b$ just when $a$ causes $b$.

These 'dataflow' theories can be seen as generalisations of automata theory. A single automata engages in its happenings sequentially, but several distributed connected automata possess the potential of concurrent, cooperative behaviour.

A major advance in this line of work was made by Winskel, who was the first to explain that happenings (which he calls 'events') and their causality could be understood almost independently of the state changes that exhibit them. In [Winskel 1980] the event structure model was presented; this model, while closely related to Petri nets, has brought different issues into focus, and has propitiated the comparison of concurrency theories, (of which more later).

The phrase 'true concurrency' has been coined for the work of people like Petri and Winskel. To understand this phrase, we need to understand nondeterminism. Concurrency is intuitively about unrelated things happening, — they might happen in different places, or simultaneously, but the key point about two things being concurrent is that they are causally unrelated. Concurrency, then, is a property of happenings. Conflict is also a property of happenings: the idea is that $a$ and $b$ are in conflict if they can't both happen. Thus reading this thesis prevents you from simultaneously reading Oliver Twist (unless you are rather unusual), while Nancy’s untimely death prevented something ineffably twee happening to her. Conflict is a useful notion when modelling competition for resources, a situation that often occurs in distributed systems. If we don’t care how conflict is resolved, (— I might
either have some chocolate for breakfast, or some Christmas pudding, but not, of course, both, and I don’t mind which,) or we don’t know, then nondeterminism results. The system can do a, or it can do b, but not both, and there is no a priori means of telling which. Systems with conflict, then, can display different behaviour on different executions; they have more than one possible history.

Another strand in concurrency theory sprang from the attempt not just to describe distributed systems, but to write correct programs for them. Generalised automata theories were suitable for reasoning about systems with distributed state, but they were rather monolithic. Programmers needed abstraction mechanisms in order to be able to describe structural as well as behavioural features. The theory of process algebras sought to provide these mechanisms; here the central notion was of autonomous processes, communicating by synchronising and passing data. Process algebras were usually compositional; large processes could be constructed from small ones using process combinators representing the sequential combination of processes, the parallel combination of processes, the nondeterministic choice between processes and so on. Moreover, the meaning of a compound process could be deduced from the meanings of its-components. Process algebra is algebraic; there are laws for reasoning about processes which reflect the nature of the combinators.

Process algebras meet a certain need, and are beautifully simple to use. However, initially at least, they relied on a suspension of disbelief in the user, as they did not have a notion of concurrency. The concurrent execution of a and b was modelled by the nondeterministic choice between the two sequentialisations, ab and ba. Not only are complex events reduced to atomic happenings, but also concurrency is reduced to interleaving. The idea was that either a happened first and then b happened, or vice versa, and since a and b were concurrent, the order was not decidable a priori. Thus representing a happening by a single (notional) instant in time, and adopting interleaving, leads to a model without a primitive notion of concurrency. Hence the origin of the phrase ‘true concurrency’ for the causality-as-a-poset tradition; here concurrency is a primitive notion, not derivable from nondeterminism.

In the next section some models derived from these two traditions will be discussed and compared. Before moving on, we will outline a few minority traditions.

- The design of hardware has always been seen by concurrency theorists as an exercise in their discipline. Only recently, however, has concurrency theory been able to deal tractably with the complexity and behavioural subtlety of hardware. I expect that more mainstream concurrency models will be used for hardware design in the future, rather than the somewhat parochial hardware-specific models used in the past. This is the most technologically interesting area where physical concurrency (see below) must be considered.
Logicians have provided a convenient means of specifying distributed systems by developing temporal logics. In their more advanced forms, these enable distributed states and their causality to be specified and reasoned about. We will not treat them in any detail, as they do not seem suited to discussing the level of behavioural detail we are interested in; they usually lack convenient operational means of structuring systems and of describing the behavioural subtleties of partial order models. (Work is underway to alleviate these deficiencies; we may have cause to change our opinion in due course.)
0.1 A taxonomy of untimed concurrency theories

This section is concerned with models of concurrency. Some popular models will be examined to see where they are successful and where they are more limiting. This will be done within the framework of a taxonomy; ours is based on that of [Pnueli 1986]. (Similar work can be found in [Reisig 1989].) A classification is attempted because there is a growing feeling that

* there are too many concurrency theories, and
* all of them have serious limitations.

For instance, although the trace semantics of CSP is easy to understand, it is difficult to use as a specification tool. Similarly, systems specification can be simple in a modal logic, but such logics often cannot describe the behaviours the 'true concurrency' school find interesting. By classifying a model we can, perhaps, see which features of it contribute to its advantages and disadvantages.

It would be nice to be able to talk about behaviours without worrying about how those behaviours are represented. This jump up to a higher level of abstraction is not possible yet, so we will have to be content with trying to discover what questions about behaviours we are interested in answering, and with formalising our notions of what a behaviour is, so that we can ask and answer those questions efficiently. A classification of models of concurrency in terms of how they deal with behaviours might, therefore, be a useful one.

One distinction to be made is that between functional and reactive systems. A functional system is one whose interaction with the environment is best viewed as a function. Such systems take some input, produce some output and then stop. Many sequential programs are of this form. On the other hand, a reactive system is one whose interaction with the environment is much less directional. Such systems should often, in the ideal case, run for ever; examples include operating systems and process control applications. We are essentially concerned with the behaviour of reactive systems. (This classification by rôle seems more appropriate than the sequential/concurrent classification, which is one by implementation.)

0.1.1 Physical and Abstract Concurrency

Another important distinction is that between physical and abstract concurrency. Abstract concurrency is the form usually studied in theoretical computer science, as it is the form which is displayed by common applications, clocked digital systems, transaction processing systems and so on; it can usually be simulated on a sequential (possibly nondeterministic) machine with a countable number of states, as it only involves at most a countable number of interactions between concurrent components. The title of [Hoare 1985] describes abstract concurrency perfectly, – 'communicating sequential processes.'
However, most of physical reality displays a different form of concurrency. This form, which we call 'physical concurrency' is that displayed by analogue systems (and digital ones if you clock them fast enough, but that is another story), biological systems and so on. It involves the wholly asynchronous exchange of real-valued signals between autonomous components, so a countable infinity of happenings is inadequate for describing it.

It is clear that certain features of physical concurrency (such as synchronisation failure, [Barros & Johnson 1983], [Marino 1981]) and chaos ([Deane et al. 1989], [Schuster 1984]) are not accessible to modelling by discrete calculi. Their description seems to rely crucially upon real-valued time. Yet, despite their descriptive power, there are considerable problems with the use of continuous calculi in concurrency theory viz:

- there is no accepted method of presenting an analysis in these calculi. In particular, there is little notion of precisely what is necessary in order to prove something about a system. Further, continuous calculi are often not axiomatised, leading to worries about nonstandard models. If proofs in continuous calculi are to be comparable in rigour with those in discrete ones, much work in necessary to ensure that it is clear what is being proved about what system. Even careful work such as [Mendler 1987] can suffer from the disadvantage that it is not always clear precisely what intuitions are being exploited.

- continuous calculi have little analytical power. In a discrete calculus such as CSP, one can state just what it means for two processes to be the same, one can combine processes in several ways, and one can prove properties true of processes. CSP processes are simple, not very descriptively rich mathematical objects, but they are related to many other mathematical objects (domains, cpos, categories, topological spaces, synchronisation trees) in ways that give some insight into their nature and possible behaviour. Their theory can be said to have analytical power because of its various semantics (which relate CSP objects to other objects) and because of the transparency of composition. (Objects can be analysed via their component parts and the analyses composed just as the parts are composed.) In contrast it cannot be said that it is clear, for instance, what the semantics of general circuit theory is, or what a natural calculus of composition for asynchronous circuits is like (although some progress has been made in this direction; cf. [Cardelli 1982]).

- there are difficulties in accepting the richness of $\mathbb{R}$. The reals are not only rich enough to describe certain physically observed pathologies of behaviour, they are also rich enough to describe many unphysical ones (such as the function that takes the value 0 on the rationals and 1 on the irrationals). This criticism applies in part to $\mathbb{Q}$ and $\mathbb{N}$ as well, since there are uncomputable functions over these domains too.

The problem is this; continuous calculi are complex enough to describe physical concurrency, but are too complex to enable those descriptions to be easily manipulated and reasoned about. Discrete calculi cannot describe some phenomena of interest, but they do possess considerable power, with concomitant insights.
Further, many of the properties of concurrent systems that are traditionally of interest (such as deadlock, divergence, reachability, satisfaction of specification, refinement and so on) are easily and naturally formulated in a discrete framework. Simplistically we might say:

*uncountable objects seem to be good for describing behaviour,
but they are harder to reason with.*

One eventual aim of concurrency theory, then, is to be able to provide a tractable theory of physical concurrency. Physical concurrency, after all, is not only an interesting problem because it exists; it also legitimises all work on abstract concurrency (much as Maxwellian electrodynamics legitimises conventional circuit theory).

Although we will not deal extensively with physical concurrency, a concern for the necessity of modelling it motivates much of this thesis. These matters are further discussed in section 0.3.

After considering the problem of physical concurrency, and presenting some ideas about mathematics that might be useful to describe it, [Petri 1986] says:

"This justifies the hope that all implementable signalling structures, discrete as well as continuous (digital as well as analogue) may be completely described in terms of concurrency, using combinatorial mathematics only."

I am not this enthusiastic, but I do feel that we are beginning to understand concurrency well enough to be able to apply it to this level. The table overleaf summarises rather glibly some of this distinctions made in this subsection: after it we continue describing some features relevant to the classification of models of abstract concurrency.
PHYSICAL CONCURRENCY

REALITY
Physical Reality (including the Proof of correctness of all theories of abstract concurrency and true neural computers)

ALL CONTINUOUS SYSTEMS

MODELS
- partial differential equations of physics

ADVANTAGES
- great predictive power
- verisimilitude

DISADVANTAGES
- complex, with few deep concepts
- little analytical power

ABSTRACT CONCURRENCY

* TRANSACTION PROCESSING CICS
* SERIALISABLE COMPUTING occam, Mejje, Ada
* DIGITAL HARDWARE (hopefully)

* PROCESS CALCULI CSP, ACP, CCS
* (Discrete) TEMPORAL LOGICS ITL, CTL*
* TRANSITION SYSTEMS Petri Nets and so on

ADVANTAGES
- great analytical power
- quite simple

DISADVANTAGES
- not how most of the world works
- little analytical power

A summary of some of the differences between physical and abstract concurrency
0.1.2 The available theories

Commonly studied properties of models of abstract reactive systems, — concurrency, non-determinism, hiding, confusion, fairness, liveness, reachability, safety, guarantee, persistence, recurrence and so on, — are valuable starting points in analysing behaviours. One should obviously be concerned with the expressiveness of a model; does it capture the behaviour of interest? (Cf. [Wolper 1986].) Is it sufficiently expressive both constructively (writing the processes we want) and prescriptively (writing the specifications we want)? Notice that there is a trade-off between the expressive power of a model and the complexity of states within it.

Some insight into five of these issues can be gained from modal logic by using the modal quantifiers $\diamondsuit$ and $\Box$, corresponding to eventually and always; the safety of a proposition $p$ corresponds to $\Box p$, guarantee to $\diamondsuit p$, recurrence to $\Box \diamondsuit p$, persistence to $\diamondsuit \Box p$ and fairness to $\diamondsuit \Box p \lor \Box \diamondsuit p$; this point is discussed in [Pnueli 1986ii]. These relationships can be visualised:

![Diagram](image)

(Aside: The classification of various requirements has been undertaken by [Kwiatkowska 1989]. She shows that various kinds of properties correspond to various notions of open set in topologies generated by the executions of a concurrent system.)

There are other issues that should be considered, en passant, when evaluating a model:

* How tractable is it; can behavioural descriptions be manipulated easily? Some models can describe many behaviours but are hideously intractable. The main work on tractability and complexity is American; [Mairson 1987] offers a taste of this field for the CSP hacker; [Clarke 1981] does the same for the temporal logician. See also [Harel et al. 1982].

† — We shall use the term ‘model’ quite loosely, using it for either a syntax with some (or many) semantics, or for a semantic technique. I shall also use the word ‘process’ for any object intended as a model of (a part of) a computational system.
* Is the model related to other models? Various individual relationships are explored in [Astesiano 1985], [Best 1986], [Degano et al. 1987i–iii], [van Glabbeek & Vaandrager 1987], [Goltz 1988], [Goltz & Reisig 1983], [Manna & Wolper 1984], [Plotkin 1982], [Thomas 1987] and [Winskel 1987ii]. General techniques for relating semantics are discussed in section 0.1.6.

* There is a spectrum of activities that are thought of as specification, running from requirement specification to detailed design. Each model of the behaviour of a reactive system may be suitable to a different phase of this process. [Olderog 1989] and [Pnueli 1986ii] maintain, for instance, that trace formulas are appropriate for requirements specification (as they are abstract, and so don't force us to state too much inappropriate detail), process algebra terms are appropriate for systems specification and architectural design, and an implementational model like the Petri nets formalism is appropriate for detailed design.

These points will not be discussed any further, but they should be born in mind when choosing a model for a specific purpose. Valuable comparisons can be made by studying such review articles as [de Bakker et al. 1986a], [Broy & Streicher 1987], [Denvir et al. 1983], [Emerson & Halpern 1985], [Pnueli 1986i], and [de Roever 1985] Cf. also [de Bakker et al. 1986b] & [Barringer 1985].

The models currently on offer will now be examined. I shall use the term 'syntax' for formalisms that seem mainly syntactic or for which people seem anxious to give a semantics. Similarly I use "semantics" for formalisms that seem to be adequate by themselves and where there is some consensus that they do not need explaining further. The borderline is a trifle blurred; this is indicated by moving those formalisms whose rôle seems particularly uncertain towards the middle of the classification, and not extending the line through them.

The major classes of theory are presented in the following table:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process algebras: CCS, SCCS, CSP, ACP, COSY.</td>
<td>Set-based; set of traces, failures, ready actions, divergences, refusals.</td>
</tr>
<tr>
<td>Programming Languages and Features; occam, Ada, Monitors, Coroutines, etc.</td>
<td>c.p.o.- or lattice-based; Scott-Strachey style.</td>
</tr>
<tr>
<td>Hardware description languages: CIRCAL etc.</td>
<td>Observational Semantics.</td>
</tr>
<tr>
<td>Petri Nets</td>
<td>Operational Semantics.</td>
</tr>
<tr>
<td>Event Structures</td>
<td>Partial order Semantics.</td>
</tr>
<tr>
<td>Logics; Temporal, Modal, and Higher Order Logics, Hennessy–Milner Logic</td>
<td>Transition Systems</td>
</tr>
<tr>
<td>Trace- and regular ω-Languages</td>
<td>Behavioural Presentations</td>
</tr>
<tr>
<td>Timed variants of the above; TCSP, timed Petri nets, real-time temporal logic, etc., interval temporal logic.</td>
<td>Categorical Semantics.</td>
</tr>
<tr>
<td></td>
<td>Metric space Semantics.</td>
</tr>
</tbody>
</table>

This, then, is the territory to be mapped; we have mainly concerned ourselves with compositional models, that is, models where complex processes can be built up from simpler ones using some natural notion(s) of composition. The properties of the whole should bear some relationship to the properties of the parts. It is not clear, for instance, if finite state automata can be so described.

References continued: Trace, failure, and divergence set semantics, [Hoare 1985], [Brookes & Roscoe 1985]; ready-set semantics, [Olderog & Hoare 1986]; Scott-Strachey style semantics, [Stoy 1977]; categorical semantics, [Winskel 1984ii]; metric space semantics, [de Bakker & Mayer 1988], [Kok & Rutten 1988]; observational semantics, [Hennessy 1988], [Degano et al. 1988]; operational semantics [Plotkin 1981]; Partial order semantics, [Pratt 1986]. My apologies for everything I have left out, and for those references that are not as definitive as the reader would like.

It is important to make the distinction between models that happen to have a certain semantics, – CCS, for instance, need not have a branching time semantics, – and ones which are purely semantic.
0.1.3 Time in untimed models

In the next few subsections the differences between various models of the behaviour of abstract reactive systems will be examined. One obvious dichotomy is the event/state one. A state-based model, where events are seen as labels on state changes (as in some transition systems) seems most appropriate for shared variable parallelism. Specification methods in these models concentrate on the development of a sequence of states. On the other hand, an event-based approach assumes that states can be deduced from examining a sequence of events leading up to them. This sort of model seems best suited to message-passing parallelism. Specification methods for event-based models concentrate on constraining a sequence of events in some way. (Note that both approaches have deficiencies in the treatment of silent or $\tau$ events.)*

There seems to be little disagreement that an instance of the observation of the behaviour of a concurrent system is best represented by a run, a linear sequence of events or states ordered by time of observation. A telling difference between models of reactive systems is how they group runs into behaviours which characterise the system. There are three main approaches based on three views of the appropriate model of time to use:

(i) linear; all possible behaviours are represented by a set of possible runs, and a specification is a predicate which requires that some property is true of all runs. Examples include linear temporal logic, trace theories and ready set models.

(ii) branching; here groups of runs form a computation tree which retains information about when runs part ways (i.e. about when choices are taken). A specification language in this domain must be able to express predicates over trees. Typical models here include branching time temporal logic and bisimulation equivalence [Park 1981]; bisimulation equivalence, for example, tries to capture the notion that two things are equivalent just when the same choices are available to each at each step in their evolution; see chapter three for details.

(iii) partial order; these models are based on two relations, a causality relation, written $\rightarrow$ (or $\leq$) and a conflict relation #. The relation $a \rightarrow b$ is true just when $a$ in some sense causes $b$. This usually means that $b$ cannot happen wholly before $a$. Conflict, such as $a \# b$, means that $a$ and $b$ cannot happen together in any run of the system. A behaviour in this approach is a maximal subset of events subject to the causality and conflict constraints. Examples include event structures and Petri nets. See [Reisig 1988] for some further discussion.

In order to further clarify things, consider how each approach deals with the following four CCS-like programs (overleaf). (Remember $*$ binds more tightly than $+$.) This example, tak-

---

* - A silent event is a ‘hidden’ or ‘internal’ event. Sometimes it is just a technical gadget, with no physical (i.e. observable) significance at all, but more usually it corresponds to a change of state with no externally visible accompanying event. Synchronisation gives rise to silent events in CCS.
Time, causality, and concurrency

David Murphy

en from [Pnueli 1986ii], makes clear how the different views of time affect which behaviours are defined to be equivalent. In that sense the choice of model of time is a semantic decision. Interleaving models cannot distinguish between concurrency (3) and choice (4). Linear models allow prefixing to distribute through choice, and thus cannot distinguish between (1) and (2), although branching models can. Partial order models are the most discriminating of all. (The symbol ~ is used for a point where a decision about what to do next is made.)

\[
\begin{align*}
&\quad \text{Process} \\
\text{a \cdot (b + c)} &\quad <ab>, <ac> \quad \text{Linear model} \\
\text{a \cdot b + a \cdot c} &\quad <ab>, <ac> \quad \text{Branching model} \\
\text{a \parallel b} &\quad <ab>, <ba> \quad \text{Partial order model} \\
\text{a \cdot b + b \cdot a} &\quad <ab>, <ba> \\
\end{align*}
\]

more discriminating

more abstract

Figure 0.1 - The difference between linear, branching and partial order models

A survey of the varieties of temporal structure is given in [van Benthem 1983], [Burgess 1984], and [Joseph & Goswami 1988]. There are, of course, many temporal structures that have not been discussed here. The only one that we have ignored which is of major importance to contemporary concurrency theory is real-time. This deficiency will be remedied in section 0.2.
0.1.4 Upon semantics

One obvious semantic decision a model must make is how to assign meaning to a syntactic object. Semantics is about equivalence; a semantics allows us to determine which syntactic objects are really the same. There are three main ways of assigning meaning: –

(i) denotational: here meaning is given in a structure-oriented way; the meaning of a compound expression is some combination of the meanings of its components, the combination being formed in a way that depends on the way the components are combined in the syntax. A typical example is Winskel’s denotational semantics for event structures or Roscoe’s divergence semantics for timed CSP, – see also [de Bakker & Zucker 1982]. Denotational semantics are often fairly inscrutable. A denotational semantics, \( D \), assigns a denotation to every process; this is typically an element in a domain [Scott 1982]. A denotational semantics can be thought of as a mapping from terms \( T \) (over some signature \( \Sigma \)) to a domain \( D \) so we could write \( D : T_\Sigma \rightarrow D \).

(ii) operational: this technique relies on saying what a process can do. Thus the effect of an operation is given on an abstract machine. A typical example is the transition system model for process algebras, popularised in [Plotkin 1982].

(iii) observational: here a process is characterised by what can be deduced from observations of it. See [Hennessy 1988] and [de Bakker et al. 1986a]. Observational equivalence can often be related to an operational semantics defined in terms of a transition system (so that a given observational equivalence is generated by a preorder over transition behaviour).

We will use approaches (i) and (ii) in chapter four. We will also use: –

(iv) axiomatic: here a set of laws, \( E \), that a process should obey are set down, defining an algebra. A semantics is then given using a term algebra, \( T_\Sigma \), with the equivalence induced by the laws factored out. This approach is very useful in transformational approaches to process algebra. A given \( \Sigma \)-algebra, \( A \), is then a natural model of the equations \( E \) if \( A \) is initial in the class of all \( \Sigma \)-algebras that satisfy \( E \), \( C(E) \). This is equivalent to demanding that \( A \) is isomorphic to \( T_\Sigma / E \). The interesting point is that the concept of ‘natural’ is represented formally by initiality. Again, see [Hennessy 1988] for details.

These techniques correspond rather loosely with varying notions of how a process is described; a denotational semantics allows us full access to the workings of the process when assigning meaning; operational approaches specify the effect of an operation on an abstract machine, while observational techniques merely rely on having something to put in parallel with the process. There is clearly a scale of introspection of mechanism, running from complete knowledge of the process, to complete ignorance of everything about it except its
interaction with some agent. This is not wholly independent from the formalism it is
describing, but is nearly so; there are many possible approaches to giving a semantics to
CCS for instance, and each deserves an entry in the final classification.

I am rather unhappy with the labels 'denotational', 'operational' and so on as
classifications, but can find no reasonable alternative; I do not want to describe a particular
sort of semantics—but, rather, an attitude to how much knowledge it is reasonable to ask for
when assigning meaning.

0.1.5 On relating semantic techniques; the full abstraction problem

It is worthwhile to pause for a moment in our search for classifications and consider the
relationship between various semantic techniques used in concurrency theory. This is the
territory of the full abstraction problem. A given operational semantics is said to be fully
abstract with respect to a denotational one just when they identify the same terms. An
operational semantics generates some congruence on terms ~ (i.e. t ~ t' if t and t' have the
same transitions in a given abstract machine), so it is fully abstract with respect to a
denotational semantics \( \mathcal{D} \) iff \( t \sim t' \Leftrightarrow \mathcal{D}(t) = \mathcal{D}(t') \).

There have been various approaches to the full abstraction problem; one technique is to
assign a contraction to both semantics in a complete metric space and show that these
contractions are the same using the uniqueness property guaranteed by Banach's contraction
mapping theorem; [Kok & Rutten 1988] and [de Bakker & Mayer 1988] advocate this
approach. Alternatively, one can use the algebraic characterisation given in the last
paragraph.

0.1.6 On relating models

Some progress has been made recently on relating models themselves. There seem to be
two main strategies:

(i) Categorical. This is the technique adopted by [Bednarczyk 1987]; models are
formulated as suitable categories (easy enough for transition systems and nets,
harder for process algebras), and then the relationship between categories is
explored using the usual category-theoretic mechanisms (essentially adjunc-
tions). The method has also been used by [Winskel 1984ii], [Winskel 1987].
Difficulties lie in knowing how to give a sensible categorical treatment to certain
operations, like sequential composition, which seem both intuitively clear and
hard to model categorically. However, the existence of a mathematical tool which
enables some features of different models to be compared ought to hold interest
in concurrency theory for some time.
(ii) **Model Theoretic.** Here a grand model is proposed with all the behavioural features of any model which we want to relate. Individual models are then translated into the terms of the grand model, to see what class of structures they can express. This technique has been used by [Shields 1988]. The problem then comes in comparing grand models.

It is worth mentioning that there is nothing in concurrency theory yet that is a model in the sense of model theory. Joseph Goguen has done some work on the properties that any concurrency theory should satisfy, with the aim of investigating the real model theory of concurrency.

0.1.7 Delving into events

There is a spectrum of positions that a model can take with regard to the status of event refinement. We shall, as usual, caricature this complexity by indicating three possible stances:

(i) Events are always *atomic*: there is no such thing as event refinement, simultaneity is difficult to define in a satisfying way, and concurrency is the same as interleaving. CSP is built from much this view.

(ii) Events are atomic, but, under certain circumstances event refinements can proceed *sub rosa* as process transformations. Here events can have structure but careful attention to context is always needed. Petri nets allow this view [Berthelot 1987], as do many state-based approaches. [Aceto & Hennessy 1988] investigates event refinement for a simple process algebra.

(iii) Events are *intervals*. Event refinement and process refinement proceed independently and events can have as much structure as desired. This approach is implicit in real-time interval temporal logics, and explicit in [Manna et al. 1983] and [Schwartz et al. 1983].

0.1.8 A plausible classification of untimed models

This rather inchoate classification will now be applied to seven major groups of models. Needless to say, this classification is tentative and often conjectural. However, while specific classifications may be in doubt, we hope that the classifying principles will give the reader some insight.
### A possible classification

<table>
<thead>
<tr>
<th>Model</th>
<th>View of Time</th>
<th>Introspection of Mechanism</th>
<th>View of Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process algebra with linear time semantics e.g. the traces semantics of CSP</td>
<td>L</td>
<td>D, Op?, Ob?</td>
<td>A</td>
</tr>
<tr>
<td>Process algebra with branching time semantics e.g. the usual semantics for CCS or COSY</td>
<td>B</td>
<td>Op, Ob, D?</td>
<td>A</td>
</tr>
<tr>
<td>Shared variable programming language style parallelism</td>
<td>L?</td>
<td>Op?</td>
<td>T?</td>
</tr>
<tr>
<td>Typical hardware description languages</td>
<td>usually L</td>
<td>Op</td>
<td>I</td>
</tr>
<tr>
<td>Distributed transition systems e.g. Petri nets</td>
<td>PO</td>
<td>Op, D?</td>
<td>T</td>
</tr>
<tr>
<td>Partial order models without cycles e.g. Event structures</td>
<td>PO</td>
<td>D, Op</td>
<td>A</td>
</tr>
<tr>
<td>Interval temporal logics</td>
<td>B, PO?</td>
<td>Op, D?</td>
<td>I</td>
</tr>
</tbody>
</table>

**Key:**  
L = linear time, B = branching time, PO = partial order;  
D = denotational, Ob = observational, Op' = operational;  
A = atomic, T = transformable, I = interval;  
? = particularly dubious classification
0.2 Timed concurrency theories

We shall discuss three timed concurrency theories, one from each of the major branches of concurrency theory. Timed CSP is based on the process algebra CSP, timed Petri nets are based on the partial-order Petri net framework, and real-time temporal logic is a temporal logic with a semantics based on a continuous set of possible worlds. These models will be discussed briefly, in terms of their usefulness for our purposes.

Before addressing the models themselves, it is reasonable to ask why real-timed models are necessary at all. After all, physical concurrency is a long way from the traditional concerns of concurrency theory, if one neglects the subtleties of asynchronous hardware. The answer is that many distributed systems have performance constraints. Operating systems have to respond to users before they decide to go away and have a cup of coffee, many other distributed systems, such as process control or air traffic control systems, also have sophisticated timing constraints which are usually expressed with the assumption that time is real-valued. We need to be able to reason about these in the same framework used for functional correctness. It is not always, or even usually, possible to speed up a system once it is ‘correct.’ Distributed systems are subject to subtle behaviours resulting from the compound behaviour of asynchronous subsystems; adding more processes or speeding up existing ones will often change the functionality of a system, not improve its performance. We must be able to deal with timing considerations ab initio. The discussion in [Joseph & Goswami 1988] is relevant here.

There is, then, clearly a need to be able to articulate timing constraints. But won’t a finite or a countable model of time do? Are the reals necessary? Apart from the fact that the reals are the obvious structure to deal with time, since timing constraints are usually expressed over them, there are two issues; density and continuity.

A dense model is certainly necessary (i.e. a model where there is always a point in time between any two given points); we cannot know what is going to happen before we have built the model; neither can we know when things will happen. Hence it is impossible to chose the correct granularity for a model that isn’t dense: time based on N is therefore excluded.

Continuity is more difficult. It is not known to me whether certain interesting behaviours (such as chaos and metastability) are expressible with a dense but not continuous time such as one based on Q. Certainly R seems the natural domain for describing such behaviours. (It would be interesting to know if work concerning the impossibility of realising certain behaviours, such as [Mendler 1987], was reproducible under the assumption of rational rather than continuous time.) These behaviours are perverse, admittedly, but we can’t say that a system won’t display them unless they are expressible in the model. If we can’t
express synchronisation failure, or metastability, properly, then how can we know that we have built a system without it? Any model which purports to be implementable must account for the essential underlying complexity of the behaviour of physically concurrent systems, and for this real-valued (or, at least, dense) time is necessary.

It seems silly to have an elegant, abstract model, when implementations of it will be hideous ad hoc, unverified, and of dubious reliability; we should be able to descend to the implementational level. Further, as implementations display behaviour that is easy to describe under the assumption that time is real-valued, and as the timing constraints that implementations must satisfy can be articulated over $\mathbb{R}$, real-timed models seem to have a place in concurrency theory. Before going on to consider the timed theories of concurrency currently available, we shall indulge in a more philosophical aside.

**ASIDE – Axiomatising a theory of time**

In the discussion above, we considered why $\mathbb{R}$ might the right choice for a model of time from the point of view of describing behaviours. Another approach is to adopt the 'pick-and-match' method for choosing a model of time discussed in [van Benthem 1983] (but going back in spirit to Dedekind): we shall state the axioms we think are true of time, and see what models that leaves us with. Suppose $<$ is the order of temporal precedence, that $u, x, y, z$ are points in time and that $A$ is a property of a point. Then it seems reasonable to demand: –

- transitivity ($\forall x, y, z . (x < z < y) \Rightarrow (x < y)$),
- irreflexivity ($\forall x . \neg (x < x)$),
- linearity ($\forall x . (x = y) \lor (x < y) \lor (y < x)$),
- succession ($\forall x, \exists y . y < x$; $\forall x, \exists y . x < y$),
- density ($\forall x, y . x < y \Rightarrow \exists z . x < z < y$) and
- continuity $\forall A . (((\forall x, y . ((A x \land \neg A y) \Rightarrow x < y) \land \exists x . A x \land \exists y . \neg A y) \Rightarrow \exists z . (\forall u . (z < u \Rightarrow \neg Au) \land \forall u . (u < z \Rightarrow Au)))$

The last axiom can best be thought of with the aid of a picture. One might read it 'if $A$ is the property of being earlier than some point, then there is a point which that property is talking about.'

```
    X
A       ↑       non-A
    z
```

$\mathbb{R}$ is certainly a model for time with these axioms, and $\mathbb{Q}$ isn’t [van Benthem 1983]. If we add separability, then we have defined $\mathbb{R}$ up to isomorphism. Thus $\mathbb{R}$ will certainly serve our purposes.
The following models will be of interest:

(i) Timed CSP is one of the most extensive timed models; [Davies & Schneider 1989] contains a summary of the literature, see also [Reed & Roscoe 1987]. It is a carefully thought-out and coherent model, based on CSP. And therein lies the reason it is not adequate for our purposes: CSP has atomic events; so does timed CSP. This means that a 'system delay constant,' \( \delta \), has to be introduced to stop everything happening at once. This is not necessarily an undesirable approach, but it does mean that timed CSP doesn’t agree with our intuitions. We want to be able to understand the effect timing has on our thinking, to discover what sensible operators should be like in a timed framework. In timed CSP, the timing has to be made to fit around the old CSP operators; there is no room for exploration, not enough behavioural richness to discuss the level of detail we are interested in. Timed CSP isn’t inadequate, — far from it, it is quite useful at a certain level of abstraction, — but it doesn’t inhabit the area of concurrency theory we are interested in.

The reader may be wondering if there is a timed concurrency theory based on CCS. The only work I know of in this direction is [Tofts 1988], but this is not a proper timed theory, since it seems that we cannot force things to happen when we want, only when the process feels so inclined. The lack of a proper treatment of liveness (i.e. the theory will not allow one to require that something definitely happens at 'a given time), and problems regarding divergence (whereby divergent processes can 'stop time') are a common and regrettable deficiencies. Hennessy and Regan are working in this area, and their treatment is liable to be more satisfactory.

(ii) Timed Petri nets. The literature on timing and Petri nets that I have been able to consult is quite scanty. There are two main approaches, the time Petri nets of [Leveson & Stolzny 1985], [Merlin & Farber 1976], and [Merlin 1974], and the timed Petri nets discussed in [Carlier & Chretienne 1989], [Carlier et al. 1984], [Chretienne 1983], and [Ramchandan 1974]. (In each case the first few references are applications, and the last the presentation of the model). The timed Petri net model is closest to our own, so we shall discuss that.

Timed Petri nets have times attached to transitions representing (fairly obviously), the time at which the transition happened. Timed nets would probably be sufficient for our purposes, but we will not use them. As is often the case with nets, we will end up with something net-like, but the construction of our model is only straightforward outside the world of nets. Event structures are easier to work with than nets (having a well developed and clean theory of composition
and abstraction, and well-isolated notions of conflict and nondeterminism). Moreover, event structures are fairly closely related to nets, so we will merely be abandoning net terminology, not the net-theoretic notions of concurrency, nondeterminism etc.

(iii) **Real Time Temporal Logic.** Work here is pretty much as one would expect from the title. The standard reference is Koymans’ thesis, while [Koymans et al. 1987] is a good introduction. Our objection here is the lack of descriptive power. One can imagine a partial order real-time temporal logic, but little work has been done in this area, and useful results (such as normal form theorems, or domain-theoretic characterisations of expressive power) seem far off.

Some tangential mention should also be made of systems designed for the development of real-timed programs, and real time specification languages. In the former area the work of [Auernheimer & Kemmerer 1986], [Berry et al. 1983], [Coulas et al. 1987] and [Gautier et al. 1986] are noteworthy, while in the latter [Beneviste & Le Guernic 1987], [Bernstein & Hartier 1981], [Dasarathy 1982], [Gamatie 1986], [Quemada & Fernandez 1987], and [Wupper & Vytopil 1987] should be mentioned. [Joseph 1988] contains some current work.
0.3 On the nature of a possible concurrency theory

It is possible to see concurrency theory as part of a wider tradition than that offered by computer science and discussed in the first section of this chapter. This position sees concurrency theory as part of natural philosophy, as an attempt to describe and reason about events happening and their temporal and causal relationships. There is no reason a priori to suppose that concurrency theory is necessarily about computational systems; any computation that events perform by virtue of their happening can be seen as inadvertent: computation is an interpretation we place on some series of happenings, not an intrinsic property of the happenings themselves.

It is in this tradition that this thesis originates; we will be interested in trying to describe the behaviour of physical entities, where ‘behaviour’ will mean ‘what can be seen to have happened.’ Another crucial matter is that of time; intuitively when things happen is as important as what happens, — tomorrow’s stock market prices are more interesting than yesterday’s. Computationally, a cashpoint that always distributes money within fifteen seconds may be more desirable than one that takes arbitrarily long, although certain high street banks seem to disagree.

Our paradigm will be observational. We shall imagine that everything of interest to us can be isolated and observed. Things will happen, and we will observe not just the fact of their happening but also the time that that happening started and the time that it finished; we might see the occurrence of the event ‘my supervisor is running a mile,’ starting at 11:52am and ending at 11:59am (on the same day). We shall not participate in the events seen by any means; running alongside, or tripping the runner up are both disallowed. This, then, is a very classical viewpoint; we stand, all-seeing but aloof, recording when and what things happen.

Two features of this stance are unusual. Firstly we shall be dealing with real time. This accords with our naïve-physics outlook; all physical systems that don’t operate on microscopic or cosmic scales can be described accurately by assuming the existence of a single, fixed, all-seeing observer with a clock measuring time over the reals. (Even when we are dealing with relativistic systems the presence of such an observer is not necessarily contradictory — other observers may disagree with our observer, but their observations can be deduced from those of our observer, and in any case the fact of observation in no way influences what is observed for us.)

A second, unusual position held here is that of the ‘all-seeing’ observer. Most theories of concurrency allow ‘hidden events’ to occur without the knowledge of the observer. Here we are dealing with what happens, a level of abstraction known as the implementational level in computer science, and here there seems to be no point in denying ourselves information that is available. At higher, more abstract levels of description we may be concerned with black
boxes and what can be deduced about them by interacting with them, but this is a less observational and less classical position than we want to adopt.

The occurrence of events is what we are interested in. We have already mentioned our notion of when they happen and how that is measured. How things happen is all that remains to discuss.

The causality-as-a-poset position has already been outlined; we will adopt it. This gives us the advantage of fairly reasonable notions of causality, concurrency and the possibility of representing nondeterminism. Notice that all of these three concepts were current in natural philosophy long before concurrency theory began. The first serious attempts to formalise the intuitions that we will exploit was made by Russell. For him, implicitly at least, concurrency was a property that was enjoyed by two distinct causally-unrelated happenings, a notion that we share. For us, if two happenings are concurrent, then they might be in different places, or occur simultaneously, or both; it doesn’t matter.

A concern for nondeterminism stretches back at least to the preordination debates of the church fathers, and like most ancient philosophical problems, it still isn’t well understood today. There is certainly a place for the notion that the future is fundamentally unknowable, and that the physical world will not reveal all its secrets to us at once. This is the stance we shall take on the nature of nondeterminism.

(Aside: it is important not to confuse several issues when discussing nondeterminism. There is the dichotomy between internal choice (pure nondeterminism) and external choice: a model may have both sorts of choice, one, the other, or neither. There is also a distinction between models with silent (or \( \tau \)) moves and ones without. Furthermore, the presence of silent moves (unobservable state changes) does not necessarily indicate nondeterminism; silent moves may come about because of hiding, (choosing to ignore some happenings) or for other reasons. Hiding may introduce nondeterminism, but it need not.

One can think of a spectrum of concurrency theories, running from the most ‘classical,’ – completely deterministic theories without silent moves or hiding, – to the most ‘modern,’ – theories with internal (purely nondeterministic) but without external choice, endowed with silent moves and hiding. Our own position is midway between these two extremes: we want a theory with internal choice, (since we believe in nondeterminism), but without hiding (since we never want to deny ourselves knowledge that is available). We will occasionally use silent moves for sub rosa technical purposes, but they will not play an important conceptual rôle in the formalism.)

All of our stances on the foundational concepts of concurrency theory seem intuitively reasonable from a classical position.
0.4 A model proposed

In the last section, a philosophical position on the nature of a possible concurrency theory was adopted and some of the consequences of this position were explored. In this section a problem area within concurrency theory is selected and a suitable formalism for reasoning about systems in that area is suggested; we shall cover the same ground as the last section, this time showing how the model came about from the perspective of computer science (and, in particular, from that of section 0.1) rather than from the perspective of natural philosophy.

Our original motivation was the design and description of asynchronous digital systems [Murphy 1987]. There we suggested that certain undesirable behaviours of such (physically concurrent) systems could only be excluded from our designs if there was a means of describing genuinely continuous variables. Unfortunately, models with that capacity tend to be hideously intractable, so, in an attempt to get some way to full continuity without losing too much analytic power, it was decided to investigate real-timed models of abstract concurrency. This field has the advantage that some problems in it have already been tackled [Koymans et al. 1985], [Davies & Schneider 1989], but not, we feel with complete success. (A proper criticism of the literature on real-timed concurrency theories would be much more extensive than section 0.2 and, I suspect, even more trying on the reader’s patience.)

The original problem area, asynchronous digital systems, suggests that a model with a notion of genuinely distributed concurrency will be important. So, a non-interleaving model is needed; a partial order one seems to offer the appropriate discrimination. Clearly, also, it is reasonable for us to demand a full knowledge of how systems are constructed; we are attempting to describe just such constructions. Thus no a priori limitations on our knowledge of the system will be imposed, and the semantics will be what we classed in section 0.1 ‘denotational.’

Concurrency theories with causality and highly descriptive notions of behaviour, like event structures, have been discussed in this chapter. These theories seem to be the ideal forum for comparing various constructions, as they are rich enough to express everything that we are interested in. Our aim will be to take such a model and add timing, so that it has the same status as causality (rather than as an afterthought), and to explore the consequences of that decision.

We want to invent a concurrency theory that will highlight the interaction of causality and timing, and will allow us to discover additional insight into the nature of concurrency and non-determinism. We have some intuitions about valid constructions in a timed model, – things can’t happen before their causes, things can’t synchronise unless their times overlap, and so on, – and once we have time these intuitions can be exploited. This will allow us to discuss what certain operations (like concurrent composition with synchronisation) should be like in any model which can be timed. Even if we are only interested in a model being temporally
well-founded, – in the possibility of being able to assign consistent times to it, – then these considerations will be important.

We have seen that many interesting behaviours can only be described by real-valued time. Naïvely, time is perceived as continuous. Furthermore, there seems no compelling reason to doubt that real-valued time will be adequate for the description of physical concurrency. A model with real-valued time based on event structures, then, seems to be appropriate for our investigations.

The only further point worthy of discussion is the duration of happenings. Anything which happens seems to take time, – “time is nature’s way of stopping everything from happening at once,” – occurrences of events have measurable durations. (The practicality of this view is evinced by [Manna et al. 1983].) We will embrace this simplistic analysis, abandoning the notion of instantaneous events popular in process algebras. Although events with duration could be treated by considering instantaneous starts and finishes, completely ignoring the fact that these starts and finishes belong to events seems unnatural. So, for us, happenings will have starts and finishes, with some time delay between them, and these starts and finishes will be formally related via the happenings that they are starts and finishes of. This position has the singular advantage of giving us a clean notion of event refinement; process refinement is now entirely separate from event refinement, and the two forms of refinement can proceed without interaction. Intuitively, process refinement and event refinement are orthogonal; once events have durations they can become so formally as well.

The place of our model, the interval event structure model, in the scheme of things is now indicated. Some relationships are conjectural, some based on personal prejudice and some are fully worked out.

In the next chapter our model will be outlined in considerably more detail.
Interlude – Personal Remarks

Acknowledgements

I have benefited from the help and encouragement of my supervisor, David Pitt, and from conversations with Luca Aceto, Stephen Blamey, Ross Casley, Bernie Cohen, Jason Cozens, Jim Davies, Joe Goguen, Ursula Goltz, Barney Hilkin, David Jefferies, Alan Jeffrey, Marta Kwiatkowska, Vaughan Pratt, Jeff Sanders, Steve Schuman, Walter Vogler, Glynn Winskel and especially Mike Shields.

My greatest intellectual debt is to Glynn, but I must also mention Samson Abramsky, Jay Gischer, Matthew Hennessy, Robin Milner, Ernst-Rüdiger Olderog and Carl Adam Petri. My special thanks are due to Luca Aceto, Rod Burstall, Jim Davies, Steve Russ, Mike Shields and Terry Stroup, for making visits to the Universities of Sussex, Edinburgh, Oxford, Warwick, Erlangen-Nürnberg and Kent enjoyable and informative. Harry Simmons has done a great deal for British theoretical computer science as coordinator of the Logic for IT initiative, and for me personally; I thank him. Chris Elvin and Mark Ellis have given me considerable and highly valued help with various systems. British Rail gave me ample time to think, while Helen Parker gave me somewhere to stay while circumventing the deficiencies of my universities’ library; I owe Helen considerable gratitude. Finally, I must thank Elly for all her support, encouragement and turron.

I began this thesis while attending the 1988 REX Conference Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. I am very grateful to the organisers of REX ’88 for that conference; it was the most stimulating meeting I have ever been to, and I doubt if this thesis would have been written without the perspective gained at there.

The structure of the thesis

The thesis is composed of chapters separated by interludes; this is the first interlude. The first chapter was introductory, being concerned with the general problem area addressed in this thesis, and with my specific motivations. The main problem chosen is the development of a behaviourally rich, analytically sound, real-timed concurrency theory. Some indication of the methods to be used in the rest of the thesis was given; in particular we settled on a real-timed model based on event structures. The next five chapters will present the substance of this model. The second chapter is concerned with the basic definitions, the third with the combinators of the model, the fourth with semantics, the fifth with a process algebra based on the model, and the sixth with matters of specification and refinement. The interludes between chapters offer more speculative (and often rather more technical material) related to that presented in the preceding chapter.
Each chapter, the first excepted, is separated from its interlude by a bibliography. A postlude summarises our conclusions and outlines further work. The thesis is completed by a complete bibliography of all works referred to, organised by subject area.

On the progress of concurrency theory

I would like to crave the reader's indulgence for three more paragraphs before the thesis proper. There are many models used in concurrency theory at the moment, and their relationship is badly understood. It should be a matter of urgency to try to understand what theoretical features of a model support its usefulness, and how those features are related. The interaction between timing, causality, concurrency and nondeterminism is very subtle, and it seems more important to me to concentrate on understanding all of them together than each of the separately. There are no good models at the moment, and no bad ones, just models suitable for some tasks and not for others. Most models can be forced or cajoled into modelling most phenomena, so, to a certain extent, the choice of a model is a matter of personal proclivity, but one should be aware that some models are more convenient for some tasks than others. I use an event-structure-based model because it enables me to talk about the level of detail I am interested in easily and naturally; no doubt I could have used practically any of the models of section 0.2, but with some, this work might have been more of a struggle.

I feel very conscious that there are too many models and not enough work done on relating them and drawing out common threads, - this makes me rather nervous about presenting yet another model. However, in my defence, I would claim that interval event structures are closely related to well-understood models, and offer previously esoteric insights quite directly. Furthermore, where possible, I have given hints in the text about techniques that might be used for relating the model presented here to others. I would like to claim that real-time models are no more complicated than untimed ones, just different from them. The more we can understand about what concurrency theories make sense, (i.e. the more points in the concurrency theory design space have been investigated), the better we can understand the quiddity of concurrency and nondeterminism.

A note on terminology

Throughout this thesis the term 'event' will be reserved for some thing that can happen more than once, and has an identifiable beginning and end. When we want to refer to distinct occurrences of events, we will endow each occurrence with a label. The term 'transition' will be used for an atomic happening, usually the beginning or the end of an occurrence of an event. The term 'happening' will be used more generally, for something that happens and can be observed. Our terminology is further discussed in chapter one, which follows.
1 Definitions: Linear and Branching Interval Event Structures

Nor second he, that rode sublime,
Upon the seraph-wings of ecstasy
The secrets of th' abyss to spy.

He pass'd the flaming bounds of place and time:
The living throne, the sapphire-blaze,
Where angels tremble, while they gaze,
He saw; but blasted with excess of light,
Closed his eyes in endless night.

Milton

In this chapter the basic framework of interval event structures (I.E.S.s) is introduced. As the name suggests, these are derived from the event structures of [Winskel 1980], and differ from them in that we shall consider events to be intervals of time rather than atomic, durationless actions. We begin by discussing our notions of time and causality. This leads us to a discussion of the nature of I.E.S.s; a branching interval event structure can be thought of as like an event structure, but with each occurrence of an event having a pair of real numbers associated with it representing the time that that occurrence started and finished. The reals, R, have a total order, <, and the extra structure that this gives us means that I.E.S.s are rather more than just event structures labelled with durations.

Definitions that formalise our notions are presented. We then investigate how to represent our underlying order on times, <, and our notion of causality as an order on events. This gives us some ideas about how to specify interval event structures. At this stage we also discuss the representation of choice. Finally, we indicate what information about an I.E.S. can be obtained by observing an execution of it. The philosophical position this model supports was discussed in the introduction and will not be further elaborated here.

1.0 Linear and Branching Time

There are two common views of time in concurrency theory: linear and branching. In the former, all events are comparable; we have a sequence of events happening along a line of time, and every instant is either before, after, or at the same time as any other instant. This is the conventional view of time outside concurrency theory. In branching time instants can be incomparable; time is more like a tree (strictly, a graph) than a line. This expresses the ideas of concurrency, as instants on different branches may be concurrent, and choice, as a structure modelled thus can also display more than one behaviour.
Each path through the tree can be thought of as representing the local clock of an observer following the computation that lives on that path. Thus the branching view of time subsumes the linear one in that each path is a linear time model. However, any execution of the branching system as a whole must also be seen as linear; a global observer will see the order imposed by the branching structure, plus additional order imposed by how concurrent happenings in different branches interleave; it is by discarding the unwanted orderings of each execution that we obtain the branching structure.

1.0.1 Causality

Our starting point will be a consideration of events. Flaunting usual terminology, we shall say that events are distinguishable happenings. They can happen more than once, once, or not at all, but each time they occur they can be observed and identified. Each occurrence of an event will be given a label to allow us to disambiguate it from other occurrences of the same (or different) events; labelled events are unique.†

Events will be assumed to be compound objects, in the sense that they start and then they finish. Putting on a shoe might be an event, or taking a drink, or pin 13 of IC-7 going from 0.5V to 4.5V. We shall suppose that the important things about an event are its start and its finish; any other details of its internal structure will be ignored for the moment (they will reappear later).

A distinction will be made between things that just happen from things which must happen. Things which must happen must have been caused by something. We shall ignore the thorny philosophical problem of the presence or absence of primal causes in general, and merely allow that some labelled events can be said to cause others, and, in particular, that one special happening can be said to cause everything of interest to us. Hence, we might (in ascending degree of plausibility) say that the desire to go out to the betting shop caused me to put my shoe on, the eventual service of a drink in a crowded bar caused me to drink it, and the departure of a pulse of current from pin 7 of IC-5 caused pin 13 of IC-7 to go high. We can see that it makes sense to speak of either the beginning or the ending of an action causing the beginning or the ending of another. This notion of causality will be modelled by endowing the start and finish points of our labelled events with a strict partial order (i.e. a transitive and irreflexive relation), which we shall write as < (\(\text{c} \) for causal). The interpretation placed on an assertion like \(a_t < b_t\), where \(a_t\) is (a name for) the start of the occurrence of some event, and \(b_t\) is similarly the finish of an occurrence of another event is that the beginning of the first event caused the finish of the second. Where we want to discuss either the start or the end of an occurrence, we write \(a_t\) (\(\text{t} \) for transition – some writers call the beginning or the end of an occurrence of an event its 'transitions;' we will adopt this usage sometimes).

† – Some writers use the term 'action' for what we term 'event' and reserve the term 'event' for what we call 'labelled event.'
The causal partial order \( \prec \) has three obvious properties. Firstly the ends of all occurrences are related to their beginnings (unless we allow point-like occurrences); \( \forall a. \, a \prec a \). Secondly, we spoke of a ‘special happening’ that caused everything. This will be denoted by \( \star \); the start of \( \star \) causes everything. It will be convenient for \( \star \) to last at least as long as everything of interest: it will act like an ‘on light.’ Furthermore, everything causes the end of \( \star \). Thirdly \( \prec \) is a strict partial order; at most one of \( a, \prec a', a, \prec a' \) is true.

It may be desirable to have events occurring contemporaneously with the start of \( \star \); in this case the structure starts to do something as soon as it is ‘turned on.’ In this case we will want to consider the start of an occurrence of an event which is causally the same as the start of \( \star \).

For this reason we will introduce causal equivalence, \( =_c \); the interpretation of \( a, =_c a' \) is that \( a, \) and \( a' \) are two different names for the same happening. The equivalence \( =_c \) must be reflexive, symmetric and transitive; it must also be distinct from \( \prec \) and a congruence of it; \( a, =_c a' \Rightarrow (a, \prec a' \land a', \prec a) \) and \( (a, =_c a' \land a', =_c a) \Rightarrow (a, \prec a' \land a', \prec a) \). We might think of the expression \( a, =_c a' \) as stating that \( a, \) and \( a' \) are necessarily simultaneous. Clearly \( \leq_c \) is a partial order on occurrences.

The required property of \( \star \) is then
\[
\forall a. (\star, \prec a \lor \star, =_c a) \land (a, \prec \star \lor a, =_c a).
\]
The rôle of \( \star \) is indeed seen to be special; it will be assumed (rather theologically) that it is always present. We will know when things start, \( \star \), happens to tell us that the system is active. Similarly \( \star_t \) tells us that everything is over.

Instead of the usual \( a, \prec a \) we shall sometimes (in exceptional circumstances) allow the weaker condition \( a, =_c a \) so that, in fact, \( \forall a. (a, \prec a \lor a, =_c a) \).

We have seen how to deal with causality, now we must deal with the structure of time over which things are caused. In the introduction we discussed the motivations for dealing with branching real time (see also [van Benthem 1983] and [Joseph & Goswami 1985] for further discussion), so in the next subsection we shall just deal with the how rather than the why of our temporal structure. Then, in the next section, we shall give a formalism which encapsulates some of the ideas of this section.

An alternative approach to non-atomic events, of some interest, still within the causality-as-a-poset framework, is taken in [Boudol & Castellani 1987]. I have recently discovered that [Thomason 1984] is of considerable relevance to the next section in particular, and this chapter in general; some of the operators we discuss in later sections are also defined by Thomason.
1.0.2 Branching time

Suppose that the set of all occurrences of all events is LE. The causality of events has been dealt with by endowing the set of all transition, that is starting and finishing points of occurrences of events, \( \text{Tr} = \text{def} (\text{LE}_s \cup \text{LE}_f) \), with a partial order \(<_c\). Now we want to endow these points in addition with a real; this real will represent the time at which that transition happened. This will leave us with a branching time model incorporating both causality and timing. (Note that we use the term 'occurrence' nontechnically here.)

Branching time, then, will be modelled by time variables consisting of pairs; the first member of the pair will be a real and the second a tag. These tags are just transitions, that is, elements of \( \text{Tr} \). So, the set of all branching times, \( \text{BT} \), will be \( \mathbb{R} \times \text{Tr} \) for some set of transitions of partially ordered labelled events \( \text{Tr} = (\text{LE}_s \cup \text{LE}_f) \). These transitions are, of course, ordered by \(<_c\) and endowed with \(=_c\).

Notice that branching time, \( \text{BT} \), can be thought of as parameterised by LE; defining LE automatically defines a branching time for LE to 'live on,' \( \mathbb{R} \times \text{Tr} \).

The reals have a total order, \(<\), which will represent temporal precedence. The rest of this subsection is devoted to showing how to use both \(<\) and \(<_c\) to define various orders on branching time. In future, reals will be written as \(r\) or \(r_{\text{with some subscript}}\) whilst branching times will be \(t\) or \(t_{\text{with some subscript}}\). (We shall also suppose that, for \(a \in \text{LE}\), the branching time assigned to the start of \(a\) is \((r_{(a)}, a_s)\), and to the finish, \((r_{(a)}, a_f)\). Similarly the branching time of some general transition \(a\) will be written \((r_{(a)}, a)\). The notation \(t_{(a)} = (r_{(a)}, a)\) will often be convenient, and we will assume that, \(t_{(a)}\) is shorthand for \((r_{(a)}, a)\) and vice versa.)

The first order over \(\text{BT}\) we shall consider will be written \(<_b\); this branching order will combine salient features of both \(<\) and \(<_c\). We must assign reals to the beginnings and ends of labelled events in such a way that the branching order \(<_b\) respects the order on the reals \(<\) and the causal order on the tags \(<_c\) (so that things can’t happen before their causes):

\[
\forall t_a = (r_a, a_i), t_b = (r_b, b_i) \in \text{BT}. \; t_a <_b t_b \Leftrightarrow (r_a < r_b \& a_i < c b_i)
\]

In addition to \(<\), \(\text{BT}\) will be endowed with an equivalence \(=_b\), which respects both \(=(\text{on } \mathbb{R})\) and \(=(\text{on the tags } \text{Tr})\):

\[
\forall t_a = (r_a, a_i), t_b = (r_b, b_i) \in \text{BT}. \; t_a =_b t_b \Leftrightarrow (r_a = r_b \& a_i = c b_i)
\]

It will be convenient to write \(t_a < t_b\) to mean just \(r_a < r_b\), and \(t_a = t_b\) to mean \(r_a = r_b\).

**DEFINITION 1.0 – Orders on Linear and Branching Time**

With \(<_b\) and \(=_b\) as a basis, it is easy enough to define other orders on branching time; we do this overleaf.
\[ \forall t_1, t_2, \in BT\]

1. \((t_1 \sqsubseteq_b t_2) =_{\text{df}} (t_1 \leq_b t_2)\)
   1a. \((t_1 \ll t_2) =_{\text{df}} (t_1 < t_2)\)

2. \((t_1 \neq_b t_2) =_{\text{df}} (t_1 =_b t_2)\)
   2a. \((t_1 \neq t_2) =_{\text{df}} (t_1 = t_2)\)

3. \((t_1 \leq_b t_2) =_{\text{df}} (t_1 =_b t_2) \lor (t_1 <_b t_2)\)
   3a. \((t_1 \leq t_2) =_{\text{df}} (t_1 = t_2) \lor (t_1 < t_2)\)

4. \((t_1 \geq_b t_2) =_{\text{df}} (t_1 =_b t_2) \lor (t_2 <_b t_1)\)
   4a. \((t_1 \geq t_2) =_{\text{df}} (t_1 = t_2) \lor (t_2 < t_1)\)

5. \((t_1 >_b t_2) =_{\text{df}} (t_1 \geq_b t_2) \land (t_1 \neq_b t_2)\)
   5a. \((t_1 > t_2) =_{\text{df}} (t_1 \geq t_2) \land (t_1 \neq t_2)\)

If \(t_1 \not< t_2\) and \(t_2 \not< t_1\), then \(t_1\) and \(t_2\) are said to be *unrelated* in the branching model, which indicates that there is no causal precedence order defined between them.

This incomparability indicates that whatever is happening at \(t_1\) can be *concurrent* with whatever is happening at \((or is timed by)\) \(t_2\). A predicate, \(\text{inc}\), indicates incomparability: in branching time two things can be concurrent if they are not ordered one way, nor the other, nor are they equal. (Notice that branching time incomparability, \(\text{inc}\), can only arise from causal incomparability since the reals are totally ordered.)

\[ 6. t_1 \text{ inc } t_2 =_{\text{df}} (t_1 \ll t_2) \land (t_2 \ll t_1) \land (t_1 \neq t_2)\]

At the moment two times are either ordered one way in the branching order, \(t_1 <_b t_2\) or ordered the other way \(t_2 <_b t_1\), or they are branching equal, \(t_1 =_b t_2\), or they are incomparable, \(t_1 \text{ inc } t_2\). Given that \(\text{inc}\) does not guarantee simultaneity, incomparability should be thought of as asserting that the transitions referred to can be *distributed*.

In a *linear* model = would be defineable from < by \((t_1 = t_2) =_{\text{df}} (t_1 \ll t_2) \land (t_2 \ll t_1)\). That is not the case here.

\[ \vdash \]

A review of our progress may not be out of place here. First we considered occurrences of events and their causality. This lead us to a partially ordered set of starts and finishes of event occurrences. Then we introduced branching time, incorporating both causality and real-timing. An order over branching time, \(\ll\), was introduced, which respected both the order over the reals < and the causal order \(\ll\); other orders over BT and R were then defined using \(\ll\) and <.
1.0.3 Examples of causal and temporal structures

The causal part of our model bears some resemblance to the poset tradition of concurrency theories as exemplified by [Boudol & Castellani 1988], [Gischer 1989], [Lamport 1978], [Pratt 1986], [Shields 1988] and [Winskel 1986]. There are several examples of posets which often appear in the literature to illustrate particular behaviours. In this subsection we shall show one way of representing such behaviours in our model, by displaying appropriate branching times. *(Notational aside: we will write \( t_{(1)} \) for \( (t_1) \).)*

The first example we shall consider has two occurrences of events, with one occurrence happening wholly before the other. In this case \( LE = \{a_1, a_2\} \) and the ordering on branching times is \( t_{(1)} < b t_{(1)} < b t_{(2)} < b t_{(2)} \). We also have \( *_s < b t_{(1)} \) and \( t_{(2)} < b *_f \). These relationships come about because of the causality of the situation, \( a_1 < C a_1 \) \( a_2 < C a_2 \), and the timing \( r_{(1)} < r_{(1)} < r_{(2)} < r_{(2)} \). We can picture it thus:

| \( t_{(1)} \) \( a_1 \) \( t_{(1)} \) \( a_2 \) \( t_{(2)} \) |
|---|---|---|---|

A poset exhibiting *sequentiality*

Here time flows across the page and the transitions of the labelled events \( a_1 \) and \( a_2 \), \( t_{(1)} \) & \( t_{(2)} \), \( t_{(2)} \) & \( t_{(2)} \) respectively, are indicated. In all our examples we will omit treatment of the silent event \( * \) unless it is didactically useful.

Now consider a structure with the same events but with different causality. If we just had \( t_{(1)} < b t_{(1)} \) and \( t_{(2)} < b t_{(2)} \) then we would not be able to deduce a causal relationship between \( a_1 \) and \( a_2 \): we might have \( r_{(1)} < r_{(1)} < r_{(2)} < r_{(2)} \) (the same picture) and \( t_{(1)} \) inc \( t_{(2)} \) (different causality).

If we had no relationship between the branching times of \( a_1 \) and \( a_2 \), and \( r_{(1)} < r_{(2)} < r_{(1)} < r_{(2)} \) then \( a_1 \) and \( a_2 \) are actually simultaneous but on different branches (i.e. possibly in different places).

We can picture this as

| \( t_{(1)} \) \( a_1 \) \( t_{(1)} \) |
|---|---|---|
| \( t_{(2)} \) \( a_2 \) \( t_{(2)} \) |

A poset exhibiting *incomparable, simultaneous occurrences*
One of the standard perverse posets in the literature is the \textit{N-poset}, whose causality we might represent thus: \(a_i < a_{i(1)} < a_{i(2)} < a_i(3)\) and \(a_i < a_{i(1)} < a_{i(2)} < a_{i(3)} < a_{i(4)} < a_{i(3)} < a_{i(3)}\). A suitable diagram, representing causality by a thin dashed line might be

\[
\begin{array}{c}
t_1(1) & t_1(4) & t_2(3) & t_3(2) & t_2(4) & a_1 & a_2 & a_3 & a_4 \\
\end{array}
\]

(The ‘N’ can be seen by turning the page through ninety degrees.)

This poset is important because the absence of it embedded in a given poset is a necessary condition for a decomposition theorem to hold; cf., for instance, \cite{Boudol1987} for details. (It is also related to the notion of \textit{K-density} in Petri nets; cf. \cite{Plumpecke1984}. We shall delay a consideration of Petri nets until chapter 4. Meanwhile, background, such as \cite{Reisig1985} might be useful to the reader.)

The (non-trivial) timing we require of the N-poset in order for the branching order to reflect the causal order is \(r_{1(1)} < r_{1(2)}, r_{1(1)} < r_{1(3)}\) and \(r_{1(4)} < r_{1(3)}\). Notice that this structure displays sequentiality (as between \(a_1\) and \(a_2\)), and incomparability (as between \(t_{1(4)}\) and \(t_{1(2)}\)).

Some more perverse posets will be definable when the notion of conflict is introduced in section 1.4.

Notice, before leaving events and their causality, that nowhere have we demanded that every occurrence of an event has the same causality. In fact, we have not made any demands on the behaviour of different occurrences of the same event yet.

Thus far, we have only informally associated a branching time with the start and finish of occurrences of events; we go on to make this more formal in the next section.
1.1 Structures in time

The last section gave us causality, timing and branching time, the third incorporating features of the first two. This section will deal with the association of points in branching time to transitions. In particular, branching interval event structures will be defined; each occurrence of an event in such a structure will have two associated points in branching time representing the times that occurrence started and finished, and the associated causality. Linear interval event structures, with the timing, but without the causality, will be recovered as a special case. Of necessity this section contains much repetition; I hope this does not impose too much strain on the reader's patience.

**DEFINITION 1.1 – Events and their beginning and ending times**

Consider a finite, non-empty set of events, \( \mathcal{E} \), and an observer. Every event \( e_i \in \mathcal{E} \) is considered to have a duration which is observed, and each event \( e_i \) can be distinguished from any other \( e_j \in \mathcal{E} \) with \( i \neq j \).* The observer has a clock which is used to record the time each occurrence of an event begins and ends, so that time is notionally globally observed. Thus a possibly empty set of pairs can be associated with each event \( e_i \) recording the beginning and ending times of each occurrence of \( e_i \). (Aside: notice that events can happen more than once.)

The 'special happening' * will not be an event.

The assumption that time could be globally observed is not as damning as it looks; these times are just those assigned by some observer; we make no assumption about global clocks which could in any way influence the system itself. Cf. the discussion in [Petri 1986].

We shall allow for the possibility that these times are not yet fixed using top, \( T \), the time which dominates every other, to indicate this. This will entail an expansion of branching time; define \( \mathcal{BT}_a = (\mathcal{R} \cup \{T\}) \times \mathcal{T} \). Things that haven’t happened yet are in the future, so that \( \forall r \in \mathcal{R}, r < T \). There will not be much need to represent structures with undefined times until chapter three, so the reader can safely ignore them until then.

A labelling function associates with each occurrence of an event a unique label \( l \). Let the set \( \mathcal{L} \) be the set of all labels. The set of all possible labelled events is \( \mathcal{LE} = (\mathcal{L} \times \mathcal{E}) \cup \{*\} \), of which some subset \( \mathcal{LE} \subseteq \mathcal{LE} \) (with \( * \in \mathcal{LE} \) will be of interest. For each \( a \in \mathcal{LE} \) there is a pair \((t_b, t_e)\) associated with \( a \), such that \( t_b \) is the branching time \( a \) began, and \( t_e \) the branching time it ended (i.e. the times of the transitions of that occurrence of the event).

* – This isn’t as tautological as it looks; there are difficulties in using an uncountable set of events precisely because then it is not clear when two events are physically distinct and distinguishable. We wish to restrict ourselves to the (Newtonian) situation where we can always distinguish between two different events.

If we wish to consider a non-Newtonian world, we might wish to think of the causal part of the model as specifying physically necessary relationships between occurrences, and the timing part as indicating what observations a single observer might make.
The real component of these branching times represents when the transition occurred, while the tag field represents the causality present. Notice that * is a labelled event.

The 1-to-1 functions \( \text{begin}(a) \) and \( \text{end}(a) \) are assumed to return these branching times (or just the real part of them, if causality isn’t available, as in a linear model).

\[
\begin{align*}
\text{begin}, \text{end} : \text{LE} & \rightarrow \text{BT} \\
\text{It will be convenient to suppose that LE is countable (i.e. at worst that L is countable). For convenience suppose further that } L \cap E = \emptyset \text{ and that } * \in L \times E. \text{ LE is always nonempty; it has at least } * \text{ in it.}
\end{align*}
\]

Only a countable set of branching times will be in the range of \( \text{begin} \) and \( \text{end} \), since \( \text{LE} \) is countable. This set will be denoted by \( \text{BT} \); it is the set of branching times at which a transition occurs.

\[
\text{BT} = \{ t_s \mid \text{begin}(a) = t_s, \ a \in \text{LE} \} \cup \{ t_f \mid \text{end}(a) = t_f, \ a \in \text{LE} \}
\]

We now state the structure we have diagrammatically.

\begin{align*}
\text{Events, } E & \quad \text{Labelled Events, } \text{LE} & \quad \text{Labels, } L \\
e \in E & \quad \quad a \in \text{LE} \subseteq (L \times E) \cup \{ * \} & \quad \quad l \in L \\
\downarrow & \quad \quad \text{begin}(a) = t_{(0)}, \ \text{end}(a) = t_{(0)} & \quad \quad \downarrow \\
\text{Branching Time with reals } R \text{ and tags } Tr = (\text{LE}_s \cup \text{LE}_f) & \quad \quad \text{BT} \subseteq \text{BT} = (R \cup \{ T \}) \times Tr \\
(t_{(0)}, a_i) \in \text{BT} & \quad \quad (t_{(0)}, t_{(0)}) \in (\text{BT} \times \text{BT})
\end{align*}

\(<_s \text{ respects both }<_s \text{ (over } \text{LE}_s \cup \text{LE}_f) \text{ and } < \text{ (over } R) \\
t_s < t_b \iff (t_s < t_b \ \& \ a_s < b_s)
\]

And similarly \(=_s \text{ respects both } =_s \text{ (over } \text{LE}_s \cup \text{LE}_f) \text{ and } = \text{ (over } R) \\
Further \langle _<_s \text{ and } _<_s \text{ are consistent;}
\]

\[a_i <_s b_i \Rightarrow r_{(i)} < r_{(i)} \text{ and } a_i =_s b_i \Rightarrow r_{(i)} = r_{(i)}\]

\text{DEFINITION 1.2 -- Elementary Branching and Linear Interval Event Structures}

An elementary linear interval event structure (E.L.I.E.S.) \( L \) is a pair \( (\text{LE}, <) \) consisting of a set of labelled events and a strict global precedence order \(<\), the usual one on \( R \), on their (linear) times. Linear structures have no causality, so the \( \text{begin} \) and \( \text{end} \) functions, which are assumed to be available, pick out just the real-valued part of branching times, so that \( \text{begin}, \text{end} : \text{LE} \rightarrow R \cup \{ T \} \) (the particular meaning of the overloaded \( \text{begin} \) and \( \text{end} \) is always obvious from the context).
An elementary branching interval event structure, \(\mathcal{B}\), is a pair \((\mathcal{L}, \prec)\) consisting of a set \(\mathcal{L}\) of labelled events and a strict partial order \(\prec\) over \(\mathcal{B}\), together with the projection functions \(\text{begin}, \text{end} : \mathcal{L} \rightarrow \mathcal{B}\) and the equivalence \(\equiv\) over branching time.

The total order \(<\) on the reals will also be used freely. When we want to compare just the real parts of branching times we use \(<\), writing \(\text{begin}(a) < \text{begin}(b)\) to mean \(r_{a(0)} < r_{b(0)}\) where \(\text{begin}(a) = (r_{a(0)}, a_0)\) and \(\text{begin}(b) = (r_{b(0)}, b_0)\).

For \(t \in \mathcal{B}\) we write \(t\uparrow\) if \(t\) is defined (that is, is not \(T\)), and \(t\downarrow\) otherwise. Because of the requirements that starts cause finishes,

\[
a_0 < a_t
\]

and that \(<\) is compatible with \(<\),

\[
a_0 < b_t \Rightarrow r_{a(0)} < r_{b(0)}
\]

\(<\) is nonstrict on \(T\) (i.e. \(T < T\)) so that \(\text{begin}(a) = (T, a) \Rightarrow \text{end}(a) = (T, a_t)\).

We require that \(t_0 < b_t\) (or, exceptionally, \(t_0 = b_t\)) so that events must begin before they end (this follows from the similar property of \(<\)). We will write \(a = [t_0, t_1]\) for the assertion that \(a\) is timed by \((t_0, t_1)\), and we intend the reader to associate \(a\) with the closed time interval \([r_a, r_t]\) where the branching times \(t_a = (r_a, a_0), t_t = (r_t, a_t)\). When we want to ignore the branching structure (or when we don’t have it, as in a L.I.E.S.) we will write \(a = [r_a, r_t]\) for the assertion that \(a = [(r_a, a_0), (r_t, a_t)]\).

Our slogan thus far might be;

\[
\text{The beginning and end times of occurrences of events are associated with unique branching times, that is, real numbers with tags.}
\]

We intend the assertion \(t_{(1)} < t_{(2)}\) to represent the fact that the event timed (in some sense) by \(t_{(1)}\) must, perhaps because of causality, happen before that timed by \(t_{(2)}\). Suppose that \(a_1 = [t_{(1)}, t_{(1)}]\) and \(a_2 = [t_{(2)}, t_{(2)}]\); then, if we want \(a_1\) to start before \(a_2\) we might write \(t_{(1)} < t_{(2)}\). We shall represent the order upon branching times, \(<_b\), as an order on events in section 1.2 below. Thus far we have a means for talking about what temporal orders must be observed in a system, \(<_b\), and one that enables us to talk about ones that can be observed or are observed \(<\); it is in this sense that L.I.E.S.s are observations of B.I.E.S.s; some events in a B.I.E.S. may not be ordered with respect to each other (hence \(\text{inc}\)), but all events are ordered in a L.I.E.S.

A B.I.E.S. can represent just the orderings we want to specify and leave the others unspecified. A B.I.E.S., then, is a model of something, a process say, that can be executed.

All observed events must happen sometime, (i.e. happenings are linearly ordered by time of occurrence) so a L.I.E.S. is a good model of what is actually observed during one of these executions.
A B.I.E.S. can display genuine concurrency (indicated via the inc predicate) while a L.I.E.S. can display merely the overlap of occurrences of events.

Our requirement that the real order respects the causal order should now be restated. Remember that \(a_i\) is the beginning or the end of some \(a \in LE\), and similarly for \(b_i\). Then if \(a_i \prec b_i\) we require that the branching times given to \(a_i\) and \(b_i\), \(\tau(a_i) = (r(a_i), a_i)\) and \(\tau(b_i) = (r(b_i), b_i)\) respectively, must obey \(r(a_i) < r(b_i)\) so that \(\tau(a_i) \prec \tau(b_i)\). Note that this means that nothing can happen before the thing that caused it. Similarly if \(a_i = c b_i\) then \(r(a_i) = r(b_i)\) so that \(\tau(a_i) = \tau(b_i)\).

**Definition 1.3** – Inclusion Ordering, Overlap and Simultaneity

An inclusion ordering on events can be defined; for any \(a_1 = [t_1(j), t_1(f)]\), \(a_2 = [t_2(j), t_2(f)]\) define \(a_1 \subseteq a_2\) iff \(t_1(j) \leq t_2(j)\) & \(t_1(f) \leq t_2(f)\) (i.e. \(a_1\) and \(a_2\) are on the same branch and the times of \(a_1\) are totally contained in those of \(a_2\)). Similarly if \(t_1(j) \leq t_2(j) \leq t_1(f) \leq t_2(f)\) then we write \(a_1 \sqsubset a_2\) (i.e. \(a_1\) and \(a_2\) are on the same branch, \(a_1\) starts first and overlaps only some of \(a_2\)).

If the relationship holds on linear (but not necessarily branching) time, then write \(a_1 \preceq a_2\) and \(a_1 \supseteq a_2\) (i.e. \(a_1 \preceq a_2 \iff (r_1(j) < r_2(j) \& r_1(f) < r_2(f))\) and \(a_1 \supseteq a_2 \iff (r_1(j) \leq r_2(j) \leq r_1(f) \leq r_2(f))\)). If \(a_1 \preceq a_2\) or \(a_1 \supseteq a_2\) or \(a_2 \subseteq a_1\) or \(a_2 \supseteq a_1\) then \(a_1\) and \(a_2\) are said to be *locally simultaneous*; not only do their times overlap, but they are on the same branch, whereas if we have merely that \([r_1(j), r_1(f)] \cap [r_2(j), r_2(f)] = \emptyset\) (i.e. one of \(a_1 \preceq a_2\) or \(a_1 \supseteq a_2\) or \(a_2 \subseteq a_1\) or \(a_2 \supseteq a_1\)) then we say that \(a_1\) and \(a_2\) are *simultaneous*; their times overlap, but they may not be on the same branch. (The symbol \(\subseteq\) is unfortunate since \(!\neg(a_1 \subseteq a_1)\), but \(\subseteq\) is reserved for other purposes.)

**Aside** – Point-like events

At the moment we have forbidden *point-like* events where, for instance, \(a = [t, t]\), since we enforced \(a_1 \prec a_2\) and hence \(t(a_1) < t(a_2)\). We shall sometimes allow point-like events in explaining an idea, but we do not intend them as anything other than a philosophical nicety.

**Definition 1.4** – The Outside Relation

For any \(a_1 = [t_1(j), t_1(f)], a_2 = [t_2(j), t_2(f)]\), we say \(a_1\) is *outside* \(a_2\), written \(a_1 \not\subset a_2\), iff no labelled event \(a\) can be defined so that \(a \subset a_1\) and \(a \subset a_2\). From the definition it follows that \(a_1 \not\subset a_2\) \(\iff a_2 \not\subset a_1\). If \(a_1 \not\subset a_2\) then either \(a_1\) and \(a_2\) are on different branches, or if they are on the same branch, the intervals of time they occupy do not overlap. Notice that for any pair of events \(a_1\) and \(a_2\) with \(t_1(j) \leq t_2(j)\) either \(a_1 \not\subset a_2\) or \(a_1 \subset a_2\) or \(a_1 \supseteq a_2\). To see this note that the truth of \(a_1 \not\subset a_2\) requires that \(\neg \exists t \in BT. t_1(j) \leq t \leq t_1(f) \& t_2(j) \leq t \leq t_2(f)\). (Parenthetically, note that some care of the difference between \(\leq\) and \(\prec\) is needed in definitions like these, to ensure that \(\subset\), \(\supseteq\) and \(\not\subset\) are in fact disjoint.) Again, if the outside relation holds in linear but not in branching time then we write \(a_1 \not\subset a_2\) \(\iff t_1(j) \leq t_2(j) \leq t_1(f) \& t_2(j) \leq t_1(j)\).
EXAMPLE – The Usual One

A vending machine may be characterised thus;

\[ E = \bigcup_{i \in \mathbb{N}} \{ \text{coin}_i, \text{choc}_i \}, \text{LE} = \bigcup_{i \in \mathbb{N}} \{ \text{coin}_i, \text{choc}_i \}, i \in \mathbb{N} \]

We have abused the notation slightly, writing \( \text{coin}_i \) for the pair \( (i, \text{coin}) \). Notice that the order on the labelling set (here the \text{succ} ordering on the integers) is entirely incidental; we do not need it, although it is convenient. All we need is a way of telling which \text{choc} event matches which \text{coin} event and which \text{coin} event follows which \text{choc} event.

The behaviour of the vending machine might specified thus:

\[ \forall i. \text{end}(\text{coin}_i) <_b \text{begin}(\text{choc}_i) \& \text{end}(\text{choc}_i) <_b \text{begin}(\text{coin}_{i+1}) \]

One process which may be viewed as satisfying the observation above (in a sense to be made precise later) is the familiar CSP vending machine, \( \text{VM} = \text{VM}_0 \):

\[ \text{VM}_i = \text{def} \text{coin}_i \rightarrow \text{choc}_i \rightarrow \text{VM}_{i+1} \]

We can now, of course, articulate some real-time requirements. Assuming our units of time are minutes, a vending machine that completes a transaction within thirty seconds will (together with the customer) obey the requirement

\[ \text{end}(\text{choc}_i) - \text{begin}(\text{coin}_i) \leq \frac{1}{2} \]

while a machine that shows unconscionable lethargy if the customer is in a hurry will obey

\[ \text{end}(\text{coin}_i) - \text{begin}(\text{coin}_n) \leq \frac{1}{60} \Rightarrow \text{end}(\text{choc}_i) - \text{begin}(\text{choc}_i) \geq 1 \]

Consider \( \text{choc}_i = [t_{40}, t_{60}] \). We can make the vending machine serve customers between 10am and 2am by requiring that, if we start \( (t = 0) \) at midnight, then \( t_{40} \mod 1440 < 120 \) or \( t_{60} \mod 1440 > 1320 \) implies \( t_{60} > 120 \). Notice we have imposed a “I’ve started so I’ll finish” convention allowing users whose transactions are straddled by the 2am borderline to complete them normally.

A SOMEWHAT MORE INTERESTING EXAMPLE

The typical bar is a good example of a concurrent system. Suppose we have a countable number of drinkers (more would give a bar that is too dense) and a single barman. The labelled events we shall consider are

\[ \text{LE} = \text{def} \{ \text{drink}_{i,j}, \text{tussle}_{i,j}, \text{serve}_{i,j} \}, i, j \in \mathbb{N} \]

for a habitué \( i \) drinking his \( j^{th} \) drink, tussling at the bar for the \( j^{th} \) time and being served for the \( j^{th} \) time. Clearly in order to get a drink someone must tussle for it and be served.
We will assume drinking in rounds, so that one cannot drink until someone is served that round

\[ \forall i, j. \exists i'. \text{begin(serve}_{i,j}) < _b \text{begin(drink}_{i,j}) \]

Altruism is regretfully uncommon here; one drinks before queueing and queues before being served

\[ \forall i, j. \text{end(drink}_{i,j}) < _b \text{begin(tussle}_{i,j}) \]

Finally we shall impose one fair \( ^\dagger \) barman

\[ \forall i, j. \text{begin(tussle}_{i,j}) \uparrow \Rightarrow \text{end(tussle}_{i,j}) \uparrow \]
\[ \forall i, j. \text{begin(serve}_{i,j}) \uparrow \Rightarrow \text{end(serve}_{i,j}) \uparrow \]
\[ \forall i, j. \text{end(tussle}_{i,j}) = _b \text{begin(serve}_{i,j}) \]
\[ \forall i, i', j, j'. \text{begin(tussle}_{i,j}) < \text{begin(tussle}_{i',j'}) \Rightarrow \text{begin(serve}_{i,j}) < \text{begin(serve}_{i',j'}) \]

The conditions are, respectively, finite waiting time, finite serving time, starting to be served as soon as finishing waiting, and a fair ordering of serving by the barman. (We will have some more to say about how the barman decides who to serve in chapter four.)

Here we have written constraints upon the begin and end times of events; in effect specifications involving the timing of labelled events. In order for conventional specification techniques to be related to this model (so that we can determine when an interval event structures implements an atomic-event specification, for instance) we need to shift the orientation of the specification to labelled events themselves rather than their branching times. We shall begin this in the next section using event orders, which are relations upon events. Whether these relations hold or not is determined by the begin and end times of the occurrence of the event, so we shall preserve the ability to consider detail similar to that of this section, while shifting the level of abstraction from the times that define labelled events to the labelled events themselves.

We have focused upon the beginning and ending times of events as things we wish to reason about, thus, implicitly, limiting ourselves to a dyadic or interval logic \cite{van Benthem 1983}. Any further delving into the internal structure of events (for instance being interested in other points of time during the execution of an event) can be dealt with by event refinement, as discussed in chapter five.

\( ^\dagger \) In common with many drinkers our notion of fairness is stronger than usual.
Notice also that specific reference to constants has been mostly avoided (saying "seventeen milliseconds later..."), – this is merely to keep the parallel with conventional specification methods clear thus far (by concentrating upon causality rather than time difference) and will be remedied later (when we treat specification with a little less dalliance). A good way of considering timing is to see how to time the examples of section 1.0.3, but this will be delayed until a little more theory has been presented.

**DEFINITION 1.5 – Computational structures**

The philosophical position chosen for I.E.S.s is that of a model of distributed happenings. Those happenings may fulfil a computational purpose. A subclass of I.E.S.s is a reasonable model of computation; we will forbid more than a finite number of labelled events happening in a finite time (so as to disallow what [Joseph & Goswami 1985] call ‘Zeno machines,’ machines which can do an infinite computation in a finite time). If

\[
\forall r_1, r_2 \in R \mid \{ a \mid a \in \text{LE}, a = [(r_1, a_1), (r_2, a_2)], [r_1, r_2] \cap [r_1, r_2] \neq \emptyset \} \mid \text{is finite,}
\]

then the structure concerned is said to have finite density. If, in addition,

\[
\forall t \in \text{BT} \mid \{ t' \mid t' \preceq t, t' \in \text{BT} \} \mid \text{is finite,}
\]

then it is said to satisfy the axiom of finite causes. The axiom of finite causes requires that no transition causally depends on an infinite sequence of previous transitions. It is discussed in [Winskel 1980].

If a structure has finite density, satisfies the axiom of finite causes, and has well-defined times,

\[
\forall a \in \text{LE} - \{ * \}. \begin{align*}
\text{begin}(a) & \uparrow & \text{end}(a) & \uparrow \\
n & \end{align*}
\]

then it is said to be computational.

(The class of all I.E.S.s, IES say, and the class of computational I.E.S.s, cIES, have not yet been properly defined; this is postponed until chapter two.)

Now that the sound and fury of the main definitions is over, it is possible to remark without causing too much confusion that the branching time model can be thought of as a separate entity from the I.E.S. model which is built on top of it. Branching time, $\text{BT} = R \cup \{ T \} \times \text{Tr}$, can be used for much more than just building B.I.E.S.s. Here that use alone will be pursued, by identifying tags with the transitions of labelled events, that is elements of $\text{Tr}$, but that is not necessary, – $\text{BT}$ could, for instance, be a basis for the investigations of temporal structures carried out in [van Benthem 1983], if different tags were used.
1.2 Orders on Labelled Events

It is useful to be able to specify the relations between labelled events somewhat more abstractly than is possible using the total order < or the partial order <\textsubscript{b}. The following partial orders correspond to some common situations where one event enables another. Let \(a_1 = [t_{0(1)}, t_{0(1)}]\), and \(a_2 = [t_{0(2)}, t_{0(2)}]\) be labelled events. Then the following preorders on labelled events are defined:

1. \(a_1 \preceq_{\text{h}} a_2 \iff t_{0(1)} \leq t_{0(2)}\)
2. \(a_1 \preceq_{\text{i}} a_2 \iff t_{0(1)} \leq t_{0(2)}\)
3. \(a_1 \preceq_{\text{e}} a_2 \iff t_{0(1)} \leq t_{0(2)}\)
4. \(a_1 \preceq_{\text{t}} a_2 \iff t_{0(1)} \leq t_{0(2)}\)

The four preorders, \(\preceq_{\text{h}}, \preceq_{\text{i}}, \preceq_{\text{e}}, \text{ and } \preceq_{\text{t}}\), reflect four notions of enabling (or causality):

1. \(a_1\) is head-causal of \(a_2\) when the beginning of \(a_1\) causes the beginning of \(a_2\), written \(a_1 \preceq_{\text{h}} a_2\).
2. \(a_1\) is interior-causal of \(a_2\) when the end of \(a_1\) causes the beginning of \(a_2\), written \(a_1 \preceq_{\text{i}} a_2\).
3. \(a_1\) is exterior-causal of \(a_2\) when the beginning of \(a_1\) causes the end of \(a_2\), written \(a_1 \preceq_{\text{e}} a_2\).
4. \(a_1\) is tail-causal of \(a_2\) when the end of \(a_1\) causes the end of \(a_2\), written \(a_1 \preceq_{\text{t}} a_2\).

The diagram above uses a black blob for the enabling beginning or ending of an interval and a white one for the non-causative one.

We write \(\preceq_j\) for one of \(\preceq_{\text{h}}, \preceq_{\text{i}}, \preceq_{\text{e}}, \preceq_{\text{t}}\). The orders \(\preceq_j\) are called \(j\)-morphisms or \(j\)-orders.

Properties of \(\preceq_j\)

1. For point-like events all the \(\preceq_j\) are the same. If \(a_1 = [t_{0(1)}, t_{0(1)}]\) and \(a_2 = [t_{0(2)}, t_{0(2)}]\), then \((a_1 \preceq_{\text{h}} a_2) \iff (a_1 \preceq_{\text{i}} a_2) \iff (a_1 \preceq_{\text{e}} a_2) \iff (a_1 \preceq_{\text{t}} a_2)\).

2. Where one of the events is point-like some discrimination reappears. If \(a_1 = [t_{0(1)}, t_{0(1)}]\) and \(a_2 = [t_{0(2)}, t_{0(2)}]\), then \((a_1 \preceq_{\text{h}} a_2) \iff (a_1 \preceq_{\text{e}} a_2)\) and \((a_1 \preceq_{\text{t}} a_2) \iff (a_1 \preceq_{\text{t}} a_2)\).

3. \(\preceq_{\text{h}}, \preceq_{\text{i}}, \text{ and } \preceq_{\text{t}}\) are preorders, as is easily verified from the definition. \(\preceq_{\text{e}}\) is not transitive. A further discussion of the properties of the \(\preceq_j\) is postponed for the moment.
1.3 Primitive Concurrency

We need to decide what it means for events to be concurrent. Clearly they are concurrent if their times, in the branching model of course, are not ordered. But we can imagine a situation (see diagram) where the start or end times of two genuinely concurrent events are ordered.

First a notation for the abutment of labelled events is introduced; write \( a \mathsf{adj} a_2 \iff \text{end}(a_1) = \text{begin}(a_2) \) (i.e. \( a_1 \) is adjacent to \( a_2 \)). Now consider:

*If \( a \sqsubseteq b \) and \( a \sqsubseteq c \) then we may very well have \( a \mathsf{adj} b \) and \( a \mathsf{adj} c \). (Suppose \( a = [t(a), t_0(a)] \), \( b = [t(b), t_0(b)] \), \( c = [t(c), t_0(c)] \). For various reasons we may want \( t(f(a)) = b \) \( t(f(b)) = a \) \( t(f(c)) = c \) as well as the obvious \( r(a) = r(b) = r(c) \). For this reason we formulate our definition of \( \mathsf{primitive} \) event concurrency predicate as

\[
(a_1, a_2) \iff \exists t_1 \in a_1, t_2 \in a_2 \cdot t_1 \text{inc} t_2
\]

(Where we write \( \exists t \in a \) for \( a = [t_l, t_r] \) \& \( t = t_l \) or \( t = t_r \).)

Two labelled events are primitively concurrent iff they are on different branches and (so cannot be said to be completely causally related to each other). Concurrency is entirely unrelated to simultaneity; in general \( -((a_1, a_2) \Rightarrow (a_1 \subseteq a_2)) \& -((a_1 \subseteq a_2) \Rightarrow (a_1, a_2)) \).

Primitive concurrency is not hereditary either; \( (a_1, a_2) \& a_1 \sqsubseteq a_1' \& a_2 \sqsubseteq a_2' \) does not imply \( (a_1', a_2) \). (This is because our time branches can join up again; we do not have the situation [Winskel 1986] calls ‘stable.’)

The figure above indicates why we need branching time equivalence \( \equiv_b \); we might well have \( \text{end}(b) = \text{end}(c) \) but, if we want \( b \) and \( c \) to be on different branches, then \( \text{end}(b) \neq_b \text{end}(c) \) should hold. (It is probably misleading to think of branching time as really tree-like, since branches can join as well as split, and since more than one occurrence can be simultaneously active in one branch.)

Notice, incidentally, that the predicates \( \sqsubseteq_b \), \( \sqsupseteq_b \) and \( \supseteq_b \) of definitions 1.3 and 1.4 can be expressed much more succinctly using the orders \( \sqsubseteq \) and primitive-concurrency; \( a_1 \sqsubseteq_b a_2 \Rightarrow a_2 \sqsubseteq_b a_1 \wedge a_1 \sqsubseteq a_2 \\
a_1 \sqsupseteq_b a_2 \iff a_1 \sqsubseteq_b a_2 \wedge a_1 \sqsubseteq a_2 \\
a_1 \supseteq_b a_2 \Rightarrow (a_1, a_2) \vee a_1 \sqsubseteq a_2 \vee a_2 \sqsubseteq a_1 \\

\( \sqsubseteq \) The first and third relations don’t hold with \( \iff \) since the orders \( \sqsubseteq \) use \( \leq \) rather than \( < \). A correct version of the last relation with \( \iff \) instead of \( \Rightarrow \) would have to conjoin a condition that \( \text{end}(a_1) \neq \text{begin}(a_2) \) to the middle disjunct.
1.4 Specification and Inhibition

We will now deal with the matter of specification a little more carefully. To capture the sorts of relations on times that were used in the first section, we shall use the abstractions introduced above. A labelled event $a_1$ may $j$-cause another labelled event $a_2$, written $a_1 \sqsubseteq_j a_2$. An elementary positive specification, $S$, involving labelled events LE and orders $\sqsubseteq_j$, is just a conjunction of requirements like $a_1 \sqsubseteq_j a_2$:

$$S := \text{LE} \sqsubseteq_j \text{LE} \lor S \land S$$

indicating that the obvious set of orderings upon events should be present in the implementing structure.

A negative specification expresses the notion that labelled event $a_1$ may inhibit the occurrence of another labelled event $a_2$, possibly because of competition for a resource or because the two events are mutually exclusive choices, or for other reasons. If this is the case we write $a_1 \# a_2$. Notice that $-\left((a_1 \sqsubseteq_j a_2)\right)$ merely captures the notion that $a_1$ does not $j$-cause $a_2$. It does not capture the idea of active inhibition; even if $-\left((a_1 \sqsubseteq_j a_2)\right)$, $a_2$ may be simultaneous (in linear time) with $a_1$ for other reasons. If $a_1 \# a_2$ an occurrence of $a_1$ should guarantee that $a_2$ does not occur concurrently. Notice that inhibition, like concurrency, is symmetric; $a_1 \# a_2$ implies that $a_2 \# a_1$. (In fact, $\#$ is very similar to the conflict relation in Petri net theory; it will be formally defined shortly.)

We need a notion of which branches of those which can be concurrent actually are concurrent: that is, we need to know which branches may be seen occurring together in some particular execution, and which branches can never be seen occurring together. The consistency predicate is introduced to meet this need. (The intuition behind the notion of different simultaneous branches is simply that a system might consist of several infrequently-interacting pieces such as distributed subsystems. Each of these pieces has its own causality and forms a branch.)

A consistency predicate, written $\text{Con}$, is a subset of the set of non-empty subsets of LE, written $\text{Con} \subseteq \wp(\text{LE})$. It is intended that two labelled events coexist in some member of $\text{Con}$ just when they can occur in the same history. An element of $\text{Con}$, then, is a set of events that can happen in one ‘execution’ of an I.E.S.; it constrains which labelled events on different branches can happen together. To prevent perversities, we require that $\text{Con}$ covers LE.

If a set $X$ of labelled events is consistent in a history, then any set $Y \subseteq X$ should also be consistent, so we have the subset closure requirement

$$X \in \text{Con} \& Y \subseteq X \Rightarrow Y \in \text{Con}$$

Our intention that $\text{Con}$ expresses conflict means that conflicting occurrences should be on different branches. If we had $a_1$ and $a_2$ with $a_1 \sqsubseteq_j a_2$ (i.e. $a_1$ and $a_2$ are on the same branch and
if $a_1$ happen wholly first) then it is hard to see what interpretation to place on $a_1 \neq a_2$. For this reason we will require that

$$\forall a_1, a_2 . (a_1 \neq a_2) \Rightarrow (a_1, a_2)$$

which enforces the restriction that conflicting events be on different branches. (This is a rather strong requirement; it means that things on the same branch cannot be in conflict. Some comfort may be obtained from the fact that not much of the theory depends on this requirement. The intuition is that if two things are causally related, then they cannot be in conflict.) The link between $\#$ and $\text{Con}$ is obvious; the negative specification $a_1 \neq a_2$ is just a constraint on possible consistency predicates;

$$(a_1 \neq a_2) \Rightarrow (a_1, a_2) \in \text{Con}$$

Finally, if two labelled events are causally the same, then they should be in the same $\text{Con}$-sets;

$$\forall c \in \text{Con}. (a_1 \in c \& a_1(1) =_c a_2(1) \& a_1(2) =_c a_2(2)) \Rightarrow c \cup \{a_2\} \in \text{Con}$$

Notice that our definition of conflict allows for events which retry if a conflict of one occurrence is not resolved in its favour, and for ones which give up. The use of the conflict relation between labelled events rather than just events gives us this power. (It is possible to label events so that the impossibility of execution of two labelled events concurrently is equivalent to the exclusion of certain labelled events from execution in the same history; by this means our $\text{Con}$ over labelled events can act like Winskel's $\text{Con}$ over events [Winskel 1986], or as a means of forbidding certain occurrences.)

The branching interval event structure is now taken as the standard object (because linear structures are just special cases of branching ones). We will keep the concept of a linear event structure as an observation of a branching structure. (Thus all observations are L.I.E.S.s)

**DEFINITION 1.6 - Interval Event Structures**

An interval event structure (I.E.S.) is defined to be a elementary B.I.E.S. with conflict, i.e. a triple $(\text{LE} \subseteq (L \times E) \cup \{\star\}, \text{Con}, \preceq)$ together with all the structure indicated on page 41. (So that, for instance, it is assumed that $L$ and $E$ have been defined, so that $\text{BT}$ and $\text{LE}$ are defined, $\text{begin}$ and $\text{end}$ are available, and $\Rightarrow$ has been defined. We stress only $\text{LE}$, $\text{Con}$ and $\preceq$ in writing the structure for brevity only; all of the other structure must be present too.)

Most of our notation was summarised on page 41; the remainder can be found on page 60.
DEFINITION 1.7 – Concurrency and Conflict

The primitive concurrency definition can be altered so that conflicting events are not concurrent, giving a more sophisticated definition of event concurrency. The labelled event \( a_1 \) is concurrent with \( a_2 \) if it is consistent and primitively concurrent with it:

\[
\neg \ldotp \quad a_1 \cong a_2 =_{df} ((a_1 , a_2) \land \{a_1, a_2\} \in \text{Con})
\]

As usual in concurrency models, \( \cong \) is symmetric, irreflexive and not necessarily transitive. Notice that if \((a_1 , a_2) \land \{a_1, a_2\} \in \text{Con}\) then \(a_1\) and \(a_2\) are in conflict;

\[
\neg \ldotp \quad a_1 \not\equiv a_2 =_{df} ((a_1 , a_2) \land \{a_1, a_2\} \not\in \text{Con})
\]

Consider two labelled events, \( a_1 \) and \( a_2 \) say. Then precisely one of the following six statements holds

(i) \( a_1 \not\equiv a_2 \) (that is, \( a_1 \) and \( a_2 \) are on different inconsistent branches)

(ii) \( a_1 \equiv a_2 \) (that is, \( a_1 \) and \( a_2 \) are on different consistent branches)

(iii) \( \neg(a_1 \land a_2) \not\equiv a_1 \not\equiv a_2 \) (that is, \( a_1 \) and \( a_2 \) are on the same branch, and do not overlap)

(iv) \( a_1 \equiv a_2 \) (ditto, and \( a_1 \) happens first, but overlaps only some of \( a_2 \)) or

(v) \( a_2 \equiv a_1 \) (ditto, and \( a_2 \) happens first, but overlaps only some of \( a_1 \))

(vi) \( a_1 \equiv a_2 \) (ditto, and \( a_1 \) happens within \( a_2 \))

(vii) \( a_2 \equiv a_1 \) (ditto, and \( a_2 \) happens within \( a_1 \))

So, our model can express conflict (i), concurrency (ii), sequentiality (iii), temporal overlap (iv) and temporal inclusion (v). Properties (i) and (ii) indicate occurrences on different branches, (iii), (iv) and (v) indicate occurrences on the same branch, while (iv), (v) and (vi) indicate local simultaneity on that branch.

DEFINITION 1.8 – Maximal consistent sets

For a given I.E.S., \( S = (LE, \text{Con}, \prec_0) \), a consistent set \( c \in \text{Con} \) is said to be maximal if no other events can be added to it without violating its consistency

\[
(c' \in \text{Con} \land c \subseteq c') \Rightarrow c = c'
\]

Such a set contains all of the labelled events that can happen in one history of an I.E.S. The set of all maximal \( \text{Con} \)-sets of \( S \) will be referred to as \( \mathcal{M}(S) \). If there is no conflict in an I.E.S., \( S \) then \( \text{Con} = \emptyset(LE) \) and \( LE \) itself is maximal, (all branches are consistent) while if there is conflict, sets smaller than \( LE \) will be maximal. Clearly \( \ast \), being just an on light, should be consistent with all maximal \( \text{Con} \)-sets, so we require

\[
\forall c \in \mathcal{M}(S) . \ast \in c
\]
Notice, incidentally, that conflict is not hereditary in the sense that if \( a_1 \neq a_2 \) and \( a_1 \sqsubseteq a_1' \), and \( a_1 \) and \( a_1' \) are in some maximal consistent set, then we cannot necessarily infer that \( a_1' \) will inherit the conflict, i.e. \( a_1' \neq a_2 \) may not hold. This is 'instability' again; inconsistent branches may join up.

It should be stressed that at this stage \# models some form of choice; if \( a_1 \neq a_2 \) then we may see \( a_1 \) or \( a_2 \) but not both, and nowhere have we stated how that conflict is resolved. In other words, so far, \# has not been taken as a model solely of internal choice or solely of external choice; we shall commit ourselves to a decision in chapter three.

**EXAMPLE – Timing and conflict in posets**

In this example the posets of subsection 1.0.3 will be revisited, and we will see the richness added to our behavioural descriptiveness by conflict and effects of timing considerations. First consider some new pathological posets; our aim here is to demonstrate that many well-known behaviours can be represented in our model, and hence to lend weight to the claim that interval event structures are one of the most behaviourally-rich models of concurrency yet devised.

The first new poset is called the V-poset in [Boudol & Castellani 1988]. It demonstrates a phenomena called *asymmetric confusion* in the Petri net community (see, for instance [Plünnecke 1984]). Consider three events related thus, \( a_1 \prec a_2 \prec a_3 \prec a_4 \) with \( a_3 \sqsupseteq c \) and \( c \neq b \). If we represent conflict by thicker, more heavily dashed lines than causality, the picture might be

![Diagram of V-poset](https://example.com/diagram.png)

The V-poset

The correct \( \text{Con} \) sets to describe the situation above are \( \{a\}, \{b\}, \{a, b\}, \{c\}, \{a, c\} \).

Consider a poset with labelled events \( a_1, a_2, a_3, a_4 \), with \( a_1 \neq a_2, a_2 \neq a_3, \) and \( a_3 \neq a_4 \), and with \( a_1 \sqsubseteq a_2 \) and \( a_2 \sqsupseteq a_3 \). The diagram that displays this structure has too many lines to be helpful; some further insight can be gained from examining the net analogous to the structure, which is given in [Boudol & Castellani 1988].

Our next poset demonstrates the utility of dismissing conflict inheritance; see overleaf.
Consider $a_1$, $a_2$, $a_3$, $a_4$ with $a_1(a_1) < a_2(a_1) < a_3(a_1) < a_4(a_1)$, $a_1(a_2) < a_2(a_2) < a_3(a_2) < a_4(a_2)$, and $a_1(a_3) < a_2(a_3) < a_3(a_3) < a_4(a_3)$. This might be seen as a poset without conflict inheritance.

The correct maximal Con sets to describe this situation are $\{a_1, a_2, a_3\}$ and $\{a_4, a_2, a_3\}$.

Finally, as an example of 'instability' consider the following poset. The labelled events are $a_1$, $a_2$, and $a_3$, with $a_1(a_1) < a_2(a_1) < a_3(a_1)$, $a_1(a_2) < a_2(a_2) < a_3(a_2)$, and $a_1(a_3) < a_2(a_3)$.

The correct maximal Con sets to describe this situation are $\{a_1, a_3\}$ and $\{a_2, a_3\}$.

We will consider timing just two of these structures; take the $\vee$-poset first. The only non-trivial timing constraints required are that $r_{a_1} < r_{a_2}$; remember we can have both $a_1 \preceq a_3$ and $a_2 \preceq a_3$ without any constraint on the real part of the branching times of $a_1$, $a_2$, and $a_3$. Hence, any timing $a_1 = [(r_{a_1})_1, (r_{a_1})_2]$ and $a_2 = [(r_{a_2})_1, (r_{a_2})_2]$ and $a_3 = [(r_{a_3})_1, (r_{a_3})_2]$. This timing is fine because the timing given is consistent with the causality given, but if instead $a_4$ were timed by $[7.6, a_4(a_4)]$, the timing would not agree with the requirement $a_4(a_4) < a_4(2)$.

Timing the N-poset of subsection 1.0.3 is equally easy; one can pick any assignment of times to transitions that respects the conditions imposed by causality. For instance the timing $a_1 = [(0, a_1(a_1)), (e^{1.93}, a_1(a_1))], a_2 = [(20.8, a_2(a_2)), (35.7, a_2(a_2))], a_3 = [(35.7, a_3(a_3)), (666, a_3(a_3))], and a_4 = [(7.6, a_4(a_4)), (\pi^2, a_4(a_4))]$, is fine because the timing given is consistent with the causality given, but if instead $a_4$ were timed by $[7.6, a_4(a_4)]$ the timing would not agree with the requirement $a_4(a_4) < a_4(2)$, while if instead $a_1 = [(0, a_1(a_1)), (61/3, a_1(a_1))]$ the requirement $a_1(a_1) < a_1(2)$ would be violated.

**Example** — The bar revisited

The possible branching event structures that might satisfy the reader’s notion of a hostel-ry will now be further examined. Many further properties of the bar can be deduced by manipulation of this specification using our specification logics (discussed later). Our requirements, which we will rewrite using the event abstractions, were:
1. Someone has to get every drink
\[ \forall i,j. \exists i'. (\text{end}(\text{tussle}_{i,j}) = \text{begin}(\text{serve}_{i,j})) \land (\text{serve}_{i,j} \subseteq \text{drink}_{i,j}) \]

Notice that to capture the notion properly we should write \( \exists i \), as each drink is bought by only one person; this, however, is too constraining, as we cannot distinguish things that are directly caused from indirect causality. There may be some earlier serves which obey the above condition.

Using the \( \text{adj} \) operator this reads
\[ \forall i,j. \exists i'. (\text{tussle}_{i,j} \text{adj} \text{serve}_{i,j}) \land (\text{serve}_{i,j} \subseteq \text{drink}_{i,j}) \]

2. No-one queues more than they drink
\[ \forall i,j. \text{drink}_{i,j} \subseteq \text{tussle}_{i_{j+1}} \]

3. The barman serves everyone eventually (otherwise it is easy to implement the barman with a process which never serves anyone);
\[ \forall i,j. \text{begin}(\text{tussle}_{i,j}) \Rightarrow \text{end}(\text{tussle}_{i,j}) \]

4. People are served in the order in which they begin to queue
\[ \forall i,i',j,j'. (\text{begin}(\text{tussle}_{i,j}) \leq \text{begin}(\text{tussle}_{i',j'})) \Rightarrow (\text{begin}(\text{serve}_{i,j}) \leq \text{begin}(\text{serve}_{i',j'})) \]

5. Only one person can be served at once on any given branch (and any given client’s serves do not overlap)
\[ \forall i,i',j,j'. (\text{serve}_{i,j} \text{observe}_{i',j'}) \Leftrightarrow (i \neq i' \land j \neq j') \]

Caveat – Event abutment and Liveness

Notice that we have nowhere in our specifications specified the time between one event and another. We can, of course, force events to abut one another (by imposing a predicate like \( \forall a_1, a_2 \in \text{LE}. (a_1 \subseteq a_2 \land \neg \exists a_3. a_1 \subseteq a_3 \subseteq a_2) \Rightarrow r(a_1) = r(a_2) \)). However it suits the model not to force this to be everywhere true. Then we can use interval event structures to give an alternative semantics to languages, like timed CSP [Reed 1988], [Davies & Schneider 1989], where the time taken between events can be unspecified. (The timed CSP process \( \text{WAIT}(160) ; a \rightarrow \text{SKIP} \) (in some semantics) only guarantees that a happens sometime later than 160 seconds after execution begins, not at 160 seconds after the beginning.)

We are dealing here with liveness issues. In our specifications the matter of liveness can be explicitly dealt with, for instance by predicates like
\[ \forall a_i \in \text{LE}. r(a_0) - r(a_0) < 100 \]
Time, causality, and concurrency

(no events last more than 100 time units), or

\[ \forall a_1, a_2 \in LE. (a_1 \sqsubseteq a_2 \& \exists a_3. a_1 \sqsubseteq a_3 \sqsubseteq a_2) \Rightarrow r(a_1) - r(a_2) < 24 \]

(no waits between causally-immediately-related events of more than 24 units). However we will not always want to do this. In general we shall merely assume that (unless otherwise mentioned using \( \uparrow \) or \( \downarrow \)), all labelled events mentioned do actually happen (i.e. have defined times).

EXAMPLE—I travel agent

A typical travel agent’s on a Saturday morning in spring contains a finite set of people. Some of these people often hunt in groups known as families, and the family has a common purpose, to book a holiday. Before they can do this, though, they must queue for the attention of one of the “holiday consultants”, the other sort of people present, who hold information about holidays. Then they must make enquiries about possible holidays, which may result in making a booking. We shall denote the event of the \( i \)th family queueing by \( f.\text{tussle}_i \), inquiring by \( f.\text{serve}_i \), and booking by \( f.\text{book}_i \). (The notation makes the parallel with the last example clear.) Similarly the \( j \)th holiday consultant serving her (pardon the sexism) \( j \)th inquiry will be \( c.\text{serve}_{i,j} \) and making her \( j \)th booking will be \( c.\text{book}_{i,j} \). Dealing with all these inquiries is, of course, a computer.

The computer can serve as many inquiries as necessary simultaneously, but it must only deal with bookings one at a time and, to prevent confusion, enquiries are not processed while bookings are being made. The \( k \)th inquiry and \( k \)th booking are \( i.\text{serve}_k \) and \( i.\text{book}_k \) for the computer. A tentative specification might read:

\[ 1. \text{The family acts sequentially and with no dithering} \]

\[ \forall i. (f.\text{tussle}_i, \text{adj} f.\text{serve}_i) \& f.\text{book}_i \uparrow \Rightarrow (f.\text{serve}_i, \text{adj} f.\text{book}_i) \]

\[ 2. \text{Someone serves each inquiry and, if it occurs, booking} \]

\[ \forall i. \exists ! i', j. (f.\text{serve}_i = c.\text{serve}_{i,j}) \& f.\text{book}_i \uparrow \Rightarrow (f.\text{book}_i = c.\text{book}_{i,j}) \]

Here that we have written \( \Rightarrow \) to mean \( \text{begin}(a) \Rightarrow \text{begin}(a') \) and \( \text{end}(a) \Rightarrow \text{end}(a') \), so that 2. implies that the family and the H.C. must synchronise upon the inquiry and the booking. This is a ‘tight’ notion of synchronisation in that the times must match exactly. In chapter two, when we discuss synchronisation in more detail, we will come upon a looser notion of synchronisation. Certainly, though, for two events to be synchronised they must, at least, be concurrent, so we require, if \( a \) is to synchronise with \( a' \), written \( a \text{ synch } a' \), (and discussed in chapter two,) that \( a \text{ co } a' \).
3. Each inquiry and each booking is served by the computer

\[ \forall i,j \exists k. c. serve_{i,j} \supseteq i. serve_k \]

\[ \forall i,j \exists k'. c. book_{i,j} \supseteq i. book_k. \]

Here we have allowed the computer to be faster than the H.C. Notice that we have not forbidden multiple computer inquiries within the same family inquiry, but we have ensured that a family booking corresponds to only one computer booking.

4. Now we impose the requirement that the computer must not service inquiries or other bookings while it has a booking in hand. We can do this using the overlap predicate:

\[ \forall k. \neg \exists k' \neq k . i. book_k \supseteq i. book_k \]

5. The queue is, unusually, fair, so that people are served in the order in which they start to queue:

\[ \forall i,i'. (f. tussle_i \subseteq f. tussle_i') \Rightarrow (f. serve_i \subseteq f. serve_i') \]

6. Finally there are liveness requirements. If an H.C. finishes dealing with an inquiry or a booking while some families are still queueing, then they must eventually serve one of those families:

\[ \forall i,j \exists i'. (c. book_{i,j} \subseteq f. tussle_i) \Rightarrow (c. serve_{i,j,i} = f. serve_i) \]

Notice that the family which is served by the H.C. who becomes free, is dealt with by 5. above. We have not required that the successive agent events abut one another, which leaves them free to take (arbitrarily long) tea breaks, nor have we required that anything at all happens if an H.C. does not manage to make a booking. The reader may care to try to specify some suitable predicate to constrain this eventuality.

Notice that we have again used the ordering on the labelling set; we could rewrite the specifications without it, but it would not be as clear. In general we will allow ourselves this luxury; in chapter four when we have to provide explicit labels we use N, so no real contradiction is present.

It is highly desirable here, as in all other aspects of specification, not to over-specify. The specification of interval event structures involves writing constraints upon structures; an implementation will be satisfactory if it satisfies all of the constraints. Thus over-specification is equivalent to overconstraint of the implementation.
1.5 Safety and \( j \)-morphism

Having introduced the event orderings or \( j \)-morphisms \( \sqsubset_j \), it is useful to know how they are related. In particular, the questions of interest here are, 'when can one ordering be replaced by another?', and 'when is one structure as good as another with respect to a particular ordering?'

An ordering \( \sqsubset_j \) is safer than an ordering \( \sqsubset_k \), written \( \sqsubset_j \gg \sqsubset_k \), iff it preserves all the orderings of \( \sqsubset_k \). This just means that the relation \( \sqsubset_j \) includes the relation \( \sqsubset_k \).

**Theorem** – The preorders \( \sqsubset_h, \sqsubset_i, \sqsubset_l \) and the relation \( \sqsubset_e \) form a lattice ordered by inclusion.

**Proof:** This is obvious from the definitions.

The safer than ordering \( \gg \) can be defined so as to indicate when any expression involving \( \sqsubset_k \) can be written with \( \sqsubset_j \) instead of \( \sqsubset_k \):

\[
(\sqsubset_j \gg \sqsubset_k & a_1 \sqsubset_k a_2) \Rightarrow a_1 \sqsubset_j a_2
\]

In particular, the safest ordering is \( \sqsubset_i \): if \( a_1 \sqsubset_i a_2 \) then \( a_1 \sqsubset_j a_2 \) for any \( \sqsubset_j \).

Given any set of labelled events with some \( \sqsubset_j \) relationships defined over them, where the \( \sqsubset_j \)s are ordered as above by \( \gg \), it is always possible to assign branching times to the transitions of those labelled events consistently (i.e. so that the \( \sqsubset_j \) relationships given follow from the branching timings). This is the free generation of timing from \( j \)-causality; we will consider it further in the next definition, and more comprehensively later.

**Definition 1.9 – Securing**

We will now investigate when a series of events can all be said to be related. The idea is to have a chain of causality so that we can guarantee the occurrence of a given event \( a_n \) by ensuring that its predecessors \( a_1, ..., a_{n-1} \) happen.

In this view, \( a_i \prec_a a' \) is interpreted as 'the occurrence of \( a_i \) guarantees the eventual occurrence of \( a' \)'. Thus \( (a_1, ..., a_{n-1}) \) secures \( a_n \). The interpretation of causality as guarantee makes the requirement \( (a_1 \neq a_2) \Rightarrow (a_1, a_2) \) seem more reasonable, – an occurrence cannot both guarantee something and be in conflict with it.

Let \( 3 = (\text{LE}, \text{Con}, \sqsubset) \) be a I.E.S. A sequence \( a_1, a_2, ..., a_n \in c \in \text{Con}, n \in \mathbb{N} \), is a \( j \)-securing of \( a_n \) in \( 3 \), written \( (a_1, a_2, ..., a_{n-1}) \) secures \( a_n \), iff \( a_1 \sqsubset_j a_2 \sqsubset_j ... \sqsubset_j a_n \).
If $\forall t_1 \in a_1, \ldots, t_n \in a_n \cdot t_1 \leq t_2 \leq \ldots \leq t_n$ then $a_1, a_2, \ldots, a_n$ is a securing of $a_n$ that is secure over all possible choices of a single time to represent an event. This kind of securing is just an $i$-securing; it means that for every possible choice of times to represent occurrences, those times guarantee $t_n$ and hence $a_n$. This notion of times representing labelled events is discussed in the aside below. Similarly, a securing with an existential quantifier replacing the universal one,

$$\exists t_1 \in a_1, \ldots, t_n \in a_n \cdot t_1 \leq t_2 \leq \ldots \leq t_n$$

is true over at least one choice of times representing events, and is thus an $e$-securing. Notice, however, that all $e$-securings cannot be thus represented. Note that if a structure satisfies the axiom of finite causes, then a finite sequence of labelled events $i$, $h$-, and $t$-secures any given labelled event. (It might also be worth noting that $i$ corresponds to the 'completely precedes' order $\rightarrow$ in [Lamport 1986] and the other $c$-s to his 'can affect' order $\rightarrow$. His framework is similar to a subset of ours.)

**ASIDE – On some features of the timing of occurrences**

Obviously the use of certain orderings indicates what it is about an enabling of one event by another that we think important. If we write $a_1 \subseteq_h a_2$ then we suggest that $t_1 = \text{begin}(a_1)$, $t_2 = \text{begin}(a_2)$, are natural choices of times to represent these events (so that if $r_1 < r_3$ say, where $t_3$ is some other "natural" choice of a time to represent an event $a_3$, we might expect this to be reflected at the event level by a relation like $a_1 \subseteq_h a_3$; this is what we mean by "natural"). The notion of a single time representing an interval of time is that used in atomic-event calculi. An $i$-securing of $a$ indicates that any possible choice of atomic events within the structure that has the same event ordering will still secure $a$. This is not true for any securing: consider two events, $a_1 = [0,3]$ and $a_2 = [1,2]$. Clearly $a_1 \subseteq_h a_2$ is consistent with the underlying order $<$, and so $a_1$ secures $a_2$. However, if we choose the ends of the events to represent them, then we can see that this is not a strong securing as, if $r_1 = 3$ and $r_2 = 2$ are the representing times, clearly $t_1 \not\subseteq_b t_2$.

Notice that when we represented untimed posets in subsection 1.0.3 and in the last section we used $i$-securings. We could equally well, for instance, (if we had a different view of what 'securing' means) have used $h$-securing. In that case the timing requirements for, for example, the $N$-poset, would have read $r_{e(t)} < r_{e(Q)}$, $r_{e(t')} < r_{e(Q)}$ and $r_{e(t)} < r_{e(Q)}$. The question of generating timing from causality will be revisited in the interlude preceding chapter three.

Philosophically, it is hard to see how one transition can cause another without some time delay; that is why $a_i < b_i \Rightarrow r_{i(a)} < r_{i(b)}$ and not $a_i < b_i \Rightarrow r_{i(a)} \leq r_{i(b)}$. This is not the view of causality taken by $\subseteq_{ij}$, where $\leq_b$ is used. Both alternatives are available, – the more accurate version with $<$ at the lower level, and the more convenient one with $\leq$ at the upper, – but some care is needed in moving between them, as the footnote in section 1.3 indicated.
1.6 Observing Interval Event Structures

An interval event structure is a model of the behaviours of an implementation of a concurrent system. What can we actually see? Recall that we had intended that labels merely serve to indicate which occurrence of an event we are talking about; all that is observable are events happening. In this section we indicate which (multi)sets of timed events are valid observations of I.E.S.s.

Our paradigm of observation is this; we sit down with a notebook and watch our I.E.S., recording the events that we see with their starting and finishing times. Nothing is hidden from us: no event can occur without us noticing, (we are one of section 1.0’s global observers). We shall assume that we do this through all time. Since we demanded that LE was countable, our record of what happened will be countable too. The result of this process is a L.I.E.S.; it is a list of occurrences of events with their starting and finishing times linearly ordered. (A L.I.E.S. is just another way of thinking about a linearly ordered multiset, since the labelling set is useful only to disambiguate one occurrence of an event from another. Hence we consider L.I.E.S.s which are equal up to bijection of labelling sets as equal.) But which L.I.E.S.s are valid ones?

**DEFINITION 1.10 – Possible histories**

A L.I.E.S. $L = (LE', <)$ is a possible history of an I.E.S. $S = (LE, Con, <)$, also known as an observation sequence, iff $LE'$ is a maximal Con-set, i.e. $LE' \in M(S)$. (By abuse, we often write $L \in M(S)$ for $L = (LE', <)$ and $LE' \in M(S)$.) For the moment finite choice will be demanded; $M(S)$ must be finite.

Consider some maximal Con-set. Clearly all of its labelled events can occur in the same history. Since the set is maximal, no more events can occur than the ones given. Hence it is a possible history. We cannot determine which of the many maximal Con-sets a given I.E.S. may have will be seen, since the whole idea of conflict was to model choice.

It should be noted that an observation sequence is much more like a Mazurkiewicz trace, [Mazurkiewicz 1984], than a standard trace, as it incorporates information about temporal overlap (although it doesn’t enable us to distinguish true (i.e. co-style) concurrency; if two labelled events overlap in an observation sequence, it can be either because they were concurrent or because they were locally simultaneous: further, co labelled events are not necessarily simultaneous).

This completes the basic definitions (summarised overleaf).
In the next chapter, we move on to consider how to combine interval event structures using simple categorical machinery. The interlude between this chapter and the next will deal with category theory and its place in computer science.

**Summary of definitions**

A version of the figure given on page 41 shows the relationship between branching time and labelled events:

\[
\begin{align*}
E & \quad \text{Events, } E \\
\rightarrow \quad \text{Labelled Events, } LE \\
\leftarrow \quad \text{Labels, } L \\
\rightarrow \quad \begin{array}{c}
\begin{array}{c}
\text{begin}(a) = t_{(a)} \\
\text{end}(a) = t_{(a)}
\end{array}
\end{array}
\end{align*}
\]

Branching Time with reals \( R \) and tags \( Tr = (LE_s \cup LE_f) \)

\[
(r_{(a)}, a) \in BT \subset BT = (R \cup \{T\}) \times Tr
\]

\[
\begin{array}{c}
\text{respect both}\ <c \text{ (over } LE_s \cup LE_f) \text{ and } < \text{ (over } R) \\
\text{respect both}\ =c \text{ (over } LE_s \cup LE_f) \text{ and } = \text{ (over } R) \\
< = c \text{ and } < \text{ and } =c \text{ and } = \text{ are consistent:}
\end{array}
\]

\[
a_t < c b_t \Rightarrow r_{(a)} < r_{(b)} \text{ and } a_t = c b_t \Rightarrow r_{(a)} = r_{(b)}
\]

Next we summarise the main definitions given since page 41;

*Definedness* (page 42). A labelled event \( a \) is defined, \( a \uparrow \), if all of its branching times are not \( T \) and undefined, \( a \downarrow \) otherwise.

*Inclusion ordering & Overlap* (page 43). A labelled event \( a_1 \) branching includes another \( a_2, a_1 \sqsubseteq a_2 \), if their branching times are included. Similarly for branching overlap (\( \sqsubseteq_b \)) and ordinary inclusion & overlap.

*The Outside relation* (page 43). A labelled event \( a_1 \) is branching outside another \( a_2, a_1 \not\sqsubseteq a_2 \), if they are on different branches or share no temporal overlap. If only the latter, they just overlap (\( \# \)) each other.

*Finite and Computational Structures* (page 46). These terms essentially forbid uncomputational (Zeno) structures.

*Event Orderings* (page 47). The four forms of causality possible are starts causing starts (head causality, \( a_1 \sqsubseteq_h a_2 \)), finishes causing finishes (interior causality, \( a_1 \sqsubseteq_i a_2 \)), starts causing finishes (exterior causality, \( a_1 \sqsubseteq_e a_2 \)) and finishes causing finishes (tail causality, \( a_1 \sqsubseteq_t a_2 \)).

*Branching Time Notation* (page 48 et prev). The branching time corresponding to a transition of an event \( a \) is written \( t_{(a)} \) or \( t_{(a)} \) depending on whether it is a start or a
finish. When we want to emphasize the components involved we write \((r_{i(0)}, a_i)\) or \((r_{f(0)}, a_i)\) instead. The notation \(a = [r_{i(0)}, a_i] \) or \(a = [t_{i(0)}, t_{f(0)}] \) is used for timing.

For a generic transition (a start or a finish) we write ‘t’, so \(a_t\) is either the start or the finish of \(a\), and \(t_a\) is the branching time of that start or finish. This latter statement (that \(t_a\) is the branching time \(a\) started or finished) is also notated \(t_a \in a\).

**Primitive Concurrency** (page 48). Two labelled events are primitively concurrent, \(a_1, a_2\) if they are not causally related

**Consistency** (page 48). The consistency predicate, \(\text{Con}\), tells us which branches may appear together in any given execution of the system.

**Conflict** (page 50). Two labelled events are in conflict, \(a_1 \neq a_2\), if they are primitively concurrent but may never appear together.

**Interval Event Structures** (page 50). An interval event structure (I.E.S.) is an elementary B.I.E.S. with conflict. It is normally written \((L, E) \subseteq (L \times E) \cup \{a\}, \text{Con}, \triangleleft_b\) but the extra structure indicated above (begin and end, =b, etc.) is also assumed to be present.

**Concurrency** (page 51). Two labelled events are concurrent, \(a_1 \sim a_2\), if they are primitively concurrent and consistent.

**Maximal Con-sets** (page 51). A maximal Con-set is a possible history of an interval event structure. It contains one of the largest sets of labelled events that can possibly happen together. The set of all maximal Con-sets of an I.E.S. \(S\) is written \(\mathcal{M}(S)\).
References


Interlude – Category Theory and Computer Science

The next few chapters will make some use of category theory. Therefore, it seems appropriate to discuss the use we will make of category theory, the prerequisites we assume, and the relationship between category theory and computer science.

Category theory for us

We shall use category theory only where it seems to offer a particularly clean way of expressing our ideas without unnecessary complication. It will be indicated where a categorical approach could be taken but wasn’t, as it was felt that it was inappropriate or unwieldy.

Prerequisites

One of the least well defined terms in the literature is “elementary category theory.” To a mathematician this seems to be all of [Mac Lane 1971], while to a computer scientist [Blyth 1986] may seem to be all that merits that description. We shall assume the latter definition of elementary, and suppose that the reader is familiar with objects, morphisms, products, equalisers, limits generally (and dual notions) and with functors and adjunctions (although these will not be used extensively). More complex notions will be explained on the rare occasions that they are used; in general we shall keep the explanation as simple as is consistent with understanding the point in hand. Computer scientists may find [Pierce 1989] of use as an introduction.

It should be stressed that (apart from a very small amount of material in the interludes) this thesis will not use any advanced category theory. I intend, in due course, to rework some of the material to incorporate more advanced categorical insights as suggested in the interludes.

The motivation for the use of category theory in computer science has been extensively treated by various authors, [ADJ 1976] for example. I shall not attempt to convince those who still doubt its usefulness.

Bibliography


Mac Lane [1971] S. Mac Lane, Categories for the Working Mathematician, Springer-Verlag Graduate Texts in Mathematics.

Operations on Structures: Synchronisation and Morphism

In this chapter some operations on interval event structures and some relationships between them will be investigated. Operations such as the choice between structures, the sequential combination of structures and the parallel composition of structures will be considered. Interval event structures are related by structure-preserving morphisms which naturally lead to categories of interval event structures. These categories will be used to describe our operations on interval event structures.

The material in this chapter has been heavily influenced by [Winskel 1984i], [Winskel 1984ii]. However, we have more temporal structure than him, and this leads to substantially different results, particularly for parallel composition with synchronisation. The general plan, though, is standard; we will try to define operations as universal constructions in appropriate categories. The interlude at the end of the chapter deviates from the standard course by considering some more elaborate categories, and, hence, criteria for deciding whether a structure fits into our framework or not. Before we proceeding to define the operations, there is a short preamble concerning the timing of structures: –

An interval event structure is said to be finite if it has a finite number of labelled elements. In this section we will only consider finite structures. If a structure is finite (and sometimes if it is infinite) it will have a labelled event that begins first and one that finishes last. Recall that we endowed every structure with a unique labelled event that begins at the same time as the first labelled event begins and ends at the same time as the last labelled event ends. This event was called the ‘silent event’ of a structure, and was denoted by •. Intuitively this is an ‘I am running’ event.

Recall that the ‘special happening’ • head-causes everything and everything tail-causes •, so that for ∀ t ∈ a ∈ LE. begin(*) ⊑ t & t ⊑ end(*). This ensures that the structures are connected, and that a • can be happening at any time the structure is active.
DEFINITION 2.0 – The Beginning and End of an interval structure

Recall that * lasts as long as the structure is active; it starts not latter than the first labelled event starts, and finishes not earlier than the last possible event finishes. This means that we can define the beginning and end of a structure by begin(3) = begin(*) and end(3) = end(*). These times are defined (i.e. reals) for all finite structures with defined times and for some infinite ones.

A structure is said to be bounded if begin(3) and end(3) are defined (i.e. 3 has no undefined times and no unbounded ascending chains end(a1) < end(a2) < ... so end(3) ≠ T.) From now on we will only deal with defined structures; ones where all the times (except possibly end(*)) are defined.

(If either time of any labelled event is defined, then the time begin(*) is defined, (i.e. S is always bounded-below) since we required that 3 ∈ E with the property that for any a ∈ E, begin(*) ≤ begin(a). This means that begin(*) ≤ begin(a) and thus begin(*) ≤ inf{t, a ∈ [(t, a), t]} then the inf is well-defined.)

Notice, incidentally, that we do not require end(*) = sup{t, a ∈ [(t, a), t], a ∈ LE) or begin(*) = inf{t, a ∈ [(t, a), t], a ∈ LE); the on light can go on before things start, and stay on after things have finished.

We can formulate the time 3 starts doing events in some maximal Con-set L ∈ M(3) as begin(3, L = (LE', <)) = inf{t, a ∈ [(t, a), t], a ∈ LE') so that the earliest 3 can start doing events is min{begin(3, L) | L ∈ M(3)}, and the latest it can start is max{begin(3, L) | L ∈ M(3)}. If end(3, L = (LE', <)) = sup{t, a ∈ [(t, a), t], a ∈ LE'), then the latest it can finish doing its last labelled event is max{end(3, L) | L ∈ M(3)} and the earliest it can finish doing its last event is min{end(3, L) | L ∈ M(3)}. Recall that M(3) is finite so these times are always well-defined on R ∪ {T} by max and min.

It is important to be aware that the times begin(3) and end(3) can be very different from the times 3 is actually seen to be doing events; although 3 must restrict itself to exhibiting transitions within the interval [t, t] where begin(3) = (t, *) and end(3) = (t, *), it need not occupy all of that interval. In particular there is no contradiction in end(3) = (T, *) even if sup{t, a ∈ [(t, a), t], a ∈ LE) is well-defined and finite; such an I.E.S. might be a model of a CSP [Hoare 1985] process P; stop where P is finite.
NOTATION AND CONVENTION (Summary)

Interval event structures will be symbolised by $3, 3', 3''$ etc.; their associated sets of labelled events, consistency predicates and temporal orders will be $(LE, Con, \prec)$, $(LE', Con', \prec')$, and $(LE'', Con'', \prec'')$ respectively. Labelled events in $3$ will be $a_0, a_1, a_2$, and so on, and in $3'$ they will be $b_0, b_1, b_2$, and so on. Silent events will be $\ast, \ast$, and $\ast'$ respectively in $3, 3'$, and $3''$. $Con$ sets will be $c, c', c''$ respectively, while maximal $Con$-sets will be either the same or $L, L', L''$. Helvetica-Bold will be used for categories.

The conventions for denoting branching times introduced in chapter one will be retained. In particular, we denote two things by $\text{begin}(a)$; both the real value at which $a$ begins ($r$, say) and the branching time ($r, a$ say); we will usually be able to deduce from context which is meant. Similarly $\text{begin}(3)$ will mean either the linear time $3$ begins, or the branching time (i.e. the linear time decorated with a tag, which will be $\ast$, here).

A continuing theme will be that labels are just a means of disambiguating one occurrence of an event from another. Labels will be manipulated and events relabelled without remit; we must just be careful never to make two different labelled events the same.

A convention for substitution will be useful. The term $a[b/c]$ will denote $a$ with $b$ substituted for $c$ wherever it occurs. If $a$ is the same type as $b$ and $c$, then

$$a[b/c] = \begin{cases} c & \text{if } a = b \\ a & \text{otherwise.} \end{cases}$$

In this chapter the transitivity of $\prec$ and the subset closure property of $Con$ will sometimes be exploited. (So the correct definition is obtainable from ours by taking a transitive or subset closure.)

The whole discussion of this chapter is presumed to be made relative to a fixed, prespecified set of labelled events, called a universe of discourse. At the outset a finite set of events and a countable set of labels must be provided, and no labelled event must ever be used which is not in the cross product of those sets. This requirement is made to avoid cardinality problems, and to provide a fixed set for quantifiers to range over.
2.0 Prefixing

The operation of prefixing an event \( b \) onto an I.E.S. involves building a new I.E.S. just like the original except that the occurrence \( b \) happens first:

**DEFINITION 2.1 – Prefixing**

Suppose \( S = (LE, Con, <) \) is an interval event structure. Suppose also that we have a new labelled event, \( b \), \( b \in LE \). \( S \) prefixed by \( b \), or \( b \) before \( S \), written \( S' \), is defined to be an I.E.S. \( S' = (LE', Con', <') \) defined below. We will suppose that \( b \) has a defined duration, \( \Delta_b \), but an undefined starting time so that \( b = [(r, b_\text{begin}), (r + \Delta_b, b_\text{finish})] \) where \( r \) is variable. This accords with the intuition that we know how long a prefixing event is going to take, but not when we want it to start. By prefixing it to \( S \) we force it to start before everything else. Suppose \( S = [(r_\text{begin}, *), (r_\text{end}, *)] \). Then

\[
LE' = \{ b | b = [(r - \Delta_b, b_\text{begin}), (r, b_\text{finish})] \} \cup (LE - \{*\} \cup \{*'\})
\]

\[
Con' = \{ c \cup \{b\} | c \in Con \}
\]

\[
<' = \subseteq \cup \{(r_\text{begin} - \Delta_b, b_\text{begin}), (r_\text{end}, b_\text{finish})\} \cup \{(r_\text{begin} - \Delta_b, *'), (r, *)\} | t, a \in LE - \{*\})
\]

The first condition adds \( b \) at the beginning of the structure, deletes the old silent event and adds a new one.

The second ensures that \( b \) is consistent with every consistent execution of the old structure. The new silent event is, of course, \( *' = [(r_\text{begin} - \Delta_b, *'), (r_\text{end}, *)] \). The closure properties of \( Con \) are exploited here.

The third ensures that the branching structure of \( S \) is preserved, that the beginning of \( b \) is branching-less-than the end of \( b \), and that the new silent event \( *' \) has the required properties.

In fact, this description is not quite right, as \( \subseteq \) will have references to the old silent event \( * \) in it, an event which no longer exists. Hence the \( \subseteq' \) in the definition is related to \( \subseteq \) by

\[
\subseteq' = \{(r_\text{begin}, a_{\text{begin}}[b/*/']), (r_\text{end}, a_{\text{end}}[b/*'])\} | (r_\text{begin}, a_{\text{begin}}), (r_\text{end}, a_{\text{end}}) \in \subseteq \}
\]

that is, the same ordering except that all \( * \) tags become \( b_\text{begin} \) ones, and all \( *' \) tags become \( *' \) ones. Similarly we should overwrite \( * \) by \( b_\text{begin} \) and \( *' \) by \( *' \) to obtain \( \subseteq' \) from \( \subseteq \) in the definition of \( \subseteq' \).

The fourth condition identifies \( b_\text{begin} \) with the start of \( S \), (so that \( b \) is now branching-less-than-or-equal-to all of \( S \), since \( x \subseteq y \) and \( y =_\subseteq z \) implies \( x \subseteq z \)).
It might be argued that what is wanted is a prefixing construction that sticks \( b \) onto the beginning of every maximal \( \text{Con} \)-set rather than onto the beginning of the whole structure, so that (a version of) \( b \) (i.e. a labelled event with the same event part as \( b \)) finishes as the first event of each maximal \( \text{Con} \)-set starts. This, too, is possible, and might be thought of as a local prefixing. In general a local construction will be one relative to maximal \( \text{Con} \)-sets, in opposition to a global one that is made on the whole structure.

A diagram may make the prefixing construction easier to visualise:

\[ \text{Figure 2.1} - \text{The effect of the prefixing construction} \]

There does not seem to be a simple categorical description of prefixing, which is why it was introduced before the substance of the chapter. All prefixing constructions of interest to us will be dealt with by using sequential compositions.
2.1 Morphisms

For structures with a silent event we can define the notion of homomorphism of interval event structures. A homomorphism is intuitively a function that preserves structure. Our definition resembles that of bisimulation equivalence [Milner 1989].

**DEFINITION 2.2 - j-Homomorphism of Interval Event Structures**

Given two I.E.S.s, $\mathcal{S}$ and $\mathcal{S}'$, with silent events $\ast, \ast'$, a (n asynchronous) $j$-homomorphism is a function $f$, from LE to LE', such that

(i) Silent events are preserved; $f(\ast) = \ast'$.

(ii) The causal precedence order is preserved. If, for $a, a' \in \text{LE}, a \leq_j a'$ then this order is preserved; $f(a) \leq_j f(a')$. (Here $\leq_j$ comes from $\leq_s$ and $\leq_j'$ from $\leq_{s'}$.)

(iii) Previously consistent sets must remain consistent. If $c \subseteq \text{Con}$ then the image of $c$ in $f, f(c), \in \text{Con}'$.

This choice of morphism ensures that if $f: \mathcal{S} \rightarrow \mathcal{S}'$ then there is a matching ordering in $\mathcal{S}'$ for any in $\mathcal{S}$ (although this may be the trivial one $a \leq_j a$) so that $\mathcal{S}'$ can be thought of as simulating $\mathcal{S}$. If there are arrows both ways, $f: \mathcal{S} \rightarrow \mathcal{S}'$ and $g: \mathcal{S}' \rightarrow \mathcal{S}$, then the two structures mutually simulate each other (if, that is, we think whichever $\leq_j$ we have captures the essence of simulation).

**PROPOSITION - Categories of Interval Event Structures**

There are four categories whose arrows are I.E.S.-$j$-homomorphisms and whose objects are I.E.S.s. We shall call these categories $\text{IES}_j$. Notice that, trivially, there is an identity homomorphism and that composition of morphisms is well-defined.

We shall define a synchronous morphism by replacing condition (ii) by

(iiia) Suppose $a \leq_j a' & a \neq a'$ then we require $f(a) \leq_j f(a')$ and $f(a) \neq f(a')$.

The categories with synchronous $j$-homomorphisms as arrows will be called $\text{IES}_{j\text{syn}}$. (The reason for the term ‘synchronous’ will become apparent when we consider the product in these categories.)

In the following sections, we shall define operations on I.E.S.s as operations on the categories formed here. The question of just which limits those categories have will be discussed later.
A picture may help at this stage:

![Diagram of asynchronous and synchronous j-homomorphisms]

**Figure 2.2 – Asynchronous (top or bottom) and synchronous (bottom only) j-homomorphisms**

The synchronous categories $\text{IES}_s$ are related to the asynchronous categories $\text{IES}_j$ just as the synchronous categories of [Winskel 1984] are related to the asynchronous ones.

Notice that, because of the relationships between orderings given by $\triangleright$ in the last section, $\text{IES}_s$ and $\text{IES}_j$ are wide subcategories of $\text{IES}_a$, and $\text{IES}_j$ is a wide subcategory of both $\text{IES}_s$ and $\text{IES}_a$.

It is also worth noting the difference between the figure above and normal process graphs. Normally nodes are labelled by states and edges by actions. Here a node is a labelled event (which is like an action) and an edge is a causal dependency relation.
DEFINITION 2.3 – Strong homomorphisms and Temporal morphisms

There is another definition of morphism that seems as natural as the one given above. This preserves rather more structure of time and is called temporal homomorphism. The idea is that one structure simulates another just when the labelled events of the second structure lie entirely inside the corresponding events of the first. We can formalise this as follows:

We say that a morphism $f$, is a strong homomorphism just when it is a $j$-homomorphism for all $j$. In that case, it preserves all enablings and, hence, all securings. In particular, it preserves all securings, hence the name strong homomorphism.

Given two I.E.S.s, $S$ and $S'$, a temporal homomorphism is a function, $f$, from LE to LE' which is a strong homomorphism and where $\forall a \in LE$ if $a = [t, t']$, and $f(a) = [t', t]$ then $[t, t] \supset [t', t']$ provided $f(a) \neq \ast$. The idea is here that a simulating structure must do the same labelled events with the same causality in less time, except if we want to simulate a labelled event by a silent event, when it doesn’t matter how long it takes since it is silent.

Notice that homomorphisms are total, for $f$ is defined on all of LE, and hence $LE \subseteq LE'$. A temporal morphism is a strong homomorphism that preserves absolute timing information. The category whose arrows are temporal homomorphisms and whose objects are I.E.S.s will be called $tIES$; it is a wide subcategory of $IES$. Similarly, $tIES_{syn}$ is the wide subcategory of $tIES$ with synchronous temporal morphisms. 

Unless otherwise indicated, in future when a categorical operation is mentioned it will be assumed to be in $tIES$.

We have not assumed that homomorphisms preserve the names of events (i.e. that the image of $(l, e)$ has event part $e$; $f(l, e) = (l', e)$.) This is because it should be up to the user what ‘simulation’ means. It may sometimes be appropriate to place further constraints on $f$, but it is up to the user to examine the subcategory structure thus generated.

Notice that none of the categories we have defined are small.
2.2 The disjoint sum

Consider two I.E.S.s, \( S \) and \( S' \). Their disjoint sum, \( S + S' \), is obtained by gluing their silent events together and not letting them otherwise interfere in any way. It is a coproduct in the categories \( \text{IES} \) and \( \text{tIES} \). To see this define the sum \( S + S' \) to be the I.E.S. \( S^* \) where

(i) The set of labelled events of \( S + S' \), \( \text{LE}^* \), is the set

\[
\{((0, l_0), e_0), ((0, l_1), e_1), \ldots | (l_0, e_0), (l_1, e_1), \ldots \in \text{LE} \neq \ast \}
\cup \{((1, l_0), e_0), ((1, l_1), e_1), \ldots | (l_0, e_0), (l_1, e_1), \ldots \in \text{LE} \neq \ast' \}
\]

A labelled event \(((0, l_0), e_0)\) in \( S + S' \) has the same timing as \((l_0, e_0)\) in \( S \), while a labelled event \(((1, l_0), e_0)\) carries the timing of \((l_0, e_0)\) in \( S' \).

The new silent event is \( \ast' \); it is timed by

\[\min(\text{begin}(S), \text{begin}(S')) \cup \max(\text{end}(S), \text{end}(S'))\].

(ii) The obvious injections on labelled events are \( i_o : (l, e) \to ((0, l), e) \) and \( i_i : (l, e) \to ((1, l), e) \). The consistent sets of \( S + S' \) are just the disjoint union of the consistent sets of each component. Suppose \( c \in \text{Con} \). Write \( i_0(c) \) for the image of \( c \) in \( i_0 \) and \( i_0(\text{Con}) \) for the union of all such images. Then \( \text{Con}^* = i_0(\text{Con}) \cup i_i(\text{Con}) \). This union is disjoint as \( i_0(\text{Con}) \) and \( i_i(\text{Con}) \) are disjoint by construction.

(iii) A time, \( t \in ((0, l), e) \), of \( S + S' \) is \( \leq \) another \( t' \in ((0, l'), e') \) iff \( t \leq t' \). Further, some \( t \in ((1, l), e) \) is \( \leq \) \( t' \in ((1, l'), e') \) iff \( t \leq t' \). Finally \( \text{begin}(\ast') \leq \ast \) \( t_i \) and \( t_i \leq \text{end}(\ast') \) for \( t_i \in \alpha \in \text{LE}^* \). The equality \( = \) can be built in a similar fashion to the technique used for \( \leq \).

\((\text{LE}^*, \text{Con}^*, \leq)\) is clearly an interval event structure. The injections, \( i_o, i_i \), are obviously \( j \)-homomorphisms for any \( j \) (and hence strong homomorphisms). Further they define a coproduct:

For arbitrary morphisms \( j_0 \) and \( j_1 \) to an arbitrary I.E.S., \( \Psi \), there is a unique \( j : S + S' \to \Psi \) such that the following diagram commutes

![Diagram](image-url)
To see this, define $j(v)$ componentwise: if $v = ((0, l), e)$ then $j(v) = j_0(l, e)$ while if $v = ((1, l), e)$ define $j(v) = j_1(l, e)$. Then $j \circ i_0 = j_0$, and $j \circ i_1 = j_1$, so the diagram commutes. For the uniqueness of $j$ suppose $k : \mathcal{S} + \mathcal{S}' \rightarrow \Psi$ makes the diagram commute.

Consider one labelled event in $\mathcal{S} + \mathcal{S}'$ and let $k((0, l), e) = \psi$; then $j_0(l, e) = \psi$. This works for $((1, l), e)$ too, so $k = j$.

This proof only holds if we remove the silent events of $\mathcal{S}$ and $\mathcal{S}'$ before performing the sum. (So that $i_0(*) = i_1(*) = \ast$; since $j$ is a homomorphism, $j(*) = \ast$, so there is no problem.)

From the last two paragraphs $(\mathcal{S} + \mathcal{S}', i_0, i_1)$ is a coproduct of $\mathcal{S}$ and $\mathcal{S}'$ in $\mathbf{IES}_p$. Since $i_0$ and $i_1$ are also morphisms in $\mathcal{IES}$ if $j_0$ and $j_1$ are, then $j$ will be too, and uniqueness follows from the same argument as before. Hence $\mathcal{IES}$ has small coproducts constructed thus, and there it is natural to time, for instance, $((1, l), e)$, with the same time as $(l, e)$.

The disjoint sum of two I.E.S.s clearly corresponds to a choice between them. We can have either one structure or the other. (This is precisely what (ii) above requires.)

If $\mathcal{S}$ and $\mathcal{S}'$ begin at the same time, the effect of the disjoint sum in $\mathcal{IES}$ is rather like the traditional picture of choice, but notice that there can be shrinkage of events.

![Figure 2.4 - The effect of the disjoint sum construction
(homomorphism lines omitted for clarity)](homomorphism lines omitted for clarity)

It can be seen that $\mathcal{S} + \mathcal{S}'$ somehow corresponds to a structure that can either behave like $\mathcal{S}$ or like $\mathcal{S}'$. The crucial point is that (modulo the renaming performed by $i_0$ and $i_1$) $+ \just adds maximal $\text{Con}$-sets;

$$\mathcal{M}(\mathcal{S} + \mathcal{S}') \equiv \mathcal{M}(\mathcal{S}) \cup \mathcal{M}(\mathcal{S}')$$

So, modulo renaming, the maximal $\text{Con}$-sets of $\mathcal{S} + \mathcal{S}'$ are just those in $\mathcal{M}(\mathcal{S}) \cup \mathcal{M}(\mathcal{S}')$. 

---

David Murphy

Time, causality, and concurrency

Chapter Two — Operations on Structures

Page 73
2.3 Sequential composition

Sequential composition is a fairly natural operation on I.E.S.s; we just do one set of things after another. For two I.E.S.s, \( S = (\text{LE}, \text{Con}, \llhd) \) and \( S' = (\text{LE}', \text{Con}', \llhd) \) with silent events \( \ast, \ast' \), the local sequential composition of \( S \) and \( S' \) is written \( S \oplus S' \). The definition of this operation is fairly subtle, as we must stick a copy of \( S' \) onto the end of every maximal \( \text{Con} \)-set of \( S \). First identify the last labelled events of these maximal \( \text{Con} \)-sets; for \( c \in \mathcal{M}(S) \) define

\[
a \in \text{Last}(S, c) \iff a \in c \quad \& \quad a' \in c. \quad \text{end}(a) > \text{end}(a') \quad \& \quad \text{end}(a) \neq T \quad \& \quad a \neq \ast
\]

Now we must make \( |\mathcal{M}(S)| \) copies of \( S' \) and shift the (real-valued part of) the times of all the events in \( S'_c \) so that \( S'_c \) starts at \( \text{end}(a) \) for \( a \in \text{Last}(S, c) \). A shift operator first:

**DEFINITION 2.4** – Shifts in time

An I.E.S. \( S = (\text{LE}, \text{Con}, \llhd) \) shifted by \( t \in \mathbb{R} \), written \( S_t \), is simply an I.E.S. \( S' = (\text{LE}', \text{Con}', \llhd) \) where all the times have been shifted by \( t \);

(i) \( \text{LE}' =_\text{def} \{a' = ((t, l), e) = [(r_t + t, a_t), (r_t + t, a_0)] \mid a = (l, e) = [(r_t, a_t), (r_t, a_0)] \in \text{LE} \}

(ii) \( \text{Con}' =_\text{def} \text{Con} \)

(iii) \( ((r_t, a_0), (r_t, a_0)) \in \llhd \iff ((r_t, a_0), (r_t, a_0)) \in \llhd \)


We now want to define a set of I.E.S.s formed by shifting a copy of \( S' \) so that a member of the set starts as each maximal \( \text{Con} \)-set finishes:

Define \( \mathcal{S}^+_t = S^+_t = (\text{LE}_c, \text{Con}', \llhd) \) say (where \( a \in \text{Last}(S, c) \) and \( t \) is just \( \text{end}(a) - \min \{t \mid a = [(l_t, a_t), (r_t, a_0)] \in \text{LE}_c \}) \), in order to stick a \( S^+_t \) onto the end of each \( c \in \mathcal{M}(S) \).

(In some perverse structures where a maximal \( \text{Con} \)-set is countably infinite there may be no last \( a \) and we must take \( t = (\bigcup \{\text{end}(a) \mid a \in c \}) - \min \{t \mid a = [(l, a), (r, a)] \in \text{LE}_c \} \).

The construction \( S \oplus S' = (\text{LE}^-, \text{Con}^*, \llhd) \) say, which is defined if some maximal \( \text{Con} \)-set of \( S \) is bounded, can now be defined as;

(i) \( \text{LE}^- =_\text{def} \text{LE} \cup \{a^- = ((l, c), e) = [t_u, t_t] \mid \ast \neq a = (l', e) = [t_u, t_t] \in \text{LE}_c, c \in \mathcal{M}(S) \}. \)

That is, the labelled events of \( S \oplus S' \) are those of \( \text{LE}^- \) plus (a disambiguated version of) those of \( S^+_t \) for each \( c \in \mathcal{M}(S) \). (We suppose \( \text{LE} \cap \bigcup_c \text{LE}_c = \emptyset \).

(ii) Define \( \text{Con}^* = \{(l, c), e) \mid (l, e) \in c' \} \cup c' \in \text{Con}' \)

(i.e. a disambiguated version of \( \text{Con}' \).) Then,

\( \text{Con}^* =_\text{def} \bigcup \cup c \in \mathcal{M}(S) \{c \cup c' \mid c' \in \text{Con}' \} \)

Chapter Two — Operations on Structures
That is, every maximal Con-set \( c \) generating \( 3^{++}c \) is consistent with every Con-set of \( 3^{++}c \).

(iii) For \(<\cdot \) we must first disambiguate \(<\cdot \rangle_{c} \) and then replace \((\cdot \rangle_{c}, \) by \( a \) (where \( a \) is a last labelled event, \( a \in \text{Last}(S, c) \)) and \((\cdot \rangle_{c}, \) by the new silent events' finish \( \cdot \).

(This new silent event, \(<\cdot \rangle_{c} \), by the way, is timed by \( t_{\text{end}}(c) = \text{end}(a) - \text{begin}(S, c) \).

Then, if \(<\cdot \rangle_{c} = [((r_{(c)}, \cdot), (r_{(c)}, \cdot))] \)

\(<\cdot \rangle_{c} = \text{df} \langle (r_{(c)}, \cdot), (r_{(c)}, \cdot) \rangle \) for \( a \in \text{Last}(S, c) \) and

\(<\cdot \rangle_{c} = \text{df} \langle (r_{(c)}, \cdot), (r_{(c)}, \cdot) \rangle \).

Now, \(<\cdot \rangle \) is just \(<\cdot \rangle_{c} \) and all the orders of \(<\cdot \rangle_{c} \).

\(<\cdot \rangle \) is defined to be \(<\cdot \rangle_{c} \cup \bigcup \left\{ \langle (r_{(c)}, \cdot), (r_{(c)}, \cdot) \rangle \right\} \).

While the idea of sticking a copy of \( S^{' \prime} \), appropriately shifted, onto the end of each maximal Con-set of \( S \) is fairly obvious, the mechanics of doing so are rather complex. Thus far I have not found a simple category theoretic construction that gives \( S \otimes S^{' \prime} \). However, in the case that \( S \) is deterministic (has just one maximal Con-set, namely \( S \)) and \( \text{begin}(S^{' \prime}) > \text{end}(S) \), the construction simplifies somewhat; in the terms of section 2.1, local sequential composition (sticking a copy of \( S^{' \prime} \) on the end of each maximal Con-set) is, in this case, the same as global sequential composition (sticking \( S^{' \prime} \) on the end of the whole structure).

The global sequential composition \( S \otimes S^{' \prime} \) is defined to be \( (S^{' \prime}, \text{Con}^{' \prime}, \langle \cdot \rangle^{' \prime}) \) where

(i) Define for \( a \in \text{LE} - \{\cdot\}, \) \( i_{0}(a) = (0, a), i_{0}(\cdot) = \cdot \) and for \( b \in \text{LE} - \{\cdot\}, \) \( i_{1}(b) = (1, b), i_{1}(\cdot) = \cdot \). (i.e. \( i_{0}(l, e) = ((0, l), e) \) etc.). Then define \( \text{LE} = \{i_{0}(a) | a \in \text{LE}\} \cup \{i_{1}(b) | b \in \text{LE}\} \). The new silent event is \( \cdot \); it is timed by \( [(\text{begin}(\cdot), \cdot), (\text{end}(\cdot), \cdot)] \).

(ii) \( \text{Con}^{' \prime} = \text{df} \langle (i_{0}(\text{Con}), (i_{0}(\text{Con})) \cup \bigcup \left\{ i_{0}(e) \cup i_{1}(e) | e \in \text{Con}^{' \prime} \right\} \).

(iii) The effect of \( i_{0} \) and \( i_{1} \) on orders is defined in a slightly subtle way, so that all of \( S^{' \prime} \) is greater than all of \( S \). In order to do this smoothly we will create a new labelled event \( \cdot \) = \( [(\text{end}(\cdot), \cdot), (\text{begin}(\cdot), \cdot)] \) with \( \cdot \in \text{LE} \) to take the place of the two missing silent event transitions, so that \( \text{LE} = \text{df} \langle \text{LE}, \cdot \rangle \).
Define on tags \( i_0((l, e),) = ((0, 0, e),, i_0(*, ) = *, i_0(\ast_l) = \ast_l \)
\[ i_0(<s) = \text{def} \{ ((r_{i_0(l)}, i_0(a_{i_0(l)}),) \rightarrow ((r_{i_0(l)}, a_{i_0(l)}),) \mid ((r_{i_0(l)}, a_{i_0(l)}),, (r_{l(l)}, a_{l(l)})) \in <s} \]
and similarly \( i_1((l, e),) = ((1, 1), e),, i_1(*, ) = *_{r}, i_1(\ast_l) = \ast_{r} \)
\[ i_1(<s) = \text{def} \{ ((r_{i_1(l)}, i_1(a_{i_1(l)}),) \rightarrow ((r_{i_1(l)}, a_{i_1(l)}),) \mid ((r_{i_1(l)}, a_{i_1(l)}),, (r_{l(l)}, a_{l(l)})) \in <s'} \]
Now \(<s' = \text{def} i_0(<s) \cup i_1(<s') \cup \{\text{begin}(\ast), \text{end}(\ast)\}. \)

A categorical description of (a construction similar to) global sequential composition (as a restriction of the free product) is given in section 2.6.

We can give a sliver of justification for \( \ast \); it is like \( \checkmark \) in CSP; it tells us that one process has successfully terminated.

Aside: given structures with silent events, there are four possible ways to do sequential composition; stick silent events together (i.e. wait until one silent event is over and then start the next silent event), stick the events of the second structure onto the silent event of first (i.e. wait until one silent event is over and then start the events of the second one), stick the silent event of the second structure onto the end of each maximal \( \text{Con-set} \) of the first (i.e. wait until this execution is over then start the next silent event), and stick the events of the second structure onto the end of each maximal \( \text{Con-set} \) of the first (i.e. wait until this execution is over then start the events of the second one). The first is global sequential composition, and the last local sequential composition.
2.4 Synchronisation

Parallel composition presupposes a notion of synchronisation. In this section, which was influenced by [Petri 1986], we will discuss a definition of synchronisation that encompasses the features of what is called 'synchronisation' in two disparate areas; languages for parallel computation, and (what we called in the introduction) physical concurrency. In the former area, synchronisation means the mutual participation of two autonomous parallel processes in some act, usually either a communication from one to the other or some common event. In the latter area, which is typified by hardware, (see [Marino 1981]) there is genuine primitive concurrency in the sense that the two 'processes' are in different places. Such systems cannot be simulated on a sequential machine (as more abstract (discrete) concurrency can) without the loss of many behaviours (because the signals exchanged are truly continuous). In this area 'synchronisation' usually means an exchange of signals prior to some communication or common action. The following two slogans seem to be a description of what is common to these two notions:

Synchronisation involves the possibility of waiting.
Synchronisation involves having something provided if you want it.

In all forms of synchronisation there is the possibility that one process or entity might have to wait for another to be ready. This is equivalent to ensuring that something is eventually provided when it is needed. Hence the two slogans above are equivalent. Hardware synchronisation is essentially used to ensure that signals are available until they have been properly dealt with. Occam–style synchronisation [Jones & Goldsmith 1988] is used to ensure that a communication happens in such a way that both partners can be sure of what has been communicated.

It is important to distinguish between mere communication, which just involves the passage of information, and synchronisation which we shall assume involves common participation in some action. The former may involve the latter (e.g. in CSP [Hoare 1985]), but it is not, in general, necessary; a parallel process can communicate by sending a signal out without waiting to see whether it is heard and by whom. If it waits, then, in addition to the communication that is happening, we shall say that it is synchronising with whatever it is communicating with. The distinction is that between two processes that merely communicate and two that must always synchronise, for instance (using the CSP notation) between

\[ a!7 \rightarrow \text{SKIP} \parallel a?x \rightarrow \text{SKIP} \quad \text{and} \quad a \rightarrow \text{SKIP} \parallel a \rightarrow \text{SKIP} \]

The situation can be compared to the gedankenexperiment of throwing sacks of mail between moving trains (an example of physical concurrency). A communication is just throwing a sack; synchronisation is waiting to receive a sack in return as well (which may indicate that the first sack reached its target).
The difference is illustrated in the diagram:

\[
\begin{array}{ccc}
\text{sender's event} & \rightarrow & \text{Communication (left)} \\
\text{receiver's event} & & \\
\text{sender's event} & \rightarrow & \text{arrow of time} \\
\text{receiver's event} & & \\
\text{Synchronisation} & \rightarrow & \text{(right)} \\
\end{array}
\]

Figure 2.5 – Communication and Synchronisation

Thus, finally, we reach our working definition:

*The synchronisation of matching events must involve some temporal overlap between their intervals.*

Hence, for two labelled events, \( a = [(r_{(a)}, a_1), (r_{(a)}, a_2)] \) and \( b = [(r_{(b)}, b_1), (r_{(b)}, b)] \), to be able to synchronise, the intersection of their time intervals, \( [r_{(a)}, r_{(b)}] \cap [r_{(b)}, r_{(a)}] \), must be non-empty; in that case we write \( a \text{ synch } b \).

In future we will write \( a \cap b \) both for the subset of the reals \( [r_{(a)}, r_{(b)}] \cap [r_{(b)}, r_{(a)}] \) and for the timing \( [(\max(r_{(a)}, r_{(b)}), (a, b)), (\min(r_{(a)}, r_{(b)}), (a, b))] \). The meaning will usually be obvious from the context.

The thinking expounded in this section might be summarised by saying that it is only sensible to consider the possibility of synchronisation between two labelled events when there is some temporal overlap between them. Any weaker requirements on synchronisation would result in a theory that would not be a good model of physical concurrency.

Notice, parenthetically, that our notion of concurrency relies on distribution. If two transitions are concurrent then it should be possible for them to occur spatially separated: this is just one of the earliest ideas of Petri, but it seems (unjustly, to me) to have waned in currency. This viewpoint brings out the difference between occurrences on the same branch (which may be simultaneous but can’t be concurrent) and things on different branches which may be both. Different concurrent branches can happen in different places; any notion of synchronisation of concurrent happenings should make sense within this view. (Notice, incidentally, that if spatial separations are large, then there may be important relativistic effects as well, and our definition of \text{synch} will have to be altered. We shall stick to a model where time runs at the same rate everywhere.)

(The whole issue of synchronisation is discussed in [Murphy 1987] and [Chaney & Molnar 1973]. The latter makes particularly scary reading.)
2.5 The Free Product

Consider two I.E.S.s, 3 and 3'; one way to represent the parallel composition of 3 and 3', which we shall write as 3 \parallel 3', is via an I.E.S. whose labelled events consist of pairs of labelled events, one member of the pair from 3 and one from 3'. A labelled event like (a, b) in 3 \parallel 3' should represent the synchronisation of a from 3 and b from 3'. Unsynchronised events can be thought of as synchronisations with the matching silent event; (a, *') will represent the unsynchronised occurrence of a. This insight is due to [Winskel 1984]; it leads us to examine the categorical product of I.E.S.s.

**DEFINITION 2.5 – The Free Product of Interval Event Structures**

Suppose we have I.E.S.s 3 and 3' as usual. The *product* of 3 and 3', written 3 \times 3', is defined to be an I.E.S., 3'* = (LE', \text{Con}', <') where

1. Given two events, a = (l, e) in LE, and b = (l', e') in LE', we will write the compound event ((l, l'), (e, e')) as (a, b).

2. LE' = \{ (a, b) \mid a \in LE, b \in LE' \}. (Remember that * \in LE and *' \in LE'.) Suppose that a = [(r_\text{lo}, a_\text{lo}), (r_\text{hi}, a_\text{hi})], b = [(r_\text{lo'}, b_\text{lo'}, b_\text{hi'})], then (a, b) will be timed by [(\min(r_\text{lo}, r_\text{lo'}), (a, b)), (\max(r_\text{hi}, r_\text{hi'}), (a, b))]. This timing will, in future, be written as a \cup b. The real part of this is just the union of the intervals timing a and b with any gaps between their end-points filled in. The new silent event, *' = (*, *') = * \cup *'.

3. Any consistent set of labelled events from 3 should be consistent with any consistent set of events from 3'. Hence we have the conditions that

   \text{Con}^* = \{ c \times c' \mid c \in \text{Con}, c' \in \text{Con}' \}

   So, in any given set Con-set in the product, any given event can either synchronise with anything or appear asynchronously. If it synchronises it will appear as a labelled event of the form (a, b), with b \neq *', while if it occurs asynchronously, there will be a (a, *') in c'. All possible synchronous and asynchronous behaviours are possible; hence the term *free product*.

Notice that we can represent parallel composition without synchronisation by a construction like the disjoint sum, but using this definition for Con' rather than the one in section 2.2 (iii).

4. Clearly 3 \times 3' must have all the orderings of 3 and all those of 3' via events like (a, *') and (*, b). In this definition we shall, for clarity, explicitly display some of the tags. The individual orders for 3 and 3' give us the assertions...
\( \forall (a_1, \ast'), (a_2, \ast') \in \text{LE}^\ast \text{ with } t_{(i)} = (r_{(i)}, a_{(i)}) \in a_1, t_{(i)} = (r_{(i)}, a_{(i)}) \in a_2. \)
\( t_{(i)} \leq t_{(i)} \Rightarrow (r_{(i)}, (a_1, \ast')) \leq (r_{(i)}, (a_2, \ast')) \)
\( \forall (a, b_1), (a, b_2) \in \text{LE}^\ast \text{ with } t_{(i)} = (r_{(i)}, b_{(i)}) \in b_1, t_{(i)} = (r_{(i)}, b_{(i)}) \in b_2. \)
\( t_{(i)} \leq t_{(i)} \Rightarrow (r_{(i)}, (a, b_1)) \leq (r_{(i)}, (a, b_2)) \)

We also have some orderings due to synchronisations. First we will consider the ordering between asynchronous and synchronous occurrences. Suppose that \( a = [t_{(a)}, t_{(a)}] \in \text{LE}, b = [t_{(b)}, t_{(b)}] \in \text{LE}^\ast \) where \( t_{(a)} = (r_{(a)}, a_1) \) etc. & \( (a, b) \in \text{LE}^\ast \) (so that \( a \) and \( b \) synchronise). Remember that \( (a, b) = a \cup b \) so suppose that \( (a, b) = [t_{(a,b)}, t_{(a,b)}] \). We want everything that is \( \leq \)–related to a time of \( a \), or \( \leq \)–related to a time of \( b \) to be similarly related to a time of \((a, b)\).

The only point where care is needed is in deciding which point; we only have \( t_{(a,b)}, t_{(a,b)} \) to play with, so we must be careful not to violate the linear order. (If \( (r, a) \leq (r', a') \) then \( r < r' \).) Suppose \( a' = [t_{(a')}, t_{(a')}], t_{(a')} \in \text{LE}^\ast \) and \( (a', \ast') = [t_{(a')}, t_{(a')}], t_{(a')} \in \text{LE}^\ast \) then if \( t_{(a')} \leq t_{(a)} \) we can only assert \( t_{(a')} \leq t_{(a)} \) (for if we had \( t_{(a')} \leq t_{(a)} \) \( \Rightarrow (t_{(a')}, t_{(a')}) \text{ might be } < t_{(a')} = r_{(a')} \Rightarrow (t_{(a')}, t_{(a')}) \text{ so we would violate the linear order). Thus, for } a, b, a' \text{ as above, } \)

1. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \) \begin{equation*}
\text{where } t_{(a)} = t_{(a)}, t_{(a)} = t_{(a)} \text{ otherwise}
\end{equation*}

2. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \)

3. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \)

4. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \) \begin{equation*}
\text{where } t_{(a)} = t_{(a)}, t_{(a)} = t_{(a)} \text{ otherwise}
\end{equation*}

and symmetrically for \( b' = [t_{(b')}, t_{(b')}], t_{(b')} \in \text{LE}^\ast \) with \( (b', \ast') = [t_{(b')}, t_{(b')}], t_{(b')} \in \text{LE}^\ast \).

(That is, 1. becomes

5. \( t_{(b')} \leq t_{(b')} \Rightarrow \left(t_{(b')}, t_{(b')}\right) \leq \left(t_{(b)}, t_{(b)}\right) \) \begin{equation*}
\text{where } t_{(b)} = t_{(b)}, t_{(b)} = t_{(b)} \text{ otherwise}
\end{equation*}

and so on.)

Now consider the order we get from two synchronisations. Suppose \( a = [t_{(a)}, t_{(a)}] \in \text{LE} \) and similarly for \( a', b \) and \( b' \) and suppose \( (a, b'), (a', b') \in \text{LE}^\ast \). Then

1. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \) \begin{equation*}
\text{if } t_{(a')} = t_{(a)}
\end{equation*}

2. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \) \begin{equation*}
\text{if } t_{(a')} = t_{(a)} \& t_{(a')} = t_{(a)}
\end{equation*}

3. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a'}), t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \)

4. \( t_{(a)} \leq t_{(a)} \Rightarrow \left(t_{(a')}, t_{(a')}\right) \leq \left(t_{(a)}, t_{(a)}\right) \) \begin{equation*}
\text{if } t_{(a')} = t_{(a)}
\end{equation*}

and symmetrically for \( b \) and \( b' \).
Finally, of course, end points are related

$$\forall a \in \text{LE}^*, a = [t_i, t_f]. t_i \leq^* t_f$$

The order $\leq^*$ is the smallest transitive relation satisfying these conditions.

A similar technique can be used for $\equiv^*$.

It is obvious that $3^*$ is an I.E.S. It is also a representation of the parallel composition of $3$ and $3'$ since it contains all unsynchronised occurrences of labelled events from $3$ and $3'$ together with all possible synchronisations between labelled events of $3$ and labelled events of $3'$. Notice that the silent event of $3 \times 3'$ is $(*, *)$.

One slightly unfortunate feature of this construction is that it can disconnect branches; things which were once causally-related to starts may end up only being so related to finishes, leaving the starts dangling. Moreover, an order that on two events that become two synchronisations may not be represented at all.

**Theorem** – The product of I.E.S.s is a categorical product in $\text{IES}_j$ and $\text{tIES}$

**Proof:** Consider the projections $\pi_0(a, b) = a$ and $\pi_1(a, b) = b$. These suggest possible morphisms from $3 \times 3'$ to $3$ and $3'$ (these are, in fact, morphisms, as $\leq_0$ and $\leq'$ are both obtainable from $\leq^*$, as if two transitions are ordered in $3 \times 3'$ then their projections will be ordered in $3$ or $3'$). For an arbitrary I.E.S., $\Psi$, and arbitrary morphisms $j_0 : \Psi \rightarrow 3$, $j_1 : \Psi \rightarrow 3'$ there is a unique morphism $j : \Psi \rightarrow 3 \times 3'$ that makes the diagram below commute

For a labelled event $\psi$ in $\Psi$ define $j(\psi) = (j_0(\psi), j_1(\psi))$. Then, obviously,

$$\pi_0 \circ j = j_0 \text{ and } \pi_1 \circ j = j_1$$

so the diagram commutes. Further $j$ is unique; consider $k : \Psi \rightarrow 3 \times 3'$, take $\psi$ in $\Psi$. Suppose $k(\psi) = (a, b)$. Then if $k$ makes the diagram commute $j_0(\psi) = \pi_0(k(\psi)) = \pi_0(a, b) = a$ and $j_1(\psi) = \pi_1(k(\psi)) = \pi_1(a, b) = b$ so $k(\psi) = (j_0(\psi), j_1(\psi))$ and so $k = j$. Since $(a, b) = a \cup b$ this construction works in $\text{tIES}$ too.

**Figure 2.6** – Commuting diagram for the product construction.
The categorical product is a description of parallel composition, but it is an inadequate one; it contains synchronisations that cannot occur. Even if the intersection of the intervals of \( a \) and \( b, a \cap b, \) is empty, \((a, b)\) is still a labelled event in \( 3 \times 3' \). This synchronisation cannot occur, though, because \( a \) is over before \( b \) starts, or vice versa, and thus the criteria of section 2.4 are violated.

(Products (and limits in general) are only unique up to isomorphism, so note that, in the categories \( \text{IES}_j \) in particular, there may very well be other constructions for the product which give an isomorphic result. These might be worth investigating.)

Before we go on to explore more adequate descriptions three other things ought to be said. We shall outline an interpretation of the product in the synchronous categories. Then we outline why a result similar to the CCS decomposition theorem cannot be recovered from our constructions. Finally we give an example of the product construction (also known as the free product) for two simple process trees.

The paper [Winskel 1984i] deals with a synchronous product and with a decomposition theorem, so must we: –

(i) The Synchronous Product. For Winskel, the product in synchronous categories like \( \text{IES}_{j,\text{syn}} \) plays a useful role in giving a semantics to synchronous calculi like SCCS. We shall not investigate this line of reasoning, firstly since it is very hard to see what use a timed synchronous model might have, given the problems of building synchronisers, distributing clock signals and so on, ([Murphy 1987] discusses these problems in some detail), and secondly since the relationship between time and causality makes the veracity of such a construction extremely problematic.

(ii) A decomposition theorem. It is at this stage that a decomposition theorem is usually introduced. There can be no decomposition theorem for I.E.S.s for two reasons. The first is well-known; no poset in which the N-poset (of chapter one) is embedded can be decomposed into elemental posets using choice, sequential composition and parallel composition alone. ([Boudol & Castellani 1988] discuss this result.) The second reason is less obvious, and, like many subtle features of this model, has to do with the interplay between time and causality. The problem has to do with the uniqueness of points in branching time where branches join up: instability means that we don't know where a given point has come from; there is no unique maximal securing guaranteeing some labelled events. (A \( j \)-securing is maximal if it is a \( j \)-securing and there is no set that contains it that is also a \( j \)-securing. Obviously maximal e-securings don't make much sense.)
EXAMPLE

Consider the following two process trees, whose free product is illustrated in figure 4.7 over the page:

\[ \begin{array}{c}
\text{a}_1 \quad \text{a}_2 \quad \text{a}_3 \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{b}_1 \\
\downarrow \\
\text{b}_2 \\
\end{array} \]

Next we shall move on to deal with parallelism in more detail. First we shall concentrate on describing just which synchronisations can occur, then we shall deal with asynchronous happenings more fully.
Figure 2.7 -
The Free Product Construction
2.6 Synchronising Structures

In this section we shall concentrate on describing correctly the possible synchronisations which can occur between two interval event structures. Notice that in any representation of an execution of the parallel composition of two I.E.S.s, $S$ and $S'$, with $a \in LE$, $b \in LE'$, there will be exactly one of $(a, b)$ and $((a, *)$ or $(*, b)$ (with $a \neq *, b \neq *)$).

First notice that $(a, *)$ is an invalid synchronisation if $a \cap * = \emptyset$, so, in order to have a uniform treatment of synchronous and asynchronous events, we must agree to time both $*$ and $*$ by $* \cup *'$ before proceeding. This just means that we want to regard both structures as completely contemporaneous. Parenthetically, notice that we have the (CSP-like) situation where more than two events can synchronise; a composite event results (not a $\tau$ as in CCS). Our only restriction on synchronisation is that just one event from each I.E.S. is allowed to participate in any given synchronisation. Now consider just the genuine synchronisations. These form a set: –

**DEFINITION 2.6 - Synchronisation sets**

The set of synchronisations which occur when two I.E.S.s, $S$ and $S'$ say, are composed in parallel will be denoted by $S \parallel S'$. This set, called a *synchronisation set* (or, sometimes, a *synchronisation relation*) of $S$ and $S'$, is a subset of the cross product of $LE$ and $LE'$

$$(S \parallel S') \subseteq LE \times LE'$$

We now indicate the properties of this set: it only contains valid synchronisations

$$(a, b) \in (S \parallel S') \Rightarrow (a \cap b) \neq \emptyset$$

$$(a, b) \in (S \parallel S') \Rightarrow a \neq * \& b \neq *'$$

but it may not contain all (or any) of these. The first of these conditions ensures that $(a, b) \in (S \parallel S') \Rightarrow a \; \text{synch} \; b$. We will only allow a synchronisation set to contain at most one synchronisation for each labelled event

$$(a, b) \in (S \parallel S') \Rightarrow \neg \exists b' \neq b. (a, b') \in (S \parallel S') \& \neg \exists a' \neq a. (a', b) \in (S \parallel S')$$

The labelled events in a synchronisation set form a structure by themselves, – a structure just containing synchronisations. This structure can be formed from a restriction of the product to the given synchronisation set. (If $S \parallel S'$ then $S$ and $S'$ both run entirely oblivious of the other.) Subject to these restrictions, the user must provide her own synchronisation set.

We need not impose the requirement that the synchronisation set contains at most one synchronisation for any given labelled event from either structure; we could allow a more general construction and then say that only synchronisations which occur in the Con-set given will appear, but that alternative is rather complicated and of dubious utility, so we shall not pursue it.
**DEFINITION 2.7 – Restriction**

An I.E.S., $\mathcal{S}$, **restricted** to a set $A \subseteq \text{LE}$, written $\mathcal{S} \downarrow A$, is defined to be an I.E.S., $\mathcal{S}'$, where

(i) $\text{LE}' = \text{def } \text{LE} \cap A$.

(ii) $\text{Con}' = \text{def } \{ c \cap A \mid c \in \text{Con} \}$

(iii) Two times are $<$ one another if they were previously $\leq$ one another and are both in the new structure;

$$ (r_1, a_1) < (r_2, a_2) \Leftrightarrow ((r_1, a_1), (r_2, a_2)) \in \text{LE}'$$

This definition preserves the relation $t_{a(\omega)} < t_{a(\eta)}$ for all $a \in A$. We shall suppose that restriction cannot remove silent events, and usually time $*$ with the same interval as $\omega$.

The synchronisation set justifies every synchronisation by displaying a suitable labelled event timed with the appropriate intersection. We shall call the obvious structure on the synchronisation set the **intersection** structure:

**DEFINITION 2.8 – The intersection structure**

For two I.E.S.s, $\mathcal{S}$ and $\mathcal{S}'$, the intersection structure of $\mathcal{S}$ and $\mathcal{S}'$, written $\mathcal{S} \cap \mathcal{S}'$, is defined to be the I.E.S. $(\mathcal{S} \times \mathcal{S}') / (\mathcal{S} + \mathcal{S}')$ except that

* every event of $\mathcal{S} \cap \mathcal{S}'$, $(a, b)$ say, will be timed by $a \cap b$, except that the silent event of $\mathcal{S} \cap \mathcal{S}'$, $*$ say, will be timed by $* \cup *'$.

* the definition of $<$ is slightly different, since we have the intersection timing, and deal only in synchronisations. Suppose $a \equiv [t_{a(\omega)}, t_{a(\eta)}] \in \text{LE}$ and similarly for $a', b, b'$ and suppose $(a, b), (a', b') \in (\mathcal{S} + \mathcal{S}')$. Then

1. $t_{a(\omega)} < t_{a(\eta)} \Rightarrow (t_{a(\omega), b}, t_{a'(\eta), b'}) \in <$ if $t_{a(\omega), b} = t_{a(\eta)}$
2. $t_{a(\omega)} < t_{a(\eta)} \Rightarrow (t_{a(\omega), b}, t_{a'(\eta), b'}) \in <$ if $t_{a(\omega)} = t_{a(\omega), b} \& t_{a'(\eta), b'} = t_{a'(\eta)}$
3. $t_{a(\omega)} < t_{a(\eta)} \Rightarrow (t_{a(\omega), b}, t_{a'(\eta), b'}) \in <$ if $t_{a(\eta), b'} = t_{a(\eta)}$
4. $t_{a(\omega)} < t_{a(\eta)} \Rightarrow (t_{a(\omega), b}, t_{a'(\eta), b'}) \in <$ if $t_{a(\omega), b} = t_{a(\omega)}$

and similarly for $\leq$ (In fact, the definition is even more horrific as we have to consider in 2. the case $r_{a(\omega), b} = r_{a(\eta)}$ and in 3. the case $r_{a(\omega), b} = r_{a(\eta)}$ separately. Both of these cases are dealt with by tinkering with $=\omega$.)

Again, this construction can disconnect branches; we sometimes end up with no sensibly-timed transition to be causally related as desired, so we have to forget that particular causal relation (and hence no $<$ relation represents that unfortunate piece of causality).
The intersection structure allows us to treat multiple parallel compositions correctly. If we time \((a, b)\) by \(a \cup b\) (as we did in the definition of \(S \times S'\) in the last section) then the parallel composition modelled does not satisfy

\[ A \parallel (B \parallel C) \not\equiv (A \parallel B) \parallel C \]

To see this, consider three structures with labelled events \(a = [0, 2], b = [1, 4],\) and \(c = [3, 4].\) Clearly \(\text{synch } b\) and \(\text{synch } c\) but \(-(\text{synch } c).\) If the synchronisation \((a, b)\) is timed by \(a \cup b\) then \((a, b) = [0, 4]\) so \((a, b) \text{ synch } c\) and the composite \(((a, b), c)\) is a valid synchronisation.

However, since \(-(\text{synch } c)\) we cannot form \(((a, c), b)\) which means that \(A \parallel (B \parallel C)\) is not even isomorphic to \((A \parallel C) \parallel B.\) (We do, at least, have \(A \parallel B \equiv B \parallel A.\)) This all comes about, of course, from the atransitive nature of co.

In order to ensure that \(((a, (b, c))\) can be formed just when \(((a, b), c)\) can, we will time the composite \((a, b)\) by \(a \cap b;\) now in our example \((a, b) = [1, 2]\) and \(-(\text{synch } c).\) We still have the problem that different occurrences can happen in the structure \(A \parallel (B \parallel C)\) compared with \((A \parallel B) \parallel C,\) but at least the synchronisation behaviour is independent of the order of parallel composition.

Notice, before moving back to deal with asynchronous occurrences, that the intersection structure can be a good model of global sequential composition; if \(\text{begin}(3') > \text{end}(3)\) then the set \(3 \uparrow 3'\) will be empty and \(3T\uparrow 3'\) will be a good model of the elusive global sequential composition.

The description of \(3 (\oplus) 3'\) as \(3T\uparrow 3'\) with \(3 \uparrow 3'\) empty is sufficiently clear (lacking, for instance, the unpleasant addition of \(*\)) that \(3 (\oplus) 3'\) will be defined thus.
2.7 Parallel composition

We shall now return to asynchrony. Consider a single labelled event \( a \neq * \) in an I.E.S. \( S \). Either \( a \) can synchronise with some \( b \) from \( S' \) to form \( (a, b) \) or it can occur alone.

A structure which reflects this intuition, the union structure, will be defined. The union structure derived from \( S \) and \( S' \) will be written \( S \cup S' \). It has four kinds of labelled events:

(i) silent event \(*\) = \(*\) \( \cup *\),
(ii) asynchronous occurrences from \( S \), \((a, *) = [\tau_0], \tau_0 \),
(iii) asynchronous occurrences from \( S' \), \((*, b) = [\tau_0], \tau_0 \),
(iv) and synchronisations \((a, b) = a \cup b\).

We want the union structure to respect the synch predicate, so a synchronisation \((a, b)\) will be in \( S \cup S' \) iff it is in \((S \cap S')\). Of course, \((a, b)\) lasts for all of \( a \cup b \), so (having the intersection structure to record our intersection information) this need not trouble us.

**Definition 2.9** – The union structure

The union structure of two I.E.S.s, \( S \) and \( S' \), is defined to be an I.E.S. \( S' \) where

(i) The labelled events of \( S' \) are formed from pairs of labelled events from \( S \) and \( S' \),

\[ LE' \subseteq LE \times LE' \]

all and only synchronisations in the synchronisation set will be in \( S' \),

\[ \forall a \neq * \in LE, b \neq *' \in LE'. (a, b) \in (S \cap S') \iff (a, b) \in LE' \]

and each labelled event from either structure occurs just once in \( S' \)

\[ \forall a \neq * \in LE, \exists! b \in LE'. (a, b) \in LE' \]

\[ \forall b \neq *' \in LE', \exists! a \in LE. (a, b) \in LE' \]  

From these axioms we can deduce that every labelled event is either part of a unique synchronisation or it occurs asynchronously

\[ \forall a \neq * \in LE. \exists! b \neq *' \in LE'. (a, b) \in LE' \]

\[ ((a, *) \in LE' \& \exists b \neq *' \in LE'. (a, b) \in LE') \]

and similarly for \( b \neq *' \in LE' \). A composite event \((a, b) \in LE'\) will be timed by the interval \( a \cup b \) while an asynchronous occurrence like \((a, *)\) or \((*, b)\) will be timed by the time of \( a \) or \( b \) respectively. (This asymmetry between synchronous and asynchronous happenings is to prevent asynchronous events’ times blowing up to the whole of the duration of the other structure.)
(ii) We can think of the consistency set, $\text{Con}^*$, as formed from two distinct components: asynchronous occurrences are consistent with any set from the other structure; for $\forall a \in \text{LE} \neq \ast, b \in \text{LE}' \neq \ast'$.

$$(a, \ast') \in \text{LE}^\prime \Rightarrow \bigcup_{c \in \text{Con}^*} \{(\ast, b) | b \in c', (\ast, b) \in \text{LE}^\prime \} \cup \{(a, \ast')\} \in \text{Con}^*$$

$$(\ast, b) \in \text{LE}^\prime \Rightarrow \bigcup_{c \in \text{Con}^*} \{(a, \ast') | a \in c, (a, \ast') \in \text{LE}^\prime \} \cup \{(\ast, b)\} \in \text{Con}^*$$

while synchronous occurrence must be consistent in both structures:

$$(a, b) \in \text{LE}^\prime \Rightarrow \{c \times c' | c \in \text{Con}, a \in c, \& c' \in \text{Con}; b \in c'\} \in \text{Con}^*$$

$\text{Con}^*$ is defined to be the smallest set with these properties.

(iii) The definition of $<\S$ is an exact repeat of that for $\S \times \S'$ for the same reasons.

**PROPOSITION** – The relationship between the intersection and union structures

There is a diagram

![Diagram](image)

in $\text{tIES}$ which commutes.

**Proof:** Consider the morphisms $p_0$ and $p_1$ defined by cases;

(i) $p_0(\ast) = \ast$, $p_1(\ast) = \ast'$

(ii) $p_0(a, \ast') = \ast$, $p_1(a, \ast') = \ast'$

(iii) $p_0(\ast, b) = \ast$, $p_1(\ast, b) = \ast'$

(iv) $p_0(a, b) = a$, $p_1(a, b) = b$

The injection morphisms $i_0(\ast) = \ast'$, $i_0(a) = (a, b)$ and $i_1(\ast) = \ast'$, $i_1(b) = (a, b)$ will make the diagram commute. (The image of consistent sets is consistent by construction, and from equation $(\ast)$ in definition 2.9 we know that there is a unique $b$ for $i_0$ to pair $a$ with and similarly for $i_1$.)
**THEOREM** – The morphisms $i_0, i_1$ are a pushout of $p_0, p_1$ in TIES

Given a pair of arrows $p_0: \mathfrak{S}\mathfrak{S}' \to \mathfrak{S}, p_1: \mathfrak{S}\mathfrak{S}' \to \mathfrak{S}'$, a pushout of $p_0, p_1$ is a pair of arrows, $i_0: \mathfrak{S} \to \mathfrak{T}\mathfrak{T}\mathfrak{T}', i_1: \mathfrak{S}' \to \mathfrak{T}\mathfrak{T}\mathfrak{T}'$ such that the last diagram commutes.

Further for every $j_0: \mathfrak{S} \to \Psi, j_1: \mathfrak{S}' \to \Psi$ where $j_0 \circ p_0 = j_1 \circ p_1$ there is a unique arrow, say $j: \mathfrak{T}\mathfrak{T}\mathfrak{T}' \to \Psi$ such that

![Diagram](image)

commutes. We propose that the pair $i_0$ and $i_1$ defined in the proposition above form such a pushout.

**Proof:** We already know that the first diagram above commutes. For the second diagram:

(i) Suppose that the silent event of $\Psi$ is $^\star \Psi$. Then $j_0(\cdot) = j_1(\cdot) = j(\cdot) = ^\star \Psi$.

(ii) The definition $j_0(a) = j_1(b) = \Psi$ for some $\Psi$ in $\Psi$ and $(a, b) \in (\mathfrak{S} \downarrow \mathfrak{S}')$ certainly makes the diagram commute, if $j(a, b) = \Psi$. For the uniqueness of $j$ suppose that $k \circ i_0 \circ \pi_0 = k \circ i_1 \circ \pi_1$. If $j \neq k$ then there are some $a, b$ such that $k(a, b) \neq \Psi$. But then $k \circ i_0 \neq j_0$ or $k \circ i_1 \neq j_1$, violating the assumption that the diagram commutes.

(iii) It just remains to prove $j$ is, in fact, an arrow. By definition $j$ preserves order. To see that it is temporal note there are two cases. Since $j_0$ and $j_1$ make the diagram commute, if $i_0(a) = (a, b)$ then the interval occupied by $j_0(a)$ is no larger than that occupied by $i_0(a)$. The only other case is $i_0(\cdot)$ which follows immediately by the definition of temporal morphism on $\cdot$. 

$\diamondsuit$
2.8 Coequalisers and Initial Objects

Suppose we have two I.E.S.s, $\mathcal{I} = (\mathcal{I}, \text{Con}, <\cdot>$) and $\mathcal{I}' = (\mathcal{I}', \text{Con}', <\cdot>$), and two different homomorphisms between them, $f$ and $g$. Suppose further that there is a morphism from another I.E.S., $k: \mathcal{I}' \rightarrow \Theta$:

\[
\begin{array}{ccc}
\mathcal{I} & \xrightarrow{f} & \mathcal{I}' \\
\downarrow{g} & & \downarrow{k} \\
\Theta & & \Theta
\end{array}
\]

The morphism $k$ can be thought of as the cokernel (difference cokernel in [Mac Lane 1971]) of the morphisms $f$ and $g$ if the diagram commutes. For some object $\Theta'$ and arrow $j: \mathcal{I}' \rightarrow \Theta'$ with $f \circ j = g \circ j$, if $j$ factors uniquely through $k$ (i.e. if there is a unique arrow $k': \Theta \rightarrow \Theta'$ such that $k' \circ j = k$), then $k: \mathcal{I}' \rightarrow \Theta$ is said to be a coequaliser for $f$ and $g$.

A category $C$ is said to have an initial object, $1$, if, for any object $o$ of $C$ there is a unique arrow $i: 1 \rightarrow o$. Similarly, $C$ has a terminal object, $0$, if, for any object $o$ of $C$ there is a unique arrow $i: 0 \rightarrow o$.

All of the categories $\text{IES}_j$, $\text{tIES}$, $\text{IES}_{j\text{syn}}$ and $\text{tIES}_{\text{syn}}$ have initial objects. These are simple to construct; consider an I.E.S. with just a silent event in it. This silent event, $\ast$, maps to the silent event of any other structure. Since $\ast \subseteq_j \ast$ is tautologous, conditions (ii) and (iii) of the homomorphism definition are trivially satisfied:

FACT – Initial and terminal objects

The I.E.S., $\mathcal{I} = (\{\ast\}, \{\{\ast\}\}, \{((-\infty, \ast), (\infty, \ast))\} )$, the structure containing just a silent event that lasts all time, denoted by $1$, is initial in the categories $\text{IES}_j$ and $\text{tIES}$.

The I.E.S., $\mathcal{I} = (\{\ast\}, \{\{\ast\}\}, \{((0, \ast), (0, \ast))\} )$, the structure containing just a silent event that lasts no time, denoted by $0$, is terminal in the categories $\text{IES}_j$. Note that $\text{tIES}$ has no terminal object; such an object would both have to last no time, as $0$ does, and also have non-empty intersection with all possible labelled events, as $1$ does. The combination is impossible.

LEMMA – The existence of coequalisers and pushouts in general

A category $C$ has all finite coproducts if it has an initial object and any pair of objects has a coproduct. The following statements are equivalent for any category $C$ ([Blyth 1986] section 3.4): –
(i) $C$ has all finite coproducts and coequalisers,
(ii) $C$ has all finite coproducts and finite intersections,
(iii) $C$ has pushouts and an initial object.

More generally, if we have an initial object, coproducts of all pairs of objects and coequalisers of all pairs of arrows, then, by Corollary 1, page 109 of [Mac Lane 1971], we have all finite colimits and hence a category that is small-cocomplete.

If, additionally, we have a distinct terminal object, finite products and equalizers of all pairs of arrows, we have all finite limit and hence a category that is small-complete.

(Notice that Freyd’s theorem about small small-complete categories being preorders (page 110, [Mac Lane 1971]) will not apply to us since none of our categories are small.)

We shall find pushouts useful so, having all finite coproducts, it seems sensible to construct coequalisers. Unfortunately, we can’t:

**Theorem tIES does not have coequalisers**

Consider a labelled event $a = [(r_o, a_o), (r_0, a_0)]$ in $\mathcal{S}$, and suppose that we have two different temporal morphisms $f, g : \mathcal{S} \to \mathcal{S}'$. We would like to construct a $k : \mathcal{S}' \to \Theta$ so that

$$\mathcal{S} \xrightarrow{f} \mathcal{S}' \xrightarrow{k} \Theta$$

commutes. Consider $f(a) = b = [(r_0, b_0), (r_0, b_0)]$ say and $g(a) = b' = [(r_0, b'), (r_0, b')]$ say. Since $f$ and $g$ are temporal morphisms, $[r_0, r_0, r_0] \subseteq [r_0, r_0, r_0]$ and $[r_0, r_0, r_0] \subseteq [r_0, r_0, r_0]$. But, $k$ must equalise $f$ and $g$, so $k(f(a)) = k(g(a)) = c = [(r_0, c), (r_0, c)]$ say. Since $c$ must be temporal too, $[r_0, r_0, r_0] \subseteq [r_0, r_0, r_0]$ and $[r_0, r_0, r_0] \subseteq [r_0, r_0, r_0]$. But, this is only possible if $[r_0, r_0, r_0] \cap [r_0, r_0, r_0] = \emptyset$, which is not necessarily so. Hence, in general, we cannot construct $k : \mathcal{S}' \to \Theta$ in tIES, and so tIES does not have coequalisers.

This is rather unfortunate. The obvious change, requiring that a temporal morphism expands rather than contracts durations, will not work either, since then we cannot preserve i-securings. (If we make the change, then there are structures which coequalise simulations, but they are not universal. As an example, define the subset of LE’,

$$A = \{ b \mid \exists a \in \text{LE}. f(a) = g(a) = b \}$$

and form the obvious injection $k : \mathcal{S}' \to \mathcal{S}' \uparrow A$. This coequalises $f$ and $g$, but to see that it is not universal, consider $b \in \text{LE}' - A$. Clearly, $k(b) = \star \Theta$, but $j : \mathcal{S}' \to \Theta$ which also coequalises $f$ and $g$ can take $b$ anywhere, hence we won’t be able to form the unique $h : \mathcal{S}' \uparrow A$ necessary to show that $k$ is universal, since $h(k(b))$ must be $\star \Theta$. )
2.9 Concluding remarks and reformulations

It is time to take stock of what has been seen in this chapter so far. Categories of interval events structures were defined whose objects were I.E.S.s and whose morphisms indicated when one structure could simulate another. We defined choice between I.E.S.s and sequential compositions of I.E.S.s as limits in our categories. We discussed notions of parallel composition involving synchronisation, and decided that a labelled event \( a \) from an I.E.S. \( 3 \) could synchronise with a labelled event \( b \) from an I.E.S. \( 3' \), giving rise to a compound event \( (a, b) \), just when \( a \cap b \neq \emptyset \). The synchronisation set \( 3 \cap 3' \) was introduced to indicate which events we would like to synchronise.

The categorical product was examined as a model for parallel composition and was found wanting. We saw that a representation of parallel composition that incorporated information about the intersection and union of the time intervals of synchronising events was needed. The union and intersection structures were introduced, and a pleasant categorical relationship was found between \( 3, 3', IT3', \) and \( 3|3' \).

Notice that

* We have two structures, the intersection and the unions I.E.S.s, that together form our description of parallel composition;
* the intersection structure describes which synchronisations happened and suggests an intuitively pleasing treatment of multiple parallel compositions, while
* the union structure describes what the I.E.S. formed from the parallel composition can (be observed to) do.

It will be convenient in the light of these results to assume in future that:

* An interval event structures \( 3 \) from now on should be regarded as a pair consisting of \( 3^- \) and \( 3_+ \). Each structure has the same events and the same causality, but different timings. The timings in \( 3^- \) will indicate the times labelled events are prepared to synchronise, while those in \( 3_+ \) will indicate how long they last. If \( a = [t, t] \) in \( 3_+ \) then \( a = [t', t] \) in \( 3^- \) with \( [t', t] \subseteq [t, t] \). For I.E.S.s that have been generated without using parallel composition, \( 3_+ = 3^- \), (so \( 3 \ op 3' = ((3^- \ op 3' -), (3_+ \ op 3'_-)) \), \( op \in \{+, \oplus\} \)), but for a parallel composition \( 3 \ ll_T 3' \), the intersection and union structures are non-trivial; \( (3 \ ll_T 3')^- = 3|3' \) and \( (3 \ ll_T 3')_+ = 3IT3' \). This is the definition of the parallel composition of two I.E.S., \( 3 \ ll_T 3' \). (When deciding whether \( a \) synch \( b \), of course, we should use the timings in \( 3^- \) and \( 3^- \) while those in \( 3_+ \) and \( 3'_+ \) will tell us how long the composite \( (a, b) \) will last.)
The synchronisation set will be given with every parallel composition; \( S \parallel_T S' \) indicates that \( S \) and \( S' \) are in parallel composition, with synchronisation set \( T \subseteq LE \times LE' \).

One interesting result of the investigation of parallel composition was a further insight into sequential composition. The sequential composition of two I.E.S.s can be seen as a special case of their parallel composition where one structure starts after the other finishes. In this case no synchronisations are possible and we have a construction similar to the global sequential composition of section 2.3.

We investigated the properties of limits in categories of I.E.S. a little further, and discovered, much to our chagrin, that \( \mathbf{tIES} \) did not have coequalisers, and hence wasn’t small-cocomplete. Our proof relied on the impossibility of building certain temporal morphisms, so it remains an open question whether our other categories are small-cocomplete or not.

This investigation of the existence of limits in our categories (bar a small detour) ends this chapter. In the interlude which follows some other categories of interval event structures are examined, with a view to separating causal from timing concerns. This investigation will enable us to ascertain the consistency of structures which purport to be interval event structures.

The advertised detour concerns the class of all I.E.S.s, \( \mathbf{IES} \). Just what should be in it?

**Definition 2.11** — The class of all interval event structures

In this definition we briefly recall some of the salient features of I.E.S.s in order to be able to define the class of I.E.S.s, \( \mathbf{IES} \). Not all of the properties of an I.E.S. are given here.

Recall that an I.E.S. was a triple \( (LE, \Con, \prec) \) with \( LE \subseteq (L \times E) \cup \{ \ast \} \), and \( \ast \in LE \). The set \( E \) must be finite and \( L \) must be countable. The set of transitions \( Tr = (LE_\ast \cup LE_\ast) \) was endowed with some causality and each element of it was given a time. The branching times assigned to transitions obey

\[
\forall t_s = (r_s, a_s), t_b = (r_b, b_s) \in BT. \quad t_s \prec t_b \iff (r_s < r_b \& a_s \prec b_s)
\]

\[
t_s = t_b \iff (r_s = r_b \& a_s = b_s)
\]

The special nature of \( \ast \) includes the requirement

\[
\forall a \in LE. (\ast \prec a \lor \ast = a) \& (a \prec \ast \lor a = \ast)
\]

Further, starts must be related to finishes; \( \forall a \in LE. a \prec a \).

Notice the asymmetry between causality and timing: \( \forall a, a' \in LE. (a \prec a' \Rightarrow r_i \prec r_i' \) and \( a = a' \Rightarrow r_i = r_i' \)).
Furthermore, we have the disjunction $a_1 \neq a_2$ or $a_1 \sqsubseteq a_2$ or $a_1 \sqsupset a_2$ or $(a_1 \sqsubseteq a_2 \lor a_2 \sqsubseteq a_1)$ or $a_1 \subseteq a_2$ or $a_2 \subseteq a_1$.

The consistency predicate, $\text{Con} \subseteq \wp(\text{LE})$ must be subset-closed,

$$X \in \text{Con} \land Y \subseteq X \Rightarrow Y \in \text{Con}$$

and affect separate branches,

$$\forall a_1, a_2 \cdot (a_1 \neq a_2) \Leftrightarrow (a_1, a_2)$$

Furthermore, $\ast$ is in every maximal $\text{Con}$-set, $\forall c \in \mathcal{M}(3), \ast \in c$.

We also require finite choice; the set of maximal $\text{Con}$-sets is finite:

$$|\mathcal{M}(3)| \in \mathbb{N}$$

If all the times in an I.E.S. $\mathcal{S}$ are well-defined on $\mathbb{R} \cup \{T\}$, and the above axioms hold, then $\mathcal{S} \in \text{IES}$.

If, in addition, we have

$$\forall r_1, r_2 \in \mathbb{R} \cdot \forall c \in \text{Con} \cdot \{(a_1 \neq a, a \in [(r_1, a_1), (r_2, a_2)], [r_1, r_2] \cap [r_1, r_2] \neq \emptyset) \in \mathbb{N} \}$$

(finite density)

and

$$\forall t \in \mathbb{B} \cdot \{(t \neq t, t \in \mathbb{B}) \in \mathbb{N} \}$$

(finite causes)

and

$$\forall a \in \text{LE} - \{\ast\} \cdot \text{begin}(a) \uparrow \land \text{end}(a) \uparrow$$

(defined transitions)

then $\mathcal{S}$ is said to be computational.

(Notice the change in the definition of finite density; we demand in any $\text{Con}$ set that a finite number of things can happen in any finite interval. This is more philosophically attractive than demanding finite density for the whole structure at once.)

The class of all computational I.E.S.s will be written $\text{cIES}$.
References


Mac Lane [1971] S. Mac Lane, Categories for the Working Mathematician, Springer-Verlag Graduate Texts in Mathematics.


The reader will have noticed that we have introduced categories where the morphisms correspond to relationships between interval event structures; there is an arrow from $3$ to $3'$ just when $3'$ can simulate $3$. We are dealing exclusively with simulation at this level, – we are not interested in how $3$ or $3'$ do their computation, except insofar as that effects the observed structure of events.

Another approach, which allows us to argue at the level of causal orderings, is possible. In this case we aim to represent a single I.E.S. as a diagram in a category. Then operations on I.E.S.'s will correspond to operations on diagrams rather than objects in a category, and hence would turn out to be limits or colimits.

This approach has been used extensively by Vaughan Pratt's group [Pratt et al. 1989], and we will briefly indicate how to incorporate it into our framework. It will lead us to a characterisation of a structure whose causality is consistent with its timing, and a technique that might lead to a way of freely generating timing information that satisfies a given causality. This last work is an application of [Thompson 1984], [Thomason 1987].

It should be stressed that the material in this interlude, like that in all the interludes, is rather sketchy and speculative. We will merely give the briefest possible introduction to the ideas: we will not, for instance, investigate limits in the categories we shall define, consigning the treatment of operations-as-limits to further work.

### 12.1 Categories of causality

Most concurrency theory is about causality. In particular, it is common to model causality as a partial order, or a preorder, so that the behaviour of a concurrent system is just a member of the category of all partially-ordered sets $POSet$, or the category of all preordered sets $PrSet$. However, this view assumes that there is one form of causality, whereas we have seen that, with interval-like events it is natural to have four forms of causality, the $\leq_j$. In this interlude we will explore which category seems to be a natural generalisation of $POSet$ to our framework with four forms of causality. Then we will investigate how to give timing to transitions within this categorical framework.

### 12.2 The properties of the causal orders revisited

First we need to encapsulate our logic of causality. The basis for this is the orderings $\leq_j$ between labelled events. Recall that these were related in chapter one by inclusion; we had the diagram overleaf.
This tells us about the strength of the causal precedence orders. How are they composed? Suppose \( a \leq_j b \) and \( b \leq_k c \), and write the strongest relationship that can be inferred between \( a \) and \( c \) as \( (a \leq_j \leq_k c) \). The composition of the \( \leq_j \) is defined by a multiplication table (overleaf). The order \( \leq_j \) runs down the left and \( \leq_k \) along the top, while the relationship between \( a \) and \( c \), \( a \leq_j \leq_k c \), is found in the body.

The relationship between \( a \) and \( c \) is not always one of the \( \leq_j \); the set \( \leq_j \) is not closed under \( \leq \). For this reason we introduce two new orders:

- \( \emptyset \) means the weakest relationship, so that \( a \emptyset b \) always holds.
- \( = \) means the strongest relationship, so that \( a = b \) holds only if \( a \) is the same as \( b \).

Clearly, if \( a = b \) then \( a \leq_1 b \), and if \( a \leq e b \) then \( a \emptyset b \), so we can extend the lattice a little.

Our new lattice is

\[
\begin{array}{c}
\leq_1 \\
\emptyset \\
\end{array}
\]

while the multiplication table for \( \text{Border} = \{ \emptyset, \leq_e, \leq_b, \leq_1, = \} \) under \( \leq \) is as given overleaf.
For convenience we will now let $\ll_j$ stand for some member of the set $\text{BOrder}$.

We can make the set $\text{BOrder}$ into a category, $\text{BOrder}$ say, by allowing an arrow from some $x \in \text{BOrder}$ to $y \in \text{BOrder}$ just when $x \ll y$. An interesting question is what happens to $\otimes$: the tensor product $\otimes$ on elements of $\text{BOrder}$ lifts to an obvious bifunctor $\otimes : \text{BOrder}^2 \rightarrow \text{BOrder}$, with unit $=$.

The structure $(\text{BOrder}, \otimes, =)$ is a monoid (and there is a similar monoid on arrows): for this reason, $\text{BOrder}$ is a special sort of category, a \textit{strict monoidal category}:

A strict monoidal category, $\mathcal{D} = (\mathcal{D}, \otimes, I)$ is a category $\mathcal{D}$ together with a \textit{tensor product}, (namely a bifunctor $\otimes : \mathcal{D}^2 \rightarrow \mathcal{D}$) and an object $I$ of $\mathcal{D}$, called the \textit{unit}, such that both the object and the morphism part of $\mathcal{D}$ each form a monoid under $\otimes$ with identities $I$ and $\text{id}_I$. (Often $\otimes$ is just called 'tensor'.)

A monoidal category, $\mathcal{D} = (\mathcal{D}, \otimes, I, \alpha, \lambda, \rho)$ is a category $\mathcal{D}$ together with a \textit{tensor product}, a unit $I$, and isomorphisms $\alpha, \lambda, \rho$ satisfying $\alpha : ((x \otimes y) \otimes z) \cong (x \otimes (y \otimes z)), \lambda : I \otimes d \cong d$ and $\rho : d \otimes I \cong d$. (There are also certain \textit{coherence conditions}, outlined in [Mac Lane 1971], but these need not detain us here.) It is easy (although tedious) to check that $\text{BOrder}$ is a strict monoidal category with tensor $\otimes$ and unit $=.$

\section*{12.3 Using the properties of $\text{BOrder}$}

Having discovered that $\text{BOrder}$ is a strict monoidal category it is natural to ask what use this information is. The answer, as we shall see in this subsection, is that we can characterise a valid set of labelled events and their causality using $\text{BOrder}$. In this subsection the terminology of [Pratt et al. 1989] will be used, rather than the more standard usage of [Kelly & Street 1974].

<table>
<thead>
<tr>
<th>$\otimes$</th>
<th>$=$</th>
<th>$\ll_j$</th>
<th>$\ll_k$</th>
<th>$\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\otimes$</td>
<td>$=$</td>
<td>$\ll_j$</td>
<td>$\ll_k$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$i$</td>
<td>$i$</td>
<td>$i$</td>
<td>$i$</td>
<td>$\ll_j$</td>
</tr>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$t$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$h$</td>
<td>$h$</td>
<td>$h$</td>
<td>$h$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$c$</td>
<td>$c$</td>
<td>$h$</td>
<td>$c$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Suppose we are just interested in causality. This can be expressed using the elements of \( \text{BOrder} \); we can write \( a \sqsubseteq_b b \) meaning that \( a \) must cause \( b \), and the timing must be assigned in such a way that \( a \sqsubseteq_b b \) holds. Seen this way, it is interesting to know when some set of untimed labelled events and some specified set of \( \text{BOrder} \) relations over them can be timed consistently (i.e. so that the specified \( \text{BOrder} \) relations respect the timing). In this subsection we shall give a characterisation of the sets of untimed labelled events and \( \text{BOrder} \) relations—that can be so consistently timed. This characterisation will use the concept of enrichment.

Suppose we have a set \( A \) which we want to turn into a category, and a small monoidal category \( D = (D, \otimes, I, \alpha, \lambda, \rho) \). For every pair \( (a, b) \) of elements of \( A \) we would like to associate an object of \( D \) in such a way that this object represents the arrows from \( a \) to \( b \). This is the basic idea of a \textit{enriched} over \( D \).

In particular, we would like to associate every pair of labelled events in \( A \) with a member of \( \text{BOrder} \). However, we cannot do this freely; firstly if we assign \( \sqsubseteq_j \) to \( (a, b) \) and \( \sqsubseteq_k \) to \( (b, c) \) then we must assign \( (\sqsubseteq_j \otimes \sqsubseteq_k) \) (or something stronger, if it holds) to \( (a, c) \). Secondly, we must assign \( = \) to \( (a, a) \).

A small \( \text{BOrder} \)-category, or set enriched over \( \text{BOrder} \) consists of a set \( A \) together with a function \( \delta : A^2 \rightarrow \text{BOrder} \), families of morphisms of \( \text{BOrder} \),

\[
m_{abc} : \delta(a, b) \otimes \delta(b, c) \rightarrow \delta(b, c)
\]

(for composition) and an isomorphism \( j_a : I \rightarrow \delta(a, a) \) such that the following diagrams (below and overleaf) commute. Since \( \text{BOrder} \) is strict monoidal this is a special case of a category enriched over a symmetric monoidal category (cf. Interlude 3).

\[
\begin{align*}
\delta(c, d) &\otimes \delta(b, c) &\otimes \delta(a, b) &\Longrightarrow &\delta(c, d) \otimes (\delta(b, c) \otimes \delta(a, b)) \\
\downarrow m_{bcd} &\otimes I &\delta(b, d) &\otimes \delta(a, b) &\Longrightarrow &\delta(a, d) &\leftarrow &\delta(c, d) \otimes \delta(a, c) \\
\downarrow &\otimes m_{abd} &\delta(b, d) &\otimes \delta(a, b) &\Longrightarrow &\delta(a, b) \leftarrow &\delta(c, d) &\otimes \delta(a, c) \\
\end{align*}
\]

\( a \sqsubseteq_j b \&\ b \sqsubseteq_k c \&\ c \sqsubseteq_l d \Rightarrow a \sqsubseteq_l (b \sqsubseteq_k \otimes c) d = (a \sqsubseteq_j \otimes \sqsubseteq_k) c \sqsubseteq_l d \)

\[
\begin{align*}
I &\otimes \delta(a, b) &\Longrightarrow &\delta(a, b) &\otimes I \\
j_a &\otimes I &\downdownarrows &\delta(a, a) &\otimes \delta(a, b) &\Longrightarrow &\delta(a, b) &\leftarrow &\delta(a, b) \otimes \delta(b, b) \\
\downarrow m_{aab} &\otimes m_{abb} &\delta(a, a) &\otimes \delta(a, b) &\Longrightarrow &\delta(a, b) &\leftarrow &\delta(a, b) \otimes \delta(b, b) \\
\end{align*}
\]

\( a \sqsubseteq_j a \)

The laws of enrichment, rather neatly, express just the properties we want of the relationship between $A$ and $\mathbf{Border}$: if $A$ is a category whose objects are labelled events, and whose morphisms are objects of $\mathbf{Border}$, then $A$ can be given a consistent timing just when $A$ is expressible as the set of objects of $A$, obj$(A)$, enriched over $\mathbf{Border}$.

The category of all sets enriched over $\mathbf{Border}$ will be called $\mathbf{QOSet}$, in analogy with the category of all sets enriched over $2$ (the two element category with $0 < 1$), the more familiar $\mathbf{PrSet}$. There arrows of $\mathbf{QOSet}$ are just the $\leq$–preserving functions (i.e. if $A$ and $B$ are categories represented as the sets $A$ and $B$ enriched over $\mathbf{Border}$ respectively, then there is a morphism $f$ from $A$ to $B$ just when $\forall a \in A, b \in B, \text{rel } a \leq f(a) \text{ rel } f(b)$.)

I2.4 Validating timings

The programme of validating our structures can be taken one stage further. In this section we shall see one might freely generate valid timing information from a member of $\mathbf{QOSet}$. In [Thomason 1987] an adjunction between a category of linear orders and a category very like $\mathbf{QOSet}$ is given.

(Thomason’s category is equipped with a ‘wholly precedes’ relation rather than our ‘wholly precedes or abuts,’ $\leq_l$, a ‘begins strictly before’ relation rather than our ‘begins before or at the same time as,’ $\leq_b$, a ‘ends strictly before’ relation rather than our ‘ends before or at the same time as,’ $\leq_e$, and an ‘abuts to the left relation’ which we might write as $(a \leq_l b \& (a \leq_c c \leq_c b \Rightarrow b \leq_e c))$, so he can be seen as taking the $<$ case rather than the $\leq$ one. He has the same definition of morphism, $\exists f : A \rightarrow B$ just when $\forall a \in A, b \in B . a \text{ rel } b \Leftrightarrow f(a) \text{ rel } f(b)$, but with his relations rather than ours.)

The key points of Thomason’s work are summarised below:

* Any sensible model of timing should be based on a linear order.

* Hence, given any model of time, R, Q, N or whatever, and a causal category much like an element of $\mathbf{QOSet}$, Thomason can assign timing to happenings in a universal way.

* It should be possible to add a few axioms to $\mathbf{QOSet}$ (essentially adding an ‘abuts’ relation) and then use a similar construction to generate an adjunction between $\mathbf{QOSet}$ and the category of linearly ordered sets, $\mathbf{LinSet}$. This would enable us to characterise valid timings axiomatically and universally.
Bibliography


*Thomason* [1987] S. Thomason, *Free construction of time from events*. Manuscript, Department of Mathematics, Simon Fraser University, Burnaby, B.C.

This chapter is devoted to an examination of a notion of equivalence of interval event structures. A variety of different inter-related semantics will be presented. All semantics are based on an idea of **behavioural difference**; two I.E.S.s should have a different meaning iff they can be **observed to behave differently**. Various notions of 'observation' and 'difference' will be examined, leading to various semantics. The more detailed the notion of behaviour, the more discriminating the associated semantics.

Two orthogonal approaches will be taken to investigate various notions of behaviour: the first will concern the **timing** of behaviour. We will start by considering the order in which the transitions of labelled events happen, and then move on to consider stricter timing conditions, requiring things to happen within a certain interval of time. Our most temporally-constraining semantics will be based on the precise time things happen.

The second approach we will take concerns the **functional** behaviour of an I.E.S., - what it does rather than when it does it. Our paradigm will be observational: consider an observer, in the spirit of the last section of chapter one, viewing an execution of an I.E.S. After a while, rather than continuing to observe the structure, she goes to a betting shop to bet on what the I.E.S. will do next. (In this chapter we only consider computational I.E.S.s that start at \( t = 0 \) that is, \( \text{begin}(\star) = 0 \), and are bounded. This gives us handy finiteness properties; no structures extend infinitely backwards or forwards in time, and all structures can only display a finite number of labelled events. Furthermore, we will assume that we are always dealing with the union structure \( S \); it would be straightforward to extend our results to the intersection structure \( S^- \).)

Firstly she will be allowed to bet on the occurrence of a single transition (the start or finish of a given occurrence of an event); she will bet on which transition the structure will display next. The possibilities that she **may** win the bet and that she **must** win the bet will be distinguished. Various combinations of bets will then be considered; these will allow the observer to deduce more about the functional behaviour of the I.E.S. under scrutiny. In partic-
ular, we shall characterise the extra discrimination available once certain combinations of bets are allowed.

The observable behaviour of an I.E.S. is brought about by its internal causality; it is usually assumed that causality cannot be distinguished from mere temporal order. The consequences of relaxing this assumption will be investigated; bets not just on what happens and when, but on how it was caused, will be allowed. This sort of bet will be seen to be more powerful (in the sense of necessarily winning on fewer structures) than bets which do not consider causality. For this kind of bet, of course, we must assume an omniscient bookie.

Each of the notions of bet presented (including compound bets, — combinations of bets, — but not the causal bets of the last paragraph) can be seen in the testing tradition of [Hennessy 1988i]. Semantics is all about saying which structures are the same: each kind of bet defines a notion of sameness; two structures are the same under this notion just when all bets about one that must win must also win if placed on the other. (Or, alternatively, we could replace “must” by “may” to obtain a different semantics.)

The aim, then, is to distinguish between interval event structures just when that distinction can be made without leaving the comfort of the bookies'.

The next few sections will be organised as follows. First we present the notion of observing an I.E.S. in more detail, and outline the betting methodology. A primitive bet is introduced, and the conditions when it may and must win are investigated. Next some temporal information is added by considering bets that have a ‘within δ of now’ stipulation. The case $\delta = 0$, — demanding precisely when a transition must happen, — is investigated.

The power of bets to discriminate between structures is considered, and ways of combining primitive bets to increase this power are investigated. Having defined a notion of ‘sameness’, we then go on to define a notion of difference; two structures are different when they cease to be the same. The length of time that two structures remain the same is a measure of their difference, and this leads us to a metric space structure, with untimed and strict-timed forms of betting generating different metric spaces.

A basic grounding in metric spaces is needed for the latter sections; [Kelley 1955] might be suitable. Also of (computer science) relevance are [de Bakker & Mayer 1988], [Golson & Rounds 1983], [Goltz & Loogen 1987], [Kok & Rutten 1988], [Reed 1988] and [Rounds 1983]. (An appendix giving all relevant definitions is provided at the end of the chapter.) We provide some explanatory material.

Finally, in the interlude, we consider a uniform categorical treatment of our semantics in the tradition of Lawvere, and suggest a possible connection between the categories which underlie the work of this chapter and those of chapters two and four. This leads us to suggest a more general interpretation of betting equivalence than the metric space one.
3.0 Observing interval event structures again

The matter of observing I.E.S.s was first tackled in chapter one; there it was discovered that an L.I.E.S. \( L \) was a possible history of \( S \) just when \( L \) was a maximal \( \text{Con} \)-set of \( S \). (More precisely, \( L \) is a valid history of \( S \) if it is equal to some \( c \in \mathcal{M}(S) \) up to bijection of labels, the idea being that labels are just a way of disambiguating occurrences, so the precise way they are assigned should not be important.)

Suppose \( L = (L' \times E') \cup \{\ast\}, < \) is a L.I.E.S.†, \( S = (LE \subseteq L \times E \cup \{\ast\}, \text{Con}, <_S) \) is an I.E.S., \( E' \subseteq E \), and that \( f : L' \rightarrow L \) is a bijection between the disjoint countably infinite sets of labels \( L \) and \( L' \). Then, \( L \) is a valid history of \( S \), iff \( \exists c \in \mathcal{M}(S) \), which, viewed as an L.I.E.S. is \( (LE', <) \), and \( \forall (l', e) = a' = [r_\ast, r_\ast] \in LE' \). \( \exists (l(l'), e) = [r, r] \in LE \). In future we will write \( L \in \mathcal{M}(S) \) meaning \( L = (LE', <) \) and \( LE' \) is equal up to bijection of labels to some \( c \in \mathcal{M}(S) \).

In this section we examine not just entire histories, or complete executions of \( S \), but also observations of just a part of the behaviour of \( S \). We can tell whether a behaviour is complete or not because of the presence of \( \ast \); recall that \( \forall a \in LE \). \( \text{end}(a) < \text{end}(\ast) \) so that when we see \( \ast_f \) (the 'on light' going off) we know that \( S \) has displayed all the behaviour it can in this run. The following definition will enable this to be formalised:

DEFINITION 3.0 – Truncation of L.I.E.S.

What happens if an I.E.S. \( S \) is only observed for a limited period, say from the beginning to \( t = T \)? Only part of some history \( L \in \mathcal{M}(S) \) will be seen. An observation of \( S \) is defined to be any valid history \( L \) truncated at \( T \), written \( L^{T} \), defined thus;

Suppose \( L = (LE, <) \in \mathcal{M}(S) \). Then \( L' = L^{T} \) for \( T \in \mathbb{R}^{+} \) is an observation of \( S \) iff it is an L.I.E.S. \( L^{T} = (LE', <) \) where

\[
LE' = \{ a \mid a \in LE, \begin{align*}
\text{begin}(a) < T
\end{align*}\}
\]

An event \( a \) in \( LE' \) carries the timing \( [r, r'] \) iff \( a \) in \( LE \) carries the timing \( [r, r] \) where \( r' = r \) if \( r < T \) and \( r' = T \) otherwise. That is, events in \( LE' \) keep their timing except for finishes that happen later than \( T \), which become \( T \).

Notice that every observation must contain both \( \ast \) (with the smallest timing, here assumed to be 0), and \( \ast_f \) (with the largest timing). In a truncated history, \( L^{T}, \text{end}(\ast) \) may be \( T \), — that just indicates that it is incomplete. (Recall that we are dealing with computational structures here, so no times are \( T \) in complete histories.)

† – With \( \ast \in LE' \), of course, as in chapter one.
If \( \alpha \) is some transition of a labelled event in an observation \( L \) we sometimes loosely say that \( \alpha \) has been observed, or that it is observable.

**ASIDE – Testing, Choice, Nondeterminism and Parallelism**

There is a fundamental philosophical difference between interval event structures and the structures that testing equivalence is normally applied to; this concerns the nature of choice. It is important to understand this difference before proceeding, as the semantics presented in the next few sections represents an attempt to explore what testing equivalence would be like under our assumptions about the nature of parallelism and choice, rather than the conventional CCS ones. In the rest of this section a reasonable understanding of the testing methodology of CCS is assumed; see [Hennessy 1983] for details. Chapter four contains a further discussion of nondeterminism which extends that given in this chapter. Also of relevance is [Abramsky 1987]; there observational equivalence is shown to be formulateable as a testing equivalence — here we pursue almost an opposite programme in considerably less depth, showing how testing equivalence can be redefined using single observations.

Recall from chapters one and two that no interpretation has yet been placed upon either the disjoint sum construction or upon the conflict relation. Here and henceforth it will be interpreted as internal choice or pure nondeterminism: if we have \( a \neq b \) then we may see either \( a \) or \( b \) but not both, and we cannot predict or effect which is seen; similarly we cannot know or influence which of \( 3 \) and \( 3' \) the I.E.S. \( 3 + 3' \) will behave like. More generally:

An I.E.S. with conflict will have more than one maximal Con-set. The interpretation placed on this is that \( 3 \) will display one member of the set \( \mathcal{M}(3) \) in any given execution, but we cannot tell or influence which. The reason for choosing the presence of internal rather than external choice as the interpretation of the existence of more than one maximal Con-set will now be discussed.

First consider CCS choice: in (some interpretations of the testing view of) CCS, the choice operator \( + \) can sometimes be influenced: in the case \( a . \text{nil} + b . \text{nil} \) the operator \( + \) represents an external choice; it will do whichever of \( a . \text{nil} \) and \( b . \text{nil} \) we want, and we tell it what we want by participating in the desired alternative, so that \( a . \text{nil} + b . \text{nil} \Downarrow a . \text{nil} \rightarrow \text{nil} \). (Of course CCS isn’t always so obliging, as can be seen by considering \( a . \text{nil} + \tau . b . \text{nil} \), but that isn’t the point right now. [de Nicola & Hennessy 1984] discuss the nature of the CCS \( + \) operation in detail. Note that, due to typographical limitations, we write underbar instead of overbar.)

Now consider a parallel combination of two CSP processes. If the only form of choice the processes contain is internal choice (CSP \( \parallel \)) then one partner of the parallel combination can-
not influence the choices made by the other; the possibility of synchronisation is irrelevant to internal choice. External choice (CSP ||), on the other hand, requires that a choice is resolved through the participation of a partner placed in parallel. The testing methodology was developed with this in mind – a process is tested by placing it in parallel with a tester; the process and the tester must participate in all the processes events, with the tester resolving the external choices of the process (and vice versa). (See [Hennessy 1988i] for a clear and comprehensive explanation.) In the example of the process a.nil + b.nil we can think of a.nil as the tester; the result of the ‘experiment’ a.nil + b.nil || a.nil (successful termination) tells us that a.nil + b.nil must always do an a (if we ask it nicely, i.e. by putting a.nil in parallel.) The testing methodology, then, assumes that partners in parallel participate in each others’ events and resolve each others’ external choices, – we shall call this participatory concurrency. This view does not match our perspective on parallelism very well; we would like to think of an I.E.S. as getting on with things independently of an observer and what she might ‘want’ to see. We have no mechanism for a synchronisation to influence subsequent behaviour (as it must if external choice is to be resolved via parallelism); an interval event structure should be observed not interacted with.

(It is worth noting that this is the view of semantics that Petri net theory leads us to; there we have a rich notion of nondeterminism, and a semantics (in terms of reachable markings) based on observation rather than interaction.)

Interpreting conflict

We have chosen to interpret conflict as due to the presence of internal choice: why ? It certainly fits with our notion of an I.E.S. as an independent entity that can evolve without reference to its environment, and (as we have seen above) it is a reasonable interpretation given our notion of parallelism. Also, the notion of testing has not been applied to entities with non-participatory concurrency to my knowledge, so what follows is an exploration of a new point in the concurrency semantics design space.

It would be best, of course, to have models of both forms of choice – external choice is obviously desirable. However, I feel that it is dishonest not to differentiate between the potential (P||Q may behave like P, or it may behave like Q, depending on what is chosen) and the actual (P\parallel Q will behave like P, or it will behave like Q, regardless of the environment) in a semantics. I do not know how to articulate this difference (i.e. that between “if the environment chooses a now, then ψ” (where ψ is some behavioural property) and just “ψ”) cleanly.

This difference, of course, only becomes important when we have non-participatory concurrency. As long as events can be shared, we can live with external and internal choice in the same framework; it is only when events become private, and synchronisations are merely
composites of private events (so that observation rather than testing is the primary way of finding out about a process), that we are forced to decide what form of choice conflict should model.

(We could interpret the choice present in I.E.S. or event structures due to the presence of conflict as either internal or external; it is merely (philosophically) inconsistent to model both sorts of choice the same way. The main change necessary to our reasoning, if we were to deal with external rather than internal choice, concerns the definition of must; [Hennessy 1988i] contains the necessary details.)

In the next few sections a theory of observational equivalence of I.E.S.s will be presented; this theory is an attempt to transplant the testing theory of CCS to a world without participatory concurrency or external choice. In this world the most sensible approach seems to be to avoid external choice altogether (but see the interlude at the end of chapter four for a possible hack in the world of non-participatory process algebra).
3.1 Betting on transitions

A notion of testing the behaviour of I.E.S.s without interacting with them is presented in this section: it is based on the notion of a bet about what a structure might do. First a formalism is presented that will capture the concept of betting. This is developed to the point of being able to say just when a bet might or must win when placed on a structure, given some information about what it has already done. The forms of bets allowed are then refined to allow bets to be placed not only on what can happen (i.e. the functional behaviour of an I.E.S.) but also when it can happen.

3.1.1 Observing structures

Consider some I.E.S. \( S = (LE, \text{Con}, \prec_S) \) being observed. Suppose, thus far, some (possibly empty apart from \( \ast \)) I.E.S. has been observed. This observation must result from some maximal \( \text{Con} \)-set of \( S \), but we may not have enough information to decide which.

**DEFINITION 3.1 – Observational consistency**

The beginning and ending times of a L.I.E.S. \( L = (LE', \prec) \) are the reals assigned to \( \ast \) in it. (Remember we assume that \( \text{begin}(\ast) \) is always 0 in this chapter, so \( \ast \) will be timed by \([(0, \ast), (r, \ast)] \) where \( r \in R^+ \cup \{T\} \).

A L.I.E.S. \( L = (LE', \prec) \) is observationally consistent with a given maximal \( \text{Con} \)-set \( X \) with \( X = (LE, \prec) \in M(3) \), iff \( L \) is equal (up to the usual isomorphism of labelling sets) to \( X^T \) where \( T = \text{end}(L) \). It is worth expanding this a little. We must have:

(i) All labelled events of \( L \) appear in \( X \) with the same (or more defined) timings; for \( \forall (l', e) = [r_n, r_T] \in LE' \), \( \exists (e(l'), e) = [r_n, r_T] \in LE \) such that \( r_T = r_T \) if \( r_T < \text{end}(L) \) and \( r_T = T \) otherwise. (Remember that we assumed that \( \text{begin}(\ast) \) in \( L \) \( = \text{begin}(\ast) \) in \( S = 0 \).)

(ii) Moreover, we should not have missed any occurrences that happen in \( X \) within the time covered by \( L \); \( \neg \exists (e(l), e) = [r_n, r_T] \in LE \) such that \( r_T < \text{end}(L) \) & \( (l, e) \in LE' \).

We shall write \( P(S, L) \) for the set \( \{X \mid L \text{ is observationally consistent with } X\} \) of possible histories a given observation is consistent with. If we have \( LE' \subseteq (L' \times E') \cup \{\ast\} \) and \( LE \subseteq (\ell(L) \times E) \cup \{\ast\} \), observational consistency implies that \( E' \subseteq E \).
Consider our observer. She has seen $\mathcal{K}$ do $\mathcal{L}$: what next? She is actually observing some $\mathcal{K} \in \mathcal{P}(\mathcal{S}, \mathcal{L})$, so she will either see the start of a new occurrence $a = [r, r] \in \mathcal{L}$ next, where $r = \min \{r \mid a = [r, r], a \in (\mathcal{L} - (\mathcal{L} \times E'))\}$ (i.e. the smallest start in $\mathcal{K}$ which is not in $\mathcal{L}$), or the ending of an old one $a' = [r, r] \in \mathcal{L}'$ where $r' = \min \{r' \mid a' = (I', e) \in \mathcal{L}', (E(I'), e) = [r, r], \text{end}(a') = T\}$ (i.e. the smallest finish in $\mathcal{K}$ which is undefined in $\mathcal{L}$). She will see $a$, or $a'$ depending on whether $r < r'$. In either case we will have a new observation, and, by definition, this observation must be consistent with $\mathcal{K}$. In general, given some $\mathcal{L}$ observationally consistent with $\mathcal{K} \in \mathcal{M}(\mathcal{K})$, we can state when the occurrence of a transition $a_i$ extends $\mathcal{L}$, that is, produces a larger observation, $\mathcal{L} \cup \{a_i\}$ say, which is still consistent with $\mathcal{K}$.

**DEFINITION 3.2 – Extension**

Suppose that we have a L.I.E.S. $\mathcal{L} = (\mathcal{L}', <)$ and a history $\mathcal{K} = (\mathcal{L}, <) \in \mathcal{M}(\mathcal{K})$ with $\mathcal{L}$ consistent with $\mathcal{K}$; $\mathcal{K} \in \mathcal{P}(\mathcal{S}, \mathcal{L})$. Also, suppose we have a transition of an occurrence (a start or a finish of a labelled event) $a_i = (I', e)$, at time $r_i$:

If we have a start, $a_i'$ at $r_i'$, then $a_i'$ extends $\mathcal{L}$ in $\mathcal{K}$, written $a_i'$ extends $\mathcal{L}$ in $\mathcal{K}$, just when

(i) This occurrence hasn’t been observed yet; $r_i \geq \text{end}(\mathcal{L})$ and $a' \in \mathcal{L}'$.

(ii) The beginning we are seeing has the right time; $r_i = r_i'$.

(iii) The occurrence and $a_i$ are still consistent with $\mathcal{K}$ Define $\mathcal{L} \cup \{a_i\}$ as the L.I.E.S. $(\mathcal{L}' \cup \{a_i\}, <)$, with $a_i' = [r_i, r_i]$ in $\mathcal{L}$. Then $\mathcal{L} \cup \{a_i\}$ must be consistent with $\mathcal{K}$.

Likewise, if we had a finish $a_i'$ at $r_i'$, $a_i'$ extends $\mathcal{L}$ in $\mathcal{K}$ iff $\exists (E(I'), e) = [r, r] \in \mathcal{L}$ s.t.

(i) The start of this occurrence has been observed with the right timing but the end hasn’t been seen yet; $\text{begin}(I', e) = r_i$ and $\text{end}(I', e) = T$.

(ii) The end we are seeing has the right time; $r_i = r_i'$.

(iii) It was the right thing to see next, i.e. $\mathcal{K}$ is still consistent with the occurrence and $a_i'$. Define $\mathcal{L} \cup \{a_i\}$ as the L.I.E.S. $\mathcal{L}$ except that $\text{end}(I', e) = r_i$. Then $\mathcal{L} \cup \{a_i\}$ must be consistent with $\mathcal{K}$.

(We assume that the labelling of $a_i$ is consistent with the labelling policy used by $\mathcal{L}$, that is $I' \in \mathcal{L}'$, but $(I', e) \in \mathcal{L}'$. Notice, too, that all we really needed was the condition $\mathcal{L} \cup \{a_i\}$ must be consistent with $\mathcal{K}$; the extra conditions just make it easier to see what is going on.)

Providing at most one labelled event starts or finishes at any time in a given maximal Cons-set, there is at most one transition $a_i$ which extends $\mathcal{L}$ in a given $\mathcal{K}$.
3.1.2 The betting methodology

Given some observation \( L \), the things that \( S \) may do next are just the extensions of \( L \) in each consistent observation \( \mathcal{K} \in \mathcal{P}(S, L) \). This insight enables us to formulate two predicates which will tell us when bets may and must win.

**DEFINITION 3.3** — May and must for transitions

\( S \) must engage in the transition \( a' \) next, given some history \( L \), written \( S \text{ must } a' \text{ after } L \), if it must do it in every history \( L \) is consistent with:

\[
S \text{ must } a' \text{ after } L \iff \forall \mathcal{K} \in \mathcal{P}(S, L) \cdot a' \text{ extends } L \text{ in } \mathcal{K}
\]

while \( S \) may do \( a' \) next if it must do it in some history \( L \) is consistent with:

\[
S \text{ may } a' \text{ after } L \iff \exists \mathcal{K} \in \mathcal{P}(S, L) \cdot a' \text{ extends } L \text{ in } \mathcal{K}
\]

Notice that if \( \mathcal{P}(S, L) \) is empty, \( S \text{ must } a' \text{ after } L \) is true vacuously. This can only happen if \( S \) can display no events, or if we are already in an inconsistent situation (i.e. \( L \) is not a valid observation of \( S \)). The latter case is a ‘miracle’ and anything can happen (this is just *ex falso quod libet*). The former is more irksome and is forbidden by demanding that \( L \cap \{\ast\} \neq \emptyset \).

If \( S \text{ must } a' \text{ after } L \) our observer, having seen \( L \), is certain to win a bet that \( S \) will do \( a' \) next, while if \( S \text{ may } a' \text{ after } L \) she may or may not win, depending on which \( \mathcal{K} \) is actually being executed. Notice in particular that if \( S \) can never be observed to do \( L \), neither \( S \text{ may } a' \text{ after } L \) nor \( S \text{ must } a' \text{ after } L \) holds.

The bet \( S \) will do \( a' \) after having done \( L \) will be formulated as \( S \text{ will } a' \text{ after } L \). It is not necessary to specify a time for \( a' \) in this form of bet; the bet wins if there is some \( a \) in \( L \) with timing \( t \) such that \( a' \) timed by \( t \) extends \( L \) in \( \mathcal{K} \). The only case in which two different bets \( S \text{ will } a_1(\cdot) \text{ after } L \) and \( S \text{ will } a_2(\cdot) \text{ after } L \) (with \( a_1(\cdot) \neq a_2(\cdot) \)) will both win is if \( a_1(\cdot) \) and \( a_2(\cdot) \) have the same branching times in \( S \) (i.e. are necessarily simultaneous). Note that for our purposes we suppose that \( E \) is known in advance; the universe of events whose occurrences can be bet upon is predetermined.

We will always assume that we are dealing with valid observations; the bet \( S \text{ will } a' \text{ after } L \) will always lose if \( L \) is not a valid observation of \( S \).

(The must and may predicates we have defined are rather different from their counterparts used in testing equivalence with participatory concurrency. The predicates we have presented seem to be a plausible translation of the usual ones into a world without participation. See [Aceto 1987] for an alternative (and more conventional) approach.)
3.1.3 Adding time to bets

Let us suppose that our observer, like so many young people today, is rather impatient, and doesn’t want to wait around arbitrarily long to know if she has won a bet or not. She wants to make a timed bet, claiming not only that a transition \( a' \) will be seen next, but also that it will be seen within a time \( \delta \) of completing an observation \( L \). Since we are only dealing with computational structures, transitions are ineluctable, i.e., no transition can be put off indefinitely long, and hence we can take \( \delta \in \mathbb{R} \), rather than \( \delta \in \mathbb{R} \cup \{T\} \), and reserve \( T \) for dealing with transitions whose timing we cannot yet predict.

**DEFINITION 3.4** – Must and may for timed bets

The bet \( S \) will do \( a' \) after having done \( L \), and within \( \delta \) of the end of \( L \) will be formulated as \( S \) will \( (a')_{\delta} \) after \( L \). If \( a' \) extends \( L \) in \( K \) and \( a' \) is timed by \( r \), with \( \text{end}(L) \leq r < \text{end}(L) + \delta \), then we say that \( (a')_{\delta} \) extends \( L \) in \( K \). The must and may relations for \( \delta \)-timed bets can now be defined as

\[
S \text{ must } (a')_{\delta} \text{ after } L \iff \forall K \in P(3, L). (a')_{\delta} \text{ extends } L \text{ in } K
\]

\[
S \text{ may } (a')_{\delta} \text{ after } L \iff \exists K \in P(3, L). (a')_{\delta} \text{ extends } L \text{ in } K
\]

In other words, a \( \delta \)-timed bet on \( a' \) will win if there is an occurrence of the right transition of the right event with an appropriate label which not only extends \( L \) (and hence happens next) but also happens within \( \delta \) of the end of \( L \).

Here again, \( P(3, L) \) must be nonempty for \( S \) must \( (a')_{\delta} \) after \( L \) to hold. (Notice that this is a ‘before \( \delta \) has elapsed ...’ bet, since we have \( r < \text{end}(L) + \delta \) not \( r \leq \text{end}(L) + \delta \).

\(*\)

The bet \( S \) will \( a' \) after \( L \) can be thought of as a bet \( S \) will \( (a')_{\delta} \) after \( L \) with \( \delta = \infty \). This insight leads us to investigate the case \( \delta = 0 \): –

**DEFINITION 3.5** – Must and may for strict-timed bets

The bet \( S \) will do \( a' \) after having done \( L \), at precisely \( t = \gamma \) after the end of \( L \) will be formulated as \( S \) will \( (a')_{\gamma} \) after \( L \). (The ‘\( s \)’ for strict.) This form of bet (referred to in future as \( \gamma \)-timed bets, or strict-timed bets), enables us to check if things happen precisely when we want.

\( a' \) extends \( L \) in \( K \) and \( a' \) is timed by \( r \), with \( \text{end}(L) + \gamma = r \), then we say that \( (a')_{\gamma} \) extends \( L \) in \( K \). The must and may relations for \( \gamma \)-timed bets are

\[
S \text{ must } (a')_{\gamma} \text{ after } L \iff \forall K \in P(3, L). (a')_{\gamma} \text{ extends } L \text{ in } K
\]

\[
S \text{ may } (a')_{\gamma} \text{ after } L \iff \exists K \in P(3, L). (a')_{\gamma} \text{ extends } L \text{ in } K
\]
3.1.4 Examples

It is instructive at this stage to examine some interval event structures, and some observations of them, and to see when various bets may or must win when placed on a structure.

As before, I.E.S.s will be represented diagrammatically; time will flow down the page, lightly dashed lines will indicate causality and thicker dashed lines conflict. Small horizontal lines will indicate the durations of occurrences of events. The formal I.E.S. definition will also be given, in terms of labelled events with their beginning and end times (including *), maximal Con-sets, and the orders $\triangleleft$ and $=_b$ on branching time. Obvious inequalities ($a_i \triangleleft a_j$) will be omitted.

Bets will be framed in terms of the labels used by the structure; this is just a notational convenience: usually the better will not know how the structure is labelling events, and hence must bet in terms of her own labelling system. In general the labels used in an observation will be related to those in the structure being observed via a bijection $\mathcal{E}$, and this bijection can never be discovered by the observer.

![Diagram of I.E.S. structures]

The following hold:

$\mathcal{S}_1$ must $e_{r(1)}$ after $\{\}$,
$\mathcal{S}_1$ may $e_{r(1)}$ after $\{\}$,
$\mathcal{S}_1$ must $e_{r(1)}$ after $e_{r(1)}$,
$\mathcal{S}_1$ must $e_{r(2)}$ after $e_{r(1)}$,
$\mathcal{S}_1$ must $e_{r(2)}$ after $e_{r(1)}$.

$\mathcal{S}_2$ must $e_{s(1)}$ after $\{\}$,
$\mathcal{S}_2$ may $e_{s(1)}$ after $\{\}$,
$\mathcal{S}_2$ must $e_{s(1)}$ after $e_{s(1)}$,
$\mathcal{S}_2$ must $e_{s(2)}$ after $e_{s(1)}$,$(e_{r(1)})$,
$\mathcal{S}_2$ must $e_{s(2)}$ after $e_{s(1)}$,$(e_{r(1)})$.
The following also hold;
\[ S_3 \text{ must } e_{t(1)} \text{ after } \{ \}, \]
\[ S_3 \text{ may } e_{t(1)} \text{ after } \{ \}, \]
\[ S_3 \text{ must } e_{t(1)} \text{ after } \{ e_{s(1)}, e_{s(2)} \}, \]
\[ S_3 \text{ must } e_{t(2)} \text{ after } \{ e_{s(1)}, e_{s(2)} \}, \]
\[ S_3 \text{ must } e_{t(1)} \text{ after } \{ e_{s(1)} \}. \]

\[ S_5 = ( \{ e_1 = [(1, e_{s(1)}), (2, e_{l(1)})], \]
\[ e_2 = [(2, e_{s(2)}), (4, e_{l(2)})], \]
\[ e_3 = [(1, e_{s(3)}), (2, e_{l(3)})], \]
\[ e_4 = [(2, e_{s(4)}), (4, e_{l(4)})], \]
\[ \{ e_1, e_2 \}, \{ e_3, e_4 \}, \]
\[ \{ ((2, e_{l(1)}), (2, e_{s(2)})), ((2, e_{l(3)}), (2, e_{s(4)})) \} ) \]

Some example bets relating to the I.E.S. 3s follow overleaf.
\[ S_6 = ( \{ e_2 = [(2, e_{t(2)}), (4, e_{t(2)}), e_3 = [(0, e_{t(3)}), (2, e_{t(3)}), e_4 = [(2, e_{t(4)}), (4, e_{t(4)})], \{ (e_3, e_2), (e_3, e_4), \} ) \]

Some more examples;

\[ S_5 \text{ may } e_{t(1)} \text{ after } \}, \quad S_6 \text{ may } e_{t(2)} \text{ after } (e_{t(3)}), \]
\[ \neg (S_5 \text{ must } e_{t(1)} \text{ after } \}, \quad \neg (S_5 \text{ must } e_{t(2)} \text{ after } (e_{t(1)}), \]
\[ S_5 \text{ must } e_{t(2)} \text{ after } (e_{t(1)}), \quad S_5 \text{ must } e_{t(3)} \text{ after } (e_{t(3)}), \]
\[ S_5 \text{ may } e_{t(3)} \text{ after } \}, \quad \neg (S_5 \text{ may } e_{t(4)} \text{ after } (e_{t(1)}), \]
\[ \neg (S_5 \text{ must } e_{t(4)} \text{ after } (e_{t(1)}), \]

Finally, some timed examples;

\[ S_5 \text{ must } (e_{t(2)} \text{ after } (e_{t(1)}), \quad S_6 \text{ may } (e_{t(4)} \text{ after } (e_{t(3)}), \]
\[ \neg (S_5 \text{ must } (e_{t(2)} \text{ after } (e_{t(1)}), \quad \neg (S_5 \text{ must } (e_{t(4)} \text{ after } (e_{t(3)}), \]
\[ \neg (S_5 \text{ must } (e_{t(2)} \text{ after } (e_{t(1)}), \quad \neg (S_5 \text{ must } (e_{t(4)} \text{ after } (e_{t(3)}), \]
\[ S_5 \text{ must } (e_{t(2)} \text{ after } (e_{t(1)}), \quad S_5 \text{ must } (e_{t(4)} \text{ after } (e_{t(3)}), \]
\[ S_5 \text{ must } (e_{t(2)} \text{ after } \}, \quad S_5 \text{ must } (e_{t(4)} \text{ after } \} ). \]

Notice that \( S_1 \) and \( S_2 \) are indistinguishable using bets on untimed starts alone, while if \( e_3 = (3, e) \), \( e_1 = (1, e) \), then we cannot always tell \( S_5 \) and \( S_6 \) apart using just one bet (\( S_6 \) may do anything \( S_5 \) must do, so the result of a single bet, if we are unlucky, might not be enough to distinguish them); in the next section this will be remedied as we see how to compile the results of several bets.

N.B. We cannot have conflict present at the start of time, since if something starts at the same time as \( * \), it must be causally the same as it, and hence in every maximal Con-set. This is because we had a single first cause; we cannot handle distributed state at the start of time. Note that this doesn't stop us from modelling \( P + Q \); we just have to assume there is a state before the choice is resolved: nondeterminism is notionally resolved at run-time. If we wanted a distributed starting state, we have to allow some time between \( t = 0 \) and the first action so that the silent event can cause (rather than be necessarily simultaneous with) whatever initial distributed state is required.
3.2 Characterising structures using bets

In this section the *power* of the various sorts of bets to discriminate between histories of interval event structures will be investigated. First a way of combining the results of several bets about a given structure is presented. Then we see how to construct complex bets from simpler ones in order to reason about complex behaviours. The discriminatory power of various kinds of bets is then examined. This section is based on [Abramsky 1987] which, in turn, draws on the work of Hennessy.

3.2.1 Characterising the results of betting on structures

It is time to consider the outcome of a bet in more detail. Any particular bet can win or lose. We shall use $T$ to symbolise a win and $\bot$ for a loss. (Do not confuse the result of a successful bet, $T$, with the undefined time in the future, $T$. ) Given some observation, $L$, we cannot always predict the outcome of the bet $\exists$ will $a'$ after $L$ due to nondeterminism or lack of information. Hence, the set of all results of this bet placed on $\exists$ at any particular time form a set, $R(\exists, L, a') \subseteq \{T, \bot\}$.

**DEFINITION 3.6 – Powerdomains**

The result of any particular bet is either a win, $T$, or a loss, $\bot$, and a win is better than a loss, $T \geq \bot$. The results of all bets placed on $\exists$ in any state $K \in P(\exists, L)$ will be collected into a set of results $R(\exists, L, a') \subseteq \{T, \bot\}$ defined thus;

(i) $T \in R(\exists, L, a') \iff \exists$ may $a'$ after $L$

(ii) $\bot \in R(\exists, L, a') \iff \exists$ must $a'$ after $L$

Since $T \geq \bot$, the result of a single bet placed on one execution lives on the two-point lattice $\Theta$;

$$\Theta = \begin{array}{c} T \\ \bot \end{array}$$

while the set of all possible results of a bet, $R(\exists, L, a')$, lives on one of the powerdomains of $\Theta$. Various ways of forming the powerdomain of $\Theta$ correspond to various notions of how to define the result of a set of bets:

- if we care only about the certainty of winning, then the correct powerdomain to choose is the Smyth powerdomain $\rho_S(\Theta)$; this corresponds to the notion of must,
- while if the possibility of winning is all that is important, (i.e. we are considering may) then the Hoare powerdomain $\rho_H(\Theta)$ will do. Finally,
- if we wish to distinguish between the certainty of winning, the possibility of winning and the certainty of losing, then the Egli-Milner powerdomain $\rho_{EM}(\Theta)$ is the right choice.
Time, causality, and concurrency

David Murphy

Since the Egli-Milner powerdomain (also known as the convex powerdomain) contains the greatest discrimination between the results of multiple bets, it will be used from now on. In the next subsection operations on these powerdomain that correspond to combining bets will be considered; these will be important because the discriminating power of betting will be seen to depend on which ways of combining bets are allowed, rather than on the complexity of a primitive bet. The orders on \( P_s(\Theta) \), \( P_n(\Theta) \) and \( P_{EM}(\Theta) \) will be written \( \succeq_s \), \( \succeq_n \) and \( \succeq_{EM} \) respectively.

First, however, it is necessary to consider what a computable combination of bets might be. An obvious choice is a continuous function. Note that for finite domains monotone functions are automatically continuous.

**FACT - Operations on Powerdomains**

Given any monotone function \( f: \Theta^n \rightarrow \Theta \ (n \geq 0) \), there is a pointwise extension of \( f \) onto \( P_{EM}(\Theta) \), written \( \hat{f} : (P_{EM}(\Theta)^n) \rightarrow P_{EM}(\Theta) \), that is monotone:

\[
\hat{f}(X_1, \ldots X_n) = \{ f(x_1, \ldots x_n) \mid x_i \in X_i, 1 \leq i \leq n \}
\]

The pointwise extension is also multilinear, in that it preserves unions in each argument separately.

**3.2.2 Betting on more elaborate untimed behaviours**

In this section all bets will be of the untimed form \( \exists \ a^\tau \after L \). Suppose that our observer is interested not just in a single transition happening next, but in a sequence of transitions \( a_{\alpha 1}^\tau, a_{\alpha 2}^\tau, \ldots \) happening next.
She can place a series of bets (an *accumulator*)

3 will \( a_{(0)} \) after \( L \), and
3 will \( a_{(0)} \) after \( L \cup \{ a_{(0)} \} \) and
3 will \( a_{(0)} \) after \( (L \cup \{ a_{(0)} \}) \cup \{ a_{(0)} \} \) and

which will enable her to explore this situation. Similarly, if she doesn’t care about which of two transitions happens next, she could bet

\[
\begin{align*}
3 & \text{ will } a_{(0)} \text{ after } L, \text{ and } 3 \text{ will } a_{(0)} \text{ after } (L \cup \{ a_{(0)} \}) \\
3 & \text{ will } a_{(0)} \text{ after } (L \cup \{ a_{(0)} \}) \text{ or } 3 \text{ will } a_{(0)} \text{ after } L
\end{align*}
\]

Clearly, conjunction and disjunction of untimed bets should be powerful enough to distinguish between all L.I.E.S. up to the order their events happen. Hence, the following two operators over the results of bets will be interesting:

\[
\begin{array}{c|c|c|c}
\lor & T & \bot \\
\hline
T & T & T \\
\bot & T & \bot \\
\end{array} \quad \quad \\
\begin{array}{c|c|c|c}
\land & T & \bot \\
\hline
T & T & T \\
\bot & \bot & \bot \\
\end{array}
\]

Figure 3.2 — Truth tables for the disjunction and conjunction of bets

They correspond to the disjunction of bets \((3 \text{ will } a_{(0)} \text{ after } L \text{ or } 3 \text{ will } a_{(0)} \text{ after } L)\) and the conjunction of bets \((3 \text{ will } a_{(0)} \text{ after } L \text{ and } 3 \text{ will } a_{(0)} \text{ after } L)\). By abuse, the pointwise extensions of \(\lor\) and \(\land\) will also be written \(\lor\) and \(\land\). The truth tables for these extensions are shown below.

\[
\begin{array}{c|c|c|c|c}
\lor & \{T\} & \{T, \bot\} & \{\bot\} \\
\hline
\{T\} & \{T\} & \{T\} & \{T\} \\
\{T, \bot\} & \{T\} & \{T, \bot\} & \{T, \bot\} \\
\{\bot\} & \{T\} & \{T, \bot\} & \{\bot\} \\
\end{array} \quad \quad \\
\begin{array}{c|c|c|c|c}
\land & \{T\} & \{T, \bot\} & \{\bot\} \\
\hline
\{T\} & \{T\} & \{T, \bot\} & \{\bot\} \\
\{T, \bot\} & \{T, \bot\} & \{T, \bot\} & \{\bot\} \\
\{\bot\} & \{\bot\} & \{\bot\} & \{\bot\} \\
\end{array}
\]

Figure 3.3 — Truth tables for the pointwise extensions of \(\lor\) and \(\land\).

In addition to conjunction and disjunction of bets, we will allow *quantification* over bets. The observer will be allowed to bet that “there is a run on which the bet will win,” and “on
all runs the bet will win.” This assumes that at any point in time we are allowed to copy an I.E.S. in order to examine the consequences of possible nondeterminism at that point. [Hennessy 1988i] assumes global copying, – that at the beginning of a test an unlimited number of copies of an object are available, – we will assume local copying, – that, at any stage, an unlimited number of copies of an I.E.S. in its current state are available, – as well. This is a fairly stringent, although, I think, reasonable assumption, as a computational I.E.S. can only display finite nondeterminism, (because it has a finite number of maximal Con-sets), and so only a finite number of copies are necessary. The truth tables for all (\( \forall \)) and some (\( \exists \)) are shown below.

The intuition behind the bet combinators \( \forall \) and \( \exists \) can be seen by considering must and may; \( \forall \) just picks out the \( \varphi_{S}(\Theta) \) subdomain of \( \varphi_{EM}(\Theta) \), while \( \exists \) picks out the \( \varphi_{M}(\Theta) \) subdomain. Hence if \( \exists \) will \( a^{*} \) after \( L \) wins, then \( \exists \) must \( a^{*} \) after \( L \), while if \( \exists \) will \( \exists a^{*} \) after \( L \) wins, then \( \exists \) may \( a^{*} \) after \( L \). One might read \( \forall \) as ‘for all possible (nondeterministic) paths from here …’ and \( \exists \) as ‘for at least one possible path from here …’

\[
\begin{array}{c|c|c|c}
\forall & \exists \\
\hline
(T, 1) & (T, 1) & (T) & (T) \\
(T, 1) & (1) & (T, 1) & (1) \\
(1) & (1) & (1) & (1) \\
\end{array}
\]

**Figure 3.4 – Truth tables for the bet quantifiers \( \forall \) and \( \exists \).**

Note that the operators \( \forall, \exists : \varphi_{EM}(\Theta) \to \varphi_{EM}(\Theta) \) are monotone but not linear.

A syntax of bets can now be introduced. We shall write the primitive untimed bet \( \exists \) will \( a^{*} \) after \( L \) as just \( a^{*} \) and assume that \( a^{*} = (l, e) \) where \( e \) ranges over a nonempty universe of events \( E_{bet} \), and \( l \) over a countably-infinite labelling set \( L \). Then, all untimed combination bets \( b \in B(E_{bet}) \) can be specified by

\[
b := Win \mid Lose \mid a^{*} \mid b_{1} \mid b_{2} \mid b_{1} \land b_{2} \mid \forall b \mid \exists b
\]

Similarly, \( B_{5}(E_{bet}) \) will be the class of all \( \delta \)-timed bets, and \( B_{\gamma}(E_{bet}) \) will be the class of all strict-timed bets.

The ‘semantics’ of bets is defined via an extension of the \( R \) function to the new bet combinators. The meaning function \( S : IES \times LIES \times B \to \varphi_{EM}(\Theta) \) is defined by

\[
S(3, L, Win) = \{T\}
\]
\[
S(3, L, Lose) = \{1\}
\]
\[ S(3, L, a' \ b) = R(3, L, a') \land S(3, L \cup \{a'\}, b) \]
\[ S(3, L, b_1 \lor b_2) = S(3, L, b_1) \lor S(3, L, b_2) \]
\[ S(3, L, b_1 \land b_2) = S(3, L, b_1) \land S(3, L, b_2) \]
\[ S(3, L, \forall b) = \forall S(3, L, b) \]
\[ S(3, L, \exists b) = \exists S(3, L, b) \]

Here the auxiliary function \( R \) is defined as before:

- \( R(3, L, a') = \{\top\} \) if 3 must \( a' \) after \( L \),
- \( R(3, L, a') = \{\top, \bot\} \) if \( \neg \) (3 must \( a' \) after \( L \)) and 3 may \( a' \) after \( L \),
- \( R(3, L, a') = \{\bot\} \) if \( \neg \) (3 may \( a' \) after \( L \)).

In the next subsection the uses of compound bets are discussed, while the subsection after that is devoted to exploring the discriminatory power of various kinds of (untimed) bets.

The semantic function for \( \delta \)-timed and strict-timed bets is defined in the obvious way.

### 3.2.3 What are compound bets for?

Intuitively, the idea of the compound bets \( a' \ b, b_1 \lor b_2, b_1 \land b_2, \forall b \) and \( \exists b \) is clear; \( a' \ b \) is just a bet about the next transition, followed by some bet about the state that results from that, \( b_1 \lor b_2 \) is just the bet either \( b_1 \) or \( b_2 \), \( b_1 \land b_2 \) is similarly the bet both \( b_1 \) and \( b_2 \), \( \forall b \) is the bet "in all executions from here \( b \)," while \( \exists b \) is the bet "in some execution from here \( b \)." (The use of the compound bets will be illustrated by reference to the examples first presented in subsection 3.1.4 in subsection 3.2.6.)

Several subsets of \( B \) can be identified as familiar classes of bets; if the observer is just allowed to bet on a sequence of starts (no conjunction, disjunction or quantification) then we have a betting system based on the order in which things start; this is very much like the trace theory popularised by [Hoare 1985]. Note however that, although we cannot distinguish a concurrent system from its sequentialisation using these bets, we can distinguish branching points; in terms of the examples, \( S(3s, \{e_{s(1)}\}, e_{s(2)}\text{Win}) = \{T\} \) & \( S(3s, \{e_{s(1)}\}, e_{s(2)}\text{Win}) = \{T, \bot\} \).

Once bets are allowed on finishes as well as starts we have the ability to distinguish temporal overlap, and thus have roughly the same kind of model as Lamport.

For finite I.E.Ss these bets plus conjunctions and disjunctions of them are sufficient to distinguish all functional behaviours, as is indicated in the next section. We have said that disjunction and conjunction involve local copying, — after any observation we can make two copies of an I.E.S. and observe the result of different bets placed on each copy, combining the results in a computable fashion. The multilinearity of \( \lor \) and \( \land \) then corresponds to the intuition that the results of these tests are independent.
Since no internal unobservable action is possible in an I.E.S., refusal testing [Phillips 1987] is a redundant concept here. The bet “S can refuse to do \( a' \) after \( L \)” can be formulated as \( \mathcal{A}(S, L, a') = \{1\} \).

Notice that if we allowed countable nondeterminism (a situation which might be notated in CSP as \( \prod_{i \in \omega} P_i \)) or infinite density, (an infinite number of transitions in some finite interval of time), then the betting methodology may not be able to extract all the information that can be obtained about I.E.S.s by observation. The ability to formulate quantified bets in the presence of countable nondeterminism presumes the ability to make a countable number of copies of S so that each alternative can be explored separately. This is both philosophically unattractive and technically trying, hence the restriction that \( \mathcal{A}(S, L) \) must be finite for any S and L. This still leaves us with the problem that \( \forall b \) and \( \exists b \) allow us to combine the results of all sub-bets at some point. This relies on the ability to enumerate the (finite number of) alternatives; it may be argued that this enumeration is fundamentally contrary to the spirit of observational equivalence.

The full syntax of bets, \( \mathcal{B}(E_{\omega}) \), can be seen at the most powerful (and least plausible) end of possible observation-based semantics, with various subsets (single bets, \( b := a' \)); traces, \( b := \text{Win} \lor \text{Lose} \lor a' b \); traces & local copying \( b := \text{Win} \lor \text{Lose} \lor a' b \lor b_1 \lor b_2 \lor a_1 \land a_2 \) etc.) having less power and more plausibility. In [Abramsky 1987] it is argued that (a set of test analogous to) \( \mathcal{B} \) is in some sense the largest syntax that makes sense in the participatory world; here we merely remark that \( \mathcal{B} \) (and the timed versions \( \mathcal{B}_\delta \) and \( \mathcal{B}_{\tau}(E_{\omega}) \)) have just sufficient power to make all the distinctions we want to make, as will be seen in the next two subsections.

**ASIDE — On countability**

The total number of transitions any I.E.S. can engage in in any given run is finite since we demanded bounded computational structures. With only bounded-below computational structures the number of points in time at which a transition happens is countable. This is because, in chapter one, we insisted on computable structures having finite density;

\[
\forall t_1, t_2 \in \mathbb{R}, \forall \mathbf{S} \in \text{cIES}, \forall L = (LE, <) \in \mathcal{M}(\mathbf{S}), \forall (a, a \in \text{LE}, a_t = t, t_1 \leq t \leq t_2) | \in \mathbb{N}
\]

We have only allowed untimed and \( \delta \)-timed bets to be placed just after something has happened (i.e. at \( \text{end}(L) \)). This has the advantage that we can limit consideration of bets to a countable number of points in time (analogous to testing over a countable number of states). We propose that it is desirable in all real-timed models to exclude uncountable sets except where strictly necessary. (In the case of strict-timed bets it is necessary to have an uncountably-based bet; things could happen at any \( t \in \mathbb{R} \), they just do happen only on a countable subset of \( \mathbb{R} \).)
3.2.4 On the power of untimed bets

In order to assess the power of untimed bets, it is necessary to formulate an untimed model which keeps the temporal order and simultaneity implicit in an L.I.E.S. The right structure for this will be a sequence of sets of transitions, the sequence order reflecting temporal precedence and the set structure simultaneity. The ordering presented here was first considered on [Hennessy-1988ii]. [Aceto & Hennessy 1988] details an approach to testing process algebras using a test similar to the \textit{S will a} after \textit{L} form of bet.

**DEFINITION 3.7 - L.I.E.S. sequences**

Consider \( \text{Tr} = \text{LE}_t \cup \text{LE}_f \). The sequence generated by a L.I.E.S. \( L = (LE', <) \), written \( \text{seq}(L) \), is a member of \((\varphi(\text{Tr}))^*\). The properties of a sequence are as follows.

Suppose \( P = P_1P_2 \ldots P_n \in (\varphi(\text{Tr} - \varnothing))^* \) is a sequence of nonempty sets of transitions, and that any given \( a_i \in \text{Tr} \) is timed by \( r_i \). Then we require

(i) \( \forall i. \forall a_i, a'_i \in P_i, r_i = r_i' \) (sets contain simultaneous transitions).

(ii) \( \forall i, j. i < j \Rightarrow \forall a_{i0} \in P_i, a_{i0} \in P_j, r_{i0} < r_{j0} \) (sequence order reflects a temporal order).

Note, though, that the times are forgotten in the passage from \( L \) to \( \text{seq}(L) \).

We will define the L.I.E.S. sequence projection operator (for use later): suppose \( \text{seq}(L) = P_1P_2 \ldots P_n \) then \( \pi_m(\text{seq}(L)) = P_1P_2 \ldots P_m \) if \( n > m \) and \( P_1P_2 \ldots P_n \) otherwise.

The sequence \( \text{seq}(L) \) is generated as follows. First consider the simultaneity set of a transition \( [a] = (a, r_i = r_i') \). Then define \( \text{seq}(L) \) as \( P_1P_2 \ldots P_n \in (\varphi(\text{Tr}))^* \) where

(i) \( \forall i. \exists a_i. P_i = [a_i] \)

(ii) \( \forall i, j. i < j \Rightarrow r_{i0} < r_{j0} \) where \( P_i = [a_{i0}] \).

(iii) \( \forall a' \in \text{LE}_r. \exists P_i, a' \in P_i \)

Two sequences are equal just when all their components are equal.* Notice that the \( P_i \)s are finite since we have finite density. Further, the set \( \text{seq}(\text{S}) = \{ \text{seq}(L) \mid L \in \mathcal{M}(\text{S}) \} \) is finite since we have finite nondeterminism.

The next theorem shows that a subclass of untimed bets brings out just the distinctions \( \text{seq} \) over finite observations cares about.

---

* – Pedantically, we should say "up to bijection of labelling sets," since the observer need not necessarily use the same set of labels for recording both observations.
**THEOREM** – The power of (untimed) bets I

Given two computational I.E.S.s $S$ and $S'$, and some observation that can be made of both of them, $L$, we can distinguish between the complete histories, $J \in P(S, L)$ and $K \in P(S', L)$ using untimed bets just when $\text{seq}(J) \neq \text{seq}(K)$. We can do this since we can formulate a compound bet which has a different outcome when placed on $J$ and $K$ using just sequences of primitive bets and conjunction.

**Proof.** Suppose $\text{seq}(J) = J_1J_2 \ldots J_n$ and $\text{seq}(K) = K_1K_2 \ldots K_n$ and suppose further that the first place $\text{seq}(J)$ and $\text{seq}(K)$ differ has index $i$ (i.e. $J_j = K_j$ for $j < i$, $J_j \neq K_j$).

Suppose we have a set $A$. Let $A_J$ be the set of all sequences generated by taking all the elements from $A$ (e.g., if $A = \{a, b, c\}$, $A_J = \{a,b,c, a,c,b, b,a,c, b,c,a, c,a,b, c,b,a\}$). Then the bet

$$B_J = \bigwedge_{j} \{ J_j \implies J_j \implies \text{Win} | J_j \in J_j \}$$

will win when placed on $S$ doing $J$ and it will lose when placed on $S'$ doing $K$. (That is, $\mathcal{R}(S, L, B_J) = \{T, \bot\}$ but $\mathcal{R}(S', L, B_J) = \{\bot\}$.)

This is easy to check. The bet $\bigwedge_{j} \{ J_j \implies J_j \implies \text{Win} | J_j \in J_j \}$ may win on both structures since both display the same transitions up to $J_i$, but the bet $B_J$ can’t win on both since they are different at $J_i$.

Furthermore we can distinguish between the untimed behaviour of structures with more complicated bets:

**THEOREM** – The power of (untimed) bets II

The bet

$$b = \exists J \in \bigwedge_{\text{seq}(S)} B_J$$

will distinguish between all different untimed behaviours.

**Proof:** Suppose $\text{seq}(S) = \{ \text{seq}(J) | J \in M(S) \}$. Now if $\text{seq}(S) \neq \text{seq}(S')$ without loss of generality we can assume that some $\text{seq}(J) \in \text{seq}(S)$ is not being equal to a $\text{seq}(K) \in \text{seq}(S')$. Hence $b$ will distinguish between all untimed finite behaviours, since

$$S(S, \{\star\}, b) = \{T\} \text{ while } S(S', \{\star\}, b) = \{\bot\}$$

Observe that there is a $B_J$ which cannot have the same result when placed on $S$ as when placed on $S'$, and this difference will lead to one of the conjuncts evaluating to $\bot$ when the bet is placed on $S'$, and hence to the whole conjunction evaluating to $\bot$. 

Chapter Three — Meaning of Structures
In contrast, we will always get \( \{ T, \bot \} \) when any bet \( B_J \), for \( J \in \text{seq}(S) \) is placed on \( S \).

So, the form of bet \( S \) will \( \alpha \) \( \sigma \) after \( L \) is capable of extracting all the information about causality that is present in a maximal Con-set (viewed as a L.I.E.S.), if we allow bet combinators as described. Notice, incidentally, that the power of this form of bet depends on having an unlimited number of copies of \( S \) to observe, and on being able to make compound bets composed of the disjunction and conjunction of simpler ones. Also, we must be able to bet on old things ending (\( \text{will } \alpha \)' bets) as well as new things beginning (\( \text{will } \alpha \)' bets) in order to gain this power.

This result is directly complimentary to that of [Aceto 1987]; there, single bets do not have this power, as they are made on rather less rich structures. (Aceto applies the betting methodology to event structures. Since occurrences are atomic in that world, in order to obtain the discriminating power we have with single bets, he needs bets based on the occurrences of a sequence of multisets of events. In order to obtain the power of causal bets, (which we introduce in the next section) Aceto needs bets based on the occurrences of a partial order of events. The considerable descriptive power of our bets, and the consequent simplicity of our formalism relative to Aceto's, seems to lend weight to the claim that it is reasonable to consider occurrences of events as having distinct starts and finishes.)

Notice, incidentally, that if two computational I.E.S.s differ, they do so at some finite time into some execution; for this reason we do not need to restrict ourselves to dealing with finite structures.

### 3.2.5 On the power of betting on elaborate timed behaviours

Given the result of the last section, it seems hardly surprising that strict-timed bets can distinguish between all executions. That is, given \( J \in \mathcal{P}(S, L) \), \( K \in \mathcal{P}(S', L) \) for any possible observation, \( L \), if \( J \neq K \) then we can formulate a bet which has a different result when placed on \( J \) and \( K \). (Again, really we should say \( J \neq K \) up to bijection between labelling sets.)

Notice that, intuitively, we can always state an untimed bet as a \( \delta \)-timed bet by making \( \delta \) large enough, (since no transition can be indefinitely postponed), and a strict-timed bet can be seen as (an uncountable) disjunction of \( \delta \)-timed bets. Thus an untimed bet can always be made as a \( \delta \)-timed bet or a strict-timed bet, and a \( \delta \)-timed bet can always be made as a strict-timed bet.

In order to assess the power of strict-timed bets, it will be convenient to label each element of the sequences of the previous section with the time at which it occurred. (This just gives us a structure with exactly the same descriptive power as L.I.E.S.s, but which it is easier to define sequences of strict-timed bets upon.)
Suppose \( \text{seq}(\mathcal{J}) = J_1 J_2 \ldots J_n \). Then \( \text{tseq}(\mathcal{J}) = (J_1, r_1)(J_2, r_2) \ldots (J_n, r_n) \) where \( r_i \) times every transition in \( J_i \). (There is such a time, by construction.)

**Theorem** – The power of strict-timed bets

Given two computational I.E.S.s \( \mathcal{S} \) and \( \mathcal{S}' \), and some observation that can be made of both of them, \( L \), we can distinguish between the complete histories, \( \mathcal{J} \in \mathcal{S}(\mathcal{S}, L) \) and \( \mathcal{K} \in \mathcal{S}(\mathcal{S}', L) \) using strict-timed bets just when \( \text{tseq}(\mathcal{J}) \neq \text{tseq}(\mathcal{K}) \).

**Proof.** Suppose \( \text{tseq}(\mathcal{J}) = (J_1, t_1)(J_2, t_2) \ldots (J_n, t_n) \) and \( \text{tseq}(\mathcal{K}) = (K_1, t_1)(K_2, t_2) \ldots (K_n, t_n) \) and suppose further that the first place \( \text{tseq}(\mathcal{J}) \) and \( \text{tseq}(\mathcal{K}) \) differ has index \( i \) (that is \( J_j = K_j \) for \( j < i, J_i \neq K_i \)). Then the bet

\[
(B, y)^{r} = \land ((i_1)^{r_1}r_i'((r_2 - r_1)^{r_1}) \ldots ((i_i)^{r_i}(r_1 - r_{i-1}) \ldots r_1)) \text{Win} \quad \text{if } j_i \in J_i
\]

will win when placed on \( \mathcal{S} \) doing \( \mathcal{J} \) and it will lose when placed on \( \mathcal{S}' \) doing \( \mathcal{K} \). (The interpretation of \( (j_i)^{r_i} \), if \( j_i = a_b_c_d_a \), is the strict-timed sequence \( (a_r)(b_0)(c_0)(d_0) \); this ensures a proper treatment of simultaneity.)

Further, we can distinguish between I.E.S.s up to bijection between \( \text{tseqs} \). Suppose \( \text{tseq}(\mathcal{S}) = \{ \text{tseq}(\mathcal{J}) \mid \mathcal{J} \in \mathcal{S}(\mathcal{S}) \} \). Now, if \( \text{tseq}(\mathcal{S}) \neq \text{tseq}(\mathcal{S}') \) this must be due to some \( \text{tseq}(\mathcal{J}) \in \text{tseq}(\mathcal{S}) \) not being equal to a \( \text{tseq}(\mathcal{K}) \in \text{tseq}(\mathcal{S}') \). Hence the bet

\[
b^{r} = \exists j \in \text{tseq}(\mathcal{S}) (B, y)^{r}
\]

will distinguish between all timed finite behaviours, since

\[
\mathcal{S}(\mathcal{S}, \{*,\}, b^{r}) = \{T\} \quad \text{while} \quad \mathcal{S}(\mathcal{S}', \{*,\}, b^{r}) = \{T\}
\]

for the same reasons as before. The conjunction is finite since we demanded finite nondeterminism.

Untimed bets, then, have all the power necessary to distinguish the order in which things happen, while strict-timed bets have all the power necessary to distinguish the order and the timing of happenings. The power of \( \delta \)-timed bets lies between these two extremes, and depends on the structure concerned. As we have said, untimed bets and strict-timed bets can be seen as the special cases of \( \delta \)-timed bets \( \delta = \infty \) and \( \delta = 0 \) respectively.

We will go on, after an example of compound timed and untimed bets, to consider the observation of causality and the implications its introduction into the betting framework might have.
3.2.6 Examples

Reconsider the examples 31–36 from subsection 3.1.4. The following assertions about the results of compound bets are true.

For the purely deterministic structures 31 and 32 the results are straightforward:

\[ \mathcal{R}(31, \{\ast\}, e_{s(1)}e_{f(1)}e_{s(2)}e_{f(2)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(31, \{\ast\}, e_{s(1)}e_{f(1)}e_{s(2)}e_{f(2)}\text{Lose}) = \{L\} \]
\[ \mathcal{R}(31, \{\ast\}, e_{s(1)}e_{f(1)}\text{Win} \land e_{s(1)}e_{f(2)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(32, \{\ast\}, e_{s(1)}e_{f(1)}\text{Win} \lor e_{s(1)}e_{f(2)}\text{Win}) = \{L\} \]
\[ \mathcal{R}(32, \{\ast\}, e_{s(1)}e_{f(1)}\text{Win} \lor e_{s(1)}e_{f(2)}\text{Win}) = \{T\} \]

The presence of nondeterminism, as in 34, means that quantification becomes useful:

\[ \mathcal{R}(34, \{\ast\}, e_{s(1)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(34, \{\ast\}, \exists e_{s(1)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(34, \{\ast\}, \forall e_{s(1)}\text{Win}) = \{L\} \]
\[ \mathcal{R}(34, \{\ast\}, \exists e_{s(1)}\text{Win} \land e_{s(2)}\text{Win}) = \{T\} \]

The last bet indicates that one should read \( \exists (e_{s(1)}\text{Win} \land e_{s(2)}\text{Win}) \) not as ‘there is an execution from here where \( e_{s(1)} \) wins and \( e_{s(2)} \) wins,’ but as ‘there is an execution from here where \( e_{s(1)} \) wins and there is an execution from here where \( e_{s(2)} \) wins.’ The more complicated structures 35 and 36 offer a host of interesting bets, for example:

\[ \mathcal{R}(35, \{\ast\}, e_{s(1)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(35, \{\ast\}, \exists e_{s(1)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(35, \{\ast\}, \exists e_{s(1)}\text{Win} \land e_{s(1)}e_{f(1)}e_{s(2)}e_{f(2)}\text{Win}) = \{T\} \]
\[ \mathcal{R}(35, \{\ast\}, \exists e_{s(1)}\text{Win} \lor e_{s(2)}\text{Win}) = \{T\} \]

Finally, consider some timed causal bets; we have:

\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{0}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{2}\text{Win}) = \{L\} \]
\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{6}\text{Win} \land (e_{s(2)})_{3}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{2}\text{Win} \land (e_{s(1)})_{2}\text{Win}) = \{L\} \]
\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{0}\text{Win} \lor (e_{s(1)})_{0.5}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, \exists (e_{s(1)})_{1}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, \forall (e_{s(1)})_{0}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, (e_{s(1)})_{3}\text{Win} \lor (e_{s(2)})_{3}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, \exists (e_{s(1)})_{1}\text{Win} \lor (e_{s(2)})_{3}\text{Win}) = \{T\} \]
\[ \mathcal{R}(33, \{\ast\}, \forall (e_{s(1)})_{0}\text{Win} \lor (e_{s(1)})_{0.5}\text{Win}) = \{T\} \]
3.3 Causal bets

An interesting elaboration of the betting methodology is possible if causal bets are allowed. In some instances it might be reasonable to suppose that causality is observable; we may be able to observe co, for instance if concurrent happenings happen in observably different places. In this case, the branching structure of time would be a model for a genuinely distributed system with message passing, process creation and so on. (For example, two branches splitting might correspond to a computation being rescheduled on two geographically distributed processors.) In these cases it is reasonable to allow bets that specify how things happened. These will be called causal bets.

Causal bets have a fairly respectable history, given that process algebra theorists usually assume that causality is unobservable. Observational techniques which allow the observation of causality are implicit in the positions taken in [Aceto et al. 1987], [Degano & Montanari 1987] and particularly in [Reisig 1987]. The position taken on the nature of concurrency in [Shields 1987] also seems to suggest that causality is observable.

The causal syntax is the same as before, except for sequencing; $B_c^g(E_{sec})$ (‘C’ for causal) is defined by:

$$b ::= \text{Win} \mid \text{Lose} \mid (a, \rightarrow a') b \mid b_1 \lor b_2 \mid b_1 \land b_2 \mid \forall b \lor \exists b$$

with $a_1, a_1' \in \{(l, e)_s, (l, e)_f \mid e \in E_{sec}\}$. The interpretation of $(a, \rightarrow a')$ is that $a_1$ has already happened, $a_1'$ should happen next, and $a_1$ must cause (or be causally the same as) $a_1'$. In the case $a_1 = *$, this reduces to a non-causal bet, since $\forall a = [t_{t_0}, t_{t_0}] \in LE \cdot \text{begin}(*) \leq t_{t_0}$.

Some new definitions will be needed to give a meaning to causal bets.

**Definition 3.8 – Causal Extension**

Suppose that we have a L.I.E.S. $L = (LE', <)$ and a history $K = (LE, <) \in M(3)$ with $L$ consistent with $K$; $K \in \mathcal{P}(3, L)$. Suppose further that we have some transition $a_1 = (l, e)_s$ at time $r_1$, which hasn’t been seen yet, $r_1' \geq \text{end}(L)$, and also some transition we have already seen, $a_2 = (l', e'), \in L$.

We want to state when $a_1$ causes $a_1'$, i.e. when $(a_1 \rightarrow a_1')$ extends $L$ in $K$. We write this causal extension as $(a_1 \rightarrow a_1') \text{ extends}^c L \text{ in } K$.

First notice that $a_1'$ must extend $L$ in $K$, and suppose that the labelled events in $LE$ that justify inferring $a_1'$ extend $L$ in $K$ are $a = (E(l), e) = [r, r]$ and $a' = (E(l'), e') = [r', r']$ say.

The definition now splits into two cases:
(i) $a'_i$ is a start, $a'_i$ say. Then we require that $(r_i, a_i) \leq_b (r'_i, a'_i)$.

(ii) $a'_i$ is a finish, $a'_i$ say. Then we require that $(r_i, a_i) \leq_b (r'_i, a'_i)$. In this case, of course, $a'_i$ will be in LE already.

Using this definition of extension we can now skip through the definitions that lead to the meaning of causal bets; must and may just use the new notion of extension;

\[ \exists \text{ must } (a, \rightarrow a'_i) \text{ after } \Leftrightarrow \forall \mathcal{K} \in \mathcal{P}(3, \L). (a, \rightarrow a'_i) \text{ extends } \L \text{ in } \mathcal{K} \]

\[ \exists \text{ may } (a, \rightarrow a'_i) \text{ after } \Leftrightarrow \exists \mathcal{K} \in \mathcal{P}(3, \L). (a, \rightarrow a'_i) \text{ extends } \L \text{ in } \mathcal{K} \]

The meaning functions are then defined in the obvious way;

\[ S(3, \L, (a, \rightarrow a'_i) b) = \mathcal{R}(3, \L, (a, \rightarrow a'_i)) \cup S(3, \L \cup \{a'_i\}, b) \]

\[ \mathcal{R}(3, \L, (a, \rightarrow a'_i)) = \{T\} \quad \text{if } \exists \text{ must } (a, \rightarrow a'_i) \text{ after } \L, \]

\[ \mathcal{R}(3, \L, (a, \rightarrow a'_i)) = \{T, \perp\} \quad \text{if } -(\exists \text{ must } (a, \rightarrow a'_i) \text{ after } \L) \text{ and } \exists \text{ may } (a, \rightarrow a'_i) \text{ after } \L \]

\[ \mathcal{R}(3, \L, (a, \rightarrow a'_i)) = \{\perp\} \quad \text{if } -(\exists \text{ may } (a, \rightarrow a'_i) \text{ after } \L). \]

Causal bets clearly give us more power than non-causal ones (it is easy to think of two structures distinguished by causal bets and not distinguished by non-causal ones, – for instance $S_2$ of subsection 3.1.4 and $S'_2 = \{(e_1 = [(0, e_{t(1)}), (2, e_{r(1)})], e_2 = [(1, e_{s(2)}), (4, e_{r(2)})]), \{(e_1, e_2), \{(0, e_{t(1)}), (1, e_{s(2)})\}, \{\}\) which has less causality).

In order examine the power of causal bets, we will have to examine a structure like $S_2$ and $S'_2$ but which incorporates causal information. Previously, we just had linear structures to represent executions; now we must have branching ones. Thus our logic of observations will become branching rather than linear.

Suppose we have an I.E.S. $S$ displaying the maximal Con-set $c \in M(S)$. We will see those transitions of $S$ that belong to labelled events in $c$, with the causality $S$ specifies, so a reasonable representative of a timed causal execution is the conflict-free I.E.S. $S \rhd c$. (This is conflict-free since $M(S \rhd c) = \{c\}$ provided $c \in M(S)$.

**Theorem** – The power of strict-timed causal bets

Given two I.E.S.s $S$ and $S'$ displaying maximal Con-sets $c$ and $c'$, then we can distinguish between these histories using non-causal bets iff $tseq(S \rhd c) \neq tseq(S' \rhd c')$.

However, if $tseq(S \rhd c) = tseq(S' \rhd c')$ then, provided $S \rhd c$ and $S' \rhd c'$ are not equal up to bijection of labels, we can formulate a bet which has a different result when placed on $S \rhd c$ to that which it has when placed on $S' \rhd c'$. 

Page 128 Chapter Three — Meaning of Structures
Proof. Clearly if \( \text{tseq}(S \uparrow c) = \text{tseq}(S' \uparrow c') \) then \( S \) and \( S' \) are not distinguishable by non-causal bets. (For if they were there would be some transition at which they were different, either in timing or causality, violating the assumption that \( \text{tseq}(S \uparrow c) \neq \text{tseq}(S' \uparrow c') \).)

Suppose that \( S \uparrow c \) and \( S' \uparrow c' \) differ in their causality at \( \alpha_{(d)} \), so that \( \alpha_{(d)} \) causes different things in \( S \) and \( S' \).

Suppose further that the difference in causality refers to \( \alpha_{(d)} \), so that \( \alpha_{(d)}' \) causes different things in \( S \) and \( S' \).

Now, define the \textit{necessarily simultaneous set of transitions} of a given transition as

\[
[a]_c = \{ a_i | a_i' = c_i \}
\]
and say that \([a_{(d)}], [a_{(c)}], \ldots, [a_{(d)}]_c\) is a \textit{maximal causal securing} of \( a_i \), just when

\[
\forall a_{(1)} \in [a_{(1)}]_c, a_{(2)} \in [a_{(2)}]_c, \ldots, a_{(d)} \in [a_{(d)}]_c \cdot a_{(1)} \lesssim a_{(2)} \lesssim \ldots \lesssim a_{(d)} \lesssim a_t
\]
and

\[
\exists a_{(0)} \in \text{Tr} \cdot a_{(0)} < a_t \& a_{(0)} \in [a_{(0)}]_c \cup [a_{(c)}]_c \cup \ldots \cup [a_{(d)}]_c
\]

Define, analogously with \( A \), the set of sequences \( A \) for some set of transitions \( A \), so that

\[
A = \{ a_t \rightarrow b_t \rightarrow c_t \rightarrow a_t, \ldots \}
\]

Now, returning to \( \alpha_{(d)}' \), suppose that its maximal causal securing is \([a_{(d)}]_c[a_{(0)}]_c \ldots [a_{(0)}]_c\). Then the bet

\[
(B_c C)_y = \land \{ (a^1)_y \rightarrow (a^2)_{(r_2-r_1)}, \ldots \rightarrow (a^n)_{(r_n-r_{n-1})}, \ldots \rightarrow (a_{(d)})_{(r_{(d)}-r_{(d)-1})}, \ldots \rightarrow (a_{(d)})_{(r_{(d)})} \}
\]

will win when placed on \( S \) doing \( c \) and it will lose when placed on \( S' \) doing \( c' \).

(again, the interpretation of \( (a^i)_y \), if \( a^i_j = a_t \rightarrow b_t \rightarrow c_t \rightarrow d_t \), is the strict-timed sequence \( (a_t)_y \rightarrow (a^i)_y \rightarrow (b^i)_y \rightarrow (c^i)_y \rightarrow (d^i)_y \); this ensures a proper treatment of simultaneity.)

We assume that the common time of all of the transitions in \([a_{(0)}]_c\) is \( r_j \), that \( r_{(d)} \) times \( a_{(d)} \) and that \( r_{(0)} \) times \( a_{(0)}' \).

As the reader will probably have guessed, if \( \neg (\forall c \in M(S) . \exists c' \in M(S) . S \uparrow c = S' \uparrow c') \) if \( \neg (\forall c' \in M(S) . \exists c \in M(S) . S \uparrow c = S' \uparrow c') \) (up to bijection of labels) then the bet

\[
b^C_y = \exists c \in M(S) (B_c C)_y
\]
will distinguish \( S \) and \( S' \). (since we are only dealing with equality up to bijection of labels, we cannot distinguish between \( S \) and \( S + S \) by any form of bet.)
3.4 Bets and equivalences

The bets of the previous few sections will now be used to compare interval event structures. Consider two I.E.S.s, \( S = (LE, \text{Con}, <) \) and \( S' = (LE', \text{Con}', <') \) and suppose that our universe of bets \( E_{\text{bet}} \) will be \( E \cup E' \) (i.e. we will suppose that we can bet on the occurrences of events that \( S \) can display and on those that \( S' \) can display). The results of bets made on \( S \) and \( S' \) can be compared using \( S \).

Suppose that both I.E.S.s have been observed to engage in some (possibly empty apart from *) L.I.E.S., \( L = (LE'' \subseteq L'' \times E_{\text{bet}} \cup \{\ast\}, <) \). Then the behaviour of \( S \) at \( \text{end}(L) \) can be compared by examining \( S(S, L, a, f) \) and \( S(S', L, a, f) \) for \( a, f \in LE'' \);

**DEFINITION 3.9 — Betting equivalence**

The I.E.S. \( S \) is more reliable, or a better earner than the I.E.S. \( S' \) with respect to the bet \( a, f \), given the observation \( L \), if \( S(S, L, a, f) \geq_{EM} S(S', L, a, f) \). Clearly the outcome of all possible bets is important, so we say that \( S \) is more reliable than \( S' \) given \( L \), written \( S \geq_{E} S' \), iff

\[
\forall b \in B(E_{\text{bet}}). S(S, L, b) \geq_{EM} S(S', L, b)
\]

Our observer will sometimes see no need to leave the betting shop to make an observation. Then she just has * at \( t = 0 \), and

\[
S \geq_{*} S' \iff \forall b \in B(E_{\text{bet}}). S(S, \{\ast\}, b) \geq_{EM} S(S', \{\ast\}, b)
\]

The same principle can be applied to timed bets;

\[
S \geq_{T} S' \iff \forall b \in B_{T}(E_{\text{bet}}). S(S, L, b) \geq_{EM} S(S', L, b)
\]

We write \( \geq_{Z} \) (German Zeit, time) for one of \( \geq, \geq_{S} \) or \( \geq_{T} \).

If \( S \geq_{Z} S' \) and \( S' \geq_{Z} S \) then we write \( S \sim_{Z} S' \) and say that \( S \) and \( S' \) are observationally equivalent up to \( \geq_{Z} \):

\[
S \sim_{Z} S' \iff S \geq_{Z} S' \& S' \geq_{Z} S
\]

\[
S \sim_{T} S' \iff S \geq_{T} S' \& S' \geq_{T} S
\]

\[
S \sim_{T} S' \iff S \geq_{T} S' \& S' \geq_{T} S
\]
3.5 An introduction to metric space semantics

Metric space semantics can be quite complex and subtle, so before defining metrics that reflect the testing equivalences we have defined, we shall give a short introduction. This introduction will be based on the process tree formalism, an intuitively appealing model of processes. Our aim will be to investigate various notions of equivalence for the process tree formalism, and their inter-relationship, showing how these notions can and cannot be used to define suitable metric spaces.

This section draws heavily from [Klop 1988], from whence further details should be sought. (Incidentally, process graphs capture all the information available using untimed causal bets, so they bear the same relationship to causal bets as sequences do to untimed noncausal bets.)

3.5.1 Process trees

A process graph is a rooted, directed, connected, edge labelled graph. A process tree is a process graph without cycles and where at most one arrow goes into every node. (This is called a synchronisation tree by Milner; see [Milner 1980] or, for a modern approach, [Winskel 1984].) We shall just deal with process trees. (A process tree can always be tree-unwound to produce a process graph; this just involves removing cycles, – see [Kok & Rutten 1988] for details.)

![Figure 3.5 - A tree unwinding](image)

**NOTATION**

Process graphs or trees are denoted by $g, h, ...$

The nodes of process trees are denoted by $s, t, ...$ The set of all nodes of a tree $g$ is denoted by $\mathbb{N}(g)$. The roots of trees are $s, t, ...$

The edge-labels of process trees are denoted $a, b, c, ...$ The set of all edge-labels of a tree $g$, the alphabet of $g$, is written $\alpha(g)$.

For a node $s \in \mathbb{N}(g)$ the branching degree of $s$, $\text{brd}(s)$, is the number of edges leaving $s$. Let $G_{\alpha, \beta}$ be the class of all process graphs over a fixed alphabet of cardinality $\alpha$ and with branching degree less than $\beta$ for every node.
DEFINITION 3.10 – Bisimulation and projective equivalence

The notion of bisimulation is used to compare process graphs; if two process graphs bisimulate each other, then they can in a certain sense do the same thing.

Bisimulation is closely related to our definition of homomorphism in chapter two. Let $g, h$ be process graphs with roots $s$ and $t$ respectively. A relation $R \subseteq N(g) \times N(h)$ is a bisimulation from $g$ to $h$, written $g \leftrightarrow_R h$, iff

(i) Roots are related; $s R t$.

(ii) Similar steps are simulated: if there is an edge of $g$ from $s$ to $s'$ labelled $a$, written $s \rightarrow_a s$, and $s R t$, then there is an edge of $h$, $t \rightarrow_{a'} t'$ such that $s' R t'$.

(iii) Vice versa (i.e. as (ii) but with the roles of $g$ and $h$ interchanged).

If there is some $R$ such that $g \leftrightarrow_R h$ then we write $g \leftrightarrow h$. The example (below) shows some process trees which bisimulate each other and some that don't.

EXAMPLE

No pair of $g$s or pair of $h$s bisimulate each other. Each $g$ bisimulates just the matching $h$, so that $g^1 \leftrightarrow h^1$, $g^2 \leftrightarrow h^2$, $g^3 \leftrightarrow h^3$, but $\neg (g^1 \leftrightarrow g^2)$, $\neg (g^1 \leftrightarrow g^3)$, etc. This shows that bisimulation can distinguish between choice and interleaving.
Another equivalence on process trees which is less discriminating than bisimulation equivalence will be introduced. Projective equivalence identifies two processes at level $n$ if they 'do the same thing' for at least $n$ steps.

**DEFINITION 3.11 – Projective equivalence**

Consider a path $s, s', s'', s'''$, from the root of a process tree $g$ to a node $s^*$. If $g$ is a process tree then this path is unique. If there is a transition from the root of $g$, $s \rightarrow s'$ then if $g'$ is the subtree rooted at $s'$ we write $g \rightarrow_s g'$.

The number of steps of $s^*$ from the root, or the *distance* of $s^*$, written $l(s)$, is the number of nodes in this path, and the distance from the root of a given edge is the distance of its terminal node. For all process trees $g$ and $h$, define

(i) $g \equiv_0 h$ always.

(ii) $g \equiv_{n+1} h$ if $g \equiv_n h$ and $g \rightarrow_s g'$, then there is an edge of $h$ so that $h \rightarrow_s h'$.

(iii) Vice versa.

If $g \equiv_n h$ for all $n$ then we write $g \equiv_\infty h$, and say that $g$ and $h$ are projectively equivalent.

A projection operator, $\pi_n(g)$, which cuts away everything below level $n$, can be defined. Its effect is straightforward:

![Diagram](https://via.placeholder.com/150)

**Figure 3.6 – The effect of the tree projection operator**
THEOREM – Bisimulation is equivalent to projective equivalence in $G_{\epsilon_0, \eta_0}$

Consider finitely branching $g$ and $h$ with roots $s, t$ respectively. We have to show that

$$g \leftrightarrow h \iff g = h$$

Proof: For $\Rightarrow$ notice $g = h \iff \forall n. \pi_n(g) \equiv \pi_n(h)$ and the result is immediate. For $\Leftarrow$, consider the sets of bisimulations $B_0 = \{(s, t)\}, B_{n+1} = \{ R \mid R$ is a bisimulation from $\pi_{n+1}(g)$ to $\pi_{n+1}(h)\}$. Form the union $B = \cup_n B_n$. This collection of ‘partial’ bisimulations between $g$ and $h$ forms a tree ordered by set-theoretic union. Moreover, since $g$ and $h$ are finitely branching, there are only finitely many extensions of a bisimulation from $\pi_n(g)$ to $\pi_n(h)$ to one from $\pi_{n+1}(g)$ to $\pi_{n+1}(h)$ so the tree is finitely branching. Furthermore, since $\forall n. \pi_n(g) \equiv \pi_n(h)$ the tree must have infinitely many nodes. Hence, by König’s lemma, $B$ has an infinite branch. Consider the infinite sequence of bisimulations that live along this branch; $R_0 \subseteq R_1 \subseteq R_2 \ldots$. A given $R_n$ is a bisimulation from $\pi_n(g)$ to $\pi_n(h)$. Thus $\cup_n R_n$ is a bisimulation from $g$ to $h$.

Clearly, projective equivalence can ‘see’ behaviour out to $\omega$ but no further. Bisimulation, in contrast, has perfect ‘sight,’ as it is defined locally rather than by induction.

DEFINITION 3.12 – Projective limits

A projective limit of a process $g$ is a limit of the projections, $\lim_{n \to \infty} \pi_n(g)$. The metric this limit is taken with respect to (and hence the existence of these limits) will be discussed next.

All processes are their own projective limits, but other processes may be as well. We shall see this in the next example. First however we shall impose some metric structure using the projection functions, so that we can define the limit above formally.

3.5.2 Metric spaces of process trees

The standard approach to metric space semantics starts from the assumption that processes (or their traces, failure sets etc.) can be thought of as points in a topological space. This space is a metric space – a distance function or metric allows us to determine how far processes are from each other; this difference should depend on how semantically different the processes are. Various metrics and various spaces (spaces whose points are traces, sets of failure sets etc.) correspond to various notions of semantic difference.

We shall start with a space whose points are process trees and whose metric is based on projective equivalence to illustrate the idea. This space will be seen to be inadequate; several solutions are suggested before we return to the main problem; defining metric spaces of observations which reflect the testing equivalence of the last few sections.
**DEFINITION 3.13** – The projection-induced pseudometric on process trees

There is a natural way of assigning a distance between processes based on the projection operator. The *projection-induced pseudometric* defines processes to be further away the sooner they fail to be projectively equivalent;

\[
\overline{d_p}(g, h) = \begin{cases} 
2^{-m} & \text{if } \neg (g \equiv_n h) \text{ and } \forall m < n. g \equiv_m h \\
0 & \text{otherwise}
\end{cases}
\]

This metric is unsatisfactory because it does not distinguish between some processes, that is, not all the \((G_{\alpha, \beta}, d_p)\) are metric spaces; they are merely pseudometric spaces. To show that this is so we must exhibit two processes, \(g\) and \(h\) say with \(d_p(g, h) = 0\) and \(g \neq h\).

**EXAMPLE** – Two different processes in \(G_{\alpha, \kappa_0}\) that have non-zero distance.

Consider \(g = \Sigma a^n\) and \(h = (\Sigma a^n) + a^\omega\). (This is just notation for the process tree with one branch of length one with label \(a\), one with length two and both labels \(a\) and so on, \((g)\) and the same plus an infinitely long branch \((h)\).)

A picture may help the reader to visualise the processes \(g\) and \(h\):

![Diagram of processes](image)

**Figure 3.7** – Two processes which prove that \((G_{\alpha, \beta}, d_p)\) is not a metric space; \(g \neq h\) but \(d_p(g, h) = 0\).

The process \(h\) is the projective limit of \(g\).

It is obvious that \(g \neq h\). To show that \(d_p(g, h) = 0\) consider the following lemma.
**LEMMA** – Relating bisimulation and projective equivalence.

Consider two process trees \( g, h \). If \( g \) is finitely branching and \( g \) and \( h \) are countably long, then bisimulation of all projections guarantees bisimulation of the original structures;

\[
\forall n. \pi_n(g) \leftrightarrow \pi_n(h) \Rightarrow g \leftrightarrow h.
\]

**Proof** – [Klop 1989].

It is clear that \( \pi_n(g) \leftrightarrow \pi_n(h) \) for our two processes, so, by the lemma, \( g \leftrightarrow h \). Furthermore since \( g = h \Leftrightarrow \forall n. \pi_n(g) \leftrightarrow \pi_n(h) \) we have \( g = h \) and the veracity of the counterexample follows.

Metric spaces can be produced from the projection-induced pseudometric, but not for \( G_{\aleph_0} \); the problem in our example is the infinite branching of \( h \).

There are several alternatives; we can restrict ourselves to a smaller space such as \( G_{\alpha} \) with \( \alpha < \aleph_0 \), but this has the disadvantage that projective limits sometimes do not exist, and hence we cannot have a complete metric space.

Alternatively, the congruence induced by projective equivalence can be divided out. This leaves us with a metric space, but not necessarily a complete one; a further metric completion is needed.

These considerations leave us well-armed to attack the main problem; defining and investigating metric spaces of observations of interval event structures.
3.6 Metric spaces and bets

This section will be devoted to the construction of two metric spaces based on untimed bets and strict-timed bets. The obvious construction (analogous to definition 3.13) is to construct a space whose points are interval event structures and whose metric assigns a smaller distance to processes the longer all bets made about them have the same result. A few technical details will cloud, but not totally obscure, this simple idea. In order to restrict ourselves to set rather than proper classes, we assumed a fixed set $E_{\text{bet}}$ as before.

The idea of using metric spaces to give a semantics to concurrent systems has been popularised by a group under de Bakker (see, for instance, [de Bakker & Mayer 1988]). Similar constructions can be found in [Goltz & Loogen 1987], [Kok & Rutten 1988], and [Reed 1988]. The work of Reed and Roscoe [op. cit.] is, perhaps, of most relevance to this section, as there too the presence of real-time is important. However, our approach is strictly observational rather than denotational, and is rather simpler. (The results of the first few sections of the chapter indicate that we need no more complication than strict-timed bets give us, in order to distinguish between all observationally-different behaviours.) This work, then, may be seen as a counterpart to Reed's; we have different ideas of what 'observation,' 'concurrency,' and 'behaviour' mean, and hence we are exploring a different point in the concurrency semantics design space.

It will be important to have a metric that reflects the intuition that two I.E.S. are closer the longer they simulate each other. Simulation will be captured by the betting equivalences, - what does 'the longer' mean?

For untimed bets, we must use the sequence order of seq just as we used depth for process graphs; hence we need a projection function on seq(3). In the strict-timed case, a construction for I.E.S.s like truncation of L.I.E.S.s is needed, as the temporal analogue of projection. Suitable operators are then: -

**Definition 3.14** - Projection of I.E.S.s

Given an I.E.S. $3$, and the set of sequences generated by $3$, seq($3$), the projection of $3$ to the $m^{th}$ transition, written $\pi_m(3)$, is defined to be the set of sequences

$$\{\pi_m(P_1P_2\ldots P_n) \mid P_1P_2\ldots P_n \in \text{seq}(3)\}$$

where the projection operator on sequences is as given in definition 3.7.

**Definition 3.15** - Truncation of I.E.S.s

If an I.E.S. $3$ is only allowed to run for a limited period, say from $t = 0$ to $t = T$, only part of some history will be seen. The possible histories seen will be precisely $L^{\leq T}$, for $L \in M(3)$. 

Chapter Three — Meaning of Structures
Define $3$ \textit{truncated} at $T$, written $3 \ast T$, as the I.E.S. which can display just the behaviours $L \ast T$. This I.E.S. can be characterised in the usual ‘... is the I.E.S. (LE', Con', $\triangleleft$') where’ way:

Suppose $3 = (LE, Con, \triangleleft)$; then $3 \ast T$ is the I.E.S. $(LE', Con', \triangleleft')$ where

(i) $LE' = \text{def} \{a \mid a \in LE, \text{begin}(a) < T\}$. For $a \in LE'$, $a = [r, r']$ iff $a = [r, r] \in LE$ where $r' = r$ if $r_t < T$ and $r' = T$ otherwise.

(ii) $Con' = \text{def} \{c' \mid LE' \land c \in Con\}$

(iii) $\triangleleft' = \text{def} \{((r_t', a_{(1)}), (r_{(2)}, a_{(2)})) \mid ((r_{(1)}, a_{(1)}), (r_{(2)}, a_{(2)})) \in \triangleleft, r_{t'} = r_{(1)} \text{ if } r_{t(1)} < T, r_{(1)} = T \text{ otherwise}\}$

The obvious definition of a metric generalising definition 3.13 is then in the untimed case;

$$d_{IES}(3, 3') = \begin{cases} 2^{-m} & \text{if } \neg (\pi_n(3) = \pi_n(3')) \text{ and } \forall n < m. \pi_n(3) = \pi_n(3') \\ 0 & \text{otherwise} \end{cases}$$

and in the strict-timed case;

$$d_{IES}(3, 3') = \begin{cases} 2^{-T} & \text{if } \neg (3 \ast T \sim 3' \ast T) \text{ and } \forall t < T. 3 \ast t \sim 3' \ast t \\ 0 & \text{otherwise} \end{cases}$$

These metrics have the same problem as before; there are processes $3$ and $3'$ with $3 \neq 3'$ but the distances between them, $d_{IES}(3, 3')$, and $d_{IES}(3, 3')$, are zero. For this reason we divide out the equivalence caused by $\sim$. Suppose that $cIES$ is the class of all computational interval event structures under consideration. Then

\textbf{DEFINITION 3.16} – Observational equivalence classes and metric spaces

The set of \textit{equivalence classes} of a computational structure $3 \in cIES$ under $\sim$, written $[3]_Z$, is just the set of all things that $\sim$ it; $[3]_Z = \{3' \mid 3' \sim 3 \}$. The \textit{factored class} $[cIES]_Z$ is then $\{[3]_Z \mid 3 \in cIES\}$. (From now on, $[3]_Z$ stands for $[3]$ or $[3]_T$, $d_Z$ for $d$ or $d_T$, and so on.)

The metric associated with $[cIES]$ is;

$$d([3], [3']) = \begin{cases} 2^{-m} & \text{if } \neg (\pi_n(3) = \pi_n(3')) \text{ and } \forall n < m. \pi_n(3) = \pi_n(3') \\ 0 & \text{otherwise} \end{cases}$$
while the metric associated with $[cIES]_\gamma$ is

$$d_\gamma([3], [3']) = \begin{cases} 2^{-T} & \text{if } \neg(3^{*T} \sim_\gamma 3'^*T) \& \forall t < T. 3^{*t} \sim_\gamma 3'^*t \\ 0 & \text{otherwise} \end{cases}$$

\* 

**Lemma — Well-defined metrics**

We have to show that the metrics $d$ and $d_\gamma$ given above are well-defined. For the untimed case this reduces to showing that untimed betting equivalence is a congruence of projection, i.e. that for all $m \in N$.

$$\pi_m(3) \sim \pi_m(3') \Leftrightarrow \forall 3' \in [3]. \pi_m(3') \sim \pi_m(3')$$

While for the timed case, this reduces to showing that strict-timed betting equivalence is a congruence of restriction, i.e. that for all $T \in R^*$.

$$3^{*T} \sim_\gamma 3'^*T \Leftrightarrow \forall 3' \in [3]_\gamma. 3'^*T \sim_\gamma 3'^*T$$

**Proof.** (Untimed case.) Note that $3 \sim 3' \Rightarrow \forall m \in N. \pi_m(3) \sim_\gamma \pi_m(3')$ so we just have to show that

$$3 \sim 3' \Leftrightarrow \forall 3' \in [3]. 3' \sim 3'$$

which follows by definition.

(Timed case.) Note that $3 \sim_\gamma 3' \Rightarrow \forall T \in R^+. 3^{*T} \sim_\gamma 3'^*T$ so we just have to show that

$$3 \sim_\gamma 3' \Leftrightarrow \forall 3' \in [3]_\gamma. 3' \sim_\gamma 3'$$

which follows by definition.

\* 

**Theorem — Metric spaces of observational difference**

The spaces $([cIES], d)$ and $([cIES]_\gamma, d_\gamma)$ are both metric spaces.

**Proof.** Properties (i), (iii) and (iv) of definition A3.1 (the definition of metric spaces) hold since $\sim_z$ is an equivalence relation and since $[IES]_z$ is the factored space generated by that equivalence. That leaves the triangle inequality;

$$\forall x, y, z \in [cIES]_z. d_z(x, y) + d_z(y, z) \geq d_z(x, z)$$

For the untimed case, define $m_{XY} = m \text{ s.t. } (\pi_m(X) = \pi_m(Y)) \& \forall n < m. \pi_n(X) = \pi_n(Y)$. Then note that either $m_{XY} = m_{YZ}$, in which case $m_{XZ} = 2m_{XY}$, and the inequality holds, or we have $m_{XY} > m_{YZ}$ or $m_{YZ} > m_{XY}$. From symmetry we need only take one, so suppose that $m_{XY} > m_{YZ}$. 

\*
Now, obviously, \( \pi_n(X) = \pi_n(Y) \) implies \( \pi_n(X) = \pi_n(Y) \) for \( o < n \), so \( \pi_{m_{XY}}(X) = \pi_{m_{XY}}(Y) \) implies that \( \pi_{m_{YZ}}(X) = \pi_{m_{YZ}}(Y) \). But \( \pi_{m_{YZ}}(Y) = \pi_{m_{YZ}}(Z) \) so we have \( \pi_{m_{YZ}}(X) = \pi_{m_{YZ}}(Z) \) and hence \( m_{XZ} \geq m_{YZ} \) and thus \( d(x, z) \leq d(x, y) + d(y, z) \) as required.

For the timed case, the proof is similar. Define \( t_{XY} \) to be the \( T \) such that \( -T \leq Y \leq X \), and \( \forall t < T \), \( X \leq t Y \). Then note that either \( t_{XY} = t_{YZ} \), in which case \( t_{XZ} = 2 t_{XY} \), and the inequality holds, or we have \( t_{XZ} > t_{YZ} \) or \( t_{YZ} > t_{XZ} \). From symmetry we need only take one, so suppose \( t_{XZ} > t_{YZ} \).

Now, obviously, \( S^T \sim Y \). \( S^T \) implies \( S^U \sim Y \). \( S^U \) for \( U < T \), so \( X^f_{XY} \sim_{\gamma T} Y^f_{XY} \) (or rather, \( X^f \sim_{\gamma T} Y^f \) for any \( t < t_{XY} \)) implies that \( X^f_{YZ} \sim_{\gamma T} Y^f_{YZ} \). But \( Y^f_{YZ} \sim_{\gamma T} Z^f_{YZ} \) and hence \( t_{XZ} > t_{YZ} \) and thus \( d(x, z) \leq d(x, y) + d(y, z) \) as required.

We have been rather cavalier about the treatment of infinite structures in this chapter. There is a good reason for this; we can only have infinitely ‘long’ computational I.E.S.s (ones that go on for all time); infinitely ‘fat’ ones, – ones with infinite nondeterminism or infinite density, – are disallowed. Furthermore, infinitely long structures are made up of a countable number of labelled events, all with defined transitions, and all finitely caused, so, as we remarked earlier, if two infinite computational I.E.S.s differ, they do so at some finite time. (The class of infinite computational structures, then, in some sense isolates those infinite I.E.S.s that are recursively generable.) Hence, any Cauchy sequence of computational I.E.S.s will tend to a computational I.E.S.; in particular, \( ([\text{I.E.S.}]_{\gamma T}, d_{\gamma T}) \) is complete:

**Theorem** – The metric space \( ([\text{I.E.S.}]_{\gamma T}, d_{\gamma T}) \) is complete

Consider a Cauchy sequence of equivalence classes of I.E.S.s, \([S_1]_{\gamma T}, [S_2]_{\gamma T}, \ldots \) We will construct a \( S \) so that \([S]_{\gamma T}\) is the limit of the sequence, and so that \( S \) is computational.

**Proof** (following [Goltz & Loogen 1987]). Since \([S_1]_{\gamma T}, [S_2]_{\gamma T}, \ldots \) is a Cauchy sequence, there is a monotone increasing sequence of natural numbers, \( k_1, k_2, \ldots \) such that

\[
\forall m, n \in \mathbb{N} : m \geq k_n \Rightarrow d_{\gamma T}([S_{k_n}]_{\gamma T}, [S_{k_m}]_{\gamma T}) < 2^{-n}
\]

In other words, for any \( T \in \mathbb{R} \), there is always a point in the sequence after which we can be sure that the structure from \( t = 0 \) up to \( t = T \) is fixed: it is just the structure \( S_{k_n} \) displays, where \( k_n \) is the first integer greater than \( T \).

Now, the construction \([\_]_{\gamma T}\) is insensitive to bijection of labels, so we can suppose that the \( S_{k_n} \) s are \((L_{E_n}, \text{Con}_n, \preceq_n)\) where \( L_{E_n} \subset L_{E_m} \) for \( n \leq m \). In particular, since with \( m = k_o \) we have \( k_0 \geq k_n \Rightarrow d_{\gamma T}([S_{k_o}]_{\gamma T}, [S_{k_n}]_{\gamma T}) < 2^{-n} \) so we can assume that for \( o < n \). \([S_{k_m}]_{\gamma T} \sim_{\gamma T} [S_{k_o}]_{\gamma T}\). Thus, assuming \( S_{k_n} \sim^{a} = S_{k_o} \sim^{a} \) involves no contradiction. (\( \checkmark \))
Define $S$ as the structure such that $S^{*^n} = S_{k_n}^{*^n}$. Now, $S$ is a well-defined I.E.S. since by virtue of ($\diamondsuit$) it has only a countable number of transitions and hence only countable Con-sets and orderings.

Further, since each $S_{k_n}$ are computational by definition, $S$ is computational in any interval and hence computational in general.

We need to show that $[S]_{\gamma^*}$ is a limit point. Note that

$$d_{\gamma^*}([S]_{\gamma^*}, [S_{k_n}]_{\gamma^*}) < 2^{-n}$$

Then, for any $\varepsilon < 2^{-n-1}$, there is an $m$ such that $d_{\gamma^*}([S]_{\gamma^*}, [S_m]_{\gamma^*}) < \varepsilon$, namely $m = k_n$. Hence $[S]_{\gamma^*}$ is a limit point. But we have already shown that it is computational. Hence it is in cIES and the space is complete.

This completeness means that betting equivalence is defined over all the structures that will be of interest to us.

One final observation before leaving the chapter: notice that we have the capacity with our metric spaces to do timewise refinement; we can develop an I.E.S. to the point where it behaves as we desire using untimed bets, then we can examine timing considerations using timed bets. Or, we can mix timewise refinement with conventional functional refinement, dealing with timing considerations as and when they occur.

The dichotomy between functional behaviour, – the what of the system, – and timing behaviour, – the when, – is a common one in this thesis: we will often see two kinds of structure, one derived from a concentration on functionality, the other derived from a consideration of timing.

A good example of this phenomena is the two kinds of substructure of an I.E.S. we have discussed, one generated from restriction to a time, – the $S^{*^T}$ of this chapter, – and the other generated by restriction to a set of events, – the $S^{\uparrow \Lambda}$ of chapter two.
References


Appendix; Some topology

DEFINITION A3.1 - Metric spaces, pseudo-metric spaces, and ultrametrics. A metric space $M = (M, d)$ consists of a non-empty set $M$ and a mapping $d : M \times M \to \mathbb{R}^+$ (the nonegative reals) with

(i) $\forall x, y \in M. d(x, y) = d(y, x)$ and  
(ii) $\forall x, y, z \in M. d(x, y) + d(y, z) \geq d(x, z)$ and  
(iii) $\forall x, y \in M. d(x, y) = 0$ if $x = y$ and  
(iv) $\forall x, y \in M. d(x, y) = 0 \Rightarrow x = y$.

A pseudo-metric space satisfies (i), (ii) and (iii), while an ultrametric space satisfies the stronger version of (iii) : $\forall x, y, z \in M. \max\{d(x, y), d(y, z)\} \geq d(x, z)$.

DEFINITION A3.2 - Topological spaces. A topological space $T = (T, T)$ consists of a set $T$ of points together with $T$, a set of subsets of $T$, the open sets of $T$ or the topology on $T$, satisfying

(i) $T, \emptyset \in T$ and  
(ii) $u, v \in T \Rightarrow u \cap v \in T$ and 
(iii) $\forall \subseteq T \Rightarrow \subseteq u \subseteq T$.

DEFINITION A3.3 - Cauchy sequences, convergence. A sequence $x_i$ in a metric space $M = (M, d)$ is a Cauchy sequence when given any $\epsilon > 0$ there exists an $N$ such that $d(x_n, x_m) < \epsilon$ for all $m, n > N$. A sequence $x_i$ converges to $x$ when given any $\epsilon > 0$ there exists an $N$ such that for all $n \geq N. d(x_n, x) < \epsilon$.

THEOREM - Any convergent sequence in a metric space is a Cauchy sequence.

DEFINITION A3.4 - Completeness. A complete metric space $M = (M, d)$ is one in which every Cauchy sequence converges to a point in $M$.

DEFINITION A3.5 - Covers, subcovers and compactness.

A cover for a set $A$ is a class $U$ of sets such that $A \subseteq \bigcup_{U \subseteq u} u$. A subcover of a given cover some $V \subseteq U$ that is still a cover.

A space $T$ is compact if every cover consisting of open sets of $T$ has a finite subcover.

THEOREM - In a metric space $M = (M, d)$, compactness is equivalent to requiring that for any $X \subseteq M$, which has a convergent subsequence, that subsequence must converge to an element of $M$.

THEOREM - Any compact metric space is complete.
DEFINITION A3.6 – Continuity, homeomorphism and Isometry.

A function \( f: T \to U \) from one topological space \( T = (T, \tau) \) to another \( U = (U, \sigma) \) is continuous if \( u \in U \implies f^{-1}(u) \in T \). If both \( f \) and \( f^{-1} \) are continuous and both are one-to-one, then \( f \) is said to be a homeomorphism. Similarly a one-to-one function \( f: M \to N \) from the base set of one metric space \( M = (M, d) \) to that of another \( N = (N, e) \) is an isometry if it preserves distance; \( e(f(x), f(y)) = d(x, y) \) for all \( x, y \in M \).

DEFINITION A3.7 – Closure, limit points and density.

Given a topological space \( T \) and a subset \( A \) of \( T \), a point \( x \) is a limit point of \( A \) if every open set containing \( x \) also contains some point of \( A \) other than \( x \).

The closure of \( A \) in \( T \) is the union of \( A \) and all the limit points of \( A \) in \( T \), written \( cl(A) \). A subset \( A \) is said to be dense in \( T \) if \( cl(A) = T \), and closed if \( A = cl(A) \).

THEOREM – The closure of \( A \), \( cl(A) \), is the smallest closed set containing \( A \).

DEFINITION A3.8 – Compactification and completion.

A compactification of a space \( T \) is a pair \((f, U)\) where \( U \) is a compact space and \( f \) is a homeomorphism of \( T \) onto a dense subspace of \( U \). A completion of a metric space \( M = (M, d) \) is a pair \((f, N)\) such that \( N \) is a complete metric space and \( f \) is an isometry of \( M \) into \( N \) such that \( f(M) \) is dense in \( N \).

THEOREM – Every metric space has a completion; further these completions are unique up to isometry.

DEFINITION A3.9 – Induced Hausdorff metric for subsets.

Let \( M = (M, d) \) be a metric space and \( P(M) \) be the class of all compact non-empty subsets of \( M \). Define a new metric \( d_h(x, A) \) for \( x \in M \) and \( A \in P(M) \) by \( d_h(x, A) = \inf \{ d(x, a) \mid a \in A \} \). The Hausdorff metric for subsets, \( d_h(A, B) \) for \( A, B \in P(M) \) is defined as

\[
d_h(A, B) = \max \{ \sup \{d_h(a, A) \mid a \in A \}, \sup \{d_h(A, b) \mid b \in B \} \}
\]

The space \((P(M), d_h)\) is metric.

DEFINITION A3.10 – Contraction maps. A mapping \( f \) from a metric space \( M = (M, d) \) onto a metric space \( N = (N, e) \), \( f : M \to N \) is a contraction if there is a constant, \( 0 < K < 1 \), such that for all \( x, y \in M \), \( e(f(x), f(y)) \leq K d(x, y) \).

THEOREM (Banach) – Fixed points exist. If \( f : M \to M \) is a contraction over a complete metric space \( M \) then \( f \) has a unique fixed point in \( M \), written \( \mu_f \).
Interlude – Categories of behaviours

In this interlude we will suggest two pieces of formalism that might be used to connect the material of the previous chapter with the more categorical techniques of chapters two and four. In both cases, we just sketch the (rather involved) machinery and show why it is appropriate or, at least, plausible; a real application is left for further work.

Categories of metric spaces

In order to compare our betting equivalences with the work of chapter two, it is necessary to rephrase our metric spaces in a more categorical setting. There is a good reason for treating metric spaces as a particular kind of category, which we shall outline. This connection between metric spaces and categories was popularised by [Lawvere 1973]. In this section we will only rely on the triangle inequality and the property \( d(x, x) \geq 0 \), hence Lawvere's term 'generalised metric spaces.'

The essential idea is that many mathematical structures can be represented as categories whose hom-functor is valued not necessarily over \( \text{Set} \), but over some closed category \( V \). (So \( \text{hom}(a, b) \in \text{obj} V \).) We will discuss three examples, our aim being to make our treatment of metric spaces more plausible by preparing the way.

\[
\begin{align*}
\diamond \quad & \text{Let } P = (P, \leq) \text{ be a poset, and denote by } P(x, y) \text{ the truth-value of } x \leq y, \text{ so } P(x, y) \text{ is } \text{true} \text{ just when } x \leq y. \text{ Then the transitivity and reflexivity laws for } \leq \text{ are just the entailments} \\
& \forall x, y, z \in P. P(x, y) \land P(y, z) \rightarrow P(x, z) \\
& \text{true} \rightarrow P(x, x)
\end{align*}
\]

This suggests viewing \( P \) as a category whose set of objects is \( P \) and where there is an arrow from \( x \) to \( y \) just when \( x \leq y. \) The hom-functor for this category is valued over the two-point category \( 2, \) (whose objects are \( \text{true} \) and \( \text{false}, \) and where there is just one nonidentity arrow \( \text{false} \rightarrow \text{true} \)) so that if \( \text{hom}(x, y) = \text{true} \) then \( x \leq y. \)

\[
\begin{align*}
\diamond \quad & \text{Let } M \text{ be a category, and denote by } \text{hom}(x, y) \text{ the abstract set of } M\text{-morphisms from an object } x \text{ to an object } y. \text{ Here the hom-functor is valued over } \text{Set}; \text{ the abstract set comprising the product of the hom-set of morphisms from } x \text{ to } y \text{ and the hom-set of morphisms from } y \text{ to } z \text{ has an arrow from it in } \text{Set} \text{ to the (abstract) hom-set of morphisms from } x \text{ to } z; \text{ this is just what composition means in } M. \text{ Furthermore, there is an arrow from the one-element set } 1 \text{ in } \text{Set} \text{ to the abstract set of morphisms from } x \text{ to itself; this is just what identity means. So, using } \rightarrow \text{ for an arrow in } \text{Set}: \\
& \forall x, y, z \in \text{obj}(M). \text{hom}(x, y) \times \text{hom}(y, z) \rightarrow \text{hom}(x, z) \\
& 1 \rightarrow \text{hom}(x, x)
\end{align*}
\]
Let $M = (M, d)$ be a metric space. Then we certainly have
\[
\forall x, y, z \in M. \ d(x, y) + d(y, z) \geq d(x, z), \\
\forall x \in M. \ d(x, x) = 0
\]

Thus, associated with a given metric space, there is a chaotic category whose objects are the points of the metric space. (In a chaotic category there is an arrow between any two objects.) The underlying category that the $\text{hom}$-functor is valued over is $\mathbb{R}^+$ (whose objects are the nonnegative reals plus infinity, and where there is an arrow from $x$ to $y$ just when $x \leq y$). Composition in $\mathbb{R}^+$ gives us the triangle inequality, and identity in $\mathbb{R}^+$ gives us $d(x, x) = 0$.

In summary, we can view posets as categories whose $\text{hom}$-functors take values over the category $\mathbb{2}$, categories as categories whose $\text{hom}$-functors take values over the category $\text{Set}$ and metric spaces as categories whose $\text{hom}$-functors take values over the category $\mathbb{R}^+$. Each of these underlying categories $V$ is closed (i.e. symmetric monoidal with a right adjoint to the tensoring endofunctor $- \otimes b$). Composition in $V$ represents well-known properties of the structures concerned. $V$ is endowed with a tensor product which often represents properties of the identity in the overlying structure. We have seen the analogies: —

<table>
<thead>
<tr>
<th>Structure</th>
<th>$\text{hom}$-values for structure</th>
<th>composition and identity laws</th>
<th>domain of composition law</th>
<th>domain of identity law</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric space, $M$</td>
<td>nonnegative reals</td>
<td>$\geq$</td>
<td>sum</td>
<td>zero</td>
</tr>
<tr>
<td>Category, $C$</td>
<td>abstract sets</td>
<td>mapping</td>
<td>cartesian product</td>
<td>one element set</td>
</tr>
<tr>
<td>Poset, $P$</td>
<td>truth values</td>
<td>entailment</td>
<td>conjunction</td>
<td>true</td>
</tr>
<tr>
<td>$V$-valued category</td>
<td>objects in $V$</td>
<td>morphism in $V$</td>
<td>tensor in $V$</td>
<td>unit for tensor in $V$</td>
</tr>
</tbody>
</table>

The table suggests that we should pursue the analogy for metric spaces a little further interpreting $\geq$ as morphism in $\mathbb{R}^+$, $+$ as tensor, and $0$ as $1$. Then
\[
\text{hom}(x, z) = \begin{cases} 
  z - x & \text{if } z \geq x \\
  0 & \text{otherwise}
\end{cases}
\]

If we denote by a long dash, $-\!-\!-$, this truncated subtraction, then we have
\[
(y \geq z -\!-\!- x) \leftrightarrow (x + y \geq z)
\]

This is interesting because it indicates an adjointness between the internal $\text{hom}$ in $\mathbb{R}$, $-\!-\!-$, and the internal tensor, $+$, and hence a natural one-to-one correspondence between the $\mathbb{R}^+$ morphisms $y \rightarrow \text{hom}(x, z)$ and the $\mathbb{R}^+$ morphisms $x \rightarrow y \rightarrow z$. 
To sum up, then, we can view a metric space as a category whose objects are the points of the space, and whose morphisms are valued over the closed category $\mathbb{R}^+$. This interpretation places the metric spaces of the preceding chapter on a categorical foundation, which should be useful for comparing them to other models, such as the categories of chapter two. It also has the advantage of being part of a general programme of viewing mathematical structures as categories whose hom-functors are valued over a closed category: this approach allows us to predict many of the properties of the structures we seek to represent uniformly via categorical properties of the underlying closed category.

If we view our metric space structure in this categorical framework, certain difficulties are avoided. Consider three categories, $\text{Bet}(\text{cIES})_\mathbb{R}$ whose points are computational I.E.S.s that start at time 0. Suppose there is an arrow in $\text{Bet}(\text{cIES})_\mathbb{R}$ from $3$ to $3'$ just when $3 \succeq 3'$. We can recover the metric structure we had previously by endowing these three categories with an endofunctor $\star$. Furthermore, the category derived from the strict-timed metric space, for instance, can then be viewed as chaotic 2-category structure, – there is a 2-arrow on every isomorphism of $\text{Bet}(\text{cIES})_\mathbb{R}$, and the 2-hom functor is valued over $\mathbb{R}^+$, and similarly for the untimed space. This approach gives us the logic of betting as arrows, and the metric structure as 2-arrows, plus a representation of the $\delta$-timed case, all within the same framework.

We shall not explore Lawvere’s interpretation of a metric space any further, except to remark upon a connection between $\text{BOrder}$ and $\mathbb{R}^+$; just as objects of $\text{QOSet}$ can be seen sets enriched over $\text{BOrder}$, so, as we have seen, metric spaces can be seen as sets enriched over $\mathbb{R}^+$.

Next we suggest a different way of associating time with structures. Again the category $\mathbb{R}^+$ will be important, but now we will be more interested in behaviours.

The time-evolution of interval event structures

It is possible to describe the time-evolution of I.E.S.s using sheaves. An execution of an I.E.S. is just a L.I.E.S. If we can observe causality, then this is just a poset (all conflict has been resolved, so we are left with a set of partially-ordered transitions). Sheaf theory will give us a natural means of associating a poset with a point time (or, indeed, with a point in any topological space).

A poset $A = (A, \leq)$ is said to be embedded in another poset $A' = (A', \leq')$ just when $A \subseteq A'$ and $a \leq b \Rightarrow a' \leq' b$ and $a' \leq' b' \& \{a', b'\} \subseteq A \Rightarrow a' \leq' b'$ (so that $A'$ agrees with $A$ on all of $A$ but may have additional orders on things outside $A$). An indexed set $C = \{(A_1, \leq_1), (A_2, \leq_2), \ldots, (A_n, \leq_n)\}$ of posets is said to be compatible if $(A_1, \leq_1)$ is embedded in $(A_2, \leq_2)$ is embedded in ... is embedded in $(A_n, \leq_n)$.
To describe the time-evolution of an I.E.S. $\mathcal{F}$, then we need to associate some poset with each $r \in \mathbb{R}$, or, equivalently, some set of compatible posets with each interval $(r_1, r_2)$ of $\mathbb{R}$.

We will give a very ragged introduction to the idea of sheaves; a standard text is [Tennison 1975], from which these definitions are taken. A more interesting application of sheaf theory will be very briefly sketched in the interlude preceding chapter five, for which this section lays the groundwork.

My thanks are due to Barney Hilken for suggesting this diversion.

Suppose $T$ is a topological space. A presheaf $F$ of sets on $T$ is given by two pieces of information:

(i) For each open $U$ of $X$ a set $F(U)$ (called the set of sections of $F$ over $U$).

(ii) For each pair of opens $U, V$ of $X$, a restriction map $\rho^U_V$ from $F(V)$ to $F(U)$ such that

$$\forall U, \rho^U_U = \text{id}_U$$

and

$$\forall U, V, W. (W \subseteq V \subseteq U) \Rightarrow (\rho^U_W = \rho^V_W \circ \rho^U_V)$$

Let $T$ be a topological space and $F$ a presheaf of sets over $T$. $F$ is called a separated presheaf iff it satisfies:

Suppose that $U$ is an open set of $T$, and that $U = \bigcup_{j \in J} U_j$ is an open cover for $U$, and $c_i, c_j \in F(U)$ are two sections of $F$ such that

$$\forall j \in J. \rho^{U_j}_U (c_i) = \rho^{U_j}_U (c_j)$$

Then $c_i = c_j$.

If additionally, the following condition is satisfied, then $F$ is called a sheaf of sets over $T$.

Suppose that $U$ is an open set of $T$, that $U = \bigcup_{j \in J} U_j$ is an open cover for $U$, and that $c_i, j \in J$ is a family of sections of $F$ with $\forall j \in J. c_i \in F(U_j)$ such that

$$\forall i, j \in J. \rho^{U_j}_U \cap U_j (c_i) = \rho^{U_j}_U \cap U_j (c_i)$$

then there is a $c \in F(U)$ such that $\forall j \in J. \rho^{U_j}_U (c) = c_j$.

In other words, in a sheaf we can define things piecemeal on a cover, and if they are consistent on the overlaps, then they define something unique on all of $U$.

The application for us is obvious; take $\mathbb{R}$ with the usual topology as $T$, and associate with
an open interval a set of compatible posets representing the progress of the execution over that interval.

For an open interval of \( R, (r_1, r_2) \), and a given execution \( L \) of \( S \), define \( F(r_1, r_2) \) as the compatible set \( \{ L^r \mid r \in (r_1, r_2) \} \). This is finite since we demanded finite density.

For \( U = (r_1, r_2), V = (r_3, r_4) \subseteq U \), the restriction map \( p^U_V \) just takes \( \{ L^r \mid r \in (r_1, r_2) \} \) to \( \{ L^s \mid s \in (r_3, r_4) \} \). This gives us a presheaf of sets of posets on \( R \), since \( F(r_3, r_4) \subseteq F(r_1, r_2) \).

Suppose \( (r_1, r_2) \) is any open of \( R \) and \( \{(r_{1j}, r_{2j}) \mid j \in J \} \) is a cover of \( (r_1, r_2) \). Take \( L_i, L_j \in F(U) \). Clearly \( F \) is a separated presheaf since \( p^U_{U_j}(L_i) = \{ L^r \mid r \in L_i \} = p^U_{U_j}(L_i) \).

Finally, \( F \) is a sheaf since \( p^U_{U_j \cap U_j}(L_i) = \{ L^r \mid r \in L_i \cap U_j \} = p^U_{U_j \cap U_j}(L_i) \).

There is a more categorical way of dealing with sheaves. Suppose \( C \) is a category whose objects are sets, and that the objects of \( C \) will be sections of some sheaf \( F \). If \( U \subseteq V \), then we will require that there is an arrow in \( C \) from \( F(V) \) to \( F(U) \). (In the case \( C = \text{Set} \) this reduces to the previous definition.)

Consider \( O(T)^{op} \), the opposite of the category of open sets of \( T \) with arrows as inclusions (there is one arrow in \( O(T)^{op} \) from \( U \) to \( V \) if \( V \subseteq U \)). Then the obvious functor derived from \( F \), \( F \) say, is functorial from \( O(T)^{op} \rightarrow C \). Thus, in order to define a presheaf, we have to give a topological space \( T \), a category \( C \) and a functor \( F : O(T)^{op} \rightarrow C \).

References


This chapter is devoted to presenting a more concrete application for interval event structures than we have hitherto indulged in. A timed process algebra is presented: terms in this algebra can be naturally implemented as interval event structures. The laws obeyed by the algebra are examined, together with an operational semantics, an axiomatic semantics and a denotational semantics. The full abstraction problem is investigated for the algebra with respect to a notion of behavioural equivalence; we show that those terms that are behaviourally equivalent under the denotational are just the terms that can be shown to be equivalent using the laws of the axiomatic semantics. The problem is treated without considering recursion. We also discuss, en passant, the interpretation of process-algebraic constructs in Petri nets.

The main references to the material of this chapter are [Hennessey 1988], from which some material is taken, [Kwiatkowska 1989] and [Bednarczyk 1988]. Our process algebra is derived from both CCS [Milner 1989], timed CSP [Reed 1988], and standard (theoretical) CSP [Brookes et al. 1984]; its semantics draws heavily on the material of chapters one and two, since the meaning of a term in our algebra will be an interval event structure. We begin by defining our process algebra.

4.0 IPA – A timed process algebra

Interval Event Structures are quite detailed and implementational models of concurrency. It seems natural, therefore, to provide a slightly more abstract and structured framework for the description and implementation of real-timed systems. Process algebras have been popular as tools for reasoning about and implementing atomic–event concurrency; formalisms such as CSP (op cit.), COSY [Lauer & Shields 1983], CCS [Milner 1989] & SCCS [Milner 1983] have achieved some success. However, there has been comparatively little work on timed process algebras (as distinct from real-time languages); the only mature approach we know of in this direction is timed CSP (referred to in Oxford as TCSP, a term used for Theoretical CSP elsewhere).
Our process algebra, which will be referred to as *interval process algebra*, while notation-
ally similar to both timed CSP and CCS, is rather different semantically from either of those
formalisms, because we intend it to have a natural model in the world of interval event struc-
tures. It maintains the distinction between nondeterminism and parallelism that is the central
tenet of the non-interleaving school mentioned in the introduction. There are many ways of
building a process algebra ‘on top of’ event structures: this model falls midway between the
poset approach of [Boudol & Castellani 1987] & [Pratt et al. 1989] and the interleaving alge-
bras modelled in [Winskel 1984].

The intention in creating this process algebra is to provide a framework for constructing an
interval event structure using the usual techniques associated with process algebras (such
as the use of refinement calculi based on algebraically formulated notions of equivalence).
This approach should also provide a ‘bridge’ between the world of the last three chapters
and the world of process algebra.

A process can be thought of as a means of constraining the set of all interval events struc-
tures, leaving just those that display some required behaviour. A process, then, is a means
of characterising behaviour. Hence constructive activity with processes is complementary to
constraining activity with specifications. Our aim in, for instance, writing the bar specification
of chapter one, is to define precisely which interval event structures (and, hence, at a higher
level of abstraction, processes) are valid implementations of our intuitions. We will now
develop a theory of processes which will enable us to impose our intuitions on interval event
structures via the processes that they represent.

DEFINITION 4.0 – IPA, an interval process algebra

We shall adopt a syntax† similar to that of timed CSP. Suppose that we have a set of
events $E$, with typical element $e$. For the purposes of the process algebra we will assume
that the set is given, and finite; we will not inquire into the structure of its elements. There is
also the special set of events whose elements are wait $t$ for all $t \in \mathbb{R}^+$, the positive reals;
this set of waits is discussed below. We shall use a BNF-like syntax writing

$$\text{Action} ::= e \mid \text{wait } t$$

to indicate that an action is either some event $e \in E$ or a wait. The event wait $t$ does
nothing but occupies time $t$. We shall also introduce the abbreviations skip and stop for
the additional cases $\text{skip} =_{\text{def}} \text{wait } 0$ and $\text{stop} =_{\text{def}} \text{wait } \infty$. The use of these terms will be
discussed later; they are not actions, since 0 and $\infty$ are not elements of $\mathbb{R}^+$, but they do have
a technical use. Note that although Action is uncountable in general, for any given IPA
process it will only contain a countable number of wait $t$s, and hence will be countable
provided $E$ is.

† – The particular intuitions behind this notation will be explained later in this section; the confident reader
may safely skip from the end of this definition to section 4.1.
Processes can be formed from events in the usual way. A process is either a single action, or the nondeterministic composition of two processes, or the parallel composition of two processes (with a given synchronisation set $S = (P \uplus Q)$), or the sequential composition of two processes, or a variable process, $x$, which is used in the definition of recursion, where the recursion operator $\exists \cdot \cdot \cdot$ binds $x$ in $\exists x.\cdot \cdot \cdot$

\[
\text{Proc} := \text{Action} | \text{Proc} + \text{Proc} | \text{Proc} \parallel \text{Proc} | \text{Proc};\text{Proc} | x | \exists x.\text{Proc}
\]

The motivations for the various operators and their practical uses will now be discussed.

4.1 An introduction to Interval Process Algebra

We begin with the notion of an event or action. These have observable occurrences, we can see events happening and observe their effects. We shall collect the names of all the events that we are interested in into a set, $E$. For technical reasons we shall demand that this set is finite. Each element of the set, each event, has a duration associated with it. This duration will be used to build up timed structures of events.

The occurrence of nothing, of a pause, will carry the name wait $t$, where $t$ is the duration of the pause in some suitable prespecified units. These occurrences are interesting, too, so we shall form a set $\text{Action} = E \cup \{\text{wait } t | t \in \mathbb{R}^+\}$. (Cf. [Hennessy 1983]).

We shall assume that all events have a fixed, finite non-zero duration: no event can take no time, or infinite time. We shall also assume that there is a function $\Delta : \text{Action} \to \text{Time}$ which returns this time. (Times are reals, but we prefer to write $\text{Time}$ rather than $\mathbb{R}^+$ in order to make our intention clear.) Our aim in making the proviso on durations is to ensure that only a finite computation can take place in a finite time on a finite machine (it can be argued that infinite machines incorporating unbounded angelic nondeterminism or unbounded parallelism are capable of infinite computation in a finite time). The requirement that only finite computation is possible in finite time is seen as basic to the well-foundedness of timed models in [Joseph & Goswami 1985]; we agree with this analysis.

Notice that the requirement that the duration of all events and waits is greater than zero and less than infinity means that skip and stop are not strictly part of the syntax of IPA. The wait skip will fulfil a technical rôle like that of $\psi$ in CSP; it is not strictly a valid process. The problem is that it lasts no time, (and hence is unphysical). This durationlessness can lead to livelock if carelessly used. We will return to this point later.

The wait stop, on the other hand, is less dangerous and we will allow it to be used as an ordinary process, breaking the rule about durations being finite. So, the user can write a pro-
cess involving stop, but will not be allowed to use skip. The aim of writing a process algebra expression is, naively, to provide a description of some set of events that captures certain salient features about their order of occurrence. Further structure is given which indicates how certain sets of events may be thought of as interacting in certain ways. This structure gives us some idea of how a machine which could perform the desired actions in the desired order might be built. We can think of a process as a specification for a machine, – it comprises a set of events constrained to happen in a certain order, together with a name for this specification. Process are combined using process combinators which correspond to ways of building larger machines from smaller ones, that is, to ways of adding more structure to the sets of events in each process. The various combinators, \( \_; \) , \( ++ \), and \( \_\_ \) add different orderings of events to the orderings already present in the processes they combine.

The simplest useful process is something that engages in one event and then stops. Suppose we have given the name drink to the event of taking a drink. (Of course, we could have given the event any name we liked, but dog, say, or WhoWasWittgensteinAnyway are not very helpful signifiers, especially as we shall soon (in postmodernist vein) blur the distinction between processes and their names.) A prescription for a machine, named Drinker say, which can engage in just one event named drink and then stop would be

\[
\text{Drinker} = \text{def} \text{drink; stop}
\]

The \( =\text{def} \) symbol defines the process name on its left to be the name of the definition on its right. The definition is one of a process that can engage in one event, drink, and then (indicated by the sequential combinator \( \_; \) ) stop. A definition like \( a; b \) indicates that the event \( a \) happens then \( b \) does. We can construct a more accurate model of a drinker, called Drinker2, using further sequential combinations. (Where necessary for clarity we use placeholders with operator symbols, so \( \_\_ \) indicates that \( ; \) is an infix operator that takes two arguments on either side of it.)

The process

\[
\text{Drinker2} = \text{def} \text{tussle; serve; drink; wait 5}
\]

is a drinker which can engage in the event tussle, and then can engage in the event serve, and then can engage in the event drink, and then will wait five units of time before terminating. Notice that the previous drinker never terminated, as the event stop, once started, lasts forever.

So far we can just enforce one thing beginning after another has ended. This is not quite enough. We need to introduce something that will enable us to describe more than one event happening at once. The solution we adopt is to introduce the concurrency operator, \( \_\_ \) which defines the parallel composition of two processes.
The process, $P$, defined by

$$P =_{df} a \parallel b$$

can be thought of as a process which engages in the event $a$ and the event $b$ simultaneously; it continues engaging in the events $a$ and $b$ until both have finished. Thus if we had a process, Eater, say, which represents tussling at the bar, being served a packet of quail-flavoured crisps, eating them, and pausing afterwards to allow any colic to settle,

$$Eater =_{df} tussle; serve; eat; wait 10$$

then we could represent a couple, one buying and consuming food and the other drink by

$$Couple =_{df} (tussle; serve; eat; wait 10) \parallel (tussle'; serve'; drink; wait 5)$$

This process starts engaging in the two strands at the same time and finishes when both have finished.

We shall discuss interference between events in processes on either side of \( \parallel \) later. Notice, though, that here the two tussle events and the two serve events are entirely separate and non-interfering. Each occurrence of the same event has the same name, but events that are genuinely different, like two different people tussling at a bar, must have different names; hence the decorating prime.

The same person tussling twice before being served and eating, would be represented by

$$TusslesomeEater =_{df} tussle; tussle; serve; eat; wait 10$$

More realistically, a group of three eaters and drinkers, with one person buying all the crisps and one all the drinks might be represented by the process

$$Group =_{df} ((tussle; serve) \parallel (tussle'; serve') \parallel (wait \ t)); (drink1; wait 5) \parallel (drink2; wait 5) \parallel (drink3; wait 5)); ((eat1; wait 10) \parallel (eat2; wait 10) \parallel (eat3; wait 10))$$

Suppose

$$P =_{df} (tussle; serve) \parallel (tussle'; serve')$$

Then the value of $t$ which just fills the time available is

$$t = \text{end}(P) - \text{begin}(P)$$

i.e.

$$t = \max(\Lambda(tussle; serve), \Lambda(tussle'; serve'))$$
so that the person who neither gets the drinks nor the food must wait merely until both have been procured before he can consume them. Notice that here all the drinking must be over before any eating starts. We can visualise the process Group informally as a *process tree*, showing the evolution of the system (see figure 4.1 overleaf).

(*Theoretical aside:* the diagram suggests to the naïve that an IPA expression may be represented by a Petri net — the form of concurrency we discuss is very similar to (and, indeed, was motivated by) that in Petri nets. It will indeed be the case that we can represent IPA terms as C/E nets; see section 4.5 for details.)

(*Technical aside:* here we have assumed the existence of a function \( \Lambda : \text{IPA} \to \mathcal{P} \text{Time} \) generated from duration function \( \Delta : \text{IPA} \to \text{Time} \). (Recall that \( \Delta \) assigned a non-zero duration to every event.) Processes have sets of durations, due to nondeterminism;

\[
\Lambda(e) = \{ \Delta(e) \} \quad \text{where } e \text{ is an Action}
\]

\[
\Lambda(P \parallel Q) = \bigcup \{ \max \{ t, t' \} | t \in \Lambda(P), t' \in \Lambda(Q) \}
\]

\[
\Lambda(P ; Q) = \{ t + t' | t \in \Lambda(P), t' \in \Lambda(Q) \}
\]

\[
\Lambda(P + Q) = \Lambda(P) \cup \Lambda(Q)
\]

\[
\Lambda(\mu x. P) = \Lambda(P)
\]

where \( \Lambda(x) = \Lambda(P[\mu x. P/x]) \)

The \( \Lambda \) function allows us to make a well-formedness requirement for recursion:

\[
\min(\Lambda(\mu x. P)) > 0
\]

which ensures that recursions progress through time, and forbids processes like \( \mu x. \text{skip} ; x \) and \( \mu x. (\text{skip} + e ; x) \). Here the term \( P[y/x] \) stands for the term \( P \) with every occurrence of \( x \) substituted by \( y \).)

The crucial insight to be had from the diagram concerns the locus of control. In an expression like \( a ; (b \parallel c) ; d \), we start with one locus of control as we execute \( a \), then we have two as we go through \( b \parallel c \); these then join up to give us a single locus again as we execute \( d \). Our semantics for IPA must reflect this intuition. (This is the same idea as one we discussed in chapter one, — noninterleaving semantics boils down to the notion elucidated there that concurrency is not the same as nondeterministic choice between interleavings: it is not the same because there is more than one locus of control in genuine concurrency.)
The vertical extent of an event indicates a possible duration it might have, on some time scale following the arrow.

Figure 4.1 – A process tree for the Group process
With the syntax introduced so far (which we shall call that of little processes, or LProcs,)

$$\text{LProc ::= } e \mid \text{wait } t \mid \text{LProc} \text{||} \text{LProc} \mid \text{LProc} ; \text{LProc}$$

we have an adequate if sparse process algebra. The behaviour of LProc expressions can be characterised equationally. For instance, the order in which we combine processes using $\text{||}$ is unimportant. Combining a process in parallel with itself has no effect. We also want skip to be a zero of $\text{;}_-$ that is, sequentially combining a process with a wait of no time gives you just that process. A parallel combination of a process and a wait of less time than the process can possibly last just gives the process, while sequentially combining two waits just gives a suitably longer wait. (Notice that skip is not in the range of $P$ as it is not a process.) Hence, for any LProcs, $P$, $Q$ and $R$ say

$$P \text{||} Q = Q \text{||} P \quad (*)$$
$$P \text{||} (Q \text{||} R) = (P \text{||} Q) \text{||} R \quad (*)$$
$$P \text{||} P = P \quad (*)$$
$$P ; \text{wait } t = \text{wait } t \text{||} P = P \quad t \leq \min(A(P))$$
$$P ; \text{skip} = \text{skip} ; P = P$$
$$P ; (Q ; R) = (P ; Q) ; R$$
$$\text{wait } t ; \text{wait } t' = \text{wait } (t + t')$$

Now and in future lowercase courier type will indicate names ranging over events, and uppercase COURIER type will indicate names ranging over processes. (The rules $(*)$ are not true when we have synchronisations, as we shall see later.)

The rôle of skip is to act as a null process which does nothing and instantly terminates; the rôle of stop is to act as a deadlocked process: it does nothing but never terminates.

We shall now investigate the world beyond little processes. Parallel combination entails the possibility of synchronisation. Sometimes we may want two autonomous concurrent processes to do some action together. One may need the co-operation of the other for some reason: to provide it with a resource, or to pass information for instance. In order to facilitate this, we need to be able to let two processes synchronise upon an event so that the loci of control meet. This means that they engage in the event together. But what does 'engage in the event together' mean in the timed framework? We have two independently evolving processes which must engage in a composite event formed from the synchronisation of two events, one from each process. But which events can synchronise?

In chapter two we decided that two events can synchronise if their times overlap. Our paradigm, therefore, is this: in $P \text{||} Q$ both $P$ and $Q$ evolve independently. Together with $P$ and $Q$ we are provided with a synchronisation set; this set (first described in chapter two) contains pairs of events, one member of the pair being from $P$ and one from $Q$. Suppose $(a, b)$ is
an element of the synchronisation set of \( P \) and \( Q \) (written \( (a, b) \in (P \uplus Q) \)), then if the times of \( a \) and \( b \) overlap as \( P \parallel Q \) executes, we will allow them to synchronise.

The set \( P \uplus Q \) cannot contain waits, as we do not allow synchronisations on a wait; waits must occur asynchronously.* It is worth noting that \( (P \parallel Q) \) always displays pairs of events to the outside world; a pair with one component \( a \) indicates an asynchronous occurrence.

Whenever a parallel composition is written, a synchronisation set must be provided as well, containing all the desired synchronisations; if the times of occurrences of each event in a pair in the set allow, then a synchronisation will occur. Where the synchronisation relation is nonempty, or cannot be inferred from context, it will be allowed to decorate a parallel composition, so that if \( S = P \uplus Q \), we will write the parallel composition of \( P \) and \( Q \) as \( P \parallel Q \), read "\( P \parallel Q \) with the synchronisations \( S \) desired." We understand by \( P \parallel Q \) the parallel composition of \( P \) and \( Q \) with empty synchronisation set, i.e. \( P \parallel Q \).

Let us return to the bar: an event upon which a synchronisation should happen is called shared. Consider the process \( Drinker =\text{def} \ tussle; \text{serve}; \text{drink}; \text{wait} 5 \).

Obviously it is expected that the \( Drinker \) process will be served by someone. Perhaps a simple barman process might be

\[
\text{Barman} =\text{def} \text{pour_drink}; \text{skip}
\]

with the \text{pour_drink} event in the barman synchronising with (or matching) a serve event in the drinker. The barman in combination with a drinker might be a \( \text{Bar} \) process

\[
\text{Bar} =\text{def} \text{Drinker} \uplus \text{Barman}
\]

and we would have \( S = \text{Drinker} \uplus \text{Barman} = \{(\text{serve, pour_drink})\} \).

Remember that asynchronous occurrences become decorated in a parallel composition (because they are modelled as synchronisations with the other processes' silent event), so that we can keep track of what is what in constructions like \( \text{Bar} \parallel \text{Bar} \).

The synchronisation discipline outlined above is in sharp contrast to that of CCS or CSP; both of these languages allow us to require a synchronisation to happen. Here, we cannot – it will only happen if the times of the matching events allow. This does not alter the usefulness of IPA, as we can always prefix one event of a matching pair with a wait \( t \) for some unspecified \( t \), which we fix when we know the value it must take to enforce a synchronisation. (This point will be returned to in chapter five.)

---

* In fact, the synchronisation relation of chapter two contained pairs of labelled events; this distinction is useful if we want different synchronisation behaviour from different occurrences of the same event, but it leads to considerable technical complication, so we shall assume that the synchronisation relation holds between events in future.
As the reader will have noticed, the material of chapter two will keep on intruding on our considerations here. We have decided to model our process algebra with I.E.S.s, so the combinators in the algebra must be similar to those available in the world of I.E.S.s. Therefore, our notion of parallelism is very close to that elucidated in chapter two, and our notion of nondeterminism will be designed so that the coproduct constructions of chapter two are an appropriate model for it. It may prove helpful to reread chapter two in parallel with this section. Remember that an I.E.S. can always evolve without reference to its environment; it just gets on and does its events with their specified timings, resolving its own nondeterminism. IPA processes must be capable of this degree of autonomy. Further, recall that we were led in chapter two to consider two closely associated I.E.S.s representing the synchronisation behaviour of a system and the duration of the occurrences of its events (the union and intersection I.E.S.s); here too we shall end up with two I.E.S.s representing every process.

(Aside: in IPA we never force a process to wait without an explicit wait; it can always evolve without reference to its environment (although one part of a process, such as one component of a parallel composition must sometimes wait for the other components, so that two loci of control can merge back to one). (The surroundings of a process, what it is put in parallel with, are known as its environment.) This not only simplifies the real-time semantics, it is also an accurate reflection of the behaviour of autonomous systems; unless we ask them to interact with their environment, they don’t (that’s why they are autonomous). It seems unnecessary to build such interactions in as primitives; without them the user is allowed to specify whatever process/environment interactions he or she desires.)

Clearly we would like to be able to specify a barman who can serve an unbounded number of drinks (and a customer who can drink them). We do this using recursion. The process

\[ \text{InfiniteBarman} = \text{def } \mu x. \text{pour_drink}; x \]

will do for the barman, and

\[ \text{ThirstyMan} = \text{def } \mu x. \text{serve}; \text{drink}; x \]

for the drinker. Notice that there is no pause required here between finishing the \( n - 1 \)th drink and starting to be served the \( n \)th.

The barman process is read “The process \( \text{InfiniteBarman} \) is defined to be ...” (\( \text{InfiniteBarman} = \text{def} \) “ ... the least process \( x \) which ...” (\( \mu x. \) ) “ ... can perform a \( \text{pour_drink} \) action and then ...” (\( \text{pour_drink;} \) )“ ... behave like \( x \)”.

The behaviour of the \( \text{InfiniteBarman} \) process is, then, to do a \( \text{pour_drink} \) event and then to behave like itself. Behaving like itself involves doing a \( \text{pour_drink} \) action and then behaving like itself, which involves doing a \( \text{pour_drink} \) action and then behaving like itself, ...

So we obtain an unbounded number of \( \text{pour_drink} \) actions. The operator \( \mu x. \) takes a process containing a variable \( x \) (\( P \) say) and produces the least process obtained by replacing every occurrence of \( x \) by \( P \).
Returning to the bar, then, we might chose a more realistic model with several drinkers;

\[ \text{TuesdayNightBar} =_{\text{def}} \text{Drinker}_1 \text{ } \text{Drinker}_2 \text{ } \text{Drinker}_3 \text{ } \text{Drinker}_4 \text{ } \text{InfiniteBarman} \]

Notice that the bracketing is necessary to ensure that each synchronisation set is used properly. Even if we had \( S_1 = S_2 = S_3 = S_4 \) (modulo asynchronous pairing) then the results of chapter two will not ensure that we can write \text{TuesdayNightBar} without brackets.

This point is worth a little expansion. Consider \( P \text{ } Q \text{ } R \); we would like to state that \( P \text{ } Q \text{ } R \) for some suitable \( S_3 \) and \( S_4 \), but which \( ? \) Clearly \( (a, (b, c)) \) can occur in the LHS just when \( (a, (b, c)) \in S_1, (b, c) \in S_2 \), so

\[
((a, b), c) \in S_4 \text{ } \text{and} \text{ } (a, b) \in S_3 \iff (a, (b, c)) \in S_1 \text{ } \text{and} \text{ } (b, c) \in S_2
\]

Similarly,

\[
((a, b), c) \in S_4 \text{ } \text{and} \text{ } (a, b) \in S_3 \iff (a, (b, c)) \in S_1 \text{ } \text{and} \text{ } (b, c) \in S_2
\]

If \( S_1, S_2, S_3 \) and \( S_4 \) satisfy these conditions then we say that \( S_3 \) and \( S_4 \) are the conjugate of \( S_1 \) and \( S_2 \) and write \( (S_1, S_2) \leftrightarrow (S_3, S_4) \). Notice in particular that if we have no synchronisation, \( P \text{ } Q \text{ } R \) \( \equiv \) \( P \text{ } Q \text{ } R \) \( \equiv \) \( P \text{ } Q \text{ } R \). Similarly, \( P \text{ } Q \text{ } R \equiv Q \text{ } R \equiv P \) where \( (b, a) \in S \iff (a, b) \in S \).

Previously it was clear what synchronised with what. But, in the \text{TuesdayNightBar} process we have four separate serve-like events, \text{serve}1 through \text{serve}4. Which of these synchronises with which \text{pour_drink} event?

We can obviously construct an unfair barman by only allowing \text{pour_drink} events to synchronise with, say, \text{serve}1 events, \( (S_1 = \{ \text{serve}1, (*, (*, (*, \text{pour_drink}))\}) \), all other synchronisation sets empty) but this will not do. Or we could make a wholly synchronous bar by making \text{pour_drink} events synchronise in turn with \text{serve}1 events, \text{serve}2 events, \text{serve}3 events, and \text{serve}4 events (this would require four different \text{pour_drink} events, with, for instance, \( S_2 = \{ \text{serve}2, (*, (*, \text{pour_drink}2))\} \)). This would, however, make the bar somewhat inefficient, as the barman is constrained to serve at the speed of the slowest drinker. The most efficient approach would be to let a \text{pour_drink} event synchronise with a \text{serve} event whenever one is available. Then we would have

\[
S_1 = \{ \text{serve}1, (*, (*, (*, \text{pour_drink}))\}
\]

\[
S_2 = \{ \text{serve}2, (*, (*, \text{pour_drink}))\}
\]

\[
S_3 = \{ \text{serve}3, (*, \text{pour_drink})\}
\]

\[
S_4 = \{ \text{serve}4, \text{pour_drink}\}
\]
which ensures that no pour_drink event is wasted. Because we have not enforced any particular synchronisation discipline any of these approaches is possible; it is up to the user to specify which events synchronise with which by giving an appropriate synchronisation set. Notice that the order of parallel composition is important here — if Drinker4 always demands a drink whenever the barman is ready to give him one, then no other drinker will ever be served; it is only if Drinker4 allows a pour_drink action to occur asynchronously that another drinker can synchronise with it. Hence our form of parallel composition is local in the sense that the ordering of application of $\parallel$ is important.

This strategy also allows us to wash our hands of the vexing problem of fairness; that of making sure that if two drinkers are waiting to be served they are served fairly. This issue will be discussed in the next aside.

The final operator we shall discuss is the nondeterministic choice operator $+_\perp$. Suppose we have two alternative processes $P$ and $Q$, and we want one of them to be chosen and executed. This choice can be achieved in two ways; either we can let the process do it, washing our hands of all responsibility for (and knowledge of) how the choice is taken, or we can allow the environment of the process to have some affect on what is chosen. (The former is often known as deterministic or external choice, and the latter as nondeterministic or internal choice; the use of the term 'external nondeterminism' for the former is very confusing (as no nondeterminism is involved) and should be avoided: we will not entertain that usage here).

**ASIDE — On nondeterminism**

There are at least four ideas that are referred to as 'nondeterminism' in popular concurrency theories. These are:

(i) Delayed implementation decisions. [Hoare 1985] suggest that nondeterminism is useful as a way of modelling an implementation decision that has not yet been taken;

(ii) Determinism by some unmodelled feature. Here it is assumed that the behaviour of the system is determined, but the object doing the determining is not accessible to us (this is the sort of nondeterminism that results from hiding, as in the CSP process

$$R = \text{def } ((a \rightarrow P \parallel b \rightarrow Q) \setminus \{a\})$$

This can give rise to nondeterminism in CSP: $R = P \parallel b \rightarrow Q$. Incidentally, notice that it needn't:—

$$((c \rightarrow a \rightarrow P \parallel b \rightarrow Q) \setminus \{c\}) \parallel (a \rightarrow S) = a \rightarrow ((P \setminus \{c\}) \parallel S)$$
(iii) **Genuine physical nondeterminism.** Here the behaviour of the system is really unknowable. This is similar to (ii), but philosophically distinct from it.

(iv) **Identical external choices.** If a CSP process offers the environment the choice

\[ T =_{df} a \rightarrow P \parallel a \rightarrow Q \]

then, as the \( a \)'s are identical it is not determined whether an \( a \) action from the environment results in \( P \) or \( Q \). The process \( T \) is equivalent to \( a \rightarrow (P \parallel Q) \).

It is often unclear which of these alternatives is meant when the term 'nondeterminism' is used in the literature. In particular, as [de Nicola & Hennessy 1987] point out, the correct philosophical position of the CCS + operator is uncertain; it can act either as an external choice or (if, for instance, a \( e \) is involved) as type-(iii) nondeterminism.

We intend the choice operator \( _+_{-} \) in IPA to represent purely nondeterministic choice roughly with flavour of (iii) (or (ii)) above (i.e. roughly CSP \( \parallel \)). This means, of course, that processes do not have to wait for the environment to do any choosing, and their behaviour is independent of the environment. This has the advantage that we can assign meaning to a process without reference to its environment. (Deterministic choice could be introduced into the model by the strategy of allowing the environment to choose the \( P \) or \( Q \) in \( P + Q \) by offering one of the first events of one process for synchronisation but not one of the first events of the other. This will overcomplicate things here, so we shall not attempt it. The key point to notice is that any deterministic choice must allow the process to proceed, so that if the environment doesn’t do the choosing the process must make the choice itself (presumably nondeterministically) so that it can get on. The interlude at the end of this chapter contains further thoughts on this matter.)

At this point fairness again rears its ugly head. The classical fairness problem is this: suppose infinitely often in the history of a process two events are possible, like the two drinkers waiting to be served above. A process is said to be (unconditionally event) fair (with respect to this pair of events) if both of these events are executed infinitely often. This means that one event can be chosen in preference to the other only finitely many times. The subject of fairness is a complex and engaging one (see, for instance, [Kwiatkowska 1989] for further discussion and references). For us, fairness is a multifaceted issue; there is the problem of the fairness of internal choice (whether in any infinite set of executions of \( A + B \), it is possible for \( A \) to be chosen exclusively), and the problem of the fairness of synchronisation (whether it is possibly for one synchronisation of two available to be always preferred).

We shall deal explicitly with neither issue. Our stance on the former is that the whole point about nondeterminism is that nothing can be assumed about it, that no finite observation can distinguish a fair from an unfair process, and that the fairness of choice is an
implementational matter anyway, so we unashamedly refuse to allow the user to require that internal choice is fair.

The fairness of synchronisation will be even more blatantly ignored; parallel composition is clearly not fair, since if two synchronisations are possible the most tightly bound one will be preferred; priority of synchronisation of A over B can be achieved by using \((P \parallel A) \parallel B\) rather than \((P \parallel B) \parallel A\). The user must provide a synchronisation set which encodes information about which event synchronises with which. If fair scheduling is required, the user must explicitly state how it is to be implemented. (In practise more stringent requirements than fairness are usual (such as requiring not only that no process is always preferred to another, but also that no process is often preferred to another, for some suitable notion on ‘often’), so the user will often have to provide their own scheduler anyway.) This seems reasonable since it is hard to see how to implement completely fair synchronisation — our local form of parallel composition reflects the notion that multiple synchronisations can only be built from binary ones. [Lamport 1985] has an interesting perspective here.

A typical external choice we might need in the bar is

\[ X \equiv \text{serve\_crisps;} E + \text{serve\_beer;} D \]

Here the process can execute one of the two events serve\_crisps and serve\_beer. If it chooses serve\_crisps then E will execute, whilst if it chooses serve\_beer then D will be executed. This choice is made purely nondeterministically, by the process itself.

The topic of recursion will now be revisited. A process which engages in an unbounded number of rounds might be

\[ \text{Group} \equiv \mu x. ((\text{tussle;} \text{serve}) \parallel (\text{tussle'}; \text{serve'}) \parallel (\text{wait } t)); \\
(\text{drink}1; \text{wait } 5) \parallel (\text{drink}2; \text{wait } 5) \parallel (\text{drink}3; \text{wait } 5)); \\
(\text{eat}1; \text{wait } 10) \parallel (\text{eat}2; \text{wait } 10) \parallel (\text{eat}3; \text{wait } 10)); x \]

while a process which may or may not have another drink is

\[ \text{UndecidedDrinker} \equiv \mu x. (\text{tussle;} \text{serve}; x + \text{go\_home}) \]

The environment, of course, may ‘want’ one decision from the UndecidedDrinker; if

\[ \text{IrritatedSignificantOther} \equiv \text{wait } 5; \text{argue}; \text{sleep} \]

and \( S = (\text{argue}, \text{go\_home}) \), then, assuming for simplicity that

\[ \Delta(\text{wait } 5; \text{argue}) = \Delta(\text{go\_home}) < \Delta(\text{tussle}; \text{serve}) \]

that is, that the synchronisation can happen before any drinks are consumed.
We then have

\[
\text{UndecidedDrinker} \parallel \text{IrritatedSignificantOther} = \\
(\text{go\_home} \parallel \text{wait} 5; \text{argue}; \text{sleep}) + \\
(\text{tussle}; \text{serve}; \mu x. (\text{tussle}; \text{serve}; x + \text{go\_home})) \parallel \text{wait} 5; \text{argue}; \text{sleep}
\]

with the synchronisation \((\text{argue}, \text{go\_home})\) occurring in the first clause and not in the second. (So the UndecidedDrinker either goes home and has the argument or stays and has another round. If he has another round, the IrritatedSignificantOther must have the argument alone (as the UndecidedDrinker has taken so long drinking that he or she is too late to synchronise with the argument) and then go to sleep; the UndecidedDrinker may return after some (unbounded) further number of rounds.) This argument depends of the durations of the events being as stated, and on both components in the parallel composition starting at the same time.

The choice \(+\) is purely nondeterministic; our UndecidedDrinker could go on tussling and being served for ever. An appropriate process tree for this process is shown below.

![Process Graph](image)

Figure 4.2 — A process graph for the UndecidedDrinker process, not showing possible synchronisations.

An attempt to provide some laws for nondeterminism should be made. These laws are very similar to those for parallel composition:

\[
P + Q = Q + P \\
P + P = P \\
P + (Q + R) = (P + Q) + R \\
(P + Q) ; R = (P ; R) + (Q ; R) \\
(P + Q) \parallel R = (P \parallel R) + (Q \parallel R)
\]

The last two laws are the only interesting cases;

(i) \((P + Q) ; R = (P ; R) + (Q ; R)\) is interesting principally because the obvious compliment to it

\[
R ; (P + Q) = (R ; P) + (R ; Q)
\]
is not true. In \((P + Q) \parallel R\) we are committing ourselves to making an irrevocable decision between \(P\) and \(Q\); there is no state following \(P + Q\) which does not have its ancestry unambiguously in either \(P\) or \(Q\).

On the other hand, if we had the axiom

\[ R ; (P + Q) = (R ; P) + (R ; Q) \]

then we would be able to bring forward the point at which a decision is made,—this is tantamount to introducing time travel into the model. The arguments for why this law should not hold have been rehearsed often; the point is that after the \(R\) on the LHS both \(P\) and \(Q\) are possible, whereas only one of them is on the RHS. Hence allowing the above means allowing a transformation which introduces ambiguity; before it, everything was clear, but after it the same observable sequence of actions leads to different states. This is undesirable, so we shall not allow \(R ; (P + Q) = (R ; P) + (R ; Q)\) to be true in general.

(ii) The last law, \((P + Q) \parallel R = (P \parallel R) + (Q \parallel R)\) states that parallel composition distributes through nondeterminism. The intuition is that in the LHS \(R\) cannot influence the choice, so one might as well let it run with each alternative and make the choice between the parallel compositions. Notice in particular if \(S \subseteq \alpha P \cap \alpha Q\) does not hold, then we have a synchronisation set not in the product of the alphabets of the processes it relates to. This is not, in itself, a problem, as there is not way to offer \(R\) in \((P \parallel R)\) events from \(Q\), but the extra events must be removed in the translation to a chapter 2 style synchronisation set.

We can also formulate an axiom for the behaviour of a recursive term using substitution. The process \(\mu x. P\) behaves like \(P\), but with every occurrence of \(x\) replaced by \(\mu x. P\). (This accords with our clumsy characterisation above of \(\mu x. P\) behaving like \(P\) until we hit an \(x\), and then behaving like itself again.)

\[ \mu x. P = P \left[ \mu x. P / x \right] \]

This completes our description of the process algebra IPA. The meaning of IPA expressions will occupy us for the rest of the chapter. No attempt will be made to provide a specification calculus for IPA; this deficiency should be rectified in the future.
4.2 The axiomatic semantics of interval process algebra

The laws of IPA discovered by informal arguments in the last section will now be gathered up and stated together. Recall that we had for \( LProc \cup \{\text{skip}\} \) the laws

\[
P; \text{skip} = \text{skip}; P = P
\]

\[
\text{wait}. t; \text{wait} t' = \text{wait} (t + t')
\]

\[
P; (Q; R) = (P; Q); R
\]

\[
P \parallel Q = Q \parallel P
\]

\[
P \parallel \text{wait} t = \text{wait} t \parallel P = P
\]

\[
P \parallel \text{wait} t = \text{wait} t \parallel P = P
\]

The laws above are supposed to refer to processes constructed using events from \( E \).

It is also instructive to contemplate some of the laws that are not true. For instance, some laws which are in general false in this semantics for IPA are

\[
a; (b \parallel c) = (a; b) \parallel (a; c)
\]

\[
(a; b) \parallel (c; d) = (a \parallel c); (b \parallel d)
\]

The first law follows from the second for \( a; (b \parallel c) = (a \parallel a); (b \parallel c) \) which is not the same as \( (a; b) \parallel (a; c) \). Consider

\[
\text{Figure 4.3 – The difference between}(a \parallel c); (b \parallel d) \text{ and } (a; b) \parallel (c; d)
\]

The reason we want these laws to be false is that in \( (a \parallel c); (b \parallel d) \) both \( a \) and \( c \) must be over before \( b \) or \( d \) start (furthermore \( b \) and \( d \) start at the same time). Neither of these properties hold in \( (a; b) \parallel (c; d) = (a \parallel c); (b \parallel d) \). Notice finally that the CSP-style law
\( P \parallel \text{stop} = \text{stop} \) does not hold here, again due to our stance on the nature of concurrency. The second law is proposed in [Meseguer & Montanari 1988] as "capturing a rather basic fact about concurrency"; this intuition does not extend to our framework. The essential difference is that \( P = (a ; b) \parallel (c ; d) \) is one process with one locus of control, whereas for Meseguer & Montanari \( (a ; b) \) and \( (c ; d) \) are two processes that are wholly independent – they may just happen to be executing at the same time. Compare our Petri net interpretation of parallelism in section 4.5 to those of [Olderog 1989] for a clear idea of the distinction.

Next the axioms for choice and nondeterminism are stated.

\[
\begin{align*}
P + Q &= Q + P \\
P + P &= P \\
P + (Q + R) &= (P + Q) + R \\
(P + Q) ; R &= (P ; R) + (Q ; R) \\
(P + Q) \parallel R &= (P \parallel R) + (Q \parallel R)
\end{align*}
\]

\[
\mu x. P = P \left[ \mu x. P / x \right]
\]

The axioms for nondeterminism and recursion

All of laws can be used to reason about IPA expressions; in section 4.6 we will formalise a proof system based on them. The main idea is that the laws tell us which terms behave the same way, and hence can be thought of as equivalent.

It may be helpful to define the alphabet of an IPA process \( P \), written \( \alpha P \), as this concept has been used informally several times;

\[
\begin{align*}
\alpha e &= \{ e, * \} \\
\alpha (\text{wait } t) &= \{ * \} \\
\alpha (P ; Q) &= \alpha P \cup \alpha Q \\
\alpha (P + Q) &= \alpha P \cup \alpha Q \\
\alpha (P \parallel Q) &= \{ (e, *) | e \in \alpha P \} \cup \{ (*, e') | e' \in \alpha Q \} \cup \{ (e, e') | (e, e') \in S \} \\
\alpha (\mu x. P) &= \alpha P - \{ x \}
\end{align*}
\]

The omnipresent * indicates that, in addition to 'doing' events, \( e \in E \), the process must start and stop, – the 'on light' * must behave sensibly. (The claim that \( (e, * ) \) is observably different from \( e \), and hence the veracity of the definition of \( \alpha (P \parallel Q) \), rests on allowing the observation of causality.)

We shall require in future that if \( P \parallel Q \) is written, \( S \subseteq \alpha P \times \alpha Q \). This is a commonsense requirement, allowing us to ignore pathological synchronisation sets.
4.3 The denotational semantics of interval process algebra

We shall give a denotational semantics to our process algebra. The task of such a semantics will be to determine what I.E.S.s are, in some sense, valid representations of an IPA expression. This will determine which IPA expressions are equivalent (they are equivalent when they have equivalent denotations). The proof rules for IPA can then be compared to this semantics; the congruence induced by them and that induced by the semantics will be shown to be the same.

In fact, an IPA expression has an associated set of I.E.S.s that (intuitively) satisfy it: the denotational semantics associates one member of the set of satisfying structures with a given term; this is the maximally live structure in that set (that is, it gets on with things as soon as possible†).

Two IPA expressions can be thought of as equivalent if they behave the same way. Hence, rather than demand equality of meanings (that is equality of maximally live representing I.E.S.s) as a criterion for equivalence, we shall demand equality of behaviour (in the sense of the last chapter). This is because equality of meanings is too weak; we want to identify structures that 'we cannot tell are different.' The extra structure may, though, be useful when we come to consider implementing process algebra expressions as interval event structures; the denotational semantics can be thought of as suggesting some structure an implementation might have, while the laws merely tell us how the implementations behave. The denotational semantics, then, is our link with the world of implementations.

(A point we shall not tackle, but which is of some interest, is whether we can recover the equivalence induced by the axiomatic semantics by considering isomorphism (in the categorical sense) rather than equality. The category in which this is recovered could then be claimed to have the operationally 'natural' notion of behaviour.)

A Scott–Strachey-style semantics will be given as advocated in [Scott & Strachey 1971]. Here two descriptions are considered equivalent if they map to the same object in the domain of meanings. However, we shall use categories rather than domains; for us, the set of meanings will in fact be objects of the category tIES. (Objects in this category are members of the class of all interval event structures, IES. We shall only be concerned with I.E.S.s as (re-) defined in chapter two.) The meaning function will be structure-oriented so that the meaning of a complex process is determined by the composition of the meanings of its constituents.

† Notice that nowhere have we said that writing a ; b implies a adj b; we have merely supposed that end(a) ≤ begin(b). Now we will impose a semantics where a adj b — things happen as soon as they can.

It would be fairly trivial to modify this to fit the timed CSP paradigm where end(a) = begin(b) + δ, for some (possibly variable) δ. Then the law wait t ; wait t' = wait (t + t') would need modification.
We will assume for the moment that all structures start at \( t = 0 \); our task is to determine what I.E.S. will result if a given IPA term starts to execute at that time. So, then, we want to define a function

\[
C : \text{Proc} \rightarrow \text{IES}
\]

which, given an IPA term \( s \) will produce a I.E.S., \( C(s) \). However, we will have to deal with I.E.S.s that display more than one maximal Con-set, thus possibly ending at more than one time. Hence, really, we will need a function which given a set of starting times, returns an I.E.S. and a set of possible finishing times

\[
D : \text{Proc} \rightarrow \varnothing\text{Time} \rightarrow (\text{IES} \times \varnothing\text{Time})
\]

then we just define the denotation of a structure as the result if we start at just time 0;

\[
C(s) = \text{fst}(D(s) [0])
\]

The main meaning function \( D \) is defined by cases over the structure of terms. It decomposes an IPA-composition of terms into a \( \text{ties} \)-composition of meanings using the \( \text{ties} \)-operators defined in chapter two. For convenience we shall define auxiliary functions

\[
E : \text{Proc} \rightarrow \varnothing\text{Time} \rightarrow \text{IES},
E(s) t = \text{fst}(D(s) t)
\]

\[
F : \text{Proc} \rightarrow \varnothing\text{Time} \rightarrow \varnothing\text{Time}
F(s) t = \text{snd}(D(s) t)
\]

where \( \text{fst}(x, y) = x \) and \( \text{snd}(x, y) = y \).

Primitive actions are converted into one element I.E.S., while waits become empty structures (or, rather, structures with just silent events in them). The clause for \( \text{wait} t' \) works for \( \text{stop} \) as well, with the definition \( \text{stop} = \text{def} \text{wait} \infty \). The definition of \( D \) that we shall adopt in the base cases, then, is

\[
D(e) t = \text{def} \left( \sum_{t' \in t} M(e) t', \{ t + \Delta(e) \mid t \in t \} \right)
\]

\[
D(\text{wait} t') t = \text{def} \left( \sum_{t' \in t} M(\text{wait} t') t', \{ t + t' \mid t \in t \} \right) \quad \text{for } t' > 0
\]

so that events and waits just happen whenever they can. (\( M \) is defined overleaf.) For the more complicated, structured cases, we have

\[
D(p \parallel q) t = \text{def} \left( \Psi', t' \right) \quad \text{where } \Psi = E(p) t \parallel E(q) t 	ext{ and }
\]

\[
t' = \bigcup_{t \in t} \{ \max(t, t_2) \mid t_1 \in F(p) \{ t \}, t_2 \in F(q) \{ t \} \}
\]
Here the I.E.S. synchronisation set $T$ over labelled events is derived from the IPA one, $S$, over events by $T = \{(n, a), (n', b)\} \mid (a, b) \in S, n, n' \in \mathbb{N}, a \in \alpha_\mathbb{P}, b \in \alpha_\mathbb{Q}\}$. (The integers are used as labels.)

(Recall that we indicated two approaches to dealing with timed synchronisations; conventionally (i.e. in process algebras like CSP), we insert waits before events so that all the synchronisations 'demanded' actually happen, or the process deadlocks. Alternatively, (and this is the course adopted above,) we only allow a synchronisation to happen if the two synchronising event overlap in time.)

\[
D(P + Q) t = \text{def} (\Psi, t') \quad \text{where } \Psi = E(P) t + E(Q) t \quad \text{and } t' = F(P) t \cup F(Q) t
\]

\[
D(P ; Q) t = \text{def} (\Psi, t') \quad \text{where } \Psi = E(P) t \oplus E(Q) (F(P) t) \quad \text{and } \quad t' = \bigcup_{t \in t} \{F(Q) \{t\} \mid t \in F(Q) \{t\}\}
\]

In the definition above, certain combinators (\texttt{!} and +) have been overloaded; they are defined both on IPA terms and on I.E.S.s. To decrease confusion slightly, the I.E.S. versions are shown in \textbf{bigger, bolder} type.

The function $M$ produces an interval event structure containing just one labelled event (and a silent event). It also deals with the production of I.E.S.s containing just silent events. (We will show the output of $M$ in the form (labelled events, maximal Con-set, ordering). We will also show the (real) timing of \texttt{*}.)

In the case of an event we have;

\[
M: \text{Action} \rightarrow \text{Time} \rightarrow \text{IES}
\]

\[
M(e) t = ((n, e), *), \{((n, e), *)\}, \{(t, t + \Delta(e))\}) \quad e \neq \text{wait} t'
\]

\[
\text{where } * = [t, t + \Delta(e)] \quad \text{and } n = \text{next}(e)
\]

(The function \texttt{next} is assumed to keep track of which labels have been issued to a given event, and to always provide a new one.) In the case of a wait

\[
M(e) t = ((\ast), \{\ast\}, \{(t, t + t')\}) \quad e = \text{wait} t'
\]

\[
\text{where } * = [t, t + t']
\]

The two flavours of I.E.S., $3^-$ and $3_-$, are dealt with in the obvious way;

\[
(3M(e) t^- = \{3M(e) t\} = M(e) t
\]

\[
(E(s \ op s') t^- = (E(s \ op s') t)_-
\]

for \texttt{op} $\in \{+, ;\}$ while

\[
(E(s \| s') t^- = (E(s) t \|_T E(s') t^- = E(s) t \| E(s') t)
\]
and
\[(\mathcal{E}(s \parallel s^r) t)_- = (\mathcal{E}(s) t \parallel \mathcal{E}(s^r) t)_- = \mathcal{E}(s) t \parallel \mathcal{E}(s^r) t\]
as defined in chapter two.

Several points deserve clarification. First note that the primitive I.E.S. constructor \(M\) always produces a computational structure, and that finite application of the I.E.S.-combinators preserves computationality, so all the structures that are denotations of IPA terms are computational.

Note too that the set of finishing times \(\mathcal{F}\) that we pass around in the denotational semantics has a function very similar to the set of durations \(\Lambda\) we use in the laws; we have the law
\[P \parallel\text{wait } t = \text{wait } t \parallel P = P\]
where \(t \leq \min(\Lambda(P))\)
which is closely related to the denotation
\[\mathcal{D}(P \parallel Q, t) = \text{def } (\mathcal{E}(P) t \parallel \mathcal{E}(Q) t, t')\]
where \(t' = \bigcup_{i \in t} \{\max(n, t_i) \mid t_i \in \mathcal{F}(P) (t), t_2 \in \mathcal{F}(Q) (t)\}\)
4.4 The operational semantics of interval process algebra

This section introduces a timed operational semantics for IPA. The framework used for this semantics, that of asynchronous timed labelled transition systems, is new; an introduction to timed labelled transition systems forms the first subsection of this section, while asynchronicity is introduced in the second. The formalism is then applied, giving an operational semantics for interval process algebra.

The operational semantics of IPA will be in the structured operational semantics tradition of [Plotkin 1981]. There every expression, $s$, is associated with a transition system $\{S, Ev, \rightarrow\}$ where $S$ is the set of subterms of $s$, $Ev$ is the set of events that $s$ can be observed to engage in (also called the alphabet of $s$, written $\alpha_s$), and $\rightarrow$ is the transition relation, a relation in $S \times Ev \times S$. In conventional transition system semantics [Milner 1989], [Keller 1976], something like

$$s \xrightarrow{e} s'$$

would be written to indicate that the term $s$ could perform an $e$ action and hence be transformed into the term $s'$. We, though, must have timing:

4.4.1 Timed labelled transition systems

A timed labelled transition system semantics is much like an untimed one; we shall adopt almost the same course, writing

$$s \xrightarrow{(l, e)@t'} s'$$

to indicate that the term $s$, beginning at time $t \in \mathbb{R}$ can be transformed into the term $s'$ which is incapable of any action until time $t'$. (It seems more natural to use $t \in \mathbb{R}$ instead of $r \in \mathbb{R}$ here, even though it is not uniform notation.) This transformation is accompanied by an occurrence of the event $e$ at time $t'$, and this occurrence carries the label $l$. As before we assume that the duration of events is predefined, and that some function from events $E$ to time, Time, $\Delta : E \rightarrow \text{Time}$ exists to give these durations of events. Since all events must occupy some finite interval of time, we shall require that

$$t \leq t' < (t' + \Delta(e)) \leq t'$$

(This proviso is the weakest one that makes sense. For other languages much more restricting inequalities might hold; if we were dealing with timed CSP, for instance, we would have $t = t' = t' - \delta$, where $\delta$ is the timed CSP system delay constant.)
DEFINITION 4.1 – Timed labelled transition systems

A timed labelled transition system is a triple \((S, LEv, \rightarrow)\) where

(i) \(S\) is a set of states,

(ii) \(LEv\) is a set of labelled events with durations, and

(iii) \(\rightarrow\) is a relation over \((S \times Time) \times (LEv \times Time) \times (S \times Time)\)

This form of transition system will be used to give an operational semantics to IPA. Note, however, that it is not limited to this application. TLTSs (and the ATLTSs introduced below) are suitable for use in giving operational semantics to a wide variety of timed concurrency theories, particularly timed process algebras such as timed CSP or timed CCS. Another use would be to use the extends predicate of the last chapter to construct a timed labelled transition system; the states would be observations, and \(LEv\) would be the observer’s set of labelled events.

The relation \(\rightarrow\) holds between an IPA term \(s\), a starting time \(t\), (known also as a term/time pair) a labelled event \(a\), a time \(t'\) and another term/time pair \(s', t'\) just when \(s\) starting at time \(t\) can be transformed into \(s'\) which must remain inactive until \(t'\). The labelled event \(a\) is observed to accompany this transition; it begins at time \(t'\). The labelled event \(a\) and its starting time \(t'\) can, therefore, be seen as a label of the transition as they are always observed to accompany it. When necessary, we shall refer to the starting state as \(s_0\).

How is the set \(LEv\) related to the alphabet of the process it represents? In general all that we shall require for a timed labelled transition system or one of its derivatives is that each occurrence of an event carries a label which disambiguates it from any other occurrence of any other event (or any other occurrence of the same event). In the case of the timed labelled transition system of an IPA expression, \(s\), however, we will find a little more structure on the set \(LEv\) useful; we now proceed to explain how this structure is generated from the events that \(s\) can do (its alphabet); the material concerning synchronisation in chapter two is assumed in this discussion.

Given an underlying set of events of \(s\), \(E\) say, the set of labelled events \(LEv\) is built up in quite a complex way. Firstly assume that we have some set of labels of events, \(L\). (For simplicity we require that \(L \cap E = \emptyset\).) Then \(L \times E\) will certainly be contained in \(LEv\). However, we also have to cope with the relabelling of events that happens when we enter a parallel composition. Consider the silent event \(*\) and remember that \(*\) is not a member of \(E\) or of \(L\). Then labelled events resulting from asynchronous occurrences will be of the form \(((l, *)\), \((e, *)\)), or the form \(((*, l), (*, e))\) for \(l \in L\), \(e \in E\), while synchronisations will take the form \(((l, l'), (e, e'))\).
So, any candidate for LEv would have events with structure $E \cup ((E \cup \{\ast\}) \times (E \cup \{\ast\}))$ and labels with structure $L \cup ((L \cup \{\ast\}) \times (L \cup \{\ast\}))$. There are waits in $s$ as well as "real" events; these will be represented by $\ast$, and associated transitions will carry the label $\tau$. We shall not allow waits to synchronise, (as its technically unpleasant to allow it, and not clear what it means), so for binary synchronisations

$$LEv_{IPA} = (E \cup \{\tau\} \cup ((E \cup \{\ast\} \cup \{\tau\}) \times (E \cup \{\ast\} \cup \{\tau\})) \times (L \cup ((L \cup \{\ast\}) \times (L \cup \{\ast\})))$$

(We assume that $L$ is capable of labelling waits as well as events.)

However, we have to cope with multiple parallel compositions like $(P \parallel Q) \parallel R$ as well as binary ones like $P \parallel Q$ so

$$LEv_{IPA} = \left( E \cup \{\tau\} \cup \prod_{i \in \omega} (E \cup \{\ast\} \cup \{\tau\}) \right) \times \left( L \cup \prod_{i \in \omega} (L \cup \{\ast\}) \right)$$

(The product is only potentially infinite since we demand that only a finite number of processes can be placed in parallel.)

Note that most of this complexity is due to the fact that the underlying set of events that a process can perform, which we have notated $E$, is related to the alphabet of a process $s$, $\omega s$, (the set of events that it may be seen to engage in) in quite a complex way;

$$\omega s = E \cup \prod_{i \in \omega} (E \cup \{\ast\})$$

which is in turn due to our treatment of parallel composition.

The timed labelled transition system formalism is suitable for giving a semantics to some timed systems. We want to define the particular operational semantic function for IPA. The machinery to accomplish this is developed in the next section, where the timed labelled transition system formalism is extended to allow a better treatment of parallelism. This leads us to asynchronous timed labelled transition systems or ATLTSs.

### 4.4.2 Asynchronous Timed Labelled Transition Systems

Given an IPA term $s$ and a starting time $t$, (a term/time pair) we shall want to associate with $s$ a timed labelled transition system. In the last section we saw that a timed labelled transition system was a structure $(S, LEv, \rightarrow)$. For us the set $S$ will be the set of subterms of $s$ (including $s$ itself), $LEv$ will be a set of transition labels derived from the alphabet of $s$, (but with $\tau$ built in to deal with waits), $LEv_{IPA}$, as outlined in the previous subsection, and the relation $\rightarrow$ will indicate how one term can evolve into another.
This structure is not quite enough to allow us to describe non-interleaving concurrency, so another transition relation, \( \Longrightarrow \), will be introduced.

(There have been several attempts at producing a non-interleaving (or truly concurrent) operational semantics for CCS-like languages; see, for instance [Boudol & Castellani 1989], [Degano et al. 1988], [Olderog 1987], [Goltz 1988]. Also of some relevance are the asynchronous transition systems of [Bednarczyk 1987], [Kwiatkowska 1988] and [Shields 1990] and other variants on transition systems due to [Stark 1989] and [Gaifman 1989]. We shall adopt the idea due to [Degano et al. 1987] of decomposing a parallel composition into a set of subprocesses, known there as grapes.)

The purpose of the transition relation \( \Longrightarrow \) is to allow a pair of term/time pairs to evolve independently of one another. A parallel composition \( P \parallel Q \) will give rise to a pair of derived terms each of which will proceed independently (unless they synchronise).

Once both of them have finished, this pair of terms can be reduced back into a single term. This accords with the intuition that in the process \( P \parallel Q \) the subprocesses \( P \) and \( Q \) proceed independently (except for synchronisations), and the composite is over once both \( P \) and \( Q \) have terminated. A suitable transition relation will be, therefore

\[
\Longrightarrow \subseteq (S \times \text{Time})^2 \times (\text{LEVPA} \times \text{Time}) \times (S \times \text{Time})^2
\]

In general, for any parallel composition operator \( \parallel : S^n \rightarrow S \) (rather than just the conventional composition \( \parallel : S^2 \rightarrow S \) we have had thus far) we will need a transition relation

\[
\xrightarrow{\ast} \subseteq (S \times \text{Time})^n \times (\text{LEVPA} \times \text{Time}) \times (S \times \text{Time})^n
\]

where, if \( P \parallel Q \neq Q \parallel P \) the order of tupling is important.

Parallel composition can then be modelled using \( \xrightarrow{\ast} \) transitions between tuples of term/time pairs. A labelled transition system endowed with \( \xrightarrow{\ast} \) will be called an asynchronous timed labelled transition system.

**Definition 4.2**—Asynchronous timed labelled transition systems

An asynchronous timed labelled transition system (or ATLTS) is a triple \((S, \text{LEV}, \xrightarrow{\ast})\) with

(i) \( S \) is a (countable) set of states, with a distinguished starting state \( s_0 \),

(ii) \( \text{LEV} \) is a (countable) set of labelled events, including the reserved symbol \( \tau \), and

(iii) \( \xrightarrow{\ast} \) is a relation on \( \wp(S \times \text{Time}) \times (\text{LEV} \times \text{Time}) \times \wp(S \times \text{Time}) \)

This relation holds between a set of term/time pairs, a labelled event/time pair and a set of term/time pairs.
(We assume also the existence of the duration function \( \Delta : \text{LEV} \rightarrow \text{Time} \) and label oracle as outlined above.) Transitions labelled \( \tau \) will often perform technical functions as well as representing waits.

The transition relation must respect timing;

\[
\Rightarrow \quad \langle (s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n) \rangle \xrightarrow{\tau} \langle (s'_1, t'_1), (s'_2, t'_2), \ldots, (s'_n, t'_n) \rangle \\
\Rightarrow \quad \exists i. t_i \leq t'' \land \forall i. t_i \leq t'' + \Delta(f) \land \forall i. t'' \leq t_i \land \exists i. t'' + \Delta(f) \leq t_i
\]

We can now go on to describe the operational semantics of IPA in terms of our asynchronous timed labelled transition systems. In this section the set \( S \) will always be finite. Recall that elements of \( S \) will consist of IPA terms, and that a transition will hold between one term and another just when the first can perform the event in the label of the transition and be transformed into the second.

In order to give the semantics of IPA we shall only need two special cases of the transition relation \( \rightarrow \), the cases

\[
\rightarrow : (S \times \text{Time}) \times (\text{LEV}_{\text{IPA}} \times \text{Time}) \times (S \times \text{Time})
\]

discussed above.

4.4.3 The asynchronous timed labelled transition system of IPA

The transitions for the base level terms \( e \) and \( \text{wait } t \) are straightforward (although we have to be careful that \( \text{skip} \) has no transitions in order to forbid meaningless livelock);

\[
e \xrightarrow{(l,e)@t} (\text{skip}) (t + \Delta(e)) \quad (\text{wait } t) \xrightarrow{(l,t)@t} (\text{skip}) (t + t')
\]

The label \( l \) is assumed to be generated from the event by some oracular means; we pass the oracle the event and it passes back a unique label for it – it is up to the oracle to keep track of what has happened so that it can ensure that it is issuing unique labels.

The interpretation of \( \text{stop} \) as a deadlocked process is usually reflected by the lack of a transition coming out of it. However, consider \( \text{skip} \); a moment's reflection will indicate that \( \text{skip} \) cannot have any transitions, as the only sensible one would be \( \text{skip } t \xrightarrow{(l,t)@t} \text{skip } t \), which gives us the possibility of livelock. Hence, to differentiate \( \text{stop} \) from \( \text{skip} \) we must introduce the rule

\[
\text{stop } t \xrightarrow{(l,t)@t} \text{stop } (t + t')
\]

for any \( t' > \epsilon \).
This, if you like, indicates that the ‘on’ light of stop is always on, but the condition \( t' > \varepsilon \) forbids us from making an infinite number of observations of this fact in a finite time, and hence of introducing a form of livelock. (Think of \( \varepsilon \) as the time it takes to make an observation.) Transition labels involving \( \tau \) are only carried by waits, or transitions that fulfil a technical rather than a computational function. The symbol \( * \) will be reserved for parallel compositions. The occurrence of a \( \tau \)-labelled transition can be thought of indicating that the ‘on’ light of the process is still shining.

In the following description \( f \) (and derived variables like \( f_i \)) will be assumed to range over transition labels, i.e. over the set \( LEv_{IPA} \) defined in subsection 4.4.1.

The most straightforward operator is sequential combination; here two transition rules suffice, one to deal with the first process in the combination, and one to eliminate skip;

\[
\begin{align*}
& s_1 t_1 \xrightarrow{f_1@t'} s'_1 t'_1 \\
& (s_1; s_2) \xrightarrow{f_1@t'} (s'_1; s_2) t'_1
\end{align*}
\]

Choice can be dealt with using the usual operational technique; two mutually exclusive transition rules are enabled by \( s_1 + s_2 \), one for the choice of \( s_1 \) and one for the choice of \( s_2 \). Our paradigm of \( P + Q \) meaning choose between \( P \) and \( Q \) nondeterministically is dealt with thus;

\[
\begin{align*}
& s_1 t \xrightarrow{f@t'} s'_1 t'_1 \\
& (s_1 + s_2) t \xrightarrow{f@t'} s'_1 t'_1
\end{align*}
\]

and thus

\[
\begin{align*}
& s_1 t \xrightarrow{f@t'} s'_1 t'_1 \\
& (s_2 + s_1) t \xrightarrow{f@t'} s'_1 t'_1
\end{align*}
\]

This choice of rule for \( +_+ \) automatically ensures that \( \text{skip} \) is a zero of choice. (In order to have an operational treatment that was identical to the denotational one, then, we would have to deal with choice differently, since the denotational semantics does not give \( P + \text{skip} = P \).)

Parallel composition gives rise to the most complicated set of transition rules of all. As mentioned above, we shall decompose a parallel composition into a set of terms that will be allowed to proceed independently, via the \( \Longrightarrow \) transition relation.

This set will be written with continental quotes, « and », rather than { and }, since the order of elements in it is (superficially) important: remember that asynchronous events are represented as synchronisations with the silent event of the other partner in \( _1_1 \), so we need the order to ensure that we get the right partner. This order does not lead to any observable difference
between the processes $P \parallel Q$ and $Q \parallel P$, though, so it is merely technical. Where necessary we carry the synchronisation relation $S$ around, writing $s««_»$. 

We have, then, a transition to get us into the grapes;

$$ (s_1 \parallel s_2) t \xrightarrow{(l_{1l} t)_{@t}} s«(s_1 t), (s_2 t)$$

Notice that this rule, (and the one eliminating « », together) automatically ensure that skip is a zero of $\parallel$. 

Now we introduce genuine parallel execution. There are three separate things that can happen as a parallel composition evolves; either an asynchronous event from one component occurs, or an asynchronous event from the other, or a synchronisation. 

The first two cases can be dealt with so:

$$ s_1 t f_i \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1}$$

$$ s\ «(s_1 t), (s_2 t) \xrightarrow{(f_i, *) \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1} \xrightarrow{\text{skip}}} (s_2 t)$$

and so:

$$ s_1 t f_i \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1}$$

$$ s\ «(s_2 t), (s_1 t) \xrightarrow{(f_i, *) \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1} \xrightarrow{\text{skip}}} (s_2 t)$$

Synchronisation relies on the times being right and both events being in the synchronisation set:

$$ s_1 t f_i \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1} \& s_2 t f_j \xrightarrow{t_{l_{1l}}} s_{i_2} t_{i_2} \& [t_{i_1} + \Delta(f_i)] \cap [t_{i_2} + \Delta(f_j)] \neq \emptyset \& (f_i, f_j) \in S$$

$$ s\ «(s_1 t), (s_2 t) \xrightarrow{(f_i, f_j) \xrightarrow{t_{l_{1l}}} s_{i_1} t_{i_1} \xrightarrow{\text{skip}}} (s_2 t)$$

where $t^* = \min(t_{i_1}, t_{i_2})$

So, we decompose a parallel composition into a set of terms that will be allowed to evolve independently. Our definition of a transition involving a synchronisation is slightly different from the usual one due to the nature of synchronisation in IPA (see above); we can observe the composite $(f_i, f_j)$ resulting from the synchronisation of the events $f_i$ and $f_j$ if their times overlap (the clause $[t_{i_1} + \Delta(f_i)] \cap [t_{i_2} + \Delta(f_j)] \neq \emptyset$) and if they are 'meant' to synchronise, that is, the event $(f_i, f_j)$ is in the synchronisation set (the clause $(f_i, f_j) \in S$)).

The duration of a compound event is the union of their durations. Without loss of generality, suppose $t_{i_1} < t_{i_2}$. Then $\Delta(f_i, f_j) = \max\{t_{i_1} + \Delta(f_i), t_{i_2} + \Delta(f_j)\} - t_{i_1}$. 


When both components of the composition have terminated, we return to a single locus of control:

\[ \text{skip } t_1, (\text{skip } t_2) \xrightarrow{(t_1, t_2) \in \tau} \text{skip } \tau \]

where \( \tau = \max(t_1, t_2) \)

The order in which we are allowed to fire events in a parallel composition respects the branching order \( < \) (and, hence, causality), not the linear order \( < \), so sometimes we can allow one parallel stream of execution to evolve first and then deal with another.

The choice we are allowed to make in deciding which transition to fire corresponds precisely to the different interleavings possible under an interleaving semantics.

Notice that our operational semantics, in the atomic rules, has assumed an implicit quantification over time. It does not have to be that way; if we had an event fastidious that could only happen at certain times, say during some set \( \text{allowed}(\text{fastidious}) \subseteq R \), its introduction rule would be different:

\[ t \in \text{allowed}(\text{fastidious}). \]

\[ e t. (l, \text{fastidious}) \xrightarrow{t} (\text{skip}) (t + \Delta(e)) \]

By this means we can reason about events with limited firing times, or events whose times are otherwise constrained.

It should be mentioned that a main topic we shall not tackle is observational equivalence. While there does not seem to be any obvious problem in using our transition system to define an observational equivalence (in the style of bisimulation equivalence say), it seems more natural to concentrate on equivalences of implementational structures, as considered in chapter three.

For convenience the transition rules are now gathered up and stated together. Here we shall adopt the overloaded \( \times \), which stands for both the conventional

\[ \rightarrow: (S \times \text{Time}) \times (\text{LEVPA} \times \text{Time}) \times (S \times \text{Time}) \]

and the grape

\[ \Rightarrow: (S \times \text{Time})^2 \times (\text{LEVPA} \times \text{Time}) \times (S \times \text{Time})^2 \]

This has the merit of allowing us to write \( \times \rightarrow \) in some premises, which matches all suitable transitions, not just ones involving \( \rightarrow \), as was the case above.
Atomic actions

\[ e \times (\text{skip}) \xrightarrow{t} (\text{skip})(t + \Delta(e)) \]

\[ (\text{wait} t') \times (\text{skip}) \xrightarrow{t'} (\text{skip})(t + t') \]

for any \( t' > \epsilon \)

Sequential Composition

\[ s_1 t_1 \times f_1(t) \xrightarrow{t'} s_1 t_1' \]

\[ (s_1 ; s_2) t_1 \times f_1(t) \xrightarrow{t'} (s_1 ; s_2) t_1' \]

Choice

\[ s_1 t_1 \times f_1(t) \xrightarrow{t'} s_1 t_1' \]

\[ (s_1 + s_2) t_1 \times f_1(t) \xrightarrow{t'} (s_1 + s_2) t_1' \]

Parallelism

Introduction of parallelism

\[ (s_1 || s_2) t \times f(t) \xrightarrow{t'} s\langle(s_1 t), (s_2 t)\rangle \]

Asynchrony

\[ s\langle(s_1 t), (s_2 t')\rangle \times f(t') \xrightarrow{t'} s\langle(s_1 t'), (s_2 t')\rangle \]

Synchronisation

\[ s\langle(s_1 t), (s_2 t')\rangle \times (f_1, f_2) \xrightarrow{t'} s\langle(s_1 t'), (s_2 t')\rangle \]

where \( t^* = \min(t_1^*, t_2^*) \)

Removal of parallelism

\[ s\langle\text{skip} t_1), (\text{skip} t_2)\rangle \times f(t) \xrightarrow{t} \text{skip max}(t_1, t_2) \]

where \( t' = \max(t_1, t_2) \)

DEFINITION 4.3 – The Operational Semantics of IPA
4.5 Aside - Petri nets and Process Algebra

The purpose of this aside is to discuss the net-theoretic interpretation of various process-algebraic constructs, including those found in IPA.

Petri nets are very well known models of concurrency. An extensive bibliography concerning them can be found in [Best & Fernandez 1988]; [Reisig 1985] is a good introduction. We shall concentrate on a subclass of Petri nets known as C/E nets, both because they are simpler than general Petri nets, and because they are more suited to our purpose of discussing asynchronous implementational or behavioural models. Section 4.4 in [Best & Fernandez 1988] gives an extensive discussion of the special properties of C/E nets (referred to there as 1-safe nets).

DEFINITION 4.5 - Condition/event nets

A (finite) Petri net $N$ is a triple $(S, T; F)$ where

(i) $S$ is a (finite) set of places,
(ii) $T$ is a (finite) set of transitions, such that $S \cap T = \emptyset$ and
(iii) $F$ is a flow relation, that is a relation over $(S \times T) \cup (T \times S)$.

We write $x^* = F(x)$ for the preset of $x \in S \cup T$ and $x^* = F^{-1}(x)$ for the postset of $x$. This notation generalises to sets of places or transitions; we write $^X$ for $(^-X \cup ^Y)$ with $X \subseteq S$ or $X \subseteq T$.

A net is called pure (or acyclic) if it does not contain any self-loops, that is pairs $(s, t)$, with $s \in S$, $t \in T$ such that $(s, t) \in F$ and $(t, s) \in F$. It is advantageous to interpret the flow relation $F$ as a function $F: (S \cup T) \times (T \cup S) \to \{0, 1\}$ with $F(x, y) = 1$ just when $F$ holds between $x$ and $y$.

The dynamic behaviour of a Petri net is explained by assuming a set of tokens. A marking is a function $M: S \to \mathbb{N}$ that indicates how many tokens are on each place. The initial distribution of tokens around the places of a net is called the initial marking, $M_0$. A transition can fire if it has at least one token on its input places, which results in the token appearing at an output place of the transition, so the initial marking is gradually transformed. Formally, a transition $t \in T$ is enabled in a marking $M$ (written $M$ enables $t$) iff $\forall s \in S . F(s, t) \leq M(s)$. If the transition $t$ transforms a marking $M$ to a marking $M'$ then we write $M[t>M']$. The marking $M'$ can be deduced from $M$ and $t$ by the following rule:

$$M[t>M'] \iff M \text{ enables } t \text{ & } \forall s \in S . M'(s) = M(s) - F(s, t) + F(t, s)$$

In future we shall specify nets together with the initial markings, writing $\Sigma = (S, T; F, M_0)$ for the net $(S, T; F)$ together with the initial marking $M_0$. A marking $M$ is said to be accessi-
ble or reachable from $M_0$ if there is some sequence of transitions $t_0, t_1, t_2, \ldots, t_n$ and markings $M_1, M_2, \ldots, M_n$ such that $M_0[t_0>M_1[t_1>M_2[\ldots>M_n[t_n>M$.

Two transitions $t_i$ and $t_j$ are concurrently enabled if $\forall s \in S . F(s, t_i) + F(s, t_j) \leq M(s)$.

A condition/event net (or C/E net) is a pure Petri net $\Sigma$ having at most one token per place in any reachable marking. (Often the set of places $S$ is also referred to as a set of conditions (sometimes notated $B$, from the German Bedingung), while the set of transitions $T$ is sometimes called a set of events, notated $E$.) In the case of C/E nets, markings can just be thought of as the set of conditions that have tokens on them. Such a subset $c \subseteq S$ is known as a case. For $e \in T$, $e$ is said to be $c$-enabled if $e \subseteq c$ and $e^* \cap c = \emptyset$, so $c$ can allow $e$ to occur, but any result of the firing of $c$ does not interfere with $e$. In this case the transition rule can be simplified to

$$c[t>c' \Leftrightarrow c^* = (c - e) \cup e^*$$

where $-$ denotes set-theoretic subtraction.

The type of the flow relation is then $\longrightarrow : B \times E \times B$.

An occurrence net is a pure Petri net $\Sigma$ where no state comes from or goes to more than one transition. That is, $\forall s \in S . |s^t| \leq 1$ & $|s^t| \leq 1$.

(We will not discuss here the timed nets of [Ramchandan 1974], nor connect full ATLTs with nets. Eventually we hope to understand the connection between ATLTs and timed net models, but such a connection will probably rely on the interpretation of nets as symmetric monoidal categories; discussing that here would take us too far out of our way.)

Nets can be represented diagrammatically by showing places as boxes, transitions as circles and the flow relation as an arrow connecting the relevant circles and boxes. Tokens are represented by solid smaller circles at the appropriate place. The first example (overleaf) should make the definitions clearer. Note that this net is not an occurrence net.
EXAMPLE – A simple C/E net

The following C/E net should make the definitions clearer;

![Diagram of a C/E net]

Here we have $S = \{1, 2, 3, 4, 5\}$, $E = \{(a, b), (a, c), (a, d), (b, 1), (c, 2), (c, 3), (c, 4), (d, 2), (d, 5), (b, 1)\}$. The event $a$ is $(1, 2)$-enabled, while $d$ is not thus enabled, but it is $(4)$-enabled. The initial marking is $(1, 2)$ and the following pre- and post-sets can be seen; $\bullet b = \{5\}, b^* = \{1\}, 4^* = \{c, d\}, \bullet c = \{3, 4\}$ etc.

(Here and in future Courier-Oblique will be used for places and Courier for transitions.)

From the initial marking only $\{a\}$ is enabled. Various markings can occur as a result of this transition; firing the transition $a$ gives rise to the marking $M(1) = M(2) = M(5) = 0, M(3) = M(4) = 1$. After this, either $c$ or $d$ may fire, (giving rise to $M(1) = M(2) = M(3) = M(4) = 0, M(5) = 1$ and $M(1) = M(4) = 0, M(2) = M(3) = M(5) = 1$ respectively). In both cases we can fire $b$ resulting in the marking $M(2) = M(3) = M(4) = M(5) = 0, M(1) = 1$ in the former case and $M(4) = M(5) = 0, M(1) = M(2) = M(3) = 1$ in the latter. The former marking is deadlocked, but the latter can then proceed to fire another $a$, demonstrating its non-C/E character.
EXAMPLE — The interpretation of process-algebraic operators in nets

This example briefly discusses net representations of common constructions in process algebras. We will give a series of transitions involving fragments of IPA, in the style of the previous section, together with a net representation. The marking before and after the transition will be shown.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Net before firing</th>
<th>Net after firing</th>
</tr>
</thead>
<tbody>
<tr>
<td>e [\longrightarrow] skip</td>
<td><img src="image" alt="Net before firing" /></td>
<td><img src="image" alt="Net after firing" /></td>
</tr>
</tbody>
</table>

Figure 4.5 — The net corresponding to some simple process-algebraic operators

So the occurrence of the event e, transforming the process e into the process skip can be seen as analogous to the firing of a token from a place e to a place skip via a transition e.

Transitions resulting from other rules can be similarly interpreted, giving an intuitive meaning to process-algebraic operators in terms of nets. (See overleaf.)

There is a fundamental difference between the parallelism evident in process algebras and in nets. In a process algebra we write P \[\parallel\] Q for a single process consisting of the processes P and Q running in parallel — we think of a single entity which starts both P and Q off and which can be said to be over. In net terms, parallelism means complete independence; unconnected nets bear no relationship whatsoever to each other. The difference may be a slight and philosophical one, but it affects how we choose to compile parallelism. ([Olderog 1989], which inspired this example, takes a different course and ignores this difference.)
<table>
<thead>
<tr>
<th>Transition</th>
<th>Net before firing</th>
<th>Net after firing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p + p' \xrightarrow{a} p$</td>
<td><img src="image1" alt="Net before firing" /></td>
<td><img src="image2" alt="Net after firing" /></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p + p' \xrightarrow{b} p'$</td>
<td><img src="image3" alt="Net before firing" /></td>
<td><img src="image4" alt="Net after firing" /></td>
</tr>
<tr>
<td>$p \parallel p' \xrightarrow{\tau} \langle p, p' \rangle$</td>
<td><img src="image5" alt="Net before firing" /></td>
<td><img src="image6" alt="Net after firing" /></td>
</tr>
<tr>
<td>$\langle p, p' \rangle \xrightarrow{f} \langle q, p' \rangle$</td>
<td><img src="image7" alt="Net before firing" /></td>
<td><img src="image8" alt="Net after firing" /></td>
</tr>
<tr>
<td>$\langle p, p' \rangle \xrightarrow{f} \langle q, q' \rangle$</td>
<td><img src="image9" alt="Net before firing" /></td>
<td><img src="image10" alt="Net after firing" /></td>
</tr>
</tbody>
</table>
The examples on the previous page related to the introduction and evolution of parallelism and to choice. We now need to see how to eliminate parallelism and how to deal with sequentiality:

\[ \text{Transition} \quad \quad \text{Net before firing} \quad \quad \text{Net after firing} \]

\[
\begin{array}{c}
\text{«skip,skip» } \xrightarrow{\tau} \text{skip} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Figure 4.6 – The net corresponding to some simple process-algebraic operators} \\
\end{array}
\]

For more examples of the use of Petri nets in modelling process algebras the reader is referred to [Degano et al. 1987], [Degano et al. 1988], [Goltz 1988], [Goltz & Mycroft 1984], [Goltz & Reisig 1983], [Olderog 1987], [Olderog 1989], [Shields 1987] and to [van Glabbeek & Vaandrager 1987]. A comprehensive survey, discussion and series of constructions is given in [Taubner 1989]. We must press on to investigate the relationship between our denotational and axiomatic semantics.
4.6 The relationship between the semantics of IPA

In this section we will study the relationship between the semantics of IPA, that is, between the operator \( C \) and the laws (axiomatic semantics) given in section 4.2. Some results (from [Hennessy 1988]) will be of use in proving results about the equivalence of these semantics, and we will begin by providing a framework for this proof. Recursion will not be considered here:

The syntax of finite IPA can be thought of as determined by a set of constants (functions of arity 0) and functions acting upon them and other functions. We shall refer to this set as IPA; it consists of the following functions:

- **of arity 0**: \( e, \text{wait} t \) for \( e \in E, t \in \mathbb{R}^* \)
- **of arity 2**: \( _{-}, _{-} \ll, _{-} + _{-} \)

Any expressions in IPA is a member of the term algebra of IPA, \( T_{IPA} \). This is just the application-closed set of these functions. Thus, for instance, the expression

\[(a \ll b); (c \ll d)\]

can be decomposed into the application of the function \( _{-} \ll _{-} \) onto \( a \) and \( b \), and onto \( c \) and \( d \), together with an application of the function \( _{-} + _{-} \) onto \( (a \ll b) \) and \( (c \ll d) \).

We shall use \( f^k \) for a function \( f \) of arity \( k \).

**DEFINITION 4.8** – The term algebra over a signature

The term algebra, \( T_{IPA} \) over the set of functions (or, more properly, signature) IPA is thus the least set of strings which satisfies

(i) if \( f^0 \in IPA \) then \( f^0 \in T_{IPA} \), and

(ii) if \( f^n \in IPA \) then \( f^n(s_1, ..., s_n) \in T_{IPA} \), whenever \( s_1, ..., s_n \in T_{IPA} \), for \( n > 0 \).

Thus by rule (i) \( a, b, c \) and \( d \) are in \( T_{IPA} \), and, hence, by rule (ii), the IPA terms \( (a \ll b) \), \( (c \ll d) \) and \( (a \ll b); (c \ll d) \) are.

Some authors call our signatures (i.e. sets of functions with explicit indication of domain and range) many-sorted algebras. The BNF for the syntax of IPA defines the functions of IPA and their types adequately. We shall use as little of the theory of many-sorted algebras as necessary; see [ADJ 1978] for details.

The set of terms \( T_\Sigma \) should be thought of as a language; its semantics is an algebra over that language.
One answer is a signature algebra, or $\Sigma$-algebra. This comprises a signature, and a meaning set, $A$, of all possible meanings of terms generated from $\Sigma$, and is usually written $<A, \Sigma>$. Variables that range over $\Sigma$-algebras will take the names of the underlying meaning sets and underscores, so we shall write $\mathcal{A} = <A, \Sigma>$.

Any semantics for IPA can be thought of as an IPA-algebra, $<A, IPA>$ for some set of meanings $A$. The set $A$ is also referred to as the carrier.

Notice that the term algebra, $T_{IPA}$, is also an IPA-algebra; it is $<IPA^*, IPA>$, the set $A$ being just the set of all terms. This corresponds to the semantics which assigns the same meaning to terms if and only if they are syntactically identical, and this meaning is just the term itself.

**DEFINITION 4.9 – Homomorphisms of signature algebras**

Two signature algebras, say $\mathcal{A} = <A, \Sigma>$ and $\mathcal{B} = <B, \Sigma>$ can be related by giving a function from one carrying set to the other which preserves the structure induced by the signature. Such a function is called a signature algebra homomorphism or just a $\Sigma$-homomorphism. If $h : A \rightarrow B$ then $h$ is a $\Sigma$-homomorphism if

$$\forall f^k \in \Sigma. h(f^k(a_1, ..., a_k)) = f^k(h(a_1), ..., h(a_k))$$

that is, if it preserves the structure of the meanings induced by the functions. This definition of morphism gives a category of $\Sigma$-algebras over a given signature $\Sigma$; [Bednarczyk 1987] has the details.

Notice that the denotational semantics $C$ associates an I.E.S. with a term/time pair and the axiomatic semantics defines a relation between terms, so we have, denoting by $\mathcal{A}$ the axiomatic semantics:

$$C : \text{Proc} \rightarrow \text{IES}$$
$$\mathcal{A} : \text{Proc} \times \text{Proc}$$

Our aim in this section is to solve the full abstraction problem, that is, to show that all the semantics are equivalent. We shall do this by showing that the, if two term/time pairs can be proved to be equal within the laws of the axiomatic semantics, then they will be assigned behaviourally identical I.E.S.s as their denotations.

The full abstraction problem, then, presupposes a concept of behavioural equivalence, and a notion of how to discuss terms that “can be shown to be the same using the axiomatic semantics.” Usually this notion is merely equality (of meanings), but for us it will be one of the equivalences of chapter three. (The reasons for this decision are discussed below.)
We have seen that we can encapsulate a semantics algebraically as a \( \Sigma \)-algebra; our denotational semantics, for instance, is a particular \( \Sigma \)-algebra over \( \langle \text{IES}, \Sigma \rangle \). This insight enables us to relate semantics; we could, for instance, if we had some event structure semantics for IPA, \( \langle \text{ES}, \Sigma \rangle \), relate this to the one given by the denotational semantics, \( \langle \text{IES}, \Sigma \rangle \), by giving a \( \Sigma \)-homomorphism \( h : \text{IES} \to \text{ES} \).

The two semantics would then be isomorphic (in the algebraic sense) if we could provide two \( \Sigma \)-homomorphisms \( h : \text{IES} \to \text{ES} \) and \( i : \text{ES} \to \text{IES} \); we would then have (using semantic brackets like \( [\cdot] \) to extract meaning)
\[
[sl]_{\text{IES}} = [s']_{\text{IES}} \iff [s]_{\text{ES}} = [s']_{\text{ES}}
\]

(It should be mentioned en passant that we are very unlikely to be able to do this because it is hard to imagine how two equivalent event structures could always give rise to two equivalent I.E.S. (as there is considerable freedom in generating the timing information).)

**DEFINITION 4.10 – Initiality in classes of \( \Sigma \)-algebras**

For any class of \( \Sigma \)-algebras, \( C \), an algebra \( I \in C \) is initial in \( C \) if for all \( J \in C \) there is a unique \( \Sigma \)-homomorphism from \( I \) to \( J \).

The concept of initiality, like that of an initial object in category theory, is something like that of a canonical object. An object, \( I \), is the canonical or natural object to represent a class if it is in the class and it has the least structure necessary to be there. Since morphisms preserve structure, and since the object with the least structure is the one which shares its structure with everything else, the object with the least structure will have morphism going from it to every other object, and thus will be initial. (Obviously if there are two or more initial objects it does not matter which one you pick as there are homomorphisms from one to the other and back; initial objects are unique up to isomorphism. To see this note that if \( I \) and \( J \) are initial then there are maps \( f : I \to J \) and \( g : J \to I \) and hence, by composition, maps \( f \cdot g : I \to I \) and \( g \cdot f : J \to J \). But \( \text{id}_I \) is a map, and since \( I \) is initial it must be the unique one from \( I \) to itself. Hence \( g \cdot f = \text{id}_I \) and similarly \( f \cdot g = \text{id}_J \). Thus \( I \) and \( J \) are isomorphic.)

**THEOREM – Initiality**

The term algebra, \( T_\Sigma \), is initial in the class of all \( \Sigma \)-algebras. Proof - [Hennessy 1988].

Our aim is to compare the denotational and axiomatic semantics of IPA, to show that they are compatible. First we shall have to prove that
(i) the denotational semantics agrees with the axiomatic semantics, that is, any equality induced by the axiomatic semantics must be respected by the denotational semantics, and

(ii) the axiomatic semantics agrees with the denotational semantics, that is, expressions equal under interpretation in IES must be provably equal under the axiomatic semantics. This means that of all the possible respectful denotational semantics, ours is a 'natural' one; it makes only as many identifications as it must. This notion will now be made more precise.

**DEFINITION 4.11 — A deductive system for IPA**

Let $E$ be the set of equations

$$\begin{align*}
P \cdot Q = Q \cdot P & \quad \text{where } (b, a) \in S^o \iff (a, b) \in S \\
P \cdot (Q \cdot R) = (P \cdot Q) \cdot R & \quad \text{where } (S_1, S_2) \leftrightarrow (S_3, S_4) \\
P \cdot \text{skip} = \text{skip} \cdot P & = P \\
(P ; Q ; R) = (P ; Q ; R) & \quad \text{where } t \leq \min(\lambda(P)) \\
P + Q = Q + P & \\
P + P = P & \\
(P + Q) + R = (P + Q) + R & \\
(P + Q) \cdot R = (P ; R) + (Q ; R) & \\
(P + Q) \cdot R = (P \cdot R) + (Q \cdot R) & \\
\text{wait } t ; \text{wait } t' = \text{wait } (t + t')
\end{align*}$$

\[ (4.1) \]
\[ (4.2) \]
\[ (4.3) \]
\[ (4.4) \]
\[ (4.5) \]
\[ (4.6) \]
\[ (4.7) \]
\[ (4.8) \]
\[ (4.9) \]
\[ (4.10) \]
\[ (4.11) \]

Together with the deduction rules§

$$\begin{align*}
\hline
s = s & \\
\hline
s = s' & s' = s' \\
\hline
s = s' & s' = s \\
\hline
s = s' & s = s' \\
\hline
\end{align*}$$

This set of equations will be referred to as the *deductive system* for $E$ or $\text{DED}(E)$. We write $s =_E s'$ if it is possible to deduce $s = s'$ in $\text{DED}(E)$. The class of all IPA-algebras which satisfy $E$ will be written $C(E)$. Thus (i) above reduces to proving that $\langle \text{IES, IPA} \rangle$ is in $C(E)$, and (ii) to showing that it is initial in this class.

§ — We should really have dealt with substitution here as well, to ensure that we can make deductions about expressions containing variables, but this is all standard material, and it would cloud the results.
(The law $P_{op dull} P = P$, which we previously allowed, only holds if $(e, e)$ is taken to be observationally equivalent to $e$, which is not, in general, so.)

A deductive system $\text{DED}(E)$ is said to be sound with respect to a relation $R$ if $s \equiv_{E} s' \Rightarrow (s, s') \in R$, and complete with respect to $R$ if $(s, s') \in R \Rightarrow s \equiv_{E} s'$. A denotational semantics $A$ can be thought of as giving a relation over terms, with $(s, s') \in R \Leftrightarrow |s_A| = |s'_A|$. Thus $A$ is sound if it is in $C(E)$ and complete if it is initial in $C(E)$.

**THEOREM** — Soundness and completeness of $\text{DED}(E)$

The proof system $\text{DED}(E)$ is sound and complete with respect to $\equiv_{E}$.

*Proof* — [Hennessy 1988].

The proof of (i) thus has the satisfying bonus of providing a sound and complete proof system for IPA, $\text{DED}(E)$, as a by-product. (We keep writing $\text{DED}(E)$ rather than just $\text{DED}$ to indicate that our results hold for the deductive system based on any set of equations like $E$, not just on the particular ones given.)

**DEFINITION 4.12** — $\Sigma$-congruence

A relation, $R$, is a $\Sigma$-congruence over $A = \langle A, \Sigma \rangle$ if

(i) $R$ is an equivalence relation over $A$, and

(ii) $R$ respects the structure induced by $\Sigma$: $\forall f^k \in \Sigma. (a_1, a_1), \ldots, (a_k, a_k') \in R \Rightarrow (f^k(a_1, \ldots, a_k), f^k(a_1', \ldots, a_k')) \in R$.

**LEMMA** — The congruence $\equiv_{E}$ is a $\Sigma$-congruence. *Proof* — [Hennessy 1988].

**DEFINITION 4.13** — Factored algebras

Suppose that we have a $\Sigma$-algebra $A = \langle A, \Sigma \rangle$ and a $\Sigma$-congruence, $R$, over it. Let $A/R$ be the set of equivalence classes of $A$ induced by $R$. The equivalence class of a single element, as we saw in chapter three, is $[a]_R = \{a' \mid (a, a') \in R\}$, so $A/R = \{[a]_R \mid a \in A\}$. Define functions over equivalence classes thus $f^k_{A/R} ([a_1]_R, \ldots, [a_k]_R) = [f^k(a_1, \ldots, a_k)]_R$.

**THEOREM** — The factored $\Sigma$-algebra $T_{\Sigma}/\equiv_{E}$ is initial in $C(E)$. *Proof* — [Hennessy 1988].

\[ \Phi \]

This theorem provides an example of the denotational semantics that satisfies any axiomatic semantics $E$ and is 'natural.' It is the term algebra with the equivalence induced by the laws factored out. Unfortunately this is not a particularly useful form to have it in. It
would be nice to find a IPA-algebra which was initial in C(E) (and hence a natural choice) but which was a little more informative. We see that

* The requirement (ii) amounts to proving that the denotational semantics <IES, IPA> is initial in C(E).
* We have an algebraic characterisation of the axiomatic semantics $\mathcal{A}$; it is the factored IPA-algebra $T_{IPA}/E$.

**HYPOTHESIS** – The denotational satisfies the axiomatic semantics.

The denotational semantics is in the class of $\Sigma$-algebra that satisfy the axiomatic semantics, that is $<IES, IPA> \in C(E)$ or

$$s =_E s' \Rightarrow C(s) = C(s')$$

Unfortunately this is not true; the denotational semantics has too much structure. The obvious counter example to our desired soundness result is $a + a$, because $a + a =_E a$ but, of course, $C(a + a) \neq C(a)$. The position we are in is this

$$\begin{array}{ccc}
\text{Operational semantics} & \text{Axiomatic semantics} & \text{Denotational semantics (I.E.S.)} \\
\begin{array}{c}
a \\
= \\
\end{array} \text{or} \begin{array}{c}
a \\
= \\
\end{array} & \begin{array}{c}
a + a =_E a \\
\end{array} & \begin{array}{c}
a \\
\text{or} \\
\end{array} \end{array}$$

But all is not lost. We merely want the two meanings of $s$ and $s'$ to behave the same way if $s =_E s'$, not to be the same. Hence, if we have some sort of notion of behavioural equivalence, a congruence on IES that holds when two I.E.S. have identical behaviours, say, where

$$\sim : IES \times IES$$

then the soundness result we really want is

$$s =_E s' \Rightarrow C(s) \sim C(s)$$

That means that the denotational semantics $<IES, IPA>$ will be in $C(E)$ if we are allowed to forget structure that $\sim$ doesn’t notice in constructing our $\Sigma$-homomorphisms. This amounts to showing that $<IES/\sim, IPA>$ is in $C(E)$. (We should also prove that $\sim$ is a congruence of the IPA structure.)

(The observant reader will have noticed that $C(skip)$ is undefined, and so, as $+$ is strict, $C(s + skip)$ is also undefined. This shows why we cannot let $skip$ be a proper process: --
At the moment we are safe, as `skip` is not an `Proc`, and we only want `C` and `=_{E}` to agree on `Procs`, not necessarily on `Procs \cup \{skip\}`.

However, making `skip` a process would force us to give it a denotation, which we do not want to do, as we wish to avoid infinitesimally-timed interval event structures. A theory of I.E.S. involving infinitesimal structures is beyond the scope of this work. Of course, there is an obvious denotation for `skip`; it is the I.E.S. `0`, but we do not want to make this identification yet.)

**Definition 4.14** — The observational equivalence `~`

Which congruence, of all those presented in chapter three, should be chosen here? The idea is that `~` is the most detailed *behavioural equivalence*. This means that we should not chose an equivalence that cannot be justified as a testing equivalence; we should be able to tell if `C(s) ~ C(s')` merely by observation.

The strongest testing equivalence that we can (philosophically) justify is `~_{\gamma}`, the strict timed observational equivalence. We shall choose this as the congruence as `~`, and write `~` to mean `~_{\gamma}` in future.

**Theorem** — `C` is sound with respect to `\$\$` "modulo" the observational equivalence `~_{\gamma}`.

That is, we have to prove `\(\forall s, s' \in T_{PA}. C(s) ~ C(s') \Rightarrow s =_{E} s'\)`.

**Proof** — By structural induction. As a base case consider the atomic actions `e` and `wait t` for `t > 0`. Clearly `e \neq_{E} e'` unless `e = e'`, and `wait t \neq_{E} wait t'` unless `t = t'` and furthermore for no `e` and for no `t` do we have `wait t =_{E} e` so for atomic terms `a`;

\[
\forall a, a'. a =_{E} a' \Rightarrow C(a) ~ C(a')
\]

since this only happens if `a = a'`, and certainly `C(a) ~ C(a)`.

Now we must perform the inductive step. There are cases corresponding to each of equation 4.1–11;

(i) `P s\| Q = Q s\| P`. We have `C(P s\| Q) ~ C(Q s\| P)` by construction, so this case is immediate.

(ii) `P s_{1}\| (Q s_{2}\| R) = (P s_{3}\| Q) s_{4}\| R`, where `(S_{1}, S_{2}) \leftrightarrow (S_{3}, S_{4})`. This holds by construction of `\$\$`.

(iii) `P s\| wait t = P`. Suppose `C(P s\| skip) = 3 \|_{P} 3'`. Here `3\| 3'` is empty and `3\| 3' ~_{P} 3`, so this case is immediate provided `t \leq \lambda(P)`.

(iv) `P ; skip = skip ; P = P`. This is true, if tedious to verify, by construction.
(v) \( P ; (Q ; R) = (P ; Q) ; R \). Ditto.

(vi) \( P + Q = Q + P \). This is immediate since \( C(P + Q) \sim C(Q + P) \) by construction.

(vii) \( P + P = P \). This follows from chapter three, where we noticed \( C(P + P) \sim C(P) \).

(viii) \( P + (Q + R) = (P + Q) + R \). Again, by construction.

(ix) \( (P + Q) ; R = (P ; R) + (Q ; R) \). This follows from the definition of \( E(P + Q) t \) as the union of \( E(P) t \) and \( E(Q) t \). It is worth seeing in some detail:

\[
C((P + Q) ; R) = \text{fst}(D ((P + Q) ; R) \{0\})
\]

\[
= \text{fst}(E(P + Q) \{0\} \oplus E(R) (F(P + Q) \{0\}), \{F(R) \{t_n\}, t_n \in F(P + Q) \{0\}))
\]

\[
= E(P + Q) \{0\} \oplus (E(R) (F(P + Q) \{0\}))
\]

\[
= (E(P) \{0\} + E(Q) \{0\}) \oplus (E(R) (F(P + Q) \{0\}))
\]

Take \( p = E(P) \{0\}, q = E(Q) \{0\} \) for convenience. Then, the above

\[
= (p + q) \oplus (E(R) (F(P + Q) \{0\}))
\]

\[
= (p + q) \oplus (E(R) (F(P) \{t\} \cup F(Q) \{0\}))
\]

\[
\sim (p \oplus E(R) F(P) \{0\}) + (q \oplus E(R) F(Q) \{0\})
\]

\[
= (E(P) \{0\} + E(R) F(P) \{0\}) + (E(Q) \{t\} \oplus E(R) \{t\} F(Q) \{0\})
\]

\[
= (E(P ; R) \{0\}) + (E(Q ; R) \{0\})
\]

\[
= C((P ; R) + (Q ; R) \{0\})
\]

as required.

(x) \( P + Q \lvert= R = (P \lvert= R) + (Q \lvert= R) \). Again, a tedious expansion of the definitions gives \( C((P + Q) \lvert= R) \sim C((P \lvert= R) + (Q \lvert= R)) \). The only point to note is the way the clauses \( a \in aP, b \in aQ \) in the definition of the I.E.S. synchronisation set \( T = \{(n, a), (n', b) | (a, b) \in S, n, n' \in N, a \in aP, b \in aQ\} \) ensures the same synchronisation behaviour of the LHS and RNS of the law.

(xi) wait \( t \); wait \( t' = \) wait \( t + t' \). Straightforward; both \( C(\text{wait} \ t) \) and \( C(\text{wait} \ t') \) are empty with durations \( t \) and \( t' \) respectively; \( C(\text{wait} \ t ; \text{wait} \ t') \) is thus also empty with duration \( t + t' \).

There are two ways of proving that a semantics \( A \) is initial amongst all those that satisfy a class of equations. One can either construct a homomorphism \( h \) from \( A \) to any member of the class, and prove it unique, or one can show that, for all terms \( s \) and \( s' \)

\[
\{s\} A \sim \{s'\} A \Rightarrow s =_E s'
\]
THEOREM — Completeness of the denotational semantics.

The denotational semantics is initial in the class of \( \Sigma \)-algebra that satisfy the axiomatic semantics, that is \( \langle \text{IES}, \text{IPA} \rangle \in \mathcal{C}(E) \).

\[
\mathcal{C}(s) \sim \mathcal{C}(s') \Rightarrow s =_E s'
\]

Proof: We adopt the second of the two methods given above. For convenience the subscript \( \gamma \) is dropped from \( \sim \) from now on.

Clearly \( \mathcal{C}(e) \not\sim \mathcal{C}(e') \) unless \( e = e' \), and \( \mathcal{C} \text{(wait } t_i) \not\sim \mathcal{C} \text{(wait } t_i) \) unless \( t_i = t_i' \) and furthermore \( \mathcal{C}(e) \not\sim \mathcal{C} \text{(wait } t_i) \) ever, so we have the base case.

Next note that for an action \( a, \mathcal{C}(a) \not\sim \mathcal{C}(s_1 + s_2) \) unless \( s_1 = s_2 = a \). Furthermore, we know that \( \mathcal{C}(a) \not\sim \mathcal{C}(s_1 \parallel s_2) \) unless \( a = s_1 \) and \( s_2 = \text{wait } t \) for \( t \leq \min(\lambda(a)) \). Finally \( \mathcal{C}(a) \not\sim \mathcal{C}(s_1 ; s_2) \) unless \( s_1 = \text{skip} \) or \( s_2 = \text{skip} \) or \( a = \text{wait } t'' \) \& \( s_1 = \text{wait } t \) \& \( s_1 = \text{wait } t'' \) \& \( t'' = t + t' \).

Now we have to show that \( \mathcal{C}(s_1 \oplus s_2) \sim \mathcal{C}(s_1' \oplus s_2') \Rightarrow (s_1 \oplus s_2) =_E (s_1' \oplus s_2') \) for \( \oplus, \oplus' \in \{\parallel, ;, +\} \), a straightforward double structural induction.

Consider first \( \mathcal{C}(s_1 + s_2) \). From the definition of \( \mathcal{C}(s_1 + s_2) \) as a coproduct of the meanings of \( s_1 \) and \( s_2 \) we know that \( \mathcal{C}(s_1 + s_2) \not\sim \mathcal{C}(s_1' + s_2') \) unless the sets \( \langle \mathcal{C}(s_1), \mathcal{C}(s_2) \rangle \) and \( \langle \mathcal{C}(s_1'), \mathcal{C}(s_2') \rangle \) are equal or unless \( s_2 = Q + R, s_1 = s_1 + Q \& s_2 = R \). Next note \( \mathcal{C}(s_1 + s_2) \not\sim \mathcal{C}(s_1 ; s_2) \) unless \( s_1 = \text{skip} \) and \( s_2 = s_1 + s_2 \) or \( s_2 = \text{skip} \& s_1 = s_1 + s_2 \) or \( s_1 = s_1 = s_1 \); \( s_2 = s_2 = P \& s_2 = Q \). The last case for \( + \) is \( \mathcal{C}(s_1 + s_2) \not\sim \mathcal{C}(s_1 \parallel s_2) \). This holds unless \( s_1 = P \parallel s_2, s_2 = Q \parallel s_2 \& s_1 = P + Q \) or \( s_1 = s_1 = s_1 \parallel s_2 \) or one of the \( s_1 \) is a \( \text{wait } t \) and the other is \( s_1 + s_2 \).

Next consider \( \mathcal{C}(s_1 ; s_2) \). This is even easier as \( \mathcal{C}(s_1 ; s_2) \not\sim \mathcal{C}(s_1' ; s_2') \) unless something is a \( \text{skip} \) or \( s_1 = s_1 \& s_2 = s_2 \). Furthermore \( \mathcal{C}(s_1 ; s_2) \not\sim \mathcal{C}(s_1 \parallel s_2) \) unless one of \( s_1 \) is \( \text{skip} \) or one of \( s_1 \) is a \( \text{wait } t \) and the other is \( s_1; s_2 \).

Finally consider \( \mathcal{C}(s_1 \parallel s_2) \). Again, \( \mathcal{C}(s_1 \parallel s_2) \not=E \mathcal{C}(s_1' \parallel s_2') \) unless one of the \( C \)s is \( \text{wait } t \) or the sets \( \langle \mathcal{C}(s_1), \mathcal{C}(s_2) \rangle \) and \( \langle \mathcal{C}(s_1'), \mathcal{C}(s_2') \rangle \) are equal. The only other case is the multiple composition \( P \parallel (Q \parallel s_2) = (P \parallel s_1 \parallel Q) \parallel (s_2 \parallel R) \) (where the conjugation \( (S_1, S_2) \leftrightarrow (S_3, S_4) \) holds), this follows by construction so the theorem holds.

This completes the treatment of IPA proper. In the interlude we briefly examine a technique for including external choice in IPA and for treating recursion, before returning to I.E.S.s, and in particular to their specification and refinement, in the next chapter.
References

A more comprehensive set of references for non-interleaving models of process algebras can be found in the bibliography at the end of chapter five. A discussion of the literature is given in the introduction.


Interlude – External choice and Recursion

In this interlude some suggestions are given for a treatment of external choice and recursion in IPA. A category of embeddings is discovered, and its use in giving a sheaf-theoretic model of timing is investigated.

A4.0 External Choice

The intuition behind CSP external choice || is very simple: in

\[ P = (a \rightarrow Q) || (b \rightarrow R) \]

the process \( P \) offers the environment the choice of doing \( a \) or \( b \). If the environment ‘wants’ \( a \) then it does an \( a \) too, resulting in a synchronisation and \( P \) doing \( Q \) next;

\[ P \parallel (a \rightarrow S) = a \rightarrow (Q \parallel S) \]

and similarly for \( b \);

\[ P \parallel (b \rightarrow T) = b \rightarrow (R \parallel T) \]

The natural way to include this form of choice into IPA is to introduce a new choice operator, written \( _\parallel _\parallel \), with the intention that

\[(a; Q)[F](b; R)\]

means the environment is offered \( a \) or \( b \) to synchronise with. If it synchronises with \( a \), then \( Q \) happens next, while if it synchronises with \( b \), \( R \) happens next. However, if no synchronisation takes place \( F \) occurs. This alternative is necessary since our view of a process is as an independent entity, so its progress should not be interrupted by the failure of the environment to do something. (Notice that both synchronisations cannot occur, since \( || \) is a binary operator; we are looking for \( a \) or \( b \) from the environment, not \( (*, a) \) or \( (b, *) \).) Thus we have:

\[ EC = (a; Q)[F](b; R) \]

\[ EC || a; S = (a; Q) || (a; S) \]

\[ EC || b; T = (b; R) || (b; T) \]

\[ EC || c; U = F || c; U \]

\[ EC || (a || b) = F || (a || b) \]

Here \( S = \{(a, a), (b, b)\} \). The synchronisation must be available as soon as the choice starts to execute; the environment must already be doing a matching action, or start one as soon as the choice process starts.
If \( \Delta(b) = t \), then

\[
EC \ll (\text{wait } t; a; S) = F \ll (\text{wait } t; a; S)
\]

\[
(\text{wait } t/2; EC) \ll (b; T) = (\text{wait } t/2; b; R) \ll (b; T)
\]

\[
(\text{wait } t; EC) \ll (b; T) = (\text{wait } t; b; R) \ll (b; T)
\]

\[
(\text{wait } 3t/2; EC) \ll (b; T) = (\text{wait } 3t/2; F) \ll (b; T)
\]

The operational semantics of this new form of choice are clear; if

\[
(s_1 \xrightarrow{\Delta(f_1)} s'_1 t' & s_2 \xrightarrow{\Delta(f_2)} s'_2 t_2 & t \leq t' & \hspace{1cm} [t', t'_2 + \Delta(f_2)] \cap [t'_1, t'_2 + \Delta(f_2)] \neq \emptyset & & (f_1, f_2) \in S)
\]

then

\[
(s'\ll (s_1 t), (s_2[s_3]s_4 t')) \xrightarrow{(f_1, f_2)@t''} s''(s'_1 t'_1), (s'_2 t'_2))
\]

where \( t'' = \min(t'_1, t'_2) \)

and

\[
s''(s_1 t), (s_4[s_3]s_2 t') \xrightarrow{(f_1, f_2)@t''} s''(s'_1 t'_1), (s'_2 t'_2))
\]

where \( t'' = \min(t'_1, t'_2) \)

else, \( (s_3 \xrightarrow{f_3@t_3'} s'_3 t'_3) \) then

\[
s''(s_1 t), (s_4[s_3]s_2 t') \xrightarrow{f_3@t'3} s''(s_1 t), (s'_3 t'_3))
\]

(In the above we have used if ... then ... else to make the reading of the transition rules clearer. In general one should read something like:

\[
\frac{s_1 \xrightarrow{f@t'} s'_1 t'_1}{s''(s_1 t), (s_2 t) \xrightarrow{(f_1, +)@t''} s''(s'_1 t'_1), (s_2 t))}
\]

as ‘if \( s_1 \) starting at time \( t \) can evolve via a transition \( f_1 \) seen starting at time \( t' \) into \( s'_1 \) which is incapable of further action until \( t' \) then \( s''(s_1 t), (s_2 t) \) can evolve via the transition \( (f_1, +) \) seen starting at time \( t'' \) into \( s''(s'_1 t'_1), (s_2 t)) \.)’

The correct denotational characterisation of external choice is unclear. A tentative axiomatic characterisation of external choice might be, given \( (a, a') \in S \), and \( (a, c) \in S \):

\[
(a; Q) [F] (b; R) = (b; R) [F] (a; Q)
\]

\[
((a; Q) [F] (b; R)) \ll a'; S = (a; Q) \ll (a; S)
\]

\[
((a; Q) [F] (b; R) \ll c; U = F \ll c; U
\]
A4.1 The categorical semantics of IPA with recursion

Here, and in the next few section, we give some thoughts on how recursion might be treated in a categorical setting, roughly following [Bednarczyk 1987].

In this subsection all times are measured relative to \( t = 0 \), the assumed start of all things. We will only treat singly-nested guarded recursion. Consider some recursively-defined process \( \mu x. \mathcal{P} \) and some candidate I.E.S. meaning for it \( \mathcal{I}^* = (\text{LE}^*, \text{Con}^*, <\&) \). Clearly, for every \( n \) the process

\[
\mathcal{P}^n = \mathcal{P} \left[ \mu x. \mathcal{P}/x \right]^n \left[ \text{stop}/x \right]
\]

is finite. (Where the large square brackets denote substitution as usual.)

For convenience define \( \mathcal{P}^0 = \text{stop} \). Between each unwinding and the next there is some time, \( t_n \in \mathbb{R} \) such that the \( n \)th unwinding has just finished and \( (n+1) \)th has just started. Further, each unwinding, being finite, has a well-defined meaning, \( \mathcal{S}^n \) say:

\[
\forall n \in \mathbb{N}. \mathcal{S}^n = \text{def } C(\mathcal{P}^n) \ 0
\]

We have to show that the \( n \rightarrow \infty \) is well-defined. The crux of this is the \( t_n \)s. First note that \( t_n = \text{end}(\mathcal{S}^{n-1}) \) and that \( \mathcal{S}^0 = 1 \).

Clearly the meaning, \( \mathcal{S}^n \) say, truncated at \( t_n \) must match the intended meaning;

\[
\forall n \in \mathbb{N}. \mathcal{S}^n = \mathcal{S}^* \downarrow t_n
\]

so that \( \mathcal{S}^* \) approximates \( \mathcal{P} \) at every stage. We conjecture that there is a morphism from \( \mathcal{S}^n \) to \( \mathcal{S}^{n+1} \) in \( \text{tIES} \) for all \( n \) and that this is an embedding [Coquand 1988].

This leads us to conjecture that the category of embeddings \( \text{tIES} \) whose objects are computational interval event structures, and where there is an arrow from \( \mathcal{S} \) to \( \mathcal{S}' \) just when

\[
\exists t \in \mathbb{R}. \mathcal{S} = \mathcal{S}' \downarrow t
\]

might be appropriate for dealing with recursion. The treatment should proceed much as indicated by [Smyth & Plotkin 1982].

A4.2 The operational semantics of IPA with recursion

Recall that the axiomatic semantics of recursion was

\[
\mu x. \mathcal{P} = \mathcal{P} \left[ \mu x. \mathcal{P}/x \right]
\]
This suggests the operational treatment that if an unwound process can do something, so can the wound version.

\[ P \left[ \mu x. P/\tau x t \right] \xrightarrow{f \circ \tau t} s \cdot t' \]

(\mu x. P) \xrightarrow{f \circ \tau t} s \cdot t'

We should mention here related work on various forms of transition system; the reader may find [Boudol & Castellani 1989], [Kwiatkowska 1989], [Lodaya et al. 1989] and [Shields 1988] of interest.

A4.3 Aside, for readers of the last interlude, on rIES-based sheaves

Consider the category rIES and the nonnegative reals with the Alexandroff topology, so that the opens are the sets \([0, r]\) for \(r \in \mathbb{R}\), and call this topological space \(\mathbb{R}_+\). For a given I.E.S. \(\mathfrak{S}\), associate with an open \(U = [0, r]\) the I.E.S. \(F(U) = \mathfrak{S} \cdot r\). \(F\) lifts to a functor \(F: O(\mathbb{R}_+)^{op} \rightarrow \text{rIES}\) in the obvious way, and we have a presheaf of I.E.S.s on \(\mathbb{R}_+\).

This is, in fact, a sheaf, since given an open \(U\) of \(\mathbb{R}_+\) and an open cover for \(U\), \(\{[0, r]\}\), with \(U = \bigcup_{j \in J} U_j\), and a family of sections of \(F\), \(S_j, j \in J\), the required properties hold:

Consider two opens, \(U_i, U_j\). We have arrows \(F(U_i) \rightarrow F(U_i \cap U_j)\) and \(F(U_j) \rightarrow F(U_i \cap U_j)\) in rIES and moreover, the restriction of \(F(U_i)\) to domain \(U_i \cap U_j\) and of \(F(U_j)\) to \(U_i \cap U_j\) gives us the same object, namely \(\mathfrak{S} \cdot r\) where \(U_i \cap U_j = [0, r]\). The object of rIES such that \(F\) restricted to \(U_j\) is \(S_j\) is just \(\mathfrak{S}\). Hence we have a sheaf.

[Kwiatkowska 1989] shows that progress properties correspond to Alexandroff-open sets (in her space of behaviours) and finitary safety properties to Scott-closed sets. It is rather suggestive that our category of progress properties forms a sheaf over \(\mathbb{R}\) with the Alexandroff topology. Could it be that one can view timed properties as sheaves over the space of time? One might envisage pick a model of time \((\mathbb{N}, \mathbb{Q}, \mathbb{R}, \text{Minkowski spaces}, \text{whatever})\), endowing with the appropriate topology (e.g. Alexandroff if one is interested in progress properties) and erecting a sheaf over that topology. The sections of the sheaf will be observable properties (in rIES we are dealing with properties deducible from strict-timed causal bets), and the order on these sections (for us the arrows of rIES) will correspond to the order in which those properties can be observed. Thus we have model with two dimensions of variation; keeping the same category of properties and changing the underlying space (but not the topology) allows to deal with the same behavioural properties in different models of time, while changing the category and the topology, but keeping the same points in the underlying space allows us to deal with different classes of properties uniformly.
The case of R with the Alexandroff topology with the category of progress properties rIES as a sheaf over it is, I conjecture, only one example of this phenomena. It may be that this property-oriented approach to connecting time and behaviour will be more fruitful than concentrating on primitive notions like the connection between causality and timing.

References


Manipulating Structures:
Abstraction and Refinement

Want is a bitter and a hateful good,
Because its virtues are not understood;
Yet many things, impossible to thought,
Have been by needs to full perfection brought.

Dryden

This chapter is concerned with the specification and refinement of interval event structures. There are two orthogonal ways to refine interval event structures; event refinement and functional refinement. The latter is the more usual form of refinement; it involves changing the functionality of a structure while preserving some desired behavioural invariant. It will not be treated here.

We will just tackle the problem of event refinement, giving definitions which allow a labelled event to be replaced by a whole I.E.S. The issue of event refinement, following the ground-breaking work of [Shields 1979], is becoming more important in concurrency theory; witness the recent controversy in the EATCS bulletin (exemplified by [Reisig 1988]; see also [Reisig 1987]), and the work of [Aceto & Hennessy 1987], [Nielsen et al. 1988], and [Vogler 1989]. Our own approach is modelled on [Gischer 1989].

The dual of event refinement is substructure abstraction, – replacing a whole substructure by a single event. This too is defined, and the duality between these concepts, under certain restrictions, is demonstrated.

The postlude at the end of the chapter concludes the main body of the thesis. Our conclusions are presented, and further work is suggested. A fairly expansive bibliography concludes the document.

5.0 Event refinement

Here we deal with the problem of compound events. At some levels of abstraction we may wish to treat something as a single labelled event, and as some structure of labelled events at another, lower, level. This process is known as event refinement. In this section, we shall allude to some circumstances when it is possible to do this while preserving some of the properties of the original structure.
DEFINITION 5.0 – Refining structures

Suppose \( a = [(r_1, a_1), (r_2, a_2)] \in LE \) is a labelled event in some interval event structure \( S = (LE, Con, \prec) \), and \( S' = (LE', Con', \prec') \) is another I.E.S. with silent event \( *' \). \( S \) can refine \( a \), written \( S \) refines \( a \), iff

(i) \( S \) lies exactly inside \( a \); \( r_1 = \text{begin}(S) \) & \( r_2 = \text{end}(S) \).

(ii) \( S \) is consistent with \( S \). This is easiest and cleanest to achieve if we enforce the requirement \( LE' \cap LE = \emptyset \).

(iii) \( S \) has a transition causally the same as \( *' \) and one causally the same as \( *' \): \( \exists a_i \neq *', a_i \neq *' \in LE'. \text{begin}(a_i) = \text{begin}(*') \) & \( \text{end}(a_i) = \text{end}(*') \)

The transitions must also have the same \( Con \) behaviour as \( *' \):
\[
\forall c' \in M(S'). \{a_i, a_i\} \in c'
\]

The requirement (iii) that there is an occurrence of the start of an event necessarily simultaneous with the start of \( S \), and an occurrence of the finish of an event necessarily simultaneous with the finish of \( S \), is enforced to ensure that there is a transition to ‘inherit’ the causality of \( *' \) in the refined structure. (There will be no \( *' \) in the refined structure, only a \( *' \) timed with the same time as \( *' \).)

We can think of an event as defining a bracket, – its \( \text{begin} \) and \( \text{end} \) times, – within which any refinement of it must terminate. Thus a bounded I.E.S. is a valid refinement of an event if it can do its work within the time available and does not interfere with anything.

(We can shift a structure around, (as we did in the definition of prefixing in chapter two,) by adding a constant to every time in the structure, in order to propitiate refinements. This enables us to refine the same event by the same structure each time it occurs.)

The structure obtained by performing a refinement is simple to describe; we just replace the labelled event we are going to refine by the structure that is doing the refining: –

DEFINITION 5.1 – Event refinement

The I.E.S. \( S \) with \( a \) refined by \( S \), written \( S[3/a] \), is defined if \( S \) refines \( a \), when it is the I.E.S. \( (LE^-, Con^-, \prec') \) where

(i) \( LE^- =_{def} (LE - \{a\}) \cup (LE' - \{*'\}) \)

(ii) Define the \( *' \)-less \( Con \)-sets of \( S \), \( Con^I = \{c' - \{*'\} \mid c' \in Con' \} \).
Now, in the refined structure there will be Con-sets that didn't mention $a$ from the old structure, plus, instead of any Con-set that mentioned $a$, that Con-set without $a$ but with all the $\ast$-less Con-sets of $\mathcal{S}$:

$$\text{Con}^* = \{c \mid c \in \text{Con}, a \in c \} \cup \{(c - \{a\}) \cup c' \mid c \in \text{Con}, a \in c, c' \in \text{Con}\}$$

(iii) For the part of $<\mathcal{S}$ given by $\mathcal{S}$ we need to rewrite $a$, tags by $a_{0,1}$ ones and $a_1$ tags by $a_{0,1}$ ones. Suppose $g a_1 = a_1[a_0/\mathcal{A}_1][a_0/\mathcal{A}_1]$. Then $<\mathcal{S}$ will certainly have in it

$$\{((r_{0,1}, g a_{0,1}), (r_{0,1}, g a_{0,1})) \mid ((r_{0,1}, a_{0,1}), (r_{0,1}, a_{0,1})) \in <\mathcal{S}\}$$

It will also inherit $<\mathcal{S}$ but with $a_{0,1}$ replacing $\ast$ and $a_{0,1}$, $\ast$. We need to rewrite some tags; define $h a_1 = a_1[a_0/\mathcal{A}_1][a_0/\mathcal{A}_1]$. Then $<\mathcal{S}$ will have in it

$$\{((r_{0,1}, h a_{0,1}), (r_{0,1}, h a_{0,1})) \mid ((r_{0,1}, a_{0,1}), (r_{0,1}, a_{0,1})) \in <\mathcal{S}\}$$

$<\mathcal{S}$ is defined to be the smallest transitive relation containing both of these sets.

A similar technique is used for $\approx_a$.

This last definition has the advantage of ensuring that the beginning and ending points of $a$ are used in a sensible manner; anything that was previously branching less-than the beginning of $a$ will now be less than every labelled event in the refining structure, and anything that was greater than the end of $a$ will now be greater than every labelled event in the refining structure.

If we had not required the existence of the transitions $a_{0,1}$ and $a_{0,1}$ we would be saddled with a very complex definition; we would have to work out what to do with the causality of the transitions of $a$, but we wouldn't have points with the right timing to acquire it. Hence refinement could disconnect branches, which would be rather undesirable; we have already seen this kind of problem in section 2.3.

A labelled event, then, can be seen as specifying an interval within which the silent event of a refining structure must fit exactly. Once the refinement is made, the causality that previously applied to $a$ will now affect every labelled event in the refining structure.

It remains to check that refinement is a congruence of betting equivalence: if we have two structures equivalent under a form of bet, $\mathcal{S} \sim z \mathcal{S}'$ then we must show that $\mathcal{S}[\mathcal{S}/a] \sim z \mathcal{S}'[\mathcal{S}/a]$.

This will be postponed until after an example event refinement.
EXAMPLE

Consider one of the examples from chapter three, $S_2$, rewritten as $(LE, Con, \prec_b, =_b)$:

$$S_2 = \{\begin{array}{l}
(a = [(0, a_i), (2, a_I)], b = [(1, b_i), (4, b_I)], \\
\ast = \{(0, \ast_0), (4, \ast_0)\} \\
\{\{\ast, a, b\}\}, \\
\{(0, a_i), (1, b_i), (2, a_I), (4, b_I)\}, \\
\emptyset
\end{array}\}$$

The labelled event $b$ can be refined by the structure $S_b = (LE', Con', \prec_b, =_b)$;

$$S_b = \{\begin{array}{l}
(c = [(2, c_i), (4, c_I)], d = [(1, d_i), (2, d_I)], e = [(2, e_i), (4, e_I)], \\
\ast = \{(1, \ast_1), (4, \ast_1)\} \\
\{\{\ast', d, c\}, \{\ast', d, e\}\}, \\
\{(2, d_i), (2, c_i), (2, e_i), (4, c_i)\}, \\
\emptyset
\end{array}\}$$

(Again we adopt the convention of not showing trivial relations like $a_i \preceq a_I$ & $\ast' \preceq a_i$, and of requiring a transitive closure to obtain the true $\preceq$ and $=_i$.) For clarity, we do, however, show the silent event. The refinement leaves us with $S_2[S/b]$, as shown below.

$$S_2[S/b] = \{\begin{array}{l}
(a = [(0, a_i), (2, a_I)], c = [(2, c_i), (4, c_I)], \\
d = [(1, d_i), (2, d_I)], e = [(2, e_i), (4, e_I)], \\
\ast = [(0, \ast_0), (4, \ast_0)] \\
\{\{\ast, a, d, c\}, \{\ast, a, d, e\}\}, \\
\{(0, a_i), (1, d_i), (2, a_I), (4, c_i), (2, a_I), (4, e_I)\}, \\
\{(2, d_i), (2, c_i), (2, d_i), (2, e_i)\}\}
\end{array}\}$$

THEOREM – The observational equivalences are congruences W.R.T. event refinement

The glib statement of this theorem, $3[S/a] \sim_z 3[S'/a]$, is not quite right. Suppose that in $3$, $a = (l, e) = [(r_{a_i}, a_i), (r_{a_I}, a_I)]$ is a labelled event. Then if $3 \sim_z 3'$ and if every occurrence of $e$ with real timing $[r_{a_i}, r_{a_I}]$ is replaced by $3$, in $3$ and $3'$, the results will $\sim_z$ each other.

Suppose $a = (l, e) = [(r_{a_i}, a_i), (r_{a_I}, a_I)]$ is in $3$: we will write $3$ with every occurrence of $e$ with real timing $[r_{a_i}, r_{a_I}]$ replaced by $3$, as $3[S/a]$. 

Page 208 Chapter Five — Constraining Structures
Proof. First notice that we merely have to show that $\mathfrak{S}[3/\mathcal{A}] \sim_\gamma \mathfrak{S}[3/\mathcal{A}]$ as $\sim_\gamma$ is strictly the most discriminating equivalence, and so if event refinement is a congruence of it, it will be a congruence of all the $\sim_\gamma$s.

To begin with, we have to show that if $\mathfrak{S}$ refines $a$ in $\mathfrak{S}$ then there will be a labelled event with the same event part and timing as $a$ for it to refine in $\mathfrak{S}'$. This is clearly the case, since if $\mathfrak{S}$ has an $a = (l, e) = [(r_{(\delta)}, a_\delta), (r_{(\delta)}, a_\delta)]$ then $\mathfrak{S}'$ will have an $a' = (l', e) = [(r_{(\delta)}, a_\delta), (r_{(\delta)}, a_\delta)]$ since otherwise we can bet on the transitions of $a$ and obtain $-(3 \sim_\gamma 3')$.

Thus $\mathfrak{S}[3/\mathcal{A}]$ and $\mathfrak{S}[3/\mathcal{A}]$ are well-defined and behave as intended.

Next notice that if $\mathcal{L} \subseteq \mathcal{L} \times \mathcal{E} \cup \{\ast\}$ then $\mathcal{L}' \subseteq \mathcal{L}' \times \mathcal{E} \cup \{\ast\}$ and there is a bijection, $\mathcal{E}$ say, between $\mathcal{L}$ and $\mathcal{L}'$ such that $a = (l, e) \in \mathcal{L} \Rightarrow a = (\mathcal{E}(l), e) \in \mathcal{L}'$, so, up to bijection of labels, we need only consider $E$ as a universe of bets.

Consider the syntax of bets, $b ::= \text{Win} \mid \text{Lose} \mid (a_\gamma) \gamma \cdot b_1 \lor b_2 \mid b_1 \land b_2 \mid \forall b \mid \exists b$ and suppose to the contrary of the theorem that there is a bet, $bd$ say, such that

$$\mathcal{S}(\mathfrak{S}[3/\mathcal{A}], L, bd) \neq \mathcal{S}(\mathfrak{S}[3/\mathcal{A}], L, bd).$$

Clearly, if this bet $bd$ is of the form $b_1 \lor b_2$ or $b_1 \land b_2$ then there are more primitive bets that will distinguish the structures, so we can discard these forms. Similarly, if $\forall b$ or $\exists b$ distinguish the structures then $b$ will, so we are left with the assumption that $\mathfrak{S}[3/\mathcal{A}]$ can display a transition that $\mathfrak{S}[3/\mathcal{A}]$ even though $3 \sim_\gamma 3'$.

Suppose this offending transition is $a_{(\delta)} = (r_{(\delta)}, a_{(\delta)})$. Clearly $\text{begin}(3) \leq r_{(\delta)} \leq \text{end}(3)$ since otherwise $-(3 \sim_\gamma 3')$. Suppose this transition $a_{(\delta)}$ is in some run $L_{(\delta)}$. This run must have been derived from one of $3$ which displayed $a$ since otherwise $-(3 \sim_\gamma 3')$. But, for every run of $3$ which displays an $a$ there will be a run of $3'$ which is the same up to bijection of labels, since otherwise $-(3 \sim_\gamma 3')$.

In $3[3/\mathcal{A}]$ every run of $3$ which displayed $a$ will be replaced by a set of runs which together have all of the transitions of $3_a$, and similarly for $3'[3/\mathcal{A}]$. Hence our transition $a_{(\delta)}$ can't be one of the transitions of $3_a$, since these are present in both $3[3/\mathcal{A}]$ and $3'[3/\mathcal{A}]$. But it can't be one that isn't from $3_a$ either, since then $-(3 \sim_\gamma 3')$.

Hence we have a contradiction and the theorem is proved.

$\hfill \blacksquare$
5.1 Substructure Abstraction

The dual concept of refinement is abstraction. Instead of the conventional notion of hiding, we will allow substructure abstraction; giving more than one labelled event a (new) name and only mentioning that name in future. There are considerable complications in knowing what an event abstraction might mean in general, so, here, we just give a definition that is the dual of the event refinement definition.

**DEFINITION 5.2 – Substructure**

Consider two I.E.S.s, \( S = (LE, Con, \leq_b) \), and \( S_b = (LE', Con', \leq_b') \). \( S_b \) is a abstractable substructure of \( S \) iff

(i) \( \forall a' = (l, e) \in LE', a' \neq *' \exists a = (l, e) \in LE, a \neq * . a = [t_n, t] = a' \). That is, \( S_b \) has a subset of the labelled events of \( S \) with the same timings. This means that we can use the same name for a labelled event in \( LE \) as we use for it in \( LE' \).

(ii) \( Con' =_{def} \{(c \cap LE') \cup \{*' \} \mid c \in Con\} \). That is, the \( Con\)-sets of \( S_b \) are just those of \( S \) which refer to the labelled events in \( LE' \).

(iii) \( \exists a'_1, a'_2 \in LE', \forall a' \in LE', a \in LE - LE' .

\[
(a'_1 <_b a'_2) \Rightarrow (a <_b a'_1 \leq a'_2) \land \\
(a'_1 <_b a) \Rightarrow (a'_1 <_b a'_2 <_b a'_1) \land \\
(a'_1 =_b a) \Rightarrow (a'_1 =_b a'_1 =_b a') \lor (a'_1 =_b a'_2 =_b a')
\]

Further,

\[
\forall a'_1, a'_2 \in LE'. ((a'_1 <'_b a'_2) \Leftrightarrow (a'_1 <_b a'_2)) \land ((a'_1 ='_b a'_2) \Leftrightarrow (a'_1 =_b a'_2))
\]

(iv) If \( a'_1 = ([r_{a'_1}, a_{a'_1}), (r_{a'_1}, a_{r_{a'_1}})] \) & \( a'_2 = ([r_{a'_2}, a_{a'_2}), (r_{a'_2}, a_{r_{a'_2}})] \) then

\[
*' = ([r_{a'_1}, *'), (r_{a'_2}, *')]
\]

The first two conditions just ensure that \( S_b \) is actually contained in \( S \). The third ensures that \( S_b \) is only connected to \( S \) via its end points, and agrees with it on causality, so that we have

![Diagram](image)

**Figure 5.1 – Abstractable and nonabstractable substructures**

where dashed lines, as usual, show causality.
The final condition (iv) ensures that $S_b$ has the right silent event.

The notion of substructure is the key to an abstraction principle that is the dual of refinement. It means that only some structures are amenable to refinement, but that is not necessarily a bad thing.

**DEFINITION 5.3 – Abstraction**

Consider an I.E.S., $\mathcal{S} = (\mathcal{L}E, \text{Con}, \prec_b)$, and a new labelled event, $b \in \mathcal{L}E$. Suppose that $\mathcal{S}$ has a refinable substructure, $\mathcal{S}_{b} = (\mathcal{L}E', \text{Con}', \prec')$. Then, we can define $\mathcal{S}$ with $\mathcal{S}_b$ abstracted by $b$, written $\mathcal{S}[b/\mathcal{S}_b]$, as the I.E.S. $(\mathcal{L}E, \text{Con}', \prec')$ where

(i) $\mathcal{L}E'' = \mathcal{L}E - (\mathcal{L}E' - \{*\}) \cup \{b\}$

(ii) Define $k \in \mathcal{L}E'$ if $a \in \mathcal{L}E'$
    $= a$ otherwise. Then
    $\text{Con}' = \{k(c) \mid c \in \text{Con}\}$

(iii) Define $i(r, a) = (r, a)$ if $a \in \mathcal{L}E'$
    $= (r, b)$ otherwise
    Define $j(r, a) = (r, a)$ if $a \in \mathcal{L}E'$
    $= (r, b)$ otherwise

Then

$\prec' = \{((j(r', a), i(r', a)) \cup ((r', a), (r, a))) \mid ((r, a), (r, a)) \in \prec_b\}$

And similarly for $\approx'$. 

The new labelled event $b$ carries the timing of $*'$.

It follows immediately that abstraction is the dual of refinement:

**THEOREM – Relating abstraction and refinement**

Suppose $\mathcal{S}$ is an I.E.S. and $\mathcal{S}_b$ is a refinable substructure of it. Then

$(\mathcal{S}[b/\mathcal{S}_b])[b/\mathcal{S}_b] = \mathcal{S}$ and $(\mathcal{S}[a/\mathcal{S}_a])[a/\mathcal{S}_a] = \mathcal{S}$

**Proof.** For the first part, $(\mathcal{S}[b/\mathcal{S}_b])[b/\mathcal{S}_b] = \mathcal{S}$, consider the three components separately.

(i) R.T.P. $\mathcal{L}E = ((\mathcal{L}E - (\mathcal{L}E' - \{*\}) \cup \{b\}) \cup (\mathcal{L}E' - \{*\}))$
    $= (\mathcal{L}E - (\mathcal{L}E' - \{*\}) \cup (\mathcal{L}E' - \{*\}))$
    $= \mathcal{L}E$

(ii) R.T.P. $\text{Con} = \{c \mid c \in \text{Con}_1, b \notin c\} \cup \{(c - \{b\}) \cup c' \mid c \in \text{Con}_2, b \in c, c' \in \text{Con}\}$
    where $\text{Con}_1 = \{c' - \{*\} \mid c' \in \text{Con}'\}$ and $\text{Con}_2 = \{k(c) \mid c \in \text{Con}\}$. 

---

Chapter Five — Constraining Structures
Rewrite $\text{Con2}$ as $\text{Con3} \cup \text{Con4}$ where $\text{Con3} = \{ c \mid c \in \text{Con}, \neg \exists a' \in \text{LE} . \ a' \in c \}$ and $\text{Con4} = \{ c \cap (\text{LE} - \text{LE}') \cup \{ b \mid c \in \text{Con}, \exists a' \in \text{LE} . \ a' \in c \} \}$. Then R.T.P.

$$\text{Con} = \{ c \mid c \in \text{Con3} \} \cup \{ (c - \{ b \}) \cup c' \mid c \in \text{Con4}, c' \in \text{Con1} \} = \text{Con3} \cup \{ (c \cap (\text{LE} - \text{LE}')) \cup c' \mid c \in \text{Con}, c' \in \text{Con1} \}.$$

But,

$$\text{Con}' = \{ (c \cap \text{LE}') \cup \{ * \} \mid c \in \text{Con3} \}$$

since $\mathcal{S}_a$ is a refinable substructure, so we have $\text{Con1} = \{ c \cap \text{LE}' \mid c \in \text{Con3} \}$ and it just remains to show that

$$\text{Con} = \text{Con3} \cup \{ (c \cap (\text{LE} - \text{LE}')) \cup (c \cap \text{LE}') \mid c \in \text{Con}, \neg \exists a' \in \text{LE} . \ a' \in c \} = \{ c \mid c \in \text{Con}, \neg \exists a' \in \text{LE} . \ a' \in c \} \cup \{ c \mid c \in \text{Con}, \neg \exists a' \in \text{LE} . \ a' \in c \} = \text{Con}. $$

(iii) R.T.P. $\lessdot_b = \lessdot_a \cup \lessdot_b$ where

$$\lessdot_a = \{ ( (r, (0), a_{00}), (r, (0), a_{00})) \mid ( (r, (0), a_{10}), (r, (0), a_{00})) \in \lessdot_2 \},$$

$$\lessdot_b = \{ ( (r, (0), h_{a_{00}}), (r, (0), h_{a_{00}})) \mid ( (r, (0), a_{10}), (r, (0), a_{00})) \in \lessdot_2 \},$$

and where

$$\lessdot_2 = \{ (j, (r, (0), a_{00}), i, (r, (0), a_{00})) \mid ( (r, (0), a_{10}), (r, (0), a_{00})) \in \lessdot_2 \}$$

with $g, h, i$ and $j$ are as before.

Notice that, since $\mathcal{S}_b$ is a refinable substructure, we can write $\lessdot_b$ (modulo a transitive closure or two) as the union of orders not involving any tags in $\mathcal{S}_b$, things related to $a_{01}$, things related to $a_{10}$, and the orders in $\mathcal{S}_b$ itself, $\lessdot'$ (but without those relating to the fictional $*$):

$$\lessdot_b = \lessdot_3 \cup \lessdot_4 a \cup \lessdot_4 b \cup \lessdot_b$$

where

$$\lessdot_3 = \{ ( (r, (0), a_{01}), (r, (0), a_{01})) \mid ( (r, (0), a_{10}), (r, (0), a_{10})) \in \lessdot_b, a_1 \neq \text{LE}' \},$$

$$\lessdot_4 a = \{ ( (r, (0), a_{01}), (r, (0), a_{01})) \mid ( (r, (0), a_{10}), (r, (0), a_{10})) \in \lessdot_b, a_1 = \text{LE}' \},$$

$$\lessdot_4 b = \{ ( (r, (0), a_{01}), (r, (0), a_{01})) \mid ( (r, (0), a_{10}), (r, (0), a_{10})) \in \lessdot_b, a_2 = \text{LE}' \}.$$

Clearly, nothing $g, h, i$ or $j$ do will affect $\lessdot_3$, so our task reduces to proving that $\lessdot_4 a \cup \lessdot_4 b = \{ ( (r, (0), g a_{01}), (r, (0), g a_{01})) \mid ( (r, (0), a_{10}), (r, (0), a_{10})) \in \lessdot_4 a \cup \lessdot_4 b \}$

But, $\lessdot_4 a$ and $\lessdot_4 b$ have nothing in them that $g$ can effect, by construction. Thus the case of $\lessdot_b$ follows.

Hence the first part of the theorem follows.

The other case, $(\exists ![3 \mid d_1]) [a / 3_1] = 3$, follows similarly.

There are other notions of refinement from the one we have outlined; it is possible, for instance, to define a structure as capable of refining a labelled event $a$ if it fits inside the time interval available, rather than fitting exactly. This definition places limitations on the causality that the transitions of $a$ can have, and makes it harder to find a dual definition of abstraction, but it may be useful on occasion.
Bibliography


Postlude – Conclusions and Further Work

The final section of the thesis, this one, deals with further work needed and conclusions resulting from this work. The latter first:

The principal conclusion is that it is possible to construct an implementational real-time concurrency theory with intervals of the reals associated with occurrences of events. Such a theory gives fresh insights into concurrency and nondeterminism.

The detailed conclusions can be subdivided into two sections, one dealing with the technical content of the thesis, the other with more philosophical matters; there is some overlap of material between the sections.

Technical conclusions

Our primary technical conclusion is that some of the considerable work to be done in understanding the relationships between various notions of time, causality, and concurrency can be done using the interval event structure framework and categorical techniques. As the further work section (below) indicates, much of this thesis needs reworking to incorporate a more categorical treatment and make such comparisons easier. We have discovered four kinds of categories:

(i) Categories of simulations. In these categories, objects are I.E.S.s and arrows indicate when one structure can simulate another. A proper refinement calculus for I.E.S.s should be built using these categories. The common notions of the most sequential, or most nondeterministic, or most concurrent I.E.S. that simulates a given one live here.

(ii) Categories of behaviours. These are the ‘conventional’ categories of concurrency theory; objects are either labelled events or their transitions, and arrows represent causality. Associated with each such causal category is a linear order. Thus, for us, a ‘behaviour’ is a poset viewed as a category, a linear order viewed as a category, and a functor assigning times to happenings, that is, objects in the latter category to objects in the former.

(iii) Categories of embeddings. Here, objects are I.E.S.s and there is an arrow from S to S’ just when S is S’ truncated at some point. A category of embeddings is a wide subcategory of a category of simulations, since any notion of simulation naturally leads to a notion of embedding. These may be the right categories for establishing fixed-point results.

(iv) Certain notions of behavioural difference seem to lead naturally to metric spaces. We have shown how to treat the logic of such equivalences categorically, by
defining a category whose objects are (a subclass of) I.E.S.s and whose arrows represent the betting distance between structures. This approach should be of some use in a more general search for the logic of concurrent behaviours.

It is our contention that a lot of interesting concurrency theory can be done using these four kinds of categories and by relating various instances of them. (Categories derived from asynchronous operational models like ATLTSs are also important and should turn out to be what we have called "categories of behaviours."

In the preceding chapters various constructions were invented and, as often as not, claimed to be natural. Here we summarise some of the more important conclusions about what constructions make sense in the timed framework.

★ Causality is represented as a poset and timing by the reals with their usual linear order. The connection between these two structures can be described sheaf-theoretically.

★ It is sensible to associate occurrences of events with intervals of time. This leads to clean notions of event refinement and substructure abstraction which are duals of each other.

★ It also leads to a notion of concurrency which only permits synchronisation when intervals overlap. This form of concurrency is properly described by two structures, one recording information about the intersection of events' intervals, and one recording information about the union of events' intervals. These structures stand in a simple relationship to one another.

★ There is a semantic paradigm based entirely on single observations of executions that allows us to discover all the information about a structure that is finitely obtainable. It can be seen as a natural extension of testing equivalence.

★ There is a major difference between the assumptions made about the nature of concurrency in traditional non-interleaving frameworks such as the Petri net model, and those made in the world of process algebras. A new timed process algebra can be constructed which takes a different position from either major camp; it uses Petri net-style nondeterminism, mixes it with the idea (from process algebra) that there is a single (non-distributed) state before \( P \parallel Q \) starts and after it finishes, and adds a semantics based in the "true concurrency" tradition. These notions seem to live together quite happily.

★ It is possible to have a continuous model and yet not to have to abandon handy induction principles. Models of computational systems will usually only do a countable number of things in uncountable time, and by concentrating on them, it is possible to have the richness of real-timing without losing the chance to be able to do structural inductions.
Philosophical conclusions

We have tried to explore a rather different point in the concurrency theory design space. The decision that I.E.S.s should be capable of evolving without reference to their environment leads to:

* the notion of synchronisations only happening when they are desired, and when timing allows. This is a much more primitive notion of synchronisation than usual, and a rather better description of the mechanisms available in physically concurrent systems.

* a semantics based entirely on observation. It is important to notice that concurrency and choice interact in subtle ways: having changed our notion of concurrency from the conventional one in process algebra, we must see what notion of choice makes sense. For us, external choice at the I.E.S.-level turns out not to be natural, and so the testing paradigm (which assumes participatory concurrency and makes most sense when external choice is present) is no longer appropriate. Hence we have to formulate a semantics based on observation rather than participation. It turns out to be possible to do this while keeping some of the features of the testing paradigm intact, highlighting the fact that bisimulation isn’t the only sensible notion of observational equivalence.

It is worth emphasizing that we have shown that an implementational concurrency theory can make different assumptions about what kind of an entity a distributed system is and still be coherent, (— we assumed it could evolve without reference to the environment, and that we could see everything the system does). One just has to consider what the natural technical notions of ‘concurrency’ and ‘nondeterminism’ are, once the philosophical position of the theory is decided upon.

Turning to IPA, perhaps unsurprisingly, we see the same pattern at a higher level. It is possible to build a process algebra with the same philosophical stance as the more implementational I.E.S. model; again, we have timing, non-participatory concurrency (and, thus, a very raw notion of liveness), and choice as nondeterminism.

While our stance on the nature of concurrency may seem radical, the introduction of real-time is equally so. The importance of timing is that it gives us another structure to test our constructions in; R has a strict total order <, and all constructions that we make must respect both causality and this order. We have succeeded in defining various constructions that fulfil this criteria, but the impact of timing is greater than just to validate our particular constructions; a valid consideration for any concurrency theory is whether it is, in principle, timeable, or not: for those that are not, it should be possible to justify the veracity of their constructions on other grounds.
One of our stated aims was to produce a *behaviourally-rich* concurrency theory. The I.E.S. model is certainly rich enough to make some interesting distinctions: –

* we can distinguish between concurrency and nondeterministic interleaving; hence this is a ‘true concurrency’ model.

* we can distinguish between occurrences of events whose intervals are entirely simultaneous, overlap each other, and are entirely disjoint.

* we can also distinguish between concurrency, simultaneity, and overlap. Concurrent occurrences need not overlap in time, and overlapping occurrences need not be concurrent. A clear distinction between timing and causality is maintained, enabling descriptions of concurrent systems to distinguish between happenings in the same place (which are possibly simultaneous, but never concurrent), and distributed happenings (which are possibly simultaneous and possibly concurrent).

In short, we can talk about the where and the when of concurrency theory as well as the usual what.

Our notion of concurrency, – that two things are concurrent if they are not completely dependent on each other, – is, I claim, a useful one. This notion of concurrency can be interpreted allowing two things to happen in different places if they are concurrent. The fact that our decision on which things are concurrent derives from the fundamental causality of the situation, not from any particular observations, means that it can be seen as a rather deeper one than, for instance, notions of concurrency which rely on the possibility of an observer seeing two different interleavings. ([Reisig 1987] calls our sort of notion ‘strong concurrency.’)

**Further work**

This will be discussed chapter by chapter: –

*Chapter 0.* It would be interesting to try to develop more formal criteria for comparing models. It is vital to have certain ‘fixed-points’ to facilitate this process, – notions that should have a definite well-understood meaning in every model. Part of the problem with trying to compare concurrency theories, at the moment, is that it is hard to extract the essence of a given theories’ notion of, for instance, concurrency, while abandoning the formalism used to define it.

It would be particularly interesting to try to develop an untimed concurrency theory which held to the precepts of our philosophical section, 0.3, in order to see what notions of behavioural equivalence they lead to.
In the long term, physical concurrency should become of increasing interest to concurrency theorists, – the search for a suitable formalism is only just beginning. We hope that some of the ideas presented here will be of use in the development of theories of physical concurrency.

Chapter 1. A sensible basis for a timed model, in retrospect, would probably be the categories POSet of posets and LinSet of linearly ordered sets. All the work of chapter one can be seen as providing a structure built over both these categories; it would be better rephrased as such. The work of Thomason, discussed in the interlude preceding chapter three, indicates that there might, under certain conditions, be an adjunction between QOSet (the category built from POSet by formally associating transitions with occurrences of events) and LinSet; this hypothesis ought to be explored, and, if possible, extended to the non-free construction of timing from posets (i.e. the free construction of timing in the presence of temporal as well as causal constraints). The implications of instability should be further investigated, and class of functions computable by I.E.S.s (under suitable constraints, such as finite density and finite causes) should be elucidated.

Chapter 2. Much more needs to be discovered about categories for timed models before the right choice is clear; categories like tIES seem a sound choice, but it is hard to tell without defining more operators in them (such as CSP III or CCS relabelling) and discovering more about their properties. Hiding should be dealt with categorically; unfortunately, the proper treatment of causality in this case, as with sequential composition, is less clear.

It would be very technically convenient to have a category of timed simulations that had all small limits and colimits. There seems to be no obvious modification that can be made to the definition of morphism in tIES that would give it these properties; perhaps different category of simulations might be more appropriate.

The insight stated above, that the real test of the veracity of operations is that they make sense both temporally and causally, should be exploited, once the correct forum for comparing the temporal and the causal is known: the two categories QOSet and LinSet, together with an adjunction between them, (if one exists,) would have sufficient structure for such a comparison.

The relationship between tIES and other categories used in describing concurrency theories should also be investigated. This would enable us to relate I.E.S.s formally to event structures, Petri nets etc. Such a categorical comparison might enable us to import other notions of equivalence into the I.E.S. model, which could broaden our understanding of it.

My real discovery in writing this chapter was that I didn’t know enough category theory. It would be interesting to rewrite the chapter using a much more general notion of behaviour. The work of [Pratt et al. 1989] again provides the clue; we should specify operations uniformly over a category of behavioural categories. In our case, elements of this large category...
would be categories each incorporating a notion of the behaviour of an I.E.S. Such an approach, although technically difficult, would provide the perfect forum for comparing notions of timed (and untimed) behaviour.

Chapter 3. The categories of bets should be investigated in much more detail. The relationship between traditional testing equivalence and betting equivalence is fairly unclear, and should be elucidated. Furthermore, we should investigate the relationship between the operations of chapter two and the equivalences of chapter three in more detail and, in particular, the extent to which the betting equivalences are congruences of the various operations of chapter two. It would be interesting to know if there was an appropriate subcategory of tIES in which the various betting equivalences could be recovered as isomorphisms.

The worth of a new semantic technique is only really obvious when it is applied in numerous situations. Therefore, betting equivalence might profitably be applied in other models, leading to a nonparticipatory semantics for them, and a better understanding of betting and of what it is applied to.

Practically nothing was discovered about the various spaces of bets. It would be interesting to investigate the topological properties of these spaces, (such as compactness), and attempt to discover whether there is anything interesting to be learnt from these properties. Could one, for instance, rely on any space constructed from bets having certain properties, independent of the system being bet upon? One obviously desirable property in all our spaces is that finite structures should be dense in the infinite ones; this would bear investigation.

Furthermore, it might be informative to investigate different metrics; one could easily formulate a metric based on the integral over all time of the intervals that two structures were not betting-equivalent, for instance. This would take a much more even-handed view of failure than we do now: it would allow transgressions in the past to be mitigated by future good behaviour, rather than penalising the first failure of simulation, however small.

The hint in the interlude succeeding chapter three about describing real-timed systems by sheaves would probably bear further investigation. It would be interesting to reformulate the whole theory of causality and timing, concentrating on the sheaf nature of the 'evaluation functor,' the functor that tells us what has happened thus far. The interlude preceding the chapter can be seen as hinting at an adjunction between appropriate categories of timing and causality when causality is treated at the labelled event level (i.e. via the j-morphisms). The section about sheaves in the next interlude, in contrast, deals with the relationship between timing and causality at the lower level of transitions and the causal order. The fact that the sheaf of sets of posets generated by an I.E.S. over the reals has nice global as well as local properties might be exploitable, possibly leading to a better understanding of the relationship between timing and causality at the transition level.
It might be easier to work with executions directly rather than I.E.S.s. The Hausdorff metric for subsets (discussed in the appendix to chapter three) could be used to build spaces from structures (via their set of executions); these might be interesting to work with, especially since it should be possible to derive the usual more-nondeterministic-than ordering in this setting (— essentially one process would be more nondeterministic than another iff they had the same always-lose behaviour, but the latter process always won more often than the former).

All bets can be seen as $\delta$-timed bets. Yet we have concentrated on untimed and strict-timed bets, two special cases. Our semantics would be neater if we had a uniform treatment of $\delta$-timed difference, as a metric space for instance, to match the pleasantly clean treatment of sameness that betting equivalence gives us. In order to deal with timewise refinement properly, the connection between the strict-timed and the untimed spaces needs to be fully worked out: perhaps a sensible (categorical ?) treatment of $\delta$-timed bets would provide a unifying framework.

Chapter 4. The ATLTS technique is not restricted to IPA; it should be used to give an operational semantics to other calculi, perhaps timed CSP. ATLTSs could be given a categorical treatment much as Bednarczyk treated ATSs; this treatment should prove most informative, and will propitiate the establishment of the relationship of our timed operational semantics to other operational models. We should discuss the precise relationship of the operational semantics of IPA to the other two semantics; perhaps a categorical treatment of ATLTSs would propitiate this, too. It seems likely that the untimed part of ATLTSs have the same expressive power as C/E nets or asynchronous transition systems; this topic should be investigated, as it should give some perspective on how to add time to these models.

In IPA itself we should deal with recursion properly. We should be able to use Banach's contraction mapping theorem and metric spaces of bets to deal with recursion, rather than the Smyth & Plotkin result and categories of simulations suggested in the interlude; a comparison of the two approaches would be useful. It is rather suggestive that one obvious category used to erect a sheaf of executions of I.E.S.s over the reals, rIIES, seems promising as a candidate for one needed to give a meaning to recursive terms in IPA. The suggestion that various topological notions of closure correspond to various classes of properties, and that one of these notions is the appropriate one to impose on a model of time in order to erect a sheaf of the matching class of properties upon it should be investigated further: it offers the hope of a comprehensive theory of observation.

Various other connections remain to be investigated. How does the work on models of Petri nets as monoidal categories, or the work from Imperial on quantale-theoretic models relate to ours? Our sheaf-theoretic viewpoint needs much work, and may eventually be related to these models.
We can only specify a small class of I.E.S.s with IPA so far; we cannot, for instance, specify local simultaneity. It would be interesting to extend the syntax of IPA so that the semantics included more of cIES or even IES. IPA also needs a refinement methodology, and much larger examples written in it, before its worth as a process algebra, rather than a curiosity, is established. Our full abstraction result should be extended to IPA with mutual recursion. A hiding operator would be useful in IPA, and its introduction, without making great changes to the semantics, should be investigated. Timed process algebras don’t seem very algebraic; there are few laws that they satisfy. It would be interesting to investigate a hierarchy of less and less discriminating behavioural equivalences, to see at which stage of abstraction various laws are recovered; such an approach might lead to the goal, suggested under chapter 0, of discovering an untimed concurrency theory which held to our philosophical position, but with rather more behavioural laws.

It would be interesting to know the class of nets we have ‘compiled’ IPA into; it would be interesting to discover, for instance, that all our expression correspond to confusion-free nets.

It should be possible to build ATLTSs directly from I.E.S.s using the notion of extends given in chapter three. This would extend the ATLTS framework to deal with I.E.S.s explicitly.

Chapter 5. The issues of specification and refinement, the former particularly, have been only very lightly touched upon. A proper treatment would rely on logic of specifications. It might be fruitful to investigate whether any of the real-time temporal logics known thus far are suitable for specifying I.E.S.s; the philosophical literature is so rich (e.g. [Allen 1985], [Goldblatt 1987], [van Benthem 1986]) that some progress should be possible.

The insight that various maximal I.E.S.s (the most sequential, or most nondeterministic, or most concurrent I.E.S. that simulates a given one) live in IIES, mentioned at the start of the postlude, should be exploited to give a refinement calculus for I.E.S.s with notions of both timewise and functional refinement. The problem of timing a given causality, subject to given timing constraints, is rather poorly understood, but should be accessible to attack from the I.E.S. model. This might lead to a useful calculus not only for describing, but also for designing timed distributed systems.

— FIN —
Bibliography

This bibliography is intended as a reasonably complete survey of literature in concurrency theory. The treatment of the field of non-interleaving semantics is, I hope, particularly comprehensive. For simplicity (and in order to impose some structure) the bibliography is organised by sections: these are Process algebras; Testing equivalence and Bisimulation; Programming languages; Applied category theory and Event structures; Specification languages; Petri nets and Non-interleaving semantics; Modal & temporal logics, Automata and Traces; Metric spaces and Topology; General concurrency theory and Miscellaneous topics; Real Time; Mathematical and Philosophical Tracts; Hardware. This scheme has the advantage of keeping unrelated works apart, but it does not necessarily keep related ones together; my apologies for any misclassifications. It may help the reader to note that where one reference is contained in another, (such as Berthelot [1987] G. Berthelot, Transformations and decompositions of nets. In [Brauer et al. 1986]) the second reference (here [Brauer et al. 1986]) will be found in the General concurrency theory and Miscellaneous topics section. Further, note that papers that relate to the non-interleaving semantics of process algebras will be found in the Petri nets and Non-interleaving semantics section not Process algebras.

The following abbreviations are used throughout the bibliography: LNCS = Lecture Notes in Computer Science, Springer Verlag, Berlin. TCS = Theoretical Computer Science, North-Holland. LICS = Logic in Computer Science. TAPSOFT = Theory and Practice of Software Development. ICALP = International conference on automata, languages and programming. POPL = Annual ACM symposium on the principles of programming languages.

Process algebras


Testing equivalence and Bisimulation


Programming languages


Applied category theory and Event structures


Specification languages


Petri nets and Non-interleaving semantics


Modal & temporal logics, Automata, and Traces


Metric spaces and Topology


General concurrency theory and Miscellaneous topics


Real Time


Mathematical and Philosophical Tracts


*Thomason* [1987] S. Thomason, *Free construction of time from events*. Manuscript, Department of Mathematics, Simon Fraser University, Burnaby, B.C.

Hardware


