Causal Automata

Jeremy Gunawardena,
Hewlett-Packard Laboratories,
Filton Road, Stoke Gifford,
Bristol BS12 6QZ.

http://www.jeremy-gunawardena.com/

January 29, 1991

Abstract

We introduce causal automata, a formalism based on a syntactic approach to causality in contrast to conventional approaches based on partial orders. Our main result is the following characterisation of Milner's notion of confluence in CCS:

$$\text{Confluence} \equiv \text{Determinism} + \{\mathsf{AND}, \mathsf{OR}\}\ \text{Causality}.$$

Contents

1 Introduction **1**
1.1 Determinism and Confluence 1
1.2 Causal Automata 4
1.3 AND, OR Causality 5
1.4 Restrictions, extensions and related work 6
1.5 Summary and acknowledgements 7

2 Causal automata **8**

3 AND, OR causality **11**

4 Deterministic automata **14**

5 Numbered traces **17**

6 Conclusion **19**

1 Introduction

In this paper we introduce causal automata, a formalism based on a syntactic approach to causality in contrast to conventional approaches based on partial orders. The motivation for this comes from Milner's notion of confluence in CCS, [15, Chapter 10], [16, Chapter 11]. This is a strong labelled version of the confluence that is customarily studied in term rewriting systems, [11]. Causal automata provide the following insight into confluence in Milner's sense:

$$\text{Confluence} \equiv \text{Determinism} + \{\mathsf{AND}, \mathsf{OR}\}\ \text{Causality}.$$

A precise statement is given in Theorem 1.2. We restrict attention to the finite case, for reasons which are discussed below. A more general and systematic study of causality will appear in a sequel to the present paper, [8].

Although the motivation for our results comes from CCS the results are stated and proved without any direct reference to Milner's calculus. We have, therefore, devoted an extended introductory section to giving an informal explanation of the origin of our ideas.

1.1 Determinism and Confluence

Milner's notion of confluence in CCS is closely related to, but distinct from, the customary idea of confluence in term rewriting systems. Huet, [11], has studied confluence in the abstract setting of a binary relation (reduction) on sets. For our purposes we can think of this as a transition system: a pair $(S, \rightarrow)$ where S is a set and $\rightarrow \subseteq S \times S$ is a transition relation. We use the notation $\stackrel{*}{\rightarrow}$ for the reflexive-transitive closure of $\rightarrow$.

Definition 1.1 *A transition system is confluent (in the general sense) if whenever $E \stackrel{*}{\rightarrow} F$ and $E \stackrel{*}{\rightarrow} G$, it is possible to find $H \in S$ such that $F \stackrel{*}{\rightarrow} H$ and $G \stackrel{*}{\rightarrow} H$.*

Confluence implies the Church-Rosser property: normal forms, if they exist, are unique. Newman, [17], seems to have been the first to use the word confluence. Keller, [13], pointed out the importance of confluence in several areas of parallel computation: parallel programme schemata, vector replacement systems, switching theory. Huet's elegant paper, [11], is itself a point of confluence for much earlier work.

Milner's notion of confluence in CCS, [16, §11.3], is a stronger labelled form of confluence. To describe it we need some notation. $\mathcal{A}^*$ will denote strings over the label set $\mathcal{A}$. The empty string will be denoted by ε, concatenation of strings by juxtaposition, st, or, when clarity dictates, by $s.t$. The excess of s over t, [16, Definition 6, §11.3], denoted s/t, is defined recursively by the rules below.

$$\begin{aligned} \varepsilon/t &= \varepsilon \\ (as)/t &= \begin{cases} a(s/t) & \text{if } a \text{ does not appear in } t \\ s/(t/a) & \text{otherwise.} \end{cases} \end{aligned}$$

For instance, $abadca/caab = badca/cab = adca/ca = dca/c = da$.

Consider labelled transition systems of the form $(S, \rightarrow, \mathcal{A}^*)$ where the labels are strings and the transition relation, $\rightarrow \subseteq S \times \mathcal{A}^* \times S$, is "closed": $E \stackrel{s}{\rightarrow} F$ and $F \stackrel{t}{\rightarrow} G$ implies $E \stackrel{st}{\rightarrow} G$. Such a system might come from taking the transitive closure of a labelled binary relation or by applying more complex transformations as in the relation $P \stackrel{s}{\Rightarrow} Q$ of CCS, [16, §5.1], where hidden actions are ignored.

Definition 1.2 *A labelled transition system is confluent (in the sense of Milner) if whenever* $E \stackrel{s}{\rightarrow} F$ *and* $E \stackrel{t}{\rightarrow} G$*, it is possible to find* $H \in S$ *such that* $F \stackrel{t/s}{\rightarrow} H$ *and* $G \stackrel{s/t}{\rightarrow} H$*.*

In this paper we use the word confluent in this stronger labelled sense. We should point out that the definition of a confluent process in CCS, [15, Proposition 11], also incorporates an appropriate equivalence relation on CCS terms, such as weak bisimulation. The definition given above is sufficient for our purposes; it makes apparent the relationship with confluence in the general sense. In the rest of this paper we shall be largely concerned with confluent trace sets.

Definition 1.3 *A set of strings,* $T \subseteq \mathcal{A}^*$*, is a confluent trace set if* T *is a trace set (ie:* $\varepsilon \in T$ *and* $rs \in T \Rightarrow r \in T$*) and,*

$$r, s \in T \Rightarrow r(s/r) \in T.$$

If a labelled transition system arises by transitive closure and is confluent in the sense of Definition 1.2, then for each element $E \in S$, the set of strings $\{s \mid \exists F \in S \text{ with } E \stackrel{s}{\rightarrow} F\}$ forms a confluent trace set.

Milner's introduction of confluence was motivated by his work on determinism. (We use the word determinism in preference to Milner's *determinacy*.) There seems to be some confusion about the exact meaning of this word. For us it will always mean that the past determines the present. This corresponds to the dictionary definition of determinism[1] and is the one adopted by Milner for CCS, [16, Chapter 11, Definition 3] (weak determinacy), and by Vaandrager, [22, §3.6], for event structures. Note that it differs from the one used by Aceto *et al* in [1, Definition 3.3]. Keller uses determinism in the same sense as us, [13, Definition 1.3(D)], but uses determinacy to mean something analogous to the Church-Rosser property.

By way of example, the CCS process $\tau.a.nil + \tau.b.nil$ is not deterministic, since to begin with it may sometimes be prepared to offer a and sometimes not. One frequently encounters the assertion that the process $a.b.nil + b.a.nil$ "exhibits sequential non-determinism". As far as we are concerned, this process is perfectly deterministic; it merely offers an initial choice between the actions a and b.

Deterministic systems are interesting for two main reasons. They occur widely in real-life and they are particularly easy to reason about: their behaviour is determined by trace-level information. This "folk theorem" has to be stated differently depending on the formalism which is used. Milner shows, [16, Chapter 11, Proposition 5], that two deterministic CCS processes are weakly bisimilar if, and only if, they have the same set of traces. Vaandrager shows, [22, §5.1], that two deterministic event structures are isomorphic if, and only if, they have the same set of step sequences.

The problem with determinism in CCS is that it lacks compositionality. If we place two deterministic CCS processes in parallel, the result need not be deterministic, as the example of $a.b.nil|a.c.nil$ shows. Similarly for most of the other algebraic operators in CCS. This is a serious difficulty, because we would like to infer the property of determinism from the syntactic structure of a CCS term. Milner's approach to this problem is to introduce the stronger property of confluence. Confluent processes are necessarily deterministic although not all deterministic processes are confluent. Confluence does exhibit some compositionality. Milner shows that certain derived operations in CCS, in particular a form of restricted parallel composition, do preserve confluence, [16, Chapter 11, Proposition 17]. This provides a method

[1] The philosophical doctrine that all events $\cdots$ are fully determined by preceding events. (Collins Dictionary of the English Language, Second Edition, 1986.)

for building up confluent, and hence deterministic, processes. Rem and others have considered a similar notion, conservatism, in CSP, [20], for much the same reasons.

Up to now we have made no mention of causality. Indeed causality is frequently associated with "true concurrency" in antagonism to the interleaving semantics of CCS, [19]. Perhaps one of the messages of this paper is that causality is a valuable tool with which to study any semantics, interleaving or otherwise.

We became interested, independently of Milner's work and for rather pragmatic reasons, [7, §3], in deducing the causal relationships between observable actions in a CCS process. Our methods led us to a subset of finite CCS processes which we called purely parallel, [6, 7], for which we proved the following result, [7, Theorem 3].

Theorem 1.1 *There exists a function γ, which associates to each purely parallel process, P, a finite deterministic labelled poset, $\gamma(P)$, such that $traces(P) = seq(\gamma(P))$. This function induces a one-to-one correspondence between equivalence classes, under weak bisimulation, of purely parallel processes and finite deterministic pomsets.*

The sequences of a labelled poset are the prefixes of its linearizations. A labelled poset, and the corresponding pomset, is deterministic if no two events with the same label are concurrent. This used to be referred to as "absence of auto-concurrency". It corresponds exactly to Vaandrager's definition of determinism if a poset is regarded as a conflict-free prime event structure.

This result gives a denotational semantics to the language of purely parallel processes which is fully abstract for weak bisimulation. (In fact, it gives rather more, and the extra information is quite valuable in practice, [7, §3.2].) If we note that posets, in the absence of additional structure, can only capture AND causality, we can summarise the result in the following informal way:

$$\text{Pure Parallelism} \equiv \text{Determinism} + \{\mathsf{AND}\}\ \text{Causality}.$$

While this work was in progress, Robin Milner pointed out to us that the operations used to build up purely parallel processes are exactly those which preserve confluence. This crucial observation raised the following question. Do we get all confluent processes, or at least the finite ones, in this way? The answer is no, not even up to weak bisimulation, as the following example shows:

$$C = (a.x.nil | b.x.nil | \overline{x}.c.nil) \backslash \{x\}.$$

It is not hard to prove that this process is confluent, [6, §7]. (Readers may like to convince themselves that $traces(C)$ is at least a confluent trace set.) If C were weakly bisimilar to a purely parallel process, Theorem 1.1 would imply the existence of a poset having the following set of sequences:

$$\{\varepsilon, a, b, ac, bc, ab, acb, bca, abc\}.$$

In such a poset, the events a and b would have to be initial but c could not be, since c does not appear initially. But if c were dependent on a then the sequence bca would not be possible; similarly if c were dependent on b the sequence acb would not be possible. Hence no such poset can exist. It follows that the process C is confluent but cannot be built, even upto weak bisimulation, by the operations which are known to preserve confluence. This counterexample is the starting point of this paper.

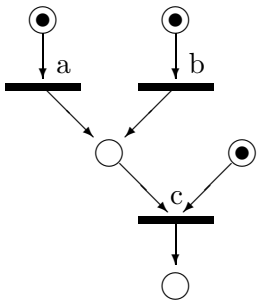

Figure 1: Net with OR causality

1.2 Causal Automata

It is instructive to follow the philosophy of [6] and to work out the causal relationships between the observable actions in the process C above. The actions a and b are causally independent and can appear in any order in the traces of C. However, c cannot appear until either a or b has appeared. The simplest, and, in our view, the most insightful way to describe this behaviour is to say that c depends causally on a OR b.

The idea of OR causality seems a very natural one. However, none of the formalisms used for specifying reactive systems, [9], seem to provide a convenient way to express it. For instance, example C is equivalent to one given by Winskel, [23, page 7], as an illustration of instability in event structures. Most authors, [1, 2, 3, 5, 18, 22, 23], have studied stable event structures and have found instability to be either unnecessary or undesirable. Of the available formalisms, Petri nets provide the most painless way to represent OR causality, as shown in Figure 1, albeit at the expense of resorting to unsafe nets. In this paper we would like to take causality more seriously by introducing causal automata, a formalism in which causal relationships like OR causality can be easily expressed.

Definition 1.4 *A causal automaton consists of a finite set, E, of events and a function, $\rho : E \to BE$, from E to the free Boolean algebra, BE, generated by E.*

The restriction to finite sets of events will hold throughout this paper. We discuss the difficulties with the infinite case in §1.4 below. Causal automata are conveniently specified in tabular form, as shown below. The first column gives the events e while the second gives $\rho(e)$.

$$C1 = \boxed{\begin{array}{ll} a & \mathsf{T} \\ b & \mathsf{T} \\ c & a \vee b \end{array}} \qquad C2 = \boxed{\begin{array}{ll} a & \neg b \\ b & \neg a \end{array}} \qquad C3 = \boxed{\begin{array}{ll} e_1 & e_3 \Rightarrow e_4 \\ e_2 & e_1 \\ e_3 & e_1 \Rightarrow e_2 \\ e_4 & e_3 \end{array}}$$

The Boolean expression $\rho(e)$ represents the causal "guard" which must become TRUE before the event e can occur. Initially, no events have occurred. This is expressed by setting each of the generators $e \in E$ to FALSE. The expressions $\rho(e)$ then simplify to some truth value. Those events for which $\rho(e)$ becomes TRUE are said to be enabled and may occur. The others, for which $\rho(e)$ evaluates to FALSE, cannot yet do so.

If an event e occurs, then it should be removed from the set E of events. However, the causal guards of the remaining events may still contain e. We may express the fact that e has occurred by setting e to TRUE, thereby removing the generator e from the values of the

function ρ. It should now be clear that we have constructed a new causal automaton F where $F = E - \{e\}$ and $\rho_F(x) = \rho_E(x)[e \to \mathsf{T}]$ for each $x \in F$. The automaton E has engaged in the event e and evolved into the automaton F.

The example $C1$ represents the same behaviour as the CCS process C described above. Setting each of a, b and c to FALSE, we see that only a and b are enabled. If either of these occur, then c will become enabled in the resulting automaton, since $(\mathsf{T} \vee p) \equiv \mathsf{T}$. (We use the symbol $\equiv$ to indicate equality in the Boolean algebra. At the propositional level, $p \equiv q$, if, and only if, $p \Leftrightarrow q$ is a tautology.)

The behaviour of $C3$ is rather different. If we set each of the events e_1, e_2, e_3 and e_4 to FALSE then e_1 and e_3 become enabled since $(\mathsf{F} \Rightarrow \mathsf{F}) \equiv \mathsf{T}$. Either of these events may occur. If e_1 occurs then $C3$ evolves into the automaton

$$D = \boxed{\begin{array}{ll} e_2 & \mathsf{T} \\ e_3 & e_2 \\ e_4 & e_3 \end{array}}$$

since $(\mathsf{T} \Rightarrow e_2) \equiv e_2$. The further behaviour of automaton D is straightforward: the events e_2, e_3 and e_4 are offered in that order. If, on the other hand, $C3$ were to engage in e_3 initially then the events e_4, e_1 and e_2, in that order, will be offered subsequently. Although the event e_3 is enabled in $C3$ it becomes disabled in D and remains so until e_2 has been offered.

Example $C2$ uses NOT causality. If both a and b are set to FALSE then both events are seen to be enabled and either may occur. However, if a occurs this has the effect of permanently disabling b and vice-versa. There is a close similarity between NOT causality and conflict in event structures.

The events in a causal automaton do not correspond directly to actions but rather to occurrences of actions. It is customary to use a labelling function, $\ell : E \to \mathcal{A}$, to associate an action with each event. As a general rule, the Latin letters $a, b, c, e_1, \cdots$ will denote events, while Greek letters $\alpha, \beta, \cdots$ will denote actions. In tabular descriptions of causal automata a third column will be used to indicate the labels of events, as shown below.

$$C4 = \boxed{\begin{array}{lll} e_1 & e_3 \Rightarrow e_4 & \alpha \\ e_2 & e_1 & \beta \\ e_3 & e_1 \Rightarrow e_2 & \alpha \\ e_4 & e_3 & \gamma \end{array}}$$

If this third column is omitted then the labelling function is deemed to be the identity and actions are identical with events.

1.3 AND, OR Causality

Our intention in this paper is not to give a systematic treatment of causal automata, leaving that task to Causal Automata II. Instead, we feel it is best to show first that they are genuinely useful. Returning to the problem of confluence, we remarked above that purely parallel processes only require AND causality, while example C uses OR causality. This suggests that we investigate automata which only use AND and OR causality. More precisely, we shall say that the automaton E is a $\{\wedge, \vee\}$-automaton if each $\rho(e)$ can be written in terms of the generators of E using only the operations $\wedge$ and $\vee$, and possibly also the constant T. $C1$ above is a $\{\wedge, \vee\}$-automaton while $C2$ and $C3$ are not. The main result of this paper is the following causal characterisation of confluence.

Theorem 1.2 *If E is a deterministic $\{\wedge, \vee\}$-automaton then the set of action traces of E is a (finite) confluent trace set. Conversely, every finite confluent trace set is the set of action traces of some deterministic $\{\wedge, \vee\}$-automaton.*

A deterministic automaton will be defined precisely in §2, but the discussion given in §1.1 should make it fairly clear what form the definition will take. Informally,

$$\text{Confluence} \equiv \text{Determinism} + \{\mathsf{AND}, \mathsf{OR}\}\ \text{Causality}.$$

This makes it clear how confluence differs from pure parallelism. It differs twice as much. Pure parallelism is one dimensional, as measured by AND causality; confluence has an extra, orthogonal dimension of variability expressed by OR causality.

This result provides a syntactic approach to confluence although the syntax expresses causal relationships between events rather than process constructions. It still leaves unresolved the problem of constructing all confluent processes in CCS. One approach might be to use the process C itself as a template for a ternary derived operation in CCS which would implement OR causality[2]. Such questions are beyond the scope of the present paper.

It may be felt that Theorem 1.2 has too much of the flavour of formal language theory. It says in effect that a confluent trace set is one which is accepted by a suitable automaton. The use of the word automaton probably reinforces this. We feel this restricted viewpoint is justified because of the deterministic nature of the problem.

It is perhaps worth pointing out that there are automata, which are not $\{\wedge, \vee\}$-automata, which nonetheless have confluent sets of action traces. Example $C5$ below

$$C5 = \boxed{\begin{array}{ll} a & b \Rightarrow c \\ b & a \\ c & b \end{array}}$$

has action traces $\{\varepsilon, a, ab, abc\}$, which is certainly confluent.

1.4 Restrictions, extensions and related work

The definition of a causal automaton allows only a finite set of events. In keeping with this, Theorem 1.2 applies only to finite confluent trace sets. Clearly, this is a serious restriction. What can we say about the infinite case?

The first point to note is that infinite sets of events allow the possibility of infinitary causal operations. In this respect infinitary OR is more palatable than infinitary AND and perhaps even essential for expressing unbounded choice. This discrepancy between AND and OR should warn us that the familiar dualities of classical propositional logic are threatened in the infinite case. Even more striking is the fact that there exists no free complete Boolean algebra on countably infinitely many generators, [12, §4.10]. These difficulties prompt us to reconsider the logic underlying our descriptions of causal relationships. Our use of classical propositional logic for causal automata is, to a great extent, a mechanical following of the path of least resistance. In the infinite case a more discriminating approach is required. In a sequel to the present paper, [8], we discuss the requirements for such a logic of causality and report on some promising experiments with geometric automata: automata based on geometric logic.

There remains the question of whether Theorem 1.2 holds for infinite trace sets. We believe that such a result does hold, albeit with some finiteness restriction such as finite branching of

[2]This suggestion comes from Chris Tofts.

the trace set. One way to approach such a result is to step back and consider a more general problem. How do our automata relate to other concurrency formalisms? Their closest relative is undoubtedly Winskel's event structures. We refer here to the most general kinds of event structure which Winskel has studied in [23, 24]. Both formalisms are based on an underlying set of events; they differ in the fuel which drives their behaviour. Event structures use a mixture of consistency and enablement, [23, Definition 1.1.1], while causal automata run on what one might call "pure causality". In [8] we show that Winskel's general event structures have a natural interpretation as geometric automata.

This general result allows us to return to the problem of confluent trace sets. We now have at our disposal the representation theorem for event structures[3] in Winskel's thesis, [4, 24]. From a confluent trace set we construct an appropriate domain which satisfies the conditions of Winskel's theorem. The finiteness restrictions are crucial here. Winskel's representation theorem gives an event structure. Our interpretation of event structures then unravels the causal relationships and gives a geometric automaton which only uses $\{\mathsf{AND}, \mathsf{OR}\}$ causality. We hope to give precise details of this argument in a future paper.

The connection between causal automata and Petri nets is also important. We feel that nets must contain, perhaps even conceal, all manner of causal relationships. Yakovlev, [25], has studied place/transition nets from an order-theoretic standpoint and has found that persistent nets give rise to $\{\mathsf{AND}, \mathsf{OR}\}$ - semi-modular - behaviour. It is interesting to note that the motivation for this study comes from the design of so-called speed independent circuits. Semi-modular phenomena were first studied by David Muller in his pioneering investigations into the design of asynchronous hardware, [14, Chapter 10]. The current resurgence of interest in the design of self-timed systems, [21], suggests that the study of $\{\mathsf{AND}, \mathsf{OR}\}$ causality is timely.

1.5 Summary and acknowledgements

The proof of Theorem 1.2 falls into two distinct parts. The heart of the argument is in §3 where the result is proved first for event traces. The labelling, and the property of determinism, are not relevant at this point. An important result in §3 is Lemma 3.1 which identifies the crucial characteristic property of $\{\mathsf{AND}, \mathsf{OR}\}$ causality. In §4 these initial results are used to prove the first part of the theorem. It is here that the assumption of determinism becomes essential. §5 contains the proof of the second part of Theorem 1.2. The details in §4 and §5 have a different flavour to those in §3 and largely concern the properties of strings and of labellings rather than of $\{\mathsf{AND}, \mathsf{OR}\}$ causality. §2 contains the initial definitions and some preparatory material on traces and confluence.

The work described here was undertaken as part of project VESPA at Hewlett-Packard's Information Systems Centre in Bristol, England. I am grateful to Claudio Condini and Miranda Mowbray for spotting various *faux pas*. Thanks are also due to Gérard Boudol, Matthew Hennessy, Ursula Martin, Robin Milner, Wolfgang Reisig, Chris Tofts and Alexandre Yakovlev for encouragement and suggestions. The comments of three anonymous referees, which led to several corrections and improvements in presentation, are gratefully acknowledged. Any indiscretions that remain are entirely my responsibility.

[3]The significance of which was pointed out to us by an anonymous ICALP referee.

2 Causal automata

This section covers the initial definitions and Lemmas needed for the proof of Theorem 1.2. We begin by recalling the definition given in the Introduction.

Definition 2.1 *A causal automaton is a triple (E, ρ, ℓ) where*

- *E is a finite set of events;*
- *$\rho : E \to BE$ is a function to the free Boolean algebra, BE, generated by E;*
- *$\ell : E \to \mathcal{A}$ is a labelling function to some set $\mathcal{A}$ of actions.*

We shall refer to an automaton by its set, E, of events and where necessary use subscripts, ρ_E and ℓ_E, to make it clear which causal automaton is being referred to.

Definition 2.2 *If S is some set of Boolean operations then E is an S-automaton if $\forall e \in E$, $\rho(e)$ can be expressed using only operations from the set S together with the constant T.*

A valuation, v, of a set E is a function $v : E \to \{\mathsf{T}, \mathsf{F}\}$ which assigns a truth value to each element of E. Note that any valuation on E lifts uniquely to a function, $BE \to \{\mathsf{T}, \mathsf{F}\}$, which is also denoted v. It is convenient to think of valuations as characteristic functions and to identify v with the subset of elements $\{e \in E \mid v(e) = \mathsf{T}\}$. With this identification, $v_1 \subseteq v_2$ indicates that if $v_1(e) = \mathsf{T}$, then $v_2(e) = \mathsf{T}$. Let v_F be the valuation which corresponds to the empty subset of E so that $v_\mathsf{F}(e) = \mathsf{F}$ for all $e \in E$.

Definition 2.3 *If (E, ρ, ℓ) is a causal automaton, the event $e \in E$ is enabled if $v_\mathsf{F}(\rho(e)) = \mathsf{T}$.*

The idea of enabled events permits the definition of an operational semantics for causal automata.

Definition 2.4 *If E_1, E_2 are causal automata and $e \in E_1$ is an event, then $E_1 \xrightarrow{e} E_2$ if, and only if,*

- *e is enabled in E_1;*
- *$E_2 = E_1 - \{e\}$;*
- *$\forall x \in E_2 \;\; \rho_{E_2}(x) = \rho_{E_1}(x)[e \to \mathsf{T}]$ (substitute T for e in $\rho_{E_1}(x)$);*
- *$\forall x \in E_2 \;\; \ell_{E_2}(x) = \ell_{E_1}(x)$.*

If $\alpha \in \mathcal{A}$ is an action, then $E_1 \xrightarrow{\alpha} E_2$ if, and only if, $E_1 \xrightarrow{e} E_2$ for some event e with $\ell(e) = \alpha$.

The event and action relations defined above extend in the usual way to relations over strings of events or strings of actions. If E, F are causal automata and $s = e_1 e_2 \cdots e_n \in E^*$ is a string of events, then $E \stackrel{s}{\Longrightarrow} F$ if there are causal automata $E_0, \cdots, E_n$ such that

$$E = E_0 \xrightarrow{e_1} E_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} E_n = F.$$

It is always true that $E \stackrel{\varepsilon}{\Longrightarrow} E$. If $E \stackrel{s}{\Longrightarrow} F$ for some $s \in E^*$ then s is said to be an event trace of E. The set of all event traces is $traces_\mathsf{E}(E)$.

Similar definitions hold for strings of actions. If E is a causal automaton over the labels $\mathcal{A}$, then $\sigma \in \mathcal{A}^*$ is an action trace of E if for some event trace $s = e_1 \cdots e_n$ we have $\sigma = \ell(e_1) \cdots \ell(e_n)$. If also $E \stackrel{s}{\Longrightarrow} F$ then we use the notation $E \stackrel{\sigma}{\Longrightarrow} F$ for the corresponding action trace. The set of action traces is $traces_{\mathsf{A}}(E)$. In general we shall use Latin letters $s, t, \cdots$ for event traces and Greek letters $\sigma, \tau, \cdots$ for action traces.

An event trace, in contrast to an action trace, is always a pure string: each event appears at most once in the string. Event traces hence give rise to subsets of events or to valuations. If $s \in traces_{\mathsf{E}}(E)$, let v_s denote the valuation corresponding to the subset of events in s.

$$v_s(e) \quad = \quad \begin{cases} \mathsf{T} & \text{if } e \text{ appears in } s \\ \mathsf{F} & \text{otherwise.} \end{cases}$$

Lemma 2.1 *Suppose that $E \stackrel{s}{\Longrightarrow} F$ for some event trace s of E. The event $a \in F$ is enabled in F if, and only if, $v_s(\rho_E(a)) = \mathsf{T}$ in E.*

Proof: If v_{F} denotes the all-FALSE valuation on F, it is clear from the operational semantics that $v_{\mathsf{F}}(\rho_F(a)) = v_s(\rho_E(a))$. The result follows.

QED

Recall that a trace set, T, over some alphabet A is a subset $T \subseteq A^*$ such that $\varepsilon \in T$ and if $st \in T$ then $s \in T$. It is clear that both $traces_{\mathsf{E}}(E)$ and $traces_{\mathsf{A}}(E)$ are trace sets. If T is a trace set and $s \in T$, then T/s denotes *"T after s"*, in the sense of [10, §1.8.3].

$$T/s = \{t \mid st \in T\}.$$

The symbol"/" is being used in two different senses in this paper depending on whether the left operand is a trace or a trace set. Both usages are sanctioned by custom and we hope that no confusion will result. T/s is undefined if $s \notin T$. Note that $T/\varepsilon = T$ and that $T/s_1s_2 = (T/s_1)/s_2$.

Lemma 2.2 *If E is a causal automaton and $E \stackrel{s}{\Longrightarrow} F$ for some event trace s of E then $traces_{\mathsf{E}}(F) = traces_{\mathsf{E}}(E)/s$.*

Proof: It is clear that $traces_{\mathsf{E}}(F) \subseteq traces_{\mathsf{E}}(E)/s$. For the other direction suppose that $t \in traces_{\mathsf{E}}(E)/s$ so that $st \in traces_{\mathsf{E}}(E)$. Then by definition, $E \stackrel{s}{\Longrightarrow} E_1 \stackrel{t}{\Longrightarrow} E_2$. Comparing E_1 and F, we see that they have both been obtained from E by removing the same set of events; namely, those events which appear in the string s. Similarly, the causality functions of both automata are obtained from that of E by replacing the same generators with T. Finally, the labelling functions are clearly identical. So $E_1 = F$ and $t \in traces_{\mathsf{E}}(F)$ as required.

QED

The excess operation defined in §1.1 has many interesting properties. The following are particularly useful, [16, §11.3].

$$\begin{array}{rcl} s/t & = & s \quad \text{if } s \text{ and } t \text{ have no elements in common} \\ s/t_1t_2 & = & (s/t_1)/t_2 \\ s_1s_2/t & = & (s_1/t).(s_2/(t/s_1)) \end{array}$$

Following the precedent of [16], we leave the proofs as exercises for the reader. In reasoning about the excess operation, it is useful to be able to count the number of occurrences of a particular element in the string. Multisets provide a convenient language for this.

Definition 2.5 *A multiset, f, over the set A is a function, $f : A \to \mathbf{N}$, from A to the natural numbers $\mathbf{N}$. The set of multisets over A is denoted $A^{\mathbf{N}}$.*

A string $s \in A$ gives rise to a multiset, $[s]$, over A as shown below.

$$\begin{array}{rcl} [\varepsilon](a) & = & 0 \ \forall a \in A \\ [bs](a) & = & \begin{cases} [s](a)+1 & \text{if } b = a \\ [s](a) & \text{otherwise.} \end{cases} \end{array}$$

A multiset $f \in A^{\mathbf{N}}$ is a set if $f(a) = 0$ or 1 for all $a \in A$. Note that a string $s \in A^*$ is pure if, and only if $[s]$ is a set. Furthermore, s and t are permutations of each other if, and only if, $[s] = [t]$. It is convenient to regard elements of A as strings of length 1, so that if $a \in A$ then $[a]$ is the multiset having the value 1 at a and 0 everywhere else. The usual operations and relations on numbers ($+$, $\leq$, $\cdots$) lift pointwise to operations on multisets.

The following result collects together some elementary properties of strings which will be used later. It makes clear in what sense the operation s/t is one of "excess".

Lemma 2.3 *If $s, t \in A^*$ are traces and $a, b \in A$ then*

1. $[st] = [s] + [t]$

2. $[t/b] = \begin{cases} [t] & \text{if } [t](b) = 0 \\ [t] - [b] & \text{otherwise} \end{cases}$

3. $[s/t](a) = \begin{cases} [s](a) - [t](a) & \text{if } [s](a) > [t](a) \\ 0 & \text{otherwise} \end{cases}$

4. $[s(t/s)] = [t(s/t)]$

Proof: *1.* By induction over the length of s. If $s = \varepsilon$ the result holds immediately. Suppose that $s = bs_1$ and the result holds for s_1. Choose $a \in A$. If $a = b$ then by the definition of the bracket function $[bs_1t](a) = [s_1t](a)+1$ while $[bs_1](a) = [s_1](a)+1$. By the inductive hypothesis $[s_1t](a) = [s_1](a) + [t](a)$. Hence, $[st](a) = [s](a) + [t](a)$. If $a \neq b$ then $[bs_1t](a) = [s_1t](a)$ and $[bs_1](a) = [s_1](a)$ and the result follows similarly. This proves *1.*

2. First suppose that b does not appear in t. In other words that $[t](b) = 0$. Then t and b have no element in common and by a remark made above, $t/b = t$. Hence, $[t/b] = [t]$ as required. Now suppose that b does appear in t. Then we can write t in the form ubv where we may suppose that b does not appear in u. Note that u may be the empty string. It follows that $t/b = ubv/b = (u/b).(bv/(b/u))$. But b and u have no elements in common and so $u/b = u$ and $b/u = b$. Hence $t/b = u(bv/b) = uv$. So we have $[t/b] = [u] + [v]$ while $[t] = [u] + [b] + [v]$. It follows that $[t/b] = [t] - [b]$ as required. This proves *2.*

3. We proceed by induction over the length of t. If $t = \varepsilon$ then $s/t = s$ for any s and the result is easily seen to hold. If $t = b$ where $b \in A$ then the result follows from part *2.* This starts the induction off. Now assume that $t = t_1t_2$, where neither t_1 nor t_2 is the empty string, and the result holds for t_1 and t_2 separately, with any s as the left hand argument.

Suppose first that $[s](a) > [t](a)$. Then since $[t] = [t_1] + [t_2]$ by part *1*, we must have $[s](a) > [t_1](a)$. Therefore, by the inductive hypothesis for t_1, $[s/t_1](a) = [s](a) - [t_1](a)$. Furthermore, $[s/t_1](a) > [t_2](a)$. Hence by the inductive hypothesis for t_2,

$$[(s/t_1)/t_2](a) = [s/t_1](a) - [t_2](a) = [s](a) - [t_1](a) - [t_2](a) = [s](a) - [t](a).$$

The required result follows since $s/t = (s/t_1)/t_2$.

Now suppose that $[s](a) \leq [t](a)$. If $[s](a) \leq [t_1](a)$, then by the inductive hypothesis for t_1, $[s/t_1](a) = 0$ and clearly $[s/t_1](a) \leq [t_2](a)$. Hence by the inductive hypothesis for t_2, $[s/t](a) = 0$ as required. If on the other hand, $[s](a) > [t_1](a)$ then by the inductive hypothesis for t_1, $[s/t_1](a) = [s](a) - [t_1](a)$. But, $[s/t_1](a) \not> [t_2](a)$ for otherwise $[s](a) > [t](a)$. So by the inductive hypothesis for t_2, $[(s/t_1)/t_2](a) = 0$ and so $[s/t](a) = 0$ as required.

4. Choose $a \in A$ and suppose that $[s](a) > [t](a)$. Then, by part *3*, $[s(t/s)](a) = [s](a) + 0$ while $[t(s/t)](a) = [t](a) + [s](a) - [t](a) = [s](a)$. Hence $[s(t/s)](a) = [t(s/t)](a)$. By symmetry of s and t the result follows.

QED

We can now prove some elementary properties of confluent trace sets.

Lemma 2.4 *Suppose that T is a confluent trace set. The following statements are true.*

1. *If $t \in T$, then T/t is also confluent.*
2. *If $p, q \in T$ and $[p] = [q]$ then $T/p = T/q$.*

Proof: *1.* Choose $r, s \in T/t$. Then $tr, ts \in T$ and since T is confluent $tr(ts/tr) \in T$. But $ts/tr = (ts/t)/r = ((t/t).(s/(t/t)))/r = s/r$ since, by Lemma 2.3(3), $t/t = \varepsilon$. It follows that $r(s/r) \in T/t$ and hence T/t is confluent.

2. If $[p] = [q]$ then it follows from Lemma 2.3(3) that $p/q = q/p = \varepsilon$. If $u \in T/p$ then $pu \in T$ and since T is confluent, $q(pu/q)$ is also in T. But, $pu/q = (p/q)(u/(q/p)) = u$. Hence $qu \in T$ and $u \in T/q$. So $T/p \subseteq T/q$ and by symmetry, $T/p = T/q$ as required.

QED

This completes the preparatory material. In the next section we begin the proof of the main results.

3 AND, OR causality

We begin this section with a simple observation about Boolean expressions built up using only AND and OR.

Lemma 3.1 *Suppose that $\rho \in BE$ is an element of the free Boolean algebra over E other than F. ρ can be expressed using only the operations T, $\wedge$ and $\vee$ if, and only if, for any valuations v_1, v_2 such that $v_1 \subseteq v_2$ and $v_1(\rho) = \mathsf{T}$, it follows that $v_2(\rho) = \mathsf{T}$.*

Proof: Suppose that ρ is expressed using only the operations T, $\wedge$ and $\vee$ and suppose given a pair of valuations such that $v_1 \subseteq v_2$ and $v_1(\rho) = \mathsf{T}$. Since $\mathsf{F} \leq \mathsf{T}$ in the partial order of the Boolean algebra and both $\wedge$ and $\vee$ are monotonic, it follows immediately that $v_1(\rho) \leq v_2(\rho)$. Hence $v_2(\rho) = \mathsf{T}$ as required.

Now suppose that $\rho \in BE$ satisfies the valuation property in the statement of the Lemma. Define a valuation w to be minimal with respect to ρ if $w(\rho) = \mathsf{T}$ but $v(\rho) = \mathsf{F}$ for any $v \subset w$. Since $\rho \not\equiv \mathsf{F}$, minimal valuations must exist.

It may happen that v_{F} is a minimal valuation for ρ. If v is any valuation, then $v(\rho) = \mathsf{T}$ since $v_{\mathsf{F}} \subseteq v$. Hence $\rho \equiv \mathsf{T}$ and there is nothing to prove.

So assume that v_{F} is not a minimal valuation for ρ. Note that any minimal valuation w must then have some $e \in E$ such that $w(e) = \mathsf{T}$. Let $\{w_1, w_2, \cdots, w_n\}$ be the set of minimal valuations of ρ and let e vary over elements of E. Define

$$\sigma = \bigvee_{1 \leq i \leq n} (\bigwedge_{w_i(e)=\mathsf{T}} (e))$$

which is well defined by the preceding remark. σ is expressed in terms of the generators of BE using only the operations $\wedge$ and $\vee$. We claim that $\rho \equiv \sigma$ in BE. It suffices to show that $v(\rho) = v(\sigma)$ for any valuation v.

Suppose given a valuation v. If $v(\rho) = \mathsf{T}$ then there must exist a (not necessarily unique) minimal valuation w_j of ρ such that $w_j \subseteq v$. But if $e \in E$ and $w_j(e) = \mathsf{T}$, then clearly $v(e) = \mathsf{T}$. Hence, $v(\bigwedge_{w_j(e)=\mathsf{T}}(e)) = \mathsf{T}$ and so $v(\sigma) = \mathsf{T}$.

Now suppose that $v(\sigma) = \mathsf{T}$. Then for some $1 \leq j \leq n$, $v(\bigwedge_{w_j(e)=\mathsf{T}}(e)) = \mathsf{T}$. Hence, for each $e \in E$ such that $w_j(e) = \mathsf{T}$ we must have $v(e) = \mathsf{T}$. In other words, $w_j \subseteq v$. But, $w_j(\rho) = \mathsf{T}$ and so, by the valuation property, $v(\rho) = \mathsf{T}$. Hence $\rho \equiv \sigma$. This completes the proof.

QED

We have spelt out this proof in some detail because the valuation behaviour is the only property of $\{\wedge, \vee\}$-causality which is subsequently used. It is therefore important to know that this property characterises $\{\wedge, \vee\}$-causality[4].

We shall say that a trace set is pure if every string in the set is pure; the event traces of any causal automaton always form a pure trace set.

Proposition 3.1 *If E is a $\{\wedge, \vee\}$-automaton then $traces_{\mathsf{E}}(E)$ is a (finite) pure confluent trace set.*

Proof: Choose $s, t \in traces_{\mathsf{E}}(E)$. We have to show that $t(s/t) \in traces_{\mathsf{E}}(E)$. The proof is by induction on the length of s. If $s = \varepsilon$ then $t(s/t) = t$ and there is nothing to prove. So suppose that $s = s_1 a$ and assume that $t(s_1/t)$ has been shown to be an event trace of E. We then have $s/t = (s_1/t).(a/(t/s_1))$. If a appears in t/s_1 then $a/(t/s_1) = \varepsilon$ and $s/t = s_1/t$. By the inductive hypothesis $t(s_1/t) \in traces_{\mathsf{E}}(E)$ and there is nothing more to do.

So assume that a does not appear in t/s_1 in which case $s/t = (s_1/t)a$. Since event traces are pure, it is clear that $[s_1](a) = 0$. By Lemma 2.3(3), $[t](a) = 0$ for otherwise a would appear in t/s_1. Furthermore, again by Lemma 2.3(3), $[s_1/t] \leq [s_1]$ and so $[s_1/t](a) = 0$. It follows that a does not appear in the trace $t(s_1/t)$.

By the inductive hypothesis, there is an automaton F such that $E \stackrel{t(s_1/t)}{\Longrightarrow} F$. Evidently, $a \in F$. All we have to do is to show that a is actually enabled in F. By Lemma 2.1, this amounts to showing that $v_{t(s_1/t)}(\rho_E(a)) = \mathsf{T}$.

Since s_1 is also an event trace, there is an automaton G such that $E \stackrel{s_1}{\Longrightarrow} G$. By Lemma 2.2, we know that $a \in traces_{\mathsf{E}}(G)$ and must hence be enabled in G. By Lemma 2.1, $v_{s_1}(\rho_E(a)) = \mathsf{T}$.

But now, from Lemma 2.3(4), $[s_1] \leq [t(s_1/t)]$. The events which appear in each of these traces are precisely those which have the value T in the corresponding valuations. Hence, $v_{s_1} \subseteq v_{t(s_1/t)}$ and the required result follows immediately from Lemma 3.1.

QED

[4]The significance and importance of this result becomes much clearer in the infinite case, [8].

The next result is a converse to Proposition 3.1. It it the central part of the proof of Theorem 1.2. Since we are not at present interested in action traces, the labellings on our causal automata are irrelevant and we shall assume that they are always the identity function.

Proposition 3.2 *If T is a finite pure confluent trace set then there exists a $\{\wedge, \vee\}$-automaton, E, such that $traces_{\mathsf{E}}(E) = T$.*

Proof: It is convenient for the proof to enlarge the conclusions of the Proposition. Suppose that A is the set of all elements which appear in the strings of T, so that $T \subseteq A^*$ and A is the smallest set with this property. What we shall prove is that if T is pure and confluent then there is a $\{\wedge, \vee\}$-automaton, of the specific form (A, ρ), such that $traces_{\mathsf{E}}(A) = T$.

The proof is by induction over the size of the trace set T. If $T = \{\varepsilon\}$ then the empty causal automaton has both the right form and the right traces. This starts off the induction. Now choose T, a non-trivial pure confluent trace set and suppose that all pure confluent trace sets of strictly smaller size have been shown to fulfil the conclusion stated in the previous paragraph. Let $e_1, e_2, \cdots, e_n$ be all the events which appear initially on some string of T. In other words, $e_i \in T$ for $1 \leq i \leq n$ and if $t \in T$ and $t \neq \varepsilon$ then $t = e_i s$ for some i. By Lemma 2.4(1), T/e_i is a confluent trace set which is clearly pure and of strictly smaller size to T. By the inductive hypothesis there exists a $\{\wedge, \vee\}$-automaton A_i such that $traces_{\mathsf{E}}(A_i) = T/e_i$. We shall use ρ_i to denote the causality function for the i'th automaton, $\rho_i : A_i \to BA_i$.

Let $f_1, f_2, \cdots, f_m$ be the elements other than $e_1, e_2, \cdots, e_n$ which appear in strings of T and, as above, let

$$A = \{e_1, e_2, \cdots, e_n, f_1, f_2, \cdots, f_m\}.$$

We need to construct a causal automaton on this specific set of events.

It is clear by the purity of T that the event e_i does not appear on any string in T/e_i. However, by confluence of T, it is easy to see that every other event in A must appear in some string of T/e_i. It follows from the expanded inductive hypothesis that $A_i = A - \{e_i\}$. In particular, $\rho_i(f_j)$ is defined for each $1 \leq i \leq n$ and $1 \leq j \leq m$ and we may take it that $\rho_i(f_j) \in BA$. This allows us to define the following $\{\wedge, \vee\}$-automaton (A, ρ) of the required form.

$$\begin{array}{lcll} \rho(e_i) & = & \mathsf{T} & 1 \leq i \leq n \\ \rho(f_j) & = & \bigvee_{1 \leq i \leq n}(e_i \wedge \rho_i(f_j)) & 1 \leq j \leq m. \end{array}$$

We claim that $traces_{\mathsf{E}}(A) = T$.

The first point to note is that the enabled events of A are precisely e_1, e_2, $\cdots$, e_n, for clearly $v_{\mathsf{F}}(\rho(f_j)) = \mathsf{F}$ for any $1 \leq j \leq m$. Hence there are causal automata B_i, for $1 \leq i \leq n$, such that $A \xrightarrow{e_i} B_i$. Moreover, it is clear that B_i and A_i have the same sets of events, namely $A - \{e_i\}$. We would be done if it were true that $B_i = A_i$ as causal automata. It is important to realise that this need not be the case. However, what we shall show is that $traces_{\mathsf{E}}(B_i) = traces_{\mathsf{E}}(A_i)$. It follows easily from Lemma 2.2 that this is sufficient to show $traces_{\mathsf{E}}(A) = T$. The remainder of the proof is hence concerned with showing that $traces_{\mathsf{E}}(B_i) = traces_{\mathsf{E}}(A_i)$. Since the proof is the same for each index i it will be convenient and no less general to do the case $i = 1$. This will cut down on the number of indices which appear in the text.

We shall start by showing $traces_{\mathsf{E}}(A_1) \subseteq traces_{\mathsf{E}}(B_1)$. The proof is by induction on the length of strings in $traces_{\mathsf{E}}(A_1)$. It is clear that $\varepsilon \in traces_{\mathsf{E}}(B_1)$ which starts off the induction. So suppose that $t \in traces_{\mathsf{E}}(A_1)$ is a non-empty string and that all strings in $traces_{\mathsf{E}}(A_1)$ of strictly smaller length have been shown to be in $traces_{\mathsf{E}}(B_1)$. We can write $t = se$ where $e \in A_1$ is an event and $s \in traces_{\mathsf{E}}(A_1)$ is a string of strictly smaller length than t. By the

inductive hypothesis $s \in traces_{\mathsf{E}}(B_1)$ and $e_1 s \in traces_{\mathsf{E}}(A)$. If e is one of the e_i, it will be permanently enabled and clearly $se \in traces_{\mathsf{E}}(B_1)$ and there is nothing to prove. So assume that e is one of the f_j. From the description of the automaton A given above, it is easy to see that

$$v_{e_1 s}(\rho(e)) = v_s(\rho_1(e)) \vee (\ \cdots\ )$$

where the exact nature of the omitted elements is not relevant.

Now by assumption, $se \in traces_{\mathsf{E}}(A_1)$, and so by Lemma 2.1, $v_s(\rho_1(e)) = \mathsf{T}$. It follows immediately that $v_{e_1 s}(\rho(e)) = \mathsf{T}$. Hence by Lemma 2.1 $e_1 se \in traces_{\mathsf{E}}(A)$ and so by Lemma 2.2, $t = se \in traces_{\mathsf{E}}(B_1)$. If follows by induction that $traces_{\mathsf{E}}(A_1) \subseteq traces_{\mathsf{E}}(B_1)$.

It remains to prove the opposite inclusion, $traces_{\mathsf{E}}(B_1) \subseteq traces_{\mathsf{E}}(A_1)$. As before the proof is by induction on the length of strings. Suppose that $t \in traces_{\mathsf{E}}(B_1)$ is a non-empty string and all strings of strictly smaller length in $traces_{\mathsf{E}}(B_1)$ have been shown to be in $traces_{\mathsf{E}}(A_1)$. We may write $t = se$ where $e \in B_1$ and $s \in traces_{\mathsf{E}}(B_1)$ is a string of strictly smaller length to which the inductive hypothesis applies. Hence $s \in traces_{\mathsf{E}}(A_1)$. Suppose first that e is one of the e_i. Because T is confluent, $e \in T/e_1$ and hence e must be enabled in A_1. By Lemma 3.1, it must be the case that $\rho_1(e) = \mathsf{T}$. Therefore e is permanently enabled in A_1 and clearly $se \in traces_{\mathsf{E}}(A_1)$.

Now suppose that e is one of the f_j. Because $se \in traces_{\mathsf{E}}(B_1)$, we know from Lemma 2.1 that $v_{e_1 s}(\rho(e)) = \mathsf{T}$ in BA. Note that the Boolean expression $\rho_i(e) \in BA_i$ does not involve the generator e_i. Furthermore, if e_i appears in s then by purity we may write $(e_1 s)/e_i = e_1(s/e_i)$. This allows us to simplify the expression for $\rho(e)$ as shown below.

$$v_{e_1 s}(\rho(e)) = v_s(\rho_1(e)) \ \ \vee \ \ \left(\bigvee_{[s](e_i)=1} v_{e_1(s/e_i)}(\rho_i(e)) \right).$$

Hence, either $v_s(\rho_1(e)) = \mathsf{T}$ or there must be some k, with $[s](e_k) = 1$, such that $v_{e_1(s/e_k)}(\rho_k(e)) = \mathsf{T}$ in A_k.

If the former then, by Lemma 2.1, $se \in traces_{\mathsf{E}}(A_1)$ and we are done. So suppose the latter. Since $e_1 s$ and e_k are both in T, and T is confluent, it follows that $e_k((e_1 s)/e_k) = e_k e_1(s/e_k) \in T$. Hence, $e_1(s/e_k) \in T/e_k$ and so by Lemma 2.1 applied to A_k we see that $e_1(s/e_k)e \in T/e_k$. It follows that $e_k e_1(s/e_k)e \in T$ which we may rewrite as saying that $e \in T/(e_k e_1(s/e_k))$.

Note that $e_1 s(e_k/(e_1 s)) = e_1 s$, since e_k appears in s, and so by Lemma 2.3(4), $[e_1 s] = [e_k e_1(s/e_k)]$. Since T is confluent, Lemma 2.4(2) tells us that $e \in T/(e_1 s)$. In other words, $se \in T/e_1$. Hence $t \in traces_{\mathsf{E}}(A_1)$ as required. It follows by induction that $traces_{\mathsf{E}}(B_1) \subseteq traces_{\mathsf{E}}(A_1)$.

We have shown that for each $1 \leq i \leq n$, $T/e_i = traces_{\mathsf{E}}(B_i)$ and it follows as remarked above that $T = traces_{\mathsf{E}}(A)$. Since A is a $\{\wedge, \vee\}$-automaton on the correct set of events, the Proposition follows by induction.

QED

4 Deterministic automata

We now consider how to extend the results above to action traces. The first point to note is that Proposition 3.1 is false, as stated, if action traces are used in place of event traces.

Consider the following $\{\wedge, \vee\}$-automaton.

$$C6 = \boxed{\begin{array}{lll} a & \mathsf{T} & \alpha \\ b & a & \beta \\ c & \mathsf{T} & \alpha \\ d & c & \gamma \end{array}}$$

It is easy to see that $\alpha\beta$ and $\alpha\gamma$ are action traces of $C6$. But $\alpha\gamma/\alpha\beta = \gamma$ and the string $\alpha\beta\gamma = \alpha\beta(\alpha\gamma/\alpha\beta)$ is not an action trace of $C6$. Hence $traces_{\mathsf{A}}(C6)$ is not confluent.

The missing ingredient is determinism. The following definition is a natural extension of Vaandrager's definition, [22, §3.6], of determinism in prime event structures.

Definition 4.1 *A causal automaton E is deterministic if, whenever $E \stackrel{s}{\Longrightarrow} F$ for some event trace s of E, then all enabled events in F have distinct labels.*

If F does have two enabled events with the same label, a, and σ is the action trace corresponding to s, then we may have $E \stackrel{\sigma a}{\Longrightarrow} E_1$ and $E \stackrel{\sigma a}{\Longrightarrow} E_2$ where E_1 and E_2 are distinct. This is contrary to the intuition for determinism presented in §1.1.

The labelling function ℓ of an automaton A gives rise to a function on traces

$$\ell^* : traces_{\mathsf{E}}(E) \to traces_{\mathsf{A}}(E).$$

This is, of course, always a surjection: every action trace arises from some underlying event trace. It is not in general a bijection. However, the reader will easily see that the following Lemma is merely a restatement of the definition above.

Lemma 4.1 *An automaton (E, ρ, ℓ) is deterministic if, and only if, ℓ^* is a bijection.*

For $\{\wedge, \vee\}$-automata we can say rather more. It will be convenient to allow ℓ^* to act, in the obvious way, on elements of E^* which are not necessarily event traces of E. Note that, with this proviso, ℓ^* always commutes with juxtaposition: $\ell^*(st) = \ell^*(s)\ell^*(t)$.

Lemma 4.2 *Suppose the (E, ρ, ℓ) is an $\{\wedge, \vee\}$-automaton. E is deterministic if, and only if, ℓ^* commutes with excess. (It is not asserted, and it is not in general true, that $traces_{\mathsf{E}}(E)$ is closed under the excess operation.)*

Proof: Suppose first that E is deterministic. Choose $s, t \in traces_{\mathsf{E}}(E)$. We have to show that $\ell^*(s/t) = \ell^*(s)/\ell^*(t)$. The proof is by induction on the length of the second argument t. If $t = \varepsilon$ then $s/\varepsilon = s$ and $\ell^*(t) = \varepsilon$ so clearly $\ell^*(s/t) = \ell^*(s) = \ell^*(s)/\ell^*(t)$.

Now suppose that $t = a$ where a is an event in E and let $\ell(a) = \alpha$. There are two cases according as $[s](a) = 0$ or $[s](a) > 0$. Suppose the first, so that a does not appear in s.

We claim that then α does not appear in $\ell^*(s)$. Suppose on the contrary that it does and write $\ell^*(s) = \mu.\alpha.\nu$ where $[\mu](\alpha) = 0$. We can similarly write $s = m.e.n$ where $\ell^*(m) = \mu$, $\ell(e) = \alpha$ and $\ell^*(n) = \nu$. It is clear that a does not appear in m for otherwise α would necessarily appear in μ. Since $me \in traces_{\mathsf{E}}(E)$ it follows from Lemma 2.1 that $v_m(\rho(e)) = \mathsf{T}$. Also $v_{\mathsf{F}}(\rho(a)) = \mathsf{T}$ since a is enabled in E. But $v_{\mathsf{F}} \subseteq v_m$ and so, by Lemma 3.1, $v_m(\rho(a)) = \mathsf{T}$. Hence, by Lemma 2.1 again, if $E \stackrel{m}{\Longrightarrow} F$ then both a and e are enabled in F. But $\ell(a) = \ell(e) = \alpha$ which contradicts the fact that E is deterministic. This shows that α does not appear in $\ell^*(s)$. It follows from Lemma 2.3(2) that both $s/a = s$ and $\ell^*(s)/\ell^*(a) = \ell^*(s)$. Hence, $\ell^*(s/a) = \ell^*(s)/\ell^*(a)$ as required.

Now suppose that a does appear in s. We may write $s = m.a.n$ where $[m](a) = 0$. Then $\ell^*(s) = \ell^*(m).\alpha.\ell^*(n)$ and by the preceding paragraph $[\ell^*(m)](\alpha) = 0$. It is then easy to see that $s/a = mn$ and, by a similar argument, that $\ell^*(s)/\alpha = \ell^*(m)\ell^*(n)$. It follows that $\ell^*(s/a) = \ell^*(s)/\ell^*(a)$ as required.

This argument suffices to start off the induction. Suppose now that we have $t \in traces_{\mathsf{E}}(E)$ of length greater than 1 and that the result has been shown to hold (for arbitrary first argument) for all event traces of length strictly less than t. We can write $t = t_1t_2$ where both t_1 and t_2 have length strictly less than t. Then $\ell^*(s/(t_1t_2)) = \ell^*((s/t_1)/t_2) = \ell^*(s/t_1)/\ell^*(t_2)$ by the inductive hypothesis applied to t_2. Proceeding further, by the inductive hypothesis applied to t_1, $\ell^*(s/t_1) = \ell^*(s)/\ell^*(t_1)$. Hence, $\ell^*(s/t) = (\ell^*(s)/\ell^*(t_1))/\ell^*(t_2) = \ell^*(s)/(\ell^*(t_1)\ell^*(t_2)) = \ell^*(s)/\ell^*(t)$ as required. It follows by induction that ℓ^* commutes with excess.

For the other direction suppose that ℓ^* commutes with excess. Suppose further that there are $s, t \in traces_{\mathsf{E}}(E)$ for which $\ell^*(s) = \ell^*(t)$ but $s \neq t$.

For any string u let u_k, where $k \geq 0$, denote the prefix of u of length k and let $|u|$ denote the length of the string u. Note that $\ell^*(s_k) = \ell^*(t_k)$ for $0 \leq k \leq |s|$ and that $|s| = |t|$. Now there must be some i with $1 \leq i \leq |s|$ such that $[s_i] \neq [t_i]$ for if not it is easy to show by induction that $s = t$.

Consider s_i/t_i. If this were the empty string, ε, then by Lemma 2.3, part *3*, $[s_i] \leq [t_i]$ and since s_i and t_i have the same length, this would force $[s_i] = [t_i]$. Hence $s_i/t_i \neq \varepsilon$. But, however, $\ell^*(s_i) = \ell^*(t_i)$ and so $\ell^*(s_i)/\ell^*(t_i) = \varepsilon$. It follows that $\ell^*(s_i/t_i) \neq \ell^*(s_i)/\ell^*(t_i)$ which contradicts the assumption that ℓ^* commutes with excess. It follows that ℓ^* must be bijective and hence, by Lemma 4.1, that E must be deterministic.

QED

It is worth pointing out that the second part of the proof above does not use the fact that E is a $\{\wedge, \vee\}$-automaton. Note that there are deterministic automata, which are not $\{\wedge, \vee\}$-automata, for which ℓ^* does not commute with excess. For instance, the automaton

$$C7 = \boxed{\begin{array}{lll} a & b \Rightarrow e & \alpha \\ b & \mathsf{T} & \beta \\ e & b & \alpha \end{array}}$$

has the following sets of traces:

$$\begin{array}{rcl} traces_{\mathsf{E}}(C7) & = & \{\varepsilon, a, b, ab, be, abe, bea\}, \\ traces_{\mathsf{A}}(C7) & = & \{\varepsilon, \alpha, \beta, \alpha\beta, \beta\alpha, \alpha\beta\alpha, \beta\alpha\alpha\}. \end{array}$$

It is clear that ℓ^* is bijective since the trace sets have the same size. Hence, by Lemma 4.1, $C7$ is deterministic. However, $\ell^*(ab/be) = \ell^*(a) = \alpha$, while $\ell^*(ab)/\ell^*(be) = \alpha\beta/\beta\alpha = \varepsilon$. So ℓ^* does not commute with excess.

It is now a simple matter to prove the first part of Theorem 1.2.

Theorem 4.1 *If E is a deterministic $\{\wedge, \vee\}$-automaton then $traces_{\mathsf{A}}(E)$ is a finite confluent trace set.*

Proof: Choose action traces $\mu, \nu \in traces_{\mathsf{A}}(E)$. Let $m, n \in traces_{\mathsf{E}}(E)$ be event traces such that $\ell^*(m) = \mu$ and $\ell^*(n) = \nu$. By Proposition 3.1, $traces_{\mathsf{E}}(E)$ is confluent and so certainly $m(n/m) \in traces_{\mathsf{E}}(E)$. Hence $\ell^*(m(n/m))$ is an action trace. But now Lemma 4.2 shows that $\ell^*(m(n/m)) = \mu(\nu/\mu)$. So $\mu(\nu/\mu) \in traces_{\mathsf{A}}(E)$ and $traces_{\mathsf{A}}(E)$ is confluent. This completes the proof.

QED

5 Numbered traces

An arbitrary confluent set is not in general pure. In this section we present a method for constructing a pure trace set from any given trace set. The main result is that this method preserves confluence. This allows us to use Proposition 3.2 to prove the second part of Theorem 1.2. Let $\mathbf{N}^+$ denote the positive natural numbers.

Definition 5.1 *Given a set A, the numbering function, $\mu : A^* \times A^* \to (A \times \mathbf{N}^+)^*$ is given inductively by the equations*

$$\begin{aligned} \mu(r, \varepsilon) &= \varepsilon \\ \mu(r, sa) &= \mu(r, s).(a, [r](a) + [s](a) + 1) \ . \end{aligned}$$

We shall use the short-hand: $\mu_r(s) = \mu(r, s)$ and $\mu(s) = \mu_\varepsilon(s)$. It is easy to see that if $[p] = [q]$ then $\mu_p = \mu_q$, so that μ factors over $A^{\mathbf{N}} \times A^*$. As an example of a numbered string, if $t = aabacaccb$, then $\mu(t)$ is

$$(a,1)(a,2)(b,1)(a,3)(c,1)(a,4)(c,2)(c,3)(b,2).$$

The effect of the first argument is to translate the numbering: if $r = bcbecc$ then $\mu_r(t)$ is

$$(a,1)(a,2)(b,3)(a,3)(c,4)(a,4)(c,5)(c,6)(b,4).$$

Unlike t, both these strings are pure. The numbering function satisfies many interesting identities and the next result collects together those which are of immediate benefit to us. To avoid unnecessary complication, we have chosen not to state these in their most general form.

Lemma 5.1 *Let A be a set and $r, s \in A^*$ be strings over A. The following statements are true.*

1. $[\mu(s)](a,k) = \begin{cases} 0 & \text{if } k > [s](a) \\ 1 & \text{otherwise} \end{cases}$.
2. $\mu(rs) = \mu(r)\mu_r(s)$.
3. $\mu_s(r/s) = \mu(r)/\mu(s)$.
4. $\mu(r(s/r)) = \mu(r)(\mu(s)/\mu(r))$.

Proof: *1.* By induction on the length of s. If $s = \varepsilon$ then from the definition $\mu(s) = \varepsilon$. Hence $[\mu(s)](a,k) = 0$. Since $k > 0$ by convention, certainly $k > [s](a)$ and so the result is true. Now suppose that $s \in A^*$ is non-empty and the result has been shown to hold for all strings of length strictly less than s. We can write $s = tb$ where $b \in A$ and the inductive hypothesis applies to t. Then, by definition, $\mu(s) = \mu(t).(b, [t](b) + 1)$.

Consider the two cases separately. If $k > [s](a)$ then certainly $k > [t](a)$ and so by the inductive hypothesis $[\mu(t)](a,k) = 0$. Hence $[\mu(s)](a,k) = [(b, [t](b) + 1)](a,k)$. If $a \neq b$ then this is certainly zero. If $a = b$ then still $k > [s](a) = [t](a) + 1$. Hence $[\mu(s)](a,k) = 0$ as required.

Now suppose $k \leq [s](a)$. If also $k \leq [t](a)$ then by the inductive hypothesis $[\mu(t)](a,k) = 1$. Since $(b, [t](b)+1) \neq (a,k)$ it follows that $[\mu(s)](a,k) = 1$ as required. If, on the other hand, $k > [t](a)$ then again by the inductive hypothesis $[\mu(t)](a,k) = 0$. However, $[s](a) = [t](a) + [b](a)$,

so it must be the case that $a = b$ and $k = [s](a)$. But then $(b, [t](b) + 1) = (a, k)$ and so $[\mu(s)](a, k) = 1$ as required.

2. By induction on the length of s. If $s = \varepsilon$ then the result is easily seen to hold. So suppose that s is a non-empty string and that the result has been shown to hold for all strings of strictly smaller length and arbitrary r. We may write $s = ta$ where $a \in A$ and the inductive hypothesis applies to t. Then, $\mu(rs) = \mu((rt)a) = \mu(r)\mu_r(t)(a, [rt](a) + 1)$. On the other hand, by definition, $\mu_r(ta) = \mu_r(t).(a, [r](a) + [t](a) + 1)$. It follows from Lemma 2.3(1) that $\mu(r.s) = \mu(r).\mu_r(s)$.

3. By induction on the length of r. If $r = \varepsilon$ then the result is easily seen to hold. So suppose that r is a non-empty string and that the result holds for all strings of length strictly less than r and arbitrary s. We may write $r = ta$ where $a \in A$ and the inductive hypothesis applies to t. Since $ta/s = (t/s)(a/(s/t))$, it is easy to see from Lemma 2.3(3) that

$$ta/s = \begin{cases} (t/s)a & \text{if } [s](a) \leq [t](a) \\ (t/s) & \text{otherwise} \end{cases}$$

We shall consider each of these cases separately. First suppose that $[s](a) \leq [t](a)$. Then,

$$\mu_s(ta/s) = \mu_s((t/s)a) = (\mu(t)/\mu(s))(a, [s](a) + [t/s](a) + 1).$$

On the other hand,

$$\mu(ta)/\mu(s) = \mu(t)(a, [t](a) + 1)/\mu(s)$$

But, by part *1*, $[\mu(s)](a, [t](a)+1) = 0$ since $[t](a)+1 > [s](a)$. Similarly, $[\mu(t)](a, [t](a)+1) = 0$ since $[t](a) + 1 > [t](a)$. Hence, $[\mu(s)](a, [t](a) + 1) \leq [\mu(t)](a, [t](a) + 1)$. So by the previous paragraph, applied to numbered strings, we see that $\mu(ta)/\mu(s) = (\mu(t)/\mu(s))(a, [t](a) + 1)$. Finally, by Lemma 2.3(3), $[t/s](a) = [t](a) - [s](a)$. So $\mu_s(r/s) = \mu(r)/\mu(s)$ as required.

Now suppose that $[s](a) > [t](a)$. Then $\mu_s(ta/s) = \mu_s(t/s) = \mu(t)/\mu(s)$. Also $\mu(ta)/\mu(s) = (\mu(t)(a, [t](a) + 1))/\mu(s)$. But now, by part *1*, $[\mu(s)](a, [t](a) + 1) = 1$ since $[t](a) + 1 \leq [s](a)$ and similarly $[\mu(t)](a, [t](a) + 1) = 0$. Hence, $[\mu(s)](a, [t](a) + 1) > [\mu(t)](a, [t](a) + 1)$ and, as remarked above, this shows that $(\mu(t)(a, [t](a) + 1))/\mu(s) = \mu(t)/\mu(s)$. Hence $\mu_s(r/s) = \mu(r)/\mu(s)$ as required.

4. By parts *2* and *3*, $\mu(r(s/r)) = \mu(r)\mu_r(s/r) = \mu(r)(\mu(s)/\mu(r))$.

QED

Let $\pi : A \times \mathbf{N}^+ \to A$ be projection on to the first component of a numbered pair: $\pi(a, k) = a$. This extends in the usual way to a function on strings: $\pi^* : (A \times \mathbf{N}^+)^* \to A^*$. It is easy to see from the definition of the numbering function that $\pi^*(\mu_r(s)) = s$. We can now prove the result which we have been working towards.

Theorem 5.1 *If T is a finite confluent trace set, there exists a deterministic $\{\wedge, \vee\}$-automaton, E, such that $traces_{\mathsf{A}}(E) = T$.*

Proof: Let A be the set of elements which appear in the traces of T and let $\mu(T) \subseteq (A \times \mathbf{N}^+)^*$ be the corresponding set of numbered traces: $\mu(T) = \{\mu(t) \text{ such that } t \in T\}$. It follows from Lemma 5.1(1) that $\mu(T)$ is pure and from Lemma 5.1(4) that $\mu(T)$ is confluent. Hence, by Proposition 3.2, there exists a $\{\wedge, \vee\}$-automaton E such that $traces_{\mathsf{E}}(E) = \mu(T)$.

In the construction of E the labelling function was not relevant and was taken to be the identity. We are at liberty to provide an alternative. It is clear that $E \subseteq A \times \mathbf{N}^+$ so we can take

$\ell_E = \pi|_E$, the restriction to E of the projection function defined above. If $\mu(t) \in traces_{\mathsf{E}}(E)$ then $\ell_E^*(\mu(t)) = \pi^*(\mu(t)) = t$. It follows that $traces_{\mathsf{A}}(E) = T$. It is also clear that ℓ_E^* is bijective and hence, by Lemma 4.1, that E is deterministic. The result follows.

QED

This completes the proof of Theorem 1.2.

6 Conclusion

We have introduced causal automata, a formalism based on a syntactic approach to causality, and we have used it to shed light upon Milner's notion of confluence in CCS. The formalism takes a less impoverished view of causality than is customary in models based upon partial orders. We feel that it has the virtue of simplicity and that the operational behaviour of an automaton is particularly easy to understand, at least at a superficial level. A more systematic study of causality which tackles the difficulties with infinite sets of events will appear in a sequel to the present paper, [8].

References

[1] Aceto L, De Nicola R, Fantechi A. Testing Equivalences for Event Structures. In: Proceedings Advanced School, Rome. Springer LNCS 280.

[2] Boudol G, Castellani I. Permutation of Transitions: an Event Structure Semantics for CCS. INRIA Research Report No 798. February 1988.

[3] Degano P, De Nicola R, Montanari U. Partial Orderings Descriptions and Observations of Nondeterministic Concurrent Processes. In: Proceedings REX Workshop. Springer LNCS 354, 1989.

[4] Droste M. Event Structures and Domains. Theoretical Computer Science, 1989;68: 37-47.

[5] van Glabbeek R, Goltz U. Equivalence Notions for Concurrent Systems and Refinement of Actions. Arbeitspapiere der GMD 366. February 1989.

[6] Gunawardena J. Purely Parallel Processes. Technical Memo HPL-89-002, Hewlett-Packard Laboratories, March 1989.

[7] Gunawardena J. Deducing Causal Relationships in CCS. In: Veni Madhavan CE (ed). Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1989. Springer LNCS 405, 1989. Also Technical Memo HPL-89-077, Hewlett-Packard Laboratories, May 1989.

[8] Gunawardena J. Geometric Logic, Causality and Event Structures. In preparation. Expanded version of a talk given at the 2nd Workshop on Concurrency and Compositionality, San Miniato, Italy, 28 February - 3 March, 1990.

[9] Harel D, Pnueli A. On the Development of Reactive Systems. In: Apt KR (ed). Logics and Models of Concurrent Systems. Springer, 1985.

[10] Hoare CAR. Communicating Sequential Processes. International Series in Computer Science. Prentice-Hall, 1985.

[11] Huet G. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. Journal ACM, 1980;27: 797-821.

[12] Johnstone PT. Stone Spaces. Cambridge Studies in Advance Mathematics, 3. Cambridge University Press, 1982.

[13] Keller RM. A fundamental theorem of asynchronous parallel computation. In: Feng T (ed). Parallel Processing. Proceedings, 1974. Springer LNCS 24, 1975.

[14] Miller RE. Switching Theory, Volume II: Sequential Circuits and Machines. John Wiley and Sons, 1965.

[15] Milner R. A Calculus of Communicating Systems. Springer LNCS 92, 1980.

[16] Milner R. Communication and Concurrency. International Series in Computer Science. Prentice-Hall, 1989.

[17] Newman MHA. On theories with a combinatorial definition of equivalence. Annals of Mathematics, 1942;43: 223-243.

[18] Nielsen M, Plotkin G, Winskel G. Petri Nets, Event Structures and Domains. Theoretical Computer Science, 1981;13: 85-108.

[19] Olderog E-R, Goltz U, van Glabbeek R (eds). Combining Compositionality and Concurrency. GMD Report 320, Gesellschaft Für Mathematik und Datenverarbeitung, Bonn. June 1988.

[20] Rem M. Trace Theory and Systolic Computations. In: de Bakker JW, Nijman AJ, Treleaven PC (eds). PARLE, Parallel Architectures and Languages Europe. Proceedings, Volume I. Springer LNCS 258, 1987.

[21] Sutherland IE. Micropipelines. Communications ACM, 1989;32: 720-738.

[22] Vaandrager FW. Determinism → (Event structure isomorphism = Step sequence equivalence). CWI Report CS-R8839. Amsterdam, October 1988.

[23] Winskel G. Event Structures. In: Brauer W, Reisig W, Rozenberg G (eds). Advances in Petri Nets. Springer LNCS 255.

[24] Winskel G. Events in Computation. PhD thesis, University of Edinburgh; 1980.

[25] Yakovlev AV. Analysing Concurrent Systems Through Lattices. Preprint, 1990.