To appear in the Journal of Requirements Engineering

1 February 1996

A Model for a Causal Logic for Requirements Engineering

Jonathan Moffett, Jon Hall, Andrew Coombes & John McDermid
Department of Computer Science
University of York
Heslington, York YO1 5DD, UK

{jdm, jgh, andyc, jam}@minster.york.ac.uk

Abstract

The language of causation is natural for the specification of requirements for complex systems. The paper provides a vocabulary of causal specification expressions, suitable for describing and analysing such systems. The notation is given a syntax and partial semantics. It covers many of the commonly-used modes of causal language including necessary and sufficient cause, prevention and enabling conditions. The concept of condition splitting is introduced, enabling a specification at an abstract level to treat two conditions as identical, while a concrete refinement of it may view them as separate. A number of other issues are examined, including: repetitive, probabilistic and hidden causes; causal agents; the validation of causal descriptions; and concurrency. Possible approaches to development of causal specifications are discussed. The work is placed in the context of related work in artificial intelligence and philosophy. The detailed framework of the paper is supported by a realistic example.

Contents

A Model for a Causal Logic for Requirements Engineering

"This above all: to thine own self be true,
And *it shall follow, as the night the day,*
Thou canst not then be false to any man"

Causal advice from Polonius to Hamlet

1. Introduction

1.1 Background and Aims

Requirements engineering is concerned, among other things, with the development of precise descriptions of systems and their environment, with at least three qualities of the description:

- The description is clear to the customers, in terms of their technical disciplines;
- The description is clear to the system developers, in terms of their technical disciplines;
- The properties of the described system can be analysed.

It appears from anecdotal evidence that one of the natural means by which the customers of complex systems express themselves is in terms of causation. They may do so in requirements: "*a* should cause *b*"; and in analysis: "given a system description, does *a* cause *b*?" Therefore one of the tools which would be useful for the specification and analysis of requirements for complex systems is a means of expressing causes precisely and reasoning about them. Also, it must be possible to interpret a causal specification in terms which can be used by the developers; for computer systems, this increasingly implies the use of a logic-based formal specification language. This paper is an attempt to make progress in this area by building upon a rather informally defined causal specification language described in [1] by giving it an extended syntax and more precise semantics.

Our aim here is to set out a relatively simple model which describes the properties which are necessary for a causal language. We are not attempting to produce a causal language suitable for writing in, nor a new logic. It is a model of causes, based on conditions and events, with a precise syntax and a meaningful, if incomplete, semantics. The model has an interpretation in terms of existing well-developed temporal logics, and so can be incorporated into existing formal specifications. The reason for this approach is that, since causal specification is only one element of a long shopping-list of characteristics of a requirements specification language, any approach needs to be able to be combined with other elements in that list. It should be clear that we are not trying to discover or model the true meaning of "cause" in all its complexity, but advocating a restricted and precise model of it which is close enough to common usage to be usefully applicable.

We introduce two flavours of causation, **direct_causes** and **sustain**, together with their opposites, **prevent** and **terminate**. The intuition for direct cause is that one condition direct_causes another if the first is always immediately followed by the second. A chain of direct causes constitutes an indirect cause, which we call leads_to. Correspondingly, one condition sustains another if the second one does not come to an end before the first. We

elaborate upon and formalise these intuitions, and discuss their consequences, in the remainder of the paper.

The style of causal specification which is addressed in this paper is not the only possible style. For example, for some forms of agent-related specification, the most appropriate causal primitive may be an action, in which case an action logic, e.g. [2] may be the best basis on which to work. However, we are concerned in this paper to follow the condition/event based approach.

1.2 Motivation

Coombes [1] suggests a notation which combines a causal logic with qualitative physics (QP). The qualitative physics notation is developed from Leitch [3] and de Kleer [4], and is not discussed further in our paper. QP expressions in the examples below are treated as atomic names for our purpose.

The causal logic is based on two major components: event and condition, where a condition can be seen as an generalised form of state, extended by the inclusion of time-variable attributes; for example, "decelerating" is a possible condition of an aircraft.

This notation is used to describe the behaviour of the braking system of an aircraft and allows expressions such as[1]:

[Aircraft.Deceleration]=+ *leads_to* [Aircraft.Speed]=0

Aircraft.ReverseThrust *direct_causes* [Aircraft.Deceleration]=+

¬Aircraft.Landing *sustains* ¬Aircraft.ReverseThrust

Using a combination of predicate logic, the properties of causal statements (e.g. transitivity of causation) and the laws of qualitative physics, the authors carry out an analysis of why the braking system failed to halt an A320 aircraft in an accident at Warsaw airport in 1993. The notation needs extensions, which we provide, to allow the expression of negative aspects of causes, i.e. prevention, and the non-occurrence of one event being the cause of another. We find it an attractive and potentially powerful notation. The aim of this paper is to provide one possible firm foundation for causal notations of this kind.

1.3 Properties of Causation

We list here a number of properties of causation, drawing upon the discussion in Shoham [5].

- Causal reasoning is ubiquitous in daily life. This is the prime motivation behind wishing to introduce a causal language into requirements specification, so that the gap between the natural language in which people describe mechanisms, problems and solutions, and a precise language of requirements, is minimised.
- Causation is antisymmetric and irreflexive; if A causes B, then B cannot cause A; and A cannot cause itself. These properties fall out naturally from the temporal ordering which is an essential part of the causal relationships which we define.

---

[1] In quoting these examples we have altered the lexical notation slightly, to conform to the notation used later in our paper.

- Entities participating in the causal relation have a temporal dimension;

  Note that this involves partially relinquishing one of the aims of many computer scientists: the replacement of procedural descriptions ("bad") by declarative descriptions ("good"). Whilst we accept the motivation behind this aim – describing "what" before we describe "how" – a simple declarative approach, as epitomised by functional languages, does not have a time-sequential aspect to it. A declarative description involving causation will include, implicitly or explicitly, time-sequencing information.

  In particular, causes cannot succeed their effects in time. Anticipating, we define causation in terms of the inevitable temporal succession of causes by their effects.

- Causal terminology includes other verbs such as "enable" and "prevent". As will be seen below, we define four flavours of cause: "cause", "prevent", "terminate" and "sustain", to reflect the different kinds of causal relationship which arise naturally from our model. In addition, we allow for enabling conditions using a "while" construct.

- Causation is context-sensitive, relative to a "causal field", the background against which causation is perceived to exist. A statement such as "letting go the ball causes it to drop" is only true if we make a number of assumptions about the context; it will, for example, be untrue if the ball is resting on a table.

- Causation is non-monotonic; addition of further conditions may falsify causal relations. "While the ball is in my hand, letting it go causes it to drop" may be true, but adding the condition "and the ball is resting on the table" will falsify it.

- Causation is not material implication. "A implies B" does not entail "A causes B"; it may be that cows lying down implies that it will rain soon, but that is not a causal relationship; there are two underlying causal relationships in which atmospheric conditions are the cause of both cows lying down, and rain. Conversely, we need to be very careful in going from "A causes B" to "A implies B", because causation may be true only relative to a given context, whereas implication is context-free.

In addition to the properties discussed by Shoham, there are two important points which must be made in relation to our own approach.

First, we make a clear distinction between sufficient cause and necessary cause. If *A* is a sufficient cause for *B* then, given the presence of its enabling conditions, the occurrence of *A* is inevitably followed by the occurrence of *B*. On the other hand we view necessary cause in terms of a double negative; if *A* is a necessary cause for *B* then, given the presence of its enabling conditions, *B* will not occur unless *A* does. We find it easiest, when working with practical examples, to think in terms of sufficient cause rather than using the double negations of necessity. Accordingly, we treat sufficient causation as primary, and give necessary cause a less complete treatment. Throughout this paper, unqualified "cause" refers to sufficient cause.

Second, we view certain factual relations about the world as following from causation, but not vice versa. The reader may have noticed that, in the paragraph above, we say "if *A* causes *B* then ...", but not the reverse. Causal conditions are inevitably followed by their consequences, but we cannot positively deduce a causal relationship from observations of hypothetical consequences; it is always possible that the hypothetical cause and its hypothetical consequences both follow from some other cause. The most that we can conclude is a contrapositive; if *A* occurs and we do not observe *B* then *A* does not cause *B*. Our view of cause is similar to Popper's view of the logic of scientific discovery; one can set up causal

hypotheses and attempt to disprove them, but never reach a conclusive proof of them. However, Newton's Laws are adequate for most practical purposes and so, no doubt, are many causal hypotheses. It is in this spirit that we believe this causal approach to be useful.

1.4 Paper Contents

The remainder of this paper is structured as follows. In section 2, we define the building blocks of our approach: Events, Conditions and Observations. Section 3 defines the syntax and a partial semantics for causal expressions. Section 4 discusses some of the main issues which emerge from our definitions. Section 5 places the model into the context of the development of causal specification. Finally sections 6 and 8 give a partial review of related work and draw some conclusions.

2. Events, Conditions and Observations

As stated in the introduction, we are not attempting to introduce a new logic, but a model with an interpretation in terms of existing well-developed temporal logics. We have used the specification language Z [6] with explicit time for encoding the specification, because of the local availability of a convenient type-checking tool CADiZ [7]. The specification was then edited into its present form for brevity. We have examined how we would encode it in other notations for temporal logic, such as Pnueli [8], and RTL [9]. In each case our approach appears equally feasible.

2.1 Time

We make minimal assumptions about time. Its instants are taken from a given primitive set, Time. We assume that Time, for the observer, is a strict total order (or **chain**), ordered on (transitive) precedes, usually written as "<" for convenience. This assumption only holds in systems with a single, global, clock; in distributed systems there is no global clock, and time is partially ordered. Partial ordering of time does not appear to pose any difficulties of principle for our approach, but this weakens the precedence relationship between those event occurrences which are timed with different clocks, and therefore weakens the hypotheses that can be made about causal relationships. Distributed clocks also weaken the statement of enabling conditions, because of additional uncertainty about the time periods in which they hold (see section 3.6).

We use the informal concept of granularity of time later in the paper, particularly in relation to Event Equality (section 4.1) and Condition-Splitting (section 5.2), recognising that it may be appropriate to work with a finer or coarser granularity of time, even in different parts of the same specification. The concept raises issues with which we do not claim to have dealt adequately. However, we believe that this does not affect our definition of causal expressions, but only our suggested approach to the development of causal specifications.

2.2 Events

We look at executions of the world in terms of the observations of events made by a single observer, who only observes one event at a time. An **event class** is a function from a name to a set of time points, associated with some characteristic of the system. An **event occurrence** is a member of the relation between names and time points.

[Name]

Event: Name $\rightarrow \wp$ Time
Event_Occurrence: Name $\leftrightarrow$ Time

where

$\forall$ n: Name; t: Time $\bullet$ (n, t) $\in$ Event_Occurrence $\Leftrightarrow$ t $\in$ Event n

The relationship between Event_Occurrence and Time is given by the function:

time_of: Event_Occurrence $\rightarrow$ Time

where

$\forall$ e_t: Event_Occurrence $\bullet$ time_of e_t = second[2] e_t

We define the ordering on event occurrences:

$\forall$ e_{t1}, e_{t2}: Event_Occurrence $\bullet$ $e_{t1} < e_{t2} \Leftrightarrow$ time_of(e_{t1}) < time_of(e_{t2})

In this sense the occurrence of an event defines and is defined by a point of the observer's time-line. Hence event occurrences are extensionally equal to the time of their occurrence. Since we work with event classes we distinguish another form of equality for event classes, that of identity of classes. Two event classes, a and b, are **identical**, (a == b) iff their sets of occurrences are identical, and we also define the subset ($\subseteq$) relationship between them:

$\forall$ a, b: Event $\bullet$ a == b $\Leftrightarrow$ second a = second b $\wedge$
a $\subseteq$ b $\Leftrightarrow$ second a $\subseteq$ second b

The event class identity relation is stronger than event occurrence equality, because it applies to all occurrences of the event, not just a single one.

Event occurrences are unique; each event occurrence occurs only once, although a number of events of the same class can occur ordered in time – the start_of_amber (traffic light) at 15:05 is a different event occurrence from, but of the same class as, the start_of_amber at 15:06. For brevity, and where there is no ambiguity, we say "a occurs" (where a is an event class) to mean "an event occurrence of class a occurs".

2.3 Conditions

We base our concept of conditions upon intervals of the time line, where intervals are defined by their start and end-points

Interval: $\wp$ Time
start_of, end_of: Interval $\rightarrow$ Time

where start_of and end_of are the greatest lower bound and least upper bound of the interval[3].

---

[2] first and second are the projection of a relation onto its first and second elements, respectively.

[3] In a total order, the greatest lower bound (resp. least upper bound) of a set is the largest (resp. least) element that is not greater than (resp. less than) any member of the set. The formal definition is omitted for brevity. We choose to use these bounds, rather than simply using the smallest and largest members of the interval, to

A **condition class** is a function from a name to a set of intervals, associated with some predicates about the system. A **condition occurrence** is a member of the relation between names and intervals.

$$\text{Condition}: \text{Name} \rightarrow \wp\ \text{Interval}$$
$$\text{Condition\_Occurrence}: \text{Name} \leftrightarrow \text{Interval}$$

where

$$\forall\ n: \text{Name}; i: \text{Interval} \bullet (n, i) \in \text{Condition\_Occurrence} \Leftrightarrow i \in \text{Condition}\ n$$

The relationships between Condition_Occurrence, Name and Interval are given by the functions:

$$\text{class\_of}: \text{Condition\_Occurrence} \rightarrow \text{Name}$$
$$\text{interval\_of} : \text{Condition\_Occurrence} \rightarrow \text{Interval}$$

where

$$\forall\ c: \text{Condition}; c_t: \text{Condition\_Occurrence} \bullet$$
$$\text{class\_of}\ c_t = \text{first}\ c_t \wedge$$
$$\text{interval\_of}\ c_t = \text{second}\ c_t$$

For brevity we treat condition classes as if they were types. So, if c is the name of an condition class and c_t an condition occurrence, we write c_t: c as shorthand for:

$$c_t: \text{Condition\_Occurrence} \mid \text{class\_of}\ (c_t) = \text{first}\ c$$

For simplicity we require that distinct occurrences of a condition do not overlap:

$$\forall\ c_{t1}, c_{t2}: c \mid \text{interval\_of}\ c_{t1} \neq \text{interval\_of}\ c_{t2} \bullet$$
$$\text{interval\_of}\ c_{t1} \cap \text{interval\_of}\ c_{t2} = \varnothing$$

Conditions and Predicates

Conditions **hold** during their intervals:

$$\text{holds\_at}: \text{Condition} \leftrightarrow \text{Time}$$

where

$$\forall\ c: \text{Condition}; t: \text{Time} \bullet c\ \text{holds\_at}\ t \Leftrightarrow \exists i: \text{Interval} \bullet t \in i \wedge i \in \text{second}\ c$$

Variables in specifications are time-dependent, and are expressed as functions from Time to their appropriate value domain, for example:

$$\text{distance}: \text{Time} \rightarrow \aleph$$

Each condition, to be meaningful has an associated time-dependent predicate, expressed as a function. In our example in section 6 the condition with name *distance>0* has this signature:

$$\text{distance\_gt\_0}: \text{Condition}$$

and the associated predicate:

$$\forall\ t: \text{Time} \bullet \text{distance\_gt\_0 holds\_at}\ t \Leftrightarrow \text{distance}\ t > 0$$

---

allow for the possibility of using continuous time, in which an open interval (e.g. {t | 1 < t < 2 }) has no smallest or largest member.

Start and End of Conditions

Intuitively, a condition class defines a set of occurrences with something in common, e.g. those intervals during which a particular traffic light is red. As such intervals will either be finite or infinite to the right, a condition defines and is defined by two event (occurrences) marking the left and, if existing, right of the smallest closed interval which contains the condition (this copes with discrete and continuous time). Informally, a condition occurrence with no right-hand event is one that does not terminate. The relationship between Condition and Event is given by two functions, $\uparrow$ and $\downarrow$, defining the starting and ending event classes for a condition class:

$\uparrow$: Condition $\rightarrow$ Event
$\downarrow$: Condition $\rightarrow$ Event

where the event classes are defined by the condition name and the sets of starting and ending times, respectively, of the intervals of the condition class:

$\forall$ c: Condition •
  $\uparrow$c = (first c, {t: Time; i: Interval | i $\in$ second c $\wedge$ t = start_of i • t }) $\wedge$
  $\downarrow$c = (first c, {t: Time; i: Interval | i $\in$ second c $\wedge$ t = end_of i • t })

We will require the "half-open" condition class [c corresponding to an interval condition class c:

[_: Condition $\rightarrow$ Condition

The intervals of [c are constructed from the union of the intervals of c with their greatest lower bounds:

$\forall$ c: Condition • second [c = { i: Interval | $\exists$ c_t: c • i = { start_of (interval_of c_t) } $\cup$ interval_of c_t }

Since we are treating Name as a primitive we do not provide a formal construction of the name of [c , but if names were defined as character strings it would be natural to define the name of [c . as the name of c prepended by [.

The "half-closed" condition class c][4] is defined similarly, for the least upper bound.

The reason for defining half-open and half-closed condition classes is so as to provide a natural definition of causal relationships in terms of set relationships, e.g. in the definitions of direct cause in section 3.1 and Sustains in section 3.4.

Condition Relationships

Two condition classes, a and b, are identical, (a == b) iff their sets of intervals are identical; and are related by inclusion, iff for each interval of a there is an interval of b of which it is a subset:

$\forall$ a, b: Condition • a == b $\Leftrightarrow$ second a = second b

[4] Where "]" is a post-fix function.

$\forall$ a, b: Condition • a $\subseteq$ b $\Leftrightarrow$
$\forall$ c_{t1}: a • $\exists$ c_{t2}: b • interval_of c_{t1} $\subseteq$ interval_of c_{t2}

An event class e and a condition class c, are related by the membership relationship $\in$, iff the times of the event class are in the intervals of the condition class, i.e.:

$\forall$ e: Event; c: Condition • e $\in$ c $\Leftrightarrow$ second e $\in$ second c

Events as Conditions

An interval with no duration is a point of the time line, and so defines two event (occurrences) which happen to be simultaneous. We define the function:

$\underline{\wedge}$: Event $\rightarrow$ Condition

such that $\uparrow\hat{e} == \downarrow\hat{e} == e$ for all event classes e, so that for any event occurrence $\hat{e}_t$ holds over the closed interval which is that of the occurrence of e_t. In this way e corresponds to two identical event classes, $\uparrow\hat{e}$ and $\downarrow\hat{e}$.

If conditions define only closed intervals (i.e., we have a discrete topology on the space) then this duality between event and condition is complete. However, when conditions may define either open or closed intervals, they are strictly more expressive than the objects defined through pairs of events, as we cannot indicate by a pair whether an interval is required to be open or closed.

Given that we have defined the interrelationship between events and conditions, it is convenient to use one as the basic object, deriving the other from it. As we will wish continuous situations to be described in our causal notation, and are not yet sure how significant is the distinction between open and closed intervals, we choose conditions to be the basic objects and derive events from them, as described above.

Complementary Conditions

During the interval between its start and end a condition occurrence **holds**. We are also interested in the complement of this, when a condition does not hold. If c is a condition class, then its complement is expressed as the condition class ~c. There is a mapping:

~_: Condition $\rightarrow$ Condition

The name of ~c is the name of c with "~" prepended. The relationship between the intervals of c and ~c is given by the identifications[5] $\uparrow$~c == $\downarrow$c and $\downarrow$~c == $\uparrow$c; ~c holds in the set of intervals that c does not. See Figure 1.

[5] Ignoring the start of the time line.

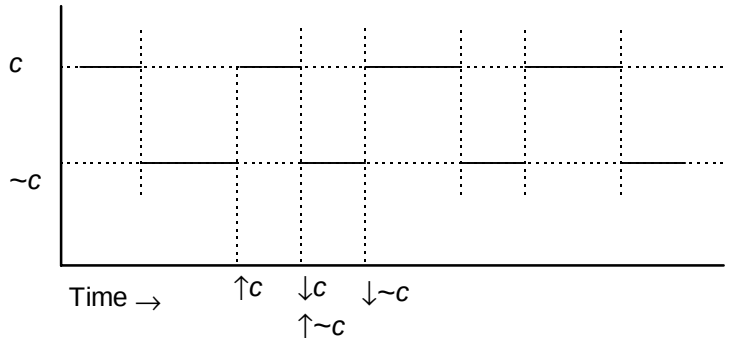

Figure 1 The relationship between c and ~c

The relationship between occurrences of the conditions c and ~c is governed by the constraints on condition occurrences that: a) the end of an occurrence is not before its start, and: b) occurrences do not overlap.

Clearly ~~c == c since ~c holds in the set of intervals that c does not.

Note that the negation of an event class is the entire time line, less the set of points at which events in the class occur.

3. Causal Expressions

A causal expression has the following components: a condition which is the cause (the **causing** condition); a condition expression which constrains the applicability of the cause; a condition which is the effect (the **effected** condition); and the causal relationship. We define our flavours of cause in terms of sufficiency, i.e. what is sufficient to cause, etc., an event or condition. We treat necessary cause as secondary; throughout this paper, unqualified "cause" should be taken to mean sufficient cause. A causal expression is one of the following expressions:

[While Compound_Condition[,][6]] *condition causal-rel condition*

There are four kinds of causal relation, as shown in this table:

Causal relationship	Activated by	Ended by	Result on effected condition[7]
causes	End of causing condition	–	Condition starts
terminates	End of causing condition	–	Condition ends
sustains	Start of causing condition	End of causing condition	Condition does not end
prevents	Start of causing condition	End of causing condition	Condition does not start

[6] The optional comma has no significance except to improve readability.

[7] Between activation and ending of the relationship, where appropriate.

For causes and terminates the end of the causing condition triggers off the result, but for prevents and sustains the effected condition occurs or does not occur throughout the duration of the causing condition. The individual relations are discussed in detail in the remainder of this section.

Since we regard events as special kinds of condition, we can extend this table. For all events e, $\uparrow\hat{e} == \downarrow\hat{e} == e$, so we can attach a meaning to an event being the cause or the effect of a causal relationship:

Causal relationship	**Activated by**	**Ended by**	**Result on effected condition / event**
condition|*event* causes *condition* | *event*	End of causing condition/ causing event occurs		Condition starts / event occurs
condition|*event* terminates *condition*	End of causing condition/ causing event occurs		Condition ends
condition sustains *condition*	Start of causing condition	End of causing condition	Condition does not end
condition prevents *condition* | *event*	Start of causing condition	End of causing condition	Condition does not start / event does not occur

3.1 Direct Cause

We define direct cause in terms of the temporal relationship between the start and end events of conditions. We start with the intuition that, for any one system, a condition c being the **necessary and sufficient direct cause** of another condition d implies that in all possible observation histories of that system, whenever c occurs, the end of c coincides with the start of d. Formally, we express this as:

$$c \text{ ns\_direct\_causes } d \Rightarrow \downarrow c == \uparrow d$$

Situation 1 in Figure 2 illustrates this relationship[8]; each occurrence of $\downarrow c$ coincides with $\uparrow d$.

We should then like to define (sufficient) direct cause similarly, as $c \text{ direct\_causes } d \Rightarrow \downarrow c \subseteq \uparrow d$; i.e. every occurrence of $\downarrow c$ is associated with a $\uparrow d$, and there may be other occurrences of $\uparrow d$ caused by other conditions. However, we required, in section 2.3, that occurrences of a condition should not overlap, so if d already holds at the point that c ends, a second occurrence of d should not start. So we define **sufficient direct cause** as $\downarrow c == \uparrow d \vee \downarrow c \in d$; see situations 1 & 2 in Figure 2. Then $\downarrow c$ is in the half-open interval defined by [d:

$$c \text{ direct\_causes } d \Rightarrow \downarrow c \in [d$$

[8] For brevity, in all figures we omit the *time_of* function from *Event_Occurrence* to *Time*.

A condition c being the **necessary direct cause** of another condition d implies that d does not start except when c has immediately preceded it, as illustrated by situations 1 & 3 in Figure 2:

$$c \text{ necessary\_direct\_causes } d \Rightarrow \uparrow d \subseteq \downarrow c$$

Situation 4 in Figure 2 is a counter-example to all flavours of direct cause.

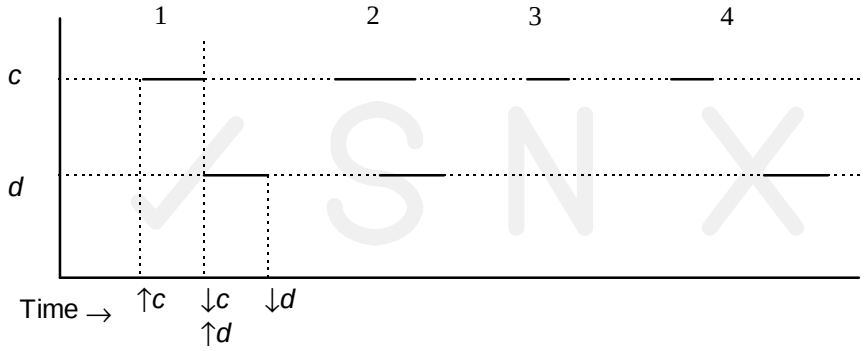

Figure 2 Condition c **direct_causes condition** d

We can regard direct causation as implying that one condition starts immediately after the end of the other, where "immediate" is governed by the granularity of time within which the observations are taking place. For example, although there may be a perceptible pause between flicking a switch and the light coming on, we may choose to make observations at a sufficiently coarse granularity of time that we can regard the two conditions as abutting. See the discussions in sections 4.1 (Event Equality) and 5.2 (Condition-Splitting).

3.2 Leads_To (General Cause)

We define sufficient general (or indirect) cause in terms of direct cause. A condition c is said to **lead to** condition d iff[9] we can find intervening direct causes to make up a causal chain. This is defined recursively:

$$c \text{ leads\_to } d \Leftrightarrow \exists\, b: \text{Condition} \bullet c \text{ leads\_to } b \wedge b \text{ direct\_causes } d$$

General cause is thus the transitive closure of the direct_causes operator. Figure 3 illustrates the general causal relationship.

The **transitivity** of "leads to" follows immediately from its definition: a leads_to b $\wedge$ leads_to c. It is not **acyclic**, because although event occurrence is based on the total ordering precedes (<), we have not defined precedence between condition classes. Repetitive cause, discussed in section 4.2, in which a leads_to b and b leads_to a, has of course a cyclic nature.

Shoham [5] defines two kinds of causal statement: "x causes y" and "x (actually) caused y". Our account is consistent with the first of these kinds of statement, but we have no formal definition of the second; we view causation as being about the consequences which follow from a relationship between two condition classes. If condition occurrence y_{t2} has followed

[9] Note the "if and only if" here; the relationship between direct-cause and cause does not depend directly upon observations, and therefore can safely be made a bi-directional implication.

condition occurrence x_{t1}, then this is consistent with "x leads_to y" but we have no means of stating that "x_{t1} **(**actually) caused y_{t2}". This is because of our agnosticism about inferring causes from observations; although y_{t2} followed x_{t1}, something else may have been its cause.

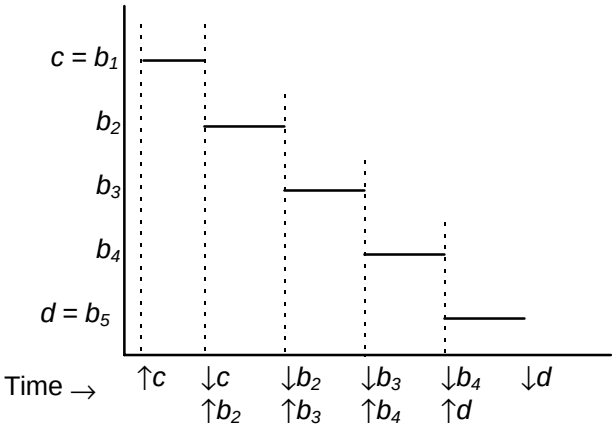

Figure 3 Condition c **causes condition** d**, but not directly ("leads to")**

Even if y always follows x, we still cannot infer a causal relationship between them. A condition may cause two other distinct conditions, so that there may be situations where two conditions inevitably follow one another, although they are not causally ordered. For example, the secondary symptoms of syphilis always follow the primary symptoms, but are not caused by them; each is caused by the original infection. If we define conditions i, and ss as the conditions of being infected, showing primary and secondary symptoms, respectively, then this situation is modelled by the existence of a fourth condition, j, which we assume always to be of longer duration than ps, such that:

i leads_to ps $\wedge$ i leads_to j $\wedge$ j leads_to ss

There may in practice be timing relationships which cause one condition, c_1, always to be followed immediately by another condition c_2 even though c_1 does not cause c_2 where, as stated above, "immediate" is governed by the granularity of time within which the observations are taking place. It will be impossible to determine whether c_1 "actually" causes c_2 except by analysing the situation at a finer granularity, in which case a common cause for the two conditions may be found. In the history of medicine there are many examples of the earlier of two symptoms of a disease being mistaken for its cause, until a common cause for both symptoms was found.

Although the discussion of the previous two paragraphs explicitly and implicitly refers to duration, we do not attempt to deal with timing in this paper, but recognise that it is an issue that must be addressed in future work.

3.3 Termination

"Direct cause" relates the end of one condition to the start of a second one. Another interesting causal relationship is when the end of one condition is identical to the end of the second one, i.e. causes the **termination** of that condition. A condition c (sufficient) terminates another condition d implies that for all possible observation histories of that system

the end of c coincides with the end of d. Formally, we express necessary and sufficient termination as:

c ns_terminates d $\Rightarrow \downarrow c == \downarrow d$

Situation 1 in Figure 4 illustrates this relationship; the occurrences of $\downarrow c$ and $\downarrow d$ coincide. It says nothing about $\uparrow d$.

Using similar arguments to those for sufficient direct cause in section 3.1, we define **sufficient terminates** as follows, as illustrated by situations 1 & 2 in Figure 2:

c terminates d $\Rightarrow \downarrow c \in [\sim d$

A condition c being necessary to terminate of another condition d implies that d does not start except when c has immediately preceded it, as illustrated by situations 1 & 3 in Figure 4:

c necessary_terminates d $\Rightarrow \downarrow d \subseteq \downarrow c$

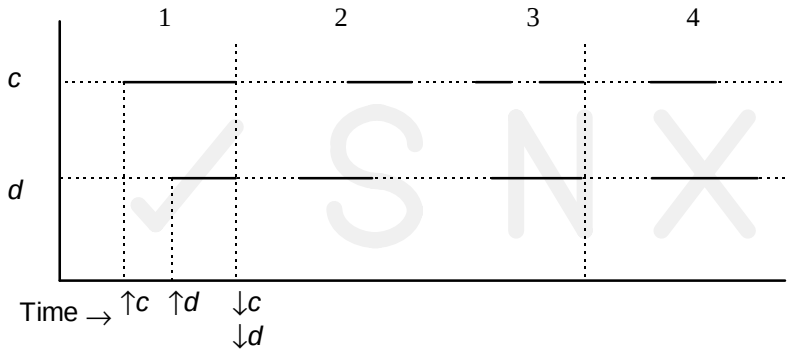

Figure 4 Condition c terminates condition d

As illustrated in the figure, there is no assumption about which of c or d starts first, nor that the start of c causes the start of d. It would be possible to construct a "strong termination" from our weak primitive, together with direct_causes, in which the start of c causes the start of d and also the end of c terminates d.

Note that, if the two conditions are always observed to end simultaneously, this is consistent with either terminating the other. If we hypothesise that c terminates d, then further analysis will be necessary to disprove it. A practical example of this can be seen in some street organs; one sees a clown playing a street organ, winding its handle. When the handle stops winding, the organ stops playing. But closer inspection shows that the clown is a dummy; the organ is playing the dummy and when the electric motor in the organ stops, so does the clown. Or is the electric motor in the dummy clown? Only closer inspection will tell us.

Clearly ns_terminates is transitive, since $\downarrow c == \downarrow d \wedge \downarrow d == \downarrow e \Rightarrow \downarrow c == \downarrow e$. On the other hand (sufficient) terminates is not transitive, since d may terminate from some other cause than c.

3.4 Sustains

We define (sufficient) **sustains** as the prevention of a condition ending. If condition c (sufficient) sustains condition d, then d does not end during c.

c sustains d $\Rightarrow \downarrow$d $\notin$ c

Figure 5 illustrates the sustenance relationship. It shows that, if c sustains d, no occurrence of the end of d ($\downarrow$d) occurs during c. It says nothing about the start of d ($\uparrow$d).

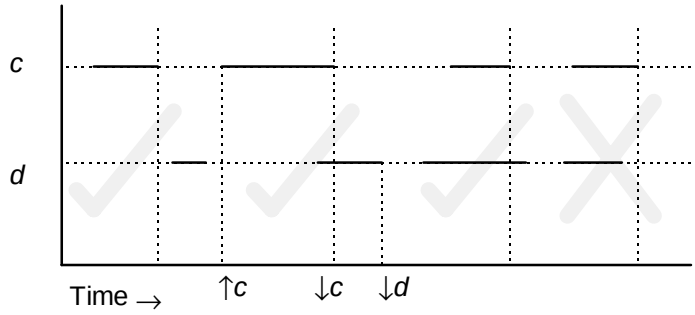

Figure 5 Condition c sustains condition d

We also require, in practice, a "strong sustains" (Sustains, with an initial capital), defined as a combination of direct_causes and the prevention of a condition ending. If condition c (sufficient) Sustains condition d then c both direct_causes and sustains d; condition c is wholly contained in condition d (see Figure 6):

c Sustains d $\Leftrightarrow \uparrow$c direct_causes d $\wedge$ c sustains d[10]

It follows, since $\downarrow$c $\in$ [d and $\downarrow$d $\notin$ c, that:

c Sustains d $\Rightarrow$ c $\subseteq$ [d

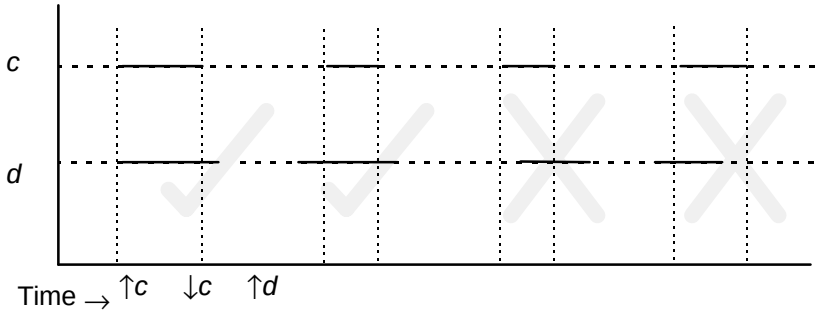

Figure 6 Condition c (strong) Sustains condition d

Clearly the Sustains relationship is "practically" transitive, since:

c Sustains d $\wedge$ d Sustains e $\Rightarrow$ c $\in$ d $\wedge$ d $\in$ e $\Rightarrow$ c $\in$ e

[10] $\uparrow$c is here regarded as a condition, i.e. has the $\hat{}$ operator applied to it.

This is not full transitivity, since the implication is only one way; we cannot go further and infer that c Sustains e

necessary_direct_causes and necessary_terminates are straightforward to define, but we have more difficulty with necessary_sustains. We can attach two alternative meanings to "condition c necessary_sustains condition d":

- Condition d *will* only hold if c holds; or
- Condition d *may* only hold if c holds.

Each of these meanings is straightforward to define formally, but we do not know at present which to choose. The means of choosing will be by examination of examples in which causal necessity is used.

3.5 Prevention

"Cause" results in a condition starting to hold. We are also interested in its complement, the (sufficient) **prevention** of a condition starting, intuitively that always, if condition c prevents d, then d does not start during c.

$c \text{ prevents } d \Rightarrow \uparrow d \notin c$

Figure 7 illustrates the prevention relationship. It shows that if c prevents d, no occurrence of the start of d ($\uparrow$d) occurs during c.. It says nothing about the end of(d).

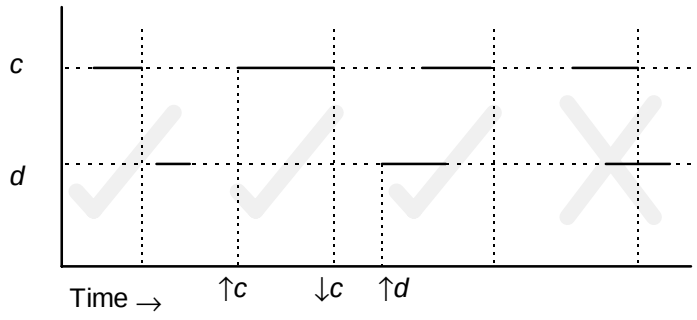

Figure 7 Condition c **prevents condition** d

We also define (strong) Prevents intuitively as the opposite of Sustains, i.e. if condition c (sufficient) Prevents condition d, then condition d never holds while c holds; the end of ~c terminates d and c prevents d (see Figure 6):

$c \text{ Prevents } d \Leftrightarrow \downarrow\sim c \text{ terminates } d \wedge c \text{ prevents } d$

It follows, since $\downarrow\sim c \in [\sim d$ and $\uparrow d \notin c$, that:

$c \text{ Prevents } d \Rightarrow c \subseteq [\sim d$

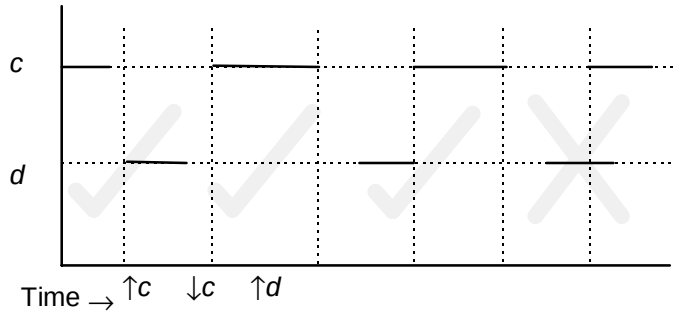

Figure 8 Condition c (strong) Prevents condition d

The Prevents relationship is not transitive, as it does not possess the simple "nesting" property of (strong) Sustains. An intuitive explanation for this is that prevention depends upon the first condition having started and implies that the second condition will not start. So "b Prevents c" prevents c from starting, so c cannot prevent anything else.

We have a similar problem for necessary_prevents as for necessary_sustains.

3.6 While

The While clause of a causal expression is intended to limit the applicability of causal expressions to intervals in which the While compound condition holds:

- cause and terminate expressions are only applicable if the While clause holds at the end of the causing condition;
- sustain and prevent expressions are only applicable during those intervals of the causing condition for which the While clause holds throughout.

We define compound conditions (CC) informally (in the Z specification they are declared as a free type, which is difficult to read):

CC::= Condition
| ~CC
| CC ∪ CC
| CC ∩ CC

We the define a function from CC to sets of time points:

CC_Times: CC ↵ Time

The operators have the following meaning when CC_Times is evaluated:

Argument of CC_Times	Value of CC_Times
c (Condition)	The time points in the intervals of c
~cc (Compound condition)	The time points that are not in cc
cc ∪ dd	The union of the time points that are not in cc and dd

cc ∩ dd The intersection of the time points that are not in cc and dd

3.6.1 The interpretation of while

The effect of while in while w , c direct_causes is to restrict the occurrences of the end of the causing expression to those at which the while clause holds. So it applies only for those end-points of c at which w holds.

There are two possible interpretations that we can use for the expression while w, c sustains d. The first interpretation is that only if w holds throughout it will an occurrence of c sustain an occurrence of d. The second possible interpretation is that sustains is effective for those parts of c in which w holds, i.e. the "sustaining power" of c is switched on and off in step with w holding and ceasing to hold during a single interval of c. In some ways this is a more natural interpretation, but it poses the technical problem of constructing sub-intervals" from those segments of intervals of c for which w holds, and we do not require this degree of complexity for the example in section 6. We therefore choose the first and simpler interpretation: that only if w holds throughout it will an occurrence of c sustain an occurrence of d.

However, note that both interpretations are different from those used for while in causes. For while in causes we are interested in the end-points of the causing condition, but for sustains we are interested in the whole interval.

terminates requires a similar interpretation to direct_causes and prevents requires a similar interpretation to sustains.

3.6.2 The construction of While

Note that the for particular instances of CC_Times and Condition, the operator that we would naturally choose for while w , c direct_causes d will have different results from the operator that we would naturally choose for while w , c direct_causes d. These instances are when the intervals of the condition are open at their upper end, and then they may be a subset of a set of time points without their upper bounds being members of that set. We therefore introduce a strong While whose intervals are the intersection of those needed for the different causal relationships. Formally, we treat While as a function with two arguments, a compound condition and a condition, yielding a new condition:

While: CC X Condition Condition

so that While w, c sustains d is interpreted as:

While (w , c) sustains d

We construct the function While (w , c). The name of While (w , c) is the name of c with "W" prepended. Its set of intervals consists of those intervals of c] (the union of the intervals c with their upper bounds) which are subsets of CC_Times w:

∀ w: CC; c: Condition • second While (w, c) =
{ i: Interval | i ∈ second c] ∧ i ⊆ CC_Times w) }

We also introduce, as syntactic sugar, Until, where:

Until x ≡ While ~x

3.6.3 Discussion of While

Notice that prevent is stronger than While; if a While condition inhibits the applicability of a cause, then we cannot say that the effect will not happen, whereas if the same condition prevents an effect, then we can say that the effect will not happen.

There is clearly a close relationship between a causing condition and a condition in a While expression.

- sustain and prevent are positive and negative enabling conditions, and their causing conditions are no different in their effect from those within a While expression:

  While a, c sustains d is identical in effect to While $a \wedge c$, true[11] sustains d (and similarly for prevents)

- On the other hand direct_causes and terminate are derived from a single triggering cause, enabled by other conditions expressed in the While expression. So for these causal relationships, the causing condition cannot be absorbed into the While expression.

There are problems for us in distributed systems. The semantics of conjunctions and disjunctions of (timed) conditions is that of the connective considered pointwise. However, the necessity to consider pointwise composition means that a description in terms of states is difficult in any system in which we cannot rely upon the global synchronisation of clocks (i.e. most systems). In a system of this kind the order in which events are observed depends upon the properties of the observer. In general, the most we can say of the state of the expression is that the start and end of the occurrence of a condition can be relied upon to occur consecutively.

4. Other Issues

4.1 Event Equality

We have proposed a view of causes based on event precedences, with the underlying mechanism being that of the relationships between conditions. However, we think that it is also necessary to allow for the specification of hypothetical causes in terms of event equality, i.e., simultaneity, the reason being that the specifier may be working at a level of description, and a time granularity, which are too coarse to allow the provision of detailed analysis of causes which are believed to be true. For example, consider a ball which is dropping until it hits the floor, and then rises again. With a coarse granularity of time we may wish to ignore the internal mechanism of compression of the ball and its subsequent expansion, and treat the contact with the ground as an instantaneous event. Then the events of end_of(dropping), ground_contact and start_of(rising) are always simultaneous, i.e., identical as illustrated in the left-hand side of Figure 9. However, it may turn out that we need to make a more detailed analysis of the ground contact; with finer granularity (see section 5.2, Condition-Splitting). The nature of the motion is then illustrated in the right-hand side of the figure.

[11] A condition class with one occurrence, which holds over all time.

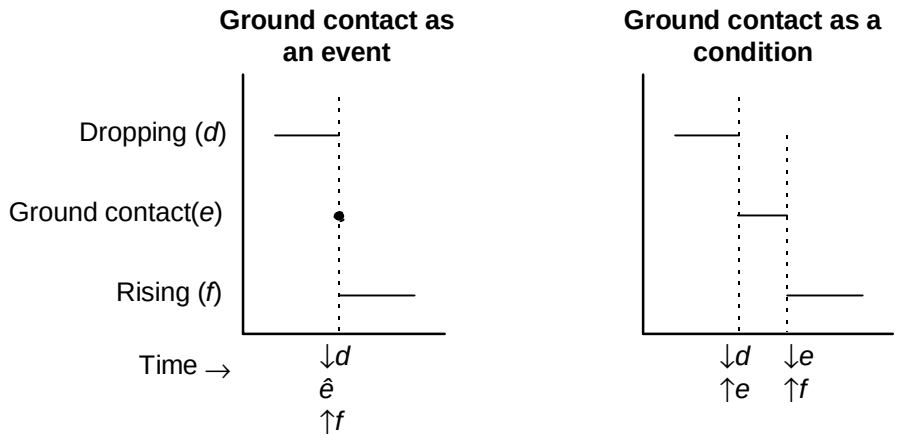

Figure 9 A ball dropping and rising

The view of a condition as an event (a process we will call **atomisation** – we make the condition atomic, i.e., without internal detail) implies that we have, analytically, lost all ability to refer to the strict temporal relationships of other conditions which might hold within that condition. The wish to preserve these relationships implies that we have been incorrect in our consideration of that particular event as atomic. The other side of the coin is that we may atomise conditions in whose internal detail we have no interest.

4.2 Repetitive Causes

As can be seen from the figures, repetitive causes fit naturally into our scheme. Consider, for instance, a clock. A portion of the expected behaviour of an escapement lever in a clock might be:

... tick *direct_causes* tock *direct_causes* tick ...

Two points arise from this:

- This example might appear to contradict the anti-symmetry of cause - for two condition occurrences, A and B one cannot have both A causes B and B causes A. This was the reason for making the distinction between condition classes and condition occurrences; the anti-symmetry applies only to occurrences, not condition classes.
- That description of the behaviour must be incomplete, as there needs to be a description of the initial condition from which the clock starts..

4.3 Probabilistic Causes

We have chosen to be extremely rigid in the expression of the cause/effect relationship, i.e., two conditions are either causally related or not. Cases in which this assumption is not justified are rife, and we will wish to be able to cope with them. Most, perhaps all, "causes" in the real world are not certain. Typically there is some probability that they will not be effective, so we need to be able cope with "a causes b with 90% probability". Indeed one writer on causation Suppes [10] bases his treatment of causation on the probability of timed events. A causing event is said to be the prima facie cause of an effect if the probability of occurrence of the effect is greater if the cause has preceded it. Prima facie causes are partitioned into genuine and spurious causes. In order to cope with probabilistic cause there

needs to be some relaxation of the insistence that a cause is *always* followed by its effect, replacing it by the statistics of observation histories. Extension of this notation to probabilistic causes is further work.

4.4 Hidden Causes

This model cannot describe hidden events as in CCS [11]. Although red always follows green in traffic lights, green does not cause red[12]. This, if correct, implies that there must be a hidden cause. Hidden causes are, in general, unsatisfactory, as they imply that a causal description is incomplete. We can look at this from two points of view:

a) In specifying the requirements for a system, we may say that red must always follow green. It will not be satisfactory to trust that this will be the case; typically we must isolate the causal agent – for example, a controller – and ensure that it too is specified, either by including it in the system design which refines the specification or by demonstrating that it follows from the system's environment. See the discussion of process issues in section 5.

b) If we are analysing an existing system, we are interested in establishing whether the observation, that so far red has always followed green, is the result of chance, in which case there is no reason to expect it to follow in the future, or has a (possibly probabilistic) explanation for why this sequence always (usually) occurs, i.e. a cause. We should not be satisfied with asserting that the observation has an explanation without knowing what the explanation is.

4.5 Causal Agents

Some writers, e.g. von Wright [12], have made a distinction between a functional relationship and a causal relationship, emphasising the importance of **agency** as an element of the causal relationship. In this view, we may transform a functional relationship into a causal one by viewing one of the variables in the functional relationship as independent, e.g., we can transform "pressure and volume are inversely related" into "an increase in the pressure causes a reduction in the volume" with the implication that some independent agent is increasing the pressure. The agent may be regarded as the independent variable in the situation, and may be a human being or some other agent which is independent from our point of view, or even physical laws. For example:

- Letting go of the ball causes it to drop

The agent may be a person or machine which lets it go. It is straightforward to tag each causal statement with its associated agent.

There are issues, which we do not claim to have resolved, about the relationship between a causal agent, and a causal mechanism. Clearly, the ball drops because it is in a gravitational field, and this kind of influence on a condition may also need to be brought into a causal description. Disentangling agent and mechanism is a subject for further study.

---

[12] As pointed out by Philip Morris (personal communication)

4.6 Validation of Causal Descriptions

We have defined "causes" in terms of precedence over all possible executions of the world. Although this is intuitively fairly satisfactory, it does raise the question of what would constitute a satisfactory validation of the assertion that a causes b. By validation we mean something which may be quite informal, but which is adequate for an engineer to be satisfied that, in practice, if a happens then b will. We have no general solution to this, but several specific comments:

- Reliance upon scientific laws is one safe approach; for the applications in view, we will incorporate a model of the relevant parts of physics into our description of the environment;
- Where the causal agent is to be a human being, or any other unreliable agent, it is impossible to rely with certainty upon the causal statement, and it will be necessary to reason in terms of uncertainty. See the distinction between possibility and (probabilistic) chance in Haddawy [13];
- If a causal assertion is made with a high degree of probability, then disproving is by finding an exception is easier than proving it;
- Proving that something will *not* happen involves examining all possible causal agents which, as we know from safety cases, may be extremely laborious. On the other hand, proving the actual cause of something that has happened, may involve tracing back only one, or a few, causal chains. So, validating a complete specification is much more arduous than analysing an incident in terms of the specification.

Using the clown and electric motor example in section 3.3, how would we set about validating our hypothesis that it is the organ playing the clown, and not vice versa?

- One way is by our knowledge of science and technology; examining the internal mechanism and discovering whether the motor is in the organ or the clown;
- Another way would be more in the spirit of this paper; observing the external mechanism, paying particular attention to the timing. If there is a perceptible lag between the movement of the organ and the clown, then the later one cannot be the cause, but the earlier one might be. If, on the other hand, there is no perceptible lag, then we cannot even reach this contrapositive conclusion.

4.7 Concurrency

We define cause in terms of chains of observations. However, the world has concurrency and is often non-determined about the ordering of events which are causes: for example, if While a, b leads_to x $\wedge$ While b, a leads_to x, typically we are not concerned about whether $\downarrow$a or $\downarrow$b occurs first. Actual executions of the world therefore must be regarded as partial orderings of events. Although the observations made by a single observer are always a chain, typically several chains are equivalent for the purpose of verifying or falsifying a causal observation; for example the observer must be indifferent to the order of interleaving of a and b in our example.

5. Development of Causal Specifications

Whether we are writing a requirements specification or analysing an existing system, it is necessary to proceed from a coarse, general view of the causes of events to a more refined one, which provides a more satisfactory explanation. In other words, we would start from one specification and refine it into another. There are several ways in which we can do this:

- Explication – providing an explanation of why c causes d. For example, we may need to justify "the braking system causes the aircraft to stop in time", and this will be done in terms of our model of the braking system; see the analysis of the 1993 A320 Warsaw crash in section 6 and in [1].
- Event-splitting – realising that an event which was treated as atomic at the coarser level must now be analysed. For example we may say ball_dropping direct_causes ball_rising. This is the kind of cause in which we equate the two events (see 4.1), and it may be satisfactory for a first approximation. But it may be necessary to recognise that this description is too simple, and that there is a significant condition between these two events so that they can no longer be identified.
- Identification of causal agents – initially the specification may simply say that something is to happen, without specifying the agent which is to cause it to happen. An aspect of development of the specification will be identification of the agent.

5.1 Explication

Explication does not seem to raise any issues for this causal logic, although it may do so for other requirements notations such as qualitative physics; it is simply replacing (what had previously been treated as) a fact with further facts, causal statements and other forms of argument which hopefully will entail the same conclusion.

5.2 Condition-Splitting

In section 4.1, Event Equality, we introduced the concept of atomisation of conditions into events. In the example, of a ball dropping, hitting the ground and rising again, we suggested that the condition of ground contact of the ball could be atomised as an event if the internal details of the mechanism of a ball rebounding were thought to be unimportant. Clearly, however, it may be necessary to "open up" that event and analyse the mechanism in more detail. We use this example to suggest a general way in which events can be split into conditions, as a means of refining a causal specification in order to validate it.

In the example there are three successive conditions: dropping (d), ground contact (e) and rising (f), and the granularity of time is sufficiently coarse that end_of(dropping) – $\downarrow$d, the event of ground_contact – ê and start_of(rising) – $\uparrow$f are treated as happening simultaneously at some time t. Our coarse causal specification states that dropping direct_causes rising, i.e. that end_of(dropping) is always immediately followed by start_of(rising).

Since this is a well-known phenomenon, it may not be necessary to question this causal relationship. However, if there is any doubt, e.g. if it appears to have failed in some instance, we need to analyse what happens in ground contact, i.e. split the event ê into a condition e in which $\uparrow$e $\neq$ $\downarrow$e. This is carried out by generating a refined specification from the coarse

specification, in which all causal relationships are preserved but e is treated in more detail. In the refined specification:

- Time is treated at a finer granularity, so that events which has previously been identified are now separated. Then, instead of

  time_of(↓d) = time_of(e) = time_of(↑f), we have

  time_of(↓d) = time_of(↑e) > time_of(↓e) = time_of(↑f).

  We appreciate that formal refinement to a finer time granularity raises technical issues which we have not addressed. At a practical level, however, it is the sort of refinement which is common in engineering mathematics, e.g. in ignoring or taking into account second orders of small quantities.

- Ground contact, e, is now a condition about which we can hypothesise that:

  * dropping direct_causes ground_contact
  * ground_contact leads_to rising.

  Either of these, more detailed causal relationships are now open to more detailed analysis, e.g. does the ball always hit the ground, or are there circumstances which could prevent this? Does ground contact always cause rising, or does the ball sometimes lose its elasticity.

If the refined specification confirms (or, rather, does not refute) the causal relationship, then the coarse specification is confirmed also.

Condition-splitting, the generalisation of event-splitting, may also be required when analysing a specification. In the example below, section 6, the basic model treats brakes as an atomic condition. However, it becomes apparent that it is necessary to split this up into brakes_commanded and brakes_are_on, which have to be treated separately.

Event-splitting or condition-splitting, like any other form of refinement, is of course like opening a can of worms; more crawls out than was expected. What appeared to be simple may on inspection be much more complicated, with the necessity to describe exception handling, and the involvement of other agents. This could imply taking a branching view of the opened-up event, dealing with the normative case and the exceptions separately. Examination of actual examples will reveal the possibilities and complications of the techniques.

The question arises, of course, whether event-*merging* is a useful technique. It would be legitimate in a situation in which it was realised that two events which had been treated as distinct were in fact identical. We cannot think of any examples.

5.3 Identification of Causal Agents and Mechanisms

We introduced causal agents and mechanisms in section 4.5, emphasising some of the difficulties which may arise in describing causal agents. However, the identification of the agents and mechanisms, which or who are to cause the system to function, is an essential part of development. Further research on clarifying the issues is required.

5.4 Structuring the Development Process

It will be clear from the above that we advocate a very hybrid approach to the refinement of causal specification, mixing a variety of ways in which it can be analysed, some more formal than others. For any given refinement step, therefore, the refinement strategy used needs to be explained and justified. The Goal Refinement approach, originally proposed for safety analysis in [14], is also appropriate for this purpose. In terms of this approach, a causal specification is a model of the system and/or its environment. A causal relationship which needs justification is a goal. The goal may be justified by one of the steps discussed above. That justification is explicitly documented as the **strategy**, with a **justification** which is associated with it. The justification may be either formal or informal. If the refinement involves event splitting, then the more detailed model produced by event-splitting is introduced as part of the strategy.

6. Example

In this section we use the example given in [1], where it is described in more detail, to show how our notation may be used in a practical example. The example is a simplified description of the braking system of an aircraft, the aim being to show how a causal description could help in the formal analysis of an incident such as the crash of an A320 aircraft at Warsaw in September 1993. In that paper, a combination of Qualitative Physics and our causal notation was used. For the purpose of exposition here, the qualitative physics notation has been translated into textual predicates.

Briefly, the incident resulted from a failure of the braking system to operate as desired, resulting in the aircraft overrunning the runway and hitting a barrier. The system operated as specified, but the specification failed to take into account a combination of circumstances, including rain and strong wind, which led the system to reject the pilot's command to brake.

In this specification we annotate the statements as follows:

- G, if they are goals that are required to be met;
- A if they are assumed correct;
- D, if they can be deduced, not necessarily formally, from assumptions.

We assume that there are no failures in the system, and examine whether we can prove from our knowledge of it and its environment that the aircraft does not overrun the runway.

System Goals

There is a global constraint not to overrun the runway, which is the primary goal:

$distance>0$ 1 G

The basic model of the system relates brakes, altitude, distance from the end of the runway, speed, and deceleration using the equations of motion. Making the first approximation of constant deceleration, given initial distance s, initial speed u, and deceleration a, we can use the formula for stopping distance ($s = ½u^2/a$) then overrun will be avoided if

$a>½u^2/s$ 2 G

If we assume that landing speed and distance are given, then the only variable that is under our control is deceleration, and our basic model of the system states as a goal (a sub-goal that will achieve 1):

While *altitude=0* ∧ *distance>0*
brakes Sustains *deceleration>a* 3 G

The Braking System

In this aircraft there are two agents in the system-as-a-whole: the human pilot and the braking system computer (BSC). The human issues the braking command: *pilot_commands_brakes_on*; the computer records that the command has been issued and not countermanded: *brakes_commanded*; and the computer can cause the brakes to come on: *brakes_are_on*. Therefore, even assuming there is no mechanical failure, we need to split the *brakes* condition into those three conditions, where the start and end of *brakes* are defined by:

↑*pilot_commands_brakes_on* = ↑*brakes* 4 G

↓*brakes_are_on* = ↓*brakes* 5 G

and we have a causal chain, to be ensured by the BSC:

pilot_commands_brakes_on direct_causes *brakes_commanded* 6 A

brakes_commanded Sustains *brakes_are_on* 7 G

We assume the truth of 6. We also assume the pilot ensures that:

↑*altitude=0* direct_causes *pilot_commands_brakes_on* 8 A

So that:

↑*altitude=0* leads_to *brakes_are_on* 9 G

We also assume that if *brakes_commanded* holds, it continues to hold until the *brakes_off_command* is issued:

until *brakes_off_command*
true sustains *brakes_commanded* 10 A

Using the proposition true as the causing condition of the expression may of course be optimistic; perhaps the brakes do *not* stay on until the "off" command. If there were reason to question that assumption it would be necessary to analyse the mechanical and/or computer systems to justify or falsify the assumption. But we assume it to be true in this instance.

[The following statement, included to exemplify the use of terminates, would also be needed in a full specification:

brakes_off_command terminates *brakes_commanded* 11 A]

What we have done in splitting *brakes* into three conditions is of course condition-splitting, as described briefly in section 5.2. There is therefore an obligation to demonstrate the truth of 7. We are assuming the truth of 6, 8 & 10 in this exposition because these aspects of the system performed as expected in the incident.

Components of Braking

There are three different contributions to braking in this aircraft: *wheel-brakes*, *spoilers* (flaps on the wings) and *reverse_thrust* from the engines. We assume that *brakes_are_on* means "all braking components are activated":

brakes_are_on ⇔ *wheel_brakes* ∧ *spoilers* ∧ *reverse_thrust* 12 A

Each of these makes a contribution, *a1*, *a2* & *a3*, to deceleration

While *altitude=0*
wheel_brakes Sustains *decel_wb>a1* 13 A
spoilers Sustains *decel_sp>a2* 14 A
reverse_thrust Sustains *decel_rt>a3* 15 A

Their contributions to deceleration are roughly additive, so that:

deceleration ≅ *decel_wb* + *decel_sp* + *decel_rt* 16 A

We know from details of the design that the values *a1*, *a2* & *a3* of the components of deceleration are such that:

a1 + *a2* + *a3* > *a* 17 A

So that:

decel_wb>a1 ∧ *decel_sp>a2* ∧ *decel_rt>a3* ⇒ *deceleration>a* 18 D

So it will be sufficient to show that:

While *altitude=0* ∧ *distance>0*
brakes_commanded Sustains *wheel_brakes* ∧ 19 G
brakes_commanded Sustains *spoilers* ∧ 20 A
brakes_commanded Sustains *reverse_thrust* 21 G

However, when we look more closely at the system, we find that we cannot show the truth of 19 or 21, which will in its turn invalidate 7, and therefore 3. This is because the BSC prevents the application of any of the braking system components in situations which could be unsafe. In particular, they are not permitted to be deployed while the aircraft is airborne, so:

altitude>0 prevents *wheel_brakes* 22 A
altitude>0 prevents *spoilers* 23 A
altitude>0 prevents *reverse_thrust* 24 A

These restrictions by themselves do not cause us a problem. However, there are also further restrictions on the application of wheel brakes and reverse thrust.

Wheel Brakes

There are restrictions on wheel brakes to prevent skidding on wet surfaces, in which the wheel-speed must be greater than a critical value, *antiskid*:

wheel_speed<antiskid prevents *wheel_brakes* 25 A

The fuller causal expression for applying wheel-brakes is:

While *brakes_commanded* ∧ *~wheel_speed<antiskid*,
altitude=0 Sustains *wheel_brakes* 26 A

However, the wet surface presented insufficient friction to bring the wheel speed up to *antiskid* for some time immediately after landing, so that 26 was not satisfied for a significant period.

Reverse Thrust

There are also restrictions on *reverse_thrust*. Reverse thrust is highly safety-critical; its deployment while an aircraft was airborne was the cause of a crash (the Lauda Air crash in 1992). Therefore it must not rely upon a single indicator, so in addition to 24, we have the requirement that the system must also detect *weight_on_wheels*:

~*weight_on_wheels* prevents *reverse_thrust* 27 A

The fuller causal expression for applying reverse thrust is:

While *brakes_commanded* $\wedge$ *weight_on_wheels*, *altitude=0* Sustains *reverse_thrust*28 A

Again, the conditions were not met; the definition of "weight on wheels" was that a weight of 12 tonnes was present on *both* of the main (front) landing gear. The pilot deliberately landed the aircraft rolled over at an angle in order to cope with the cross-wind that he had been informed existed. However, there was actually a tail wind so that, for a significant period, only one wheel was on the ground. Until both wheels touched down the antecedent of 28 was not satisfied, and *reverse_thrust* was not deployed.

Discussion

It is clear that we have not proved our primary goal and, indeed, if we used more detailed information about the value of a2 and the time that passed before the antecedents of 26 & 28 became true, we could refute it[13]. But that would complicate the argument with detail which is not relevant to our causal notation. What went wrong was:

- The wet runway surface presented insufficient friction to the wheels, for a significant period, to satisfy *wheel_speed>antiskid*, thus preventing the immediate application of wheel brakes.
- Landing on one wheel prevented *weight_on_wheels* from being satisfied, thus preventing the deployment of reverse thrust.

The outcome was that there was insufficient deceleration and the aircraft overshot the runway and hit a barrier.

This example is abbreviated and oversimplified, but it shows how a causal notation can contribute to the systematic analysis of a failure. Here are some observations on it:

- We have concentrated on *sufficient* causes and failed to prove that the system was safe. A complementary approach would be to analyse the *necessary* causes of the accident.
- It is clear that the notation must be deployed with others to be effective; even in this brief example we have touched upon several:

[13] We could, of course, also use Samuel Johnson's refutation of Bishop Berkeley's arguments about the nature of reality, and point to the facts!

- Incorporation of the arguments into a formal deductive system
- Identification of agents;
- Refinement of the specification by event splitting;
- The equations of motion;
- Temporal assumptions.

In all these cases we plead that it would have distracted attention from the causal notation.

- A number of assumptions – all those annotated with A – are injected into the specification. We see this as the most appropriate way of introducing reasoning which cannot easily be expressed in the chosen formal notation, such as solutions to differential equations and integrals, and this will be a permanent feature of specifications of this kind. On the other hand, the informal deductions of this example should be replaced by formal reasoning in the chosen notation. Since the causal expressions can be translated into this notation, all of its reasoning support can then be used.
- There are potential problems of scalability. As soon as one examines a complex specification in detail there is a statement explosion.
- There are process issues involved in developing a specification in which goals are presented and then modified, as in condition-splitting, which could help with scaling up.

7. Related Work

In preparing this paper we have reviewed publications in the areas of the philosophy of causation and of formal treatments in computer science. We comment selectively on those which have influenced us. Fuller reviews are to be found in Sosa [15], Shoham [5] and Haddawy [13].

Mackie [16] defines a (**INUS**) cause as an Insufficient and Necessary condition of a Unnecessary and Sufficient condition for the effect; C is a cause of E just in case $(C \wedge X) \vee Y$ is a necessary and sufficient condition for E. There are similarities between this and an expression in our notation such as While $((x_1 \wedge x_2) \vee (y_1 \wedge y_2))$, c causes e. The component $(x_1 \wedge x_2)$ could be described as a "sufficient" component of the While condition expression, and x_1 could be described as a "necessary" component of that sufficient component. However, the words "necessary" and "sufficient" are used differently from our notation, in which they apply to the whole causal relationship, so the similarity between Mackie's approach is more in the nature of a pun than a substantial relationship. Mackie's C is only a component of his causal expression with no distinguishing features; our c is regarded as the single causal trigger, enabled by a While condition expression.

Burks [17] introduces a logic of causal propositions in which he introduces causal implication, distinguished from both material and strict implication, and defined by a series of axioms. There is no explicit reference to time.

We have already referred to Suppes' probabilistic treatment of causation [10], above. It is clear that we shall need to extend the present work to take into account his views.

The AI (artificial intelligence) community, for practical reasons, is the major area of computer science for formal treatment of causation. Their motivation is to predict the effects of actions of, for example, robots. The foundation of this work has been McCarthy's situation calculus [18], and there has been a need to deal with the "frame problem", the difficulties imposed by the need to express the assumption that nothing changes without explicit actions. They are generally less interested in an explicit "cause" construct than in dealing directly with actions, and there is a strong emphasis on the need for non-monotonic logics, in which inferences may have to be withdrawn in the light of additional facts. Shoham [5] introduces a logic of chronological ignorance, a non-monotonic temporal logic in which the state of knowledge can be represented. He introduces a class of causal theories in this logic, in which causal inferences can be made. Haddawy [13] introduces a logic of time, chance and action in which time and uncertainty can be associated with actions, as an aid to planning in AI. Ortiz [19] provides an analysis which distinguishes agents' intentional and unintentional actions in terms of their causal pathways. Darwiche [20] characterises causal structures in order to determine the consistency of a logical database. Geffner [21] deals with the basic preferences which arise in causal default reasoning. Iwasaki [22], following on from [23] and de Kleer [4], goes in considerable depth into the introduction of causal expressions into a language for the functional description of devices. They limit themselves to the equivalent of our direct cause. Lin [24] introduces an equivalent to direct cause as a means of representing the indirect effects of actions.

Since the AI community is more advanced in its practical handling of causal reasoning, further study is require to determine the applicability of these, and other, papers to our own area. On the other hand, we believe that the concerns of Requirements Engineering are quite a lot different from those of AI. In particular, in Requirements Engineering we can rely upon the knowledge of domain experts to feed relevant facts into the specification at the points at which they are relevant. This is not a feasible approach in the AI field, in which an automaton can only call upon the knowledge that it has been given explicitly in advance.

Within the area of requirements analysis, analysis of the safety of critical systems has implicitly used the concept of causation in techniques such as Fault Trees. Gorski [25] formalises fault trees, introducing causal relationships on transitions and actions, with a formal syntax, but no semantics.

8. Conclusions

This paper has presented a suggested approach to specifying causal relationships, with the following characteristics:

- It has a precise syntax and partially specified semantics which can be accommodated within a standard temporal logic, so it can be accommodated within existing languages rather than insisting on being free-standing.
- It defines four flavours of causation: cause, prevent, sustain and terminate. It also recognises the difference between necessary and sufficient cause, defining these interdependently in terms of the flavours of causation.
- It does not claim to be a universally applicable approach to causation. It is based on the intuition that for something to happen, there must be a direct cause which triggers it off. This is an appropriate model for safety analysis of individual systems, in which causes of this kind must be found. Contrast it, however, with Suppes [10]; the examples in his work

are concerned with inferring causation from statistical models, e.g. if a disease is less likely to occur after administration of a vaccine than in the absence of a vaccine, then the vaccine is a *prima facie* cause of immunity. It is clear that his model is entirely appropriate for his area of work and that ours, as presented here, would have little to contribute; conversely, his model has little to contribute to the analysis of individual system hazards, although statistical analysis of failures is relevant to other aspects of safety analysis.

Our approach satisfies most of the properties described by Shoham; there are two main differences:

- We provide no detailed description of the contextual background to "cause", although the "while" qualifier can in principle deal with this. We argued above (section 7) that we can be much more relaxed about contextual knowledge in the field of requirements engineering than in AI; it need only be fed into the model of the system or environment when the system engineer has a need for it.
- On the other hand, we explicitly recognise that, while causal relationships imply predictable relationships between observations of conditions and events, it is never possible to deduce causal relationships from observations. The most that we can do, as with scientific theories, is to hypothesise causal relationships from observations and plausible explanations, and then support or falsify the hypotheses by further observations. In AI, it appears to us that these considerations are irrelevant.

One basis of our approach is that we have deliberately not set out to create a complete language or logic which would displace existing ones. We recognised that, to be useful, our model would have to fit into existing logics; we have made minimal assumptions about time, and so the model is compatible with any temporal logic.

On the other hand we recognise that our advocated approach to the development of causal specifications, particularly the technique of refinement by event-splitting, which is associated with the need to be able to work at varying granularities of time, poses further technical challenges. It will not be sufficient to refine the granularity of time over the entire specification, as this will add unnecessary complication to those portions which can be dealt with at a coarse level. It will be necessary to focus in on particular portions of the specification and alter the granularities in those areas. At present we do not know how to do this formally. Nor do we understand how the refinement of granularity should relate to conventional stepwise refinement.

A further characteristic of our approach is not crucial to our model, but affects the validation and refinement of causal specifications: whether or not the ultimate ideal is validation and refinement through rigorous formal proofs, today's practice must be a good deal less rigorous. On the other hand, we do not live in a black-and-white world, in which proofs are either rigorous or valueless. Justifications for validation or refinement steps vary from being rigorous, through informal but thorough, to hopeful hand-waving. All of these have a place at various points of a specification, but the degree of reliance to be placed on them must be made explicit. Therefore development methods should be able to express the method, or strategy, used for a step, and the justification for this method; a goal-directed method with explicit strategies, as described in section 5.4, above, is therefore appropriate.

The further work needed to build upon our approach falls into several categories:

- Case studies to validate the usefulness of this model; its application in [1] appears promising.
- Based on this, consideration of whether our flavour-based approach to causal expressions is adequate. Allen [25] defines 13 possible relationships between intervals, and this may be an indication of the order of magnitude of the number of types of causal expression that is required. In that case, a toolkit approach may be more appropriate, in which the user can build appropriate expressions from primitives that are supplied.
- Integration of the model into other requirements specification languages.
- Technical extensions and corrections, most notably extension of the notation to cover repetitive and probabilistic causes.

We believe that the use of causal specifications, made feasible by this approach, is another small step forward in bringing the customers requirements and the designers specifications closer together.

Acknowledgements

We thank our colleagues for helpful comments, particularly Philip Morris whose earlier work motivated this paper, and Jeremy Jacob, Dirk Jonscher and Ian Toyn for useful comments on drafts of the paper. We also acknowledge the anonymous reviewers, whose comments helped improve it. This research was supported by UK EPSRC grant GR/G49531.

References

1. Coombes, A., McDermid, J., Moffett, J., & Morris, P. (1995). Requirements Analysis and Safety: A Case Study (Using GRASP). In *SafeComp '95: 14th International Conference on Computer Safety, Reliability and Security*, . Villa Carlotta, Belgirate, Italy:
2. Jeremaes, P., Khosla, S., & Maibaum, T. S. E. (1986). A Modal (Action) Logic for Requirements Specification. In P. J. Brown & D. J. Barnes (Eds.), *Software Engineering '86* (pp. 278-294). Peter Peregrinus.
3. Leitch, R. (1993). *Recent Progress in the Development of Qualitative Reasoning* No. Heriot-Watt University, Scotland, UK.
4. de Kleer, J., & Brown, J. S. (1984). A Qualitative Physics Based on Confluences. *Artificial Intelligence, 24*, 7-83.
5. Shoham, Y. (1986) *Reasoning about Change: Time and Causation from the Point of View of Artificial Intelligence*. PhD Thesis YALEU/CSD/RR#507, Dept of Computer Science, Yale University.
6. Spivey, J. M. (1992). *The Z Notation: A Reference Manual* (2nd ed.). Prentice Hall.
7. Toyn, I., & McDermid, J. A. (1995). CADiZ: An Architecture for Z Tools and its Implementation. *Software Practice and Experience, 25*(3), 305-330.
8. Pnueli, A., & Harel, E. (1988). Applications of Temporal Logic to the Specification of Real-Time Systems. In M. Joseph (Eds.), *Formal Techniques in Real-Time and Fault-Tolerant Systems* (pp. 84-98). Springer-Verlag.

9. Jahanian, F. & Mok, A. K. (1986). Safety Analysis of Timing Properties in Real-Time Systems. *IEEE Transactions on Software Engineering*, SE-12(9), 890-904.

10. Suppes, P. (1970). A Probabilistic Theory of Causation. *Acta Philosophica Fennica, XXIV*.

11. Milner, R. (1980). *A Calculus of Communicating Systems*. Springer-Verlag.

12. von Wright, G. H. (1993). On the Logic and Epistemology of the Causal Relation. In E. Sosa & M. Tooley (Eds.), *Causation* (pp. 105-124). Oxford University Press.

13. Haddawy (1994). *Representing Plans under Uncertainty: A Logic of Time, Chance and Action*. Springer-Verlag.

14. McDermid, J. A. (1994). Support for Safety Cases and Safety Arguments using SAM. *Reliability Engineering and System Safety, 43*, 111-127.

15. Sosa, E., & Tooley, M. (Ed.). (1993). *Causation*. Oxford University Press.

16. Mackie, J. L. (1993). Causes and Conditions. In E. Sosa & M. Tooley (Eds.), *Causation* (pp. 33-55). Oxford University Press.

17. Burks, A. W. (1951). The Logic of Causal Propositions. *Mind, 196*, 363-382.

18. McCarthy, J., & Hayes, P. (1969). Some Philosophical Problems from the Standpoint of Artifical Inteligence. In *Machine Intelligence 4*. Meltzer, B. & Michie, D. (Eds). Edinburgh University Press, Edinburgh, 463-502.

19. Ortiz, C. I. (1994). Causal Pathways of Rational Action. In *12th National Conference on Artificial Intelligence (AAAI-94),* (pp. 1061-1066). Seattle, WA, USA:

20. Darwiche, A., & Pearl, J. (1994). Symbolic Causal Networks. In *12th National Conference on Artificial Intelligence (AAAI-94),* (pp. 228-244). Seattle, WA, USA:

21. Geffner, H. (1994). Causal Default Reasoning: Principles and Algorithms. In *12th National Conference on Artificial Intelligence (AAAI-94),* (pp. 245-250). Seattle, WA, USA:

22. Iwasaki, Y., Vescovi, M., Fikes, R., & Chandrasekaran, B. (1995). Causal Functional Representation Language with Behavior-Based Semantics. *Applied Artificial Intelligence, 9*(1), 5-31.

23. Iwasaki, Y., & Simon, H. A. (1986). Causality in Device Behavior. *Artificial Intelligence, 29*(1), 3-32.

24. Fangzhen Lin (1995). Embracing Causality in Specifying the Indirect Effect of Actions. Technical Report, Dept of Computer Science, University of Toronto.

25. Gorski, J. (1995). Formalising Fault Trees. In F. Redmill & T. Anderson (Eds.), *Achievement and Assurance of Safety* (pp. 311-327). Springer-Verlag.

26. Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals, CACM (83), vol 26 no. 11. *Communications of the ACM, 26*(11), 832-843.