

# Root Cause Analysis for Fulfillment Decisions

D. Gueorguiev 8/6/2023

## Table of Contents

<b>Root Cause Analysis for Fulfillment Decisions.....</b>	<b>1</b>
<b>Preliminaries.....</b>	<b>1</b>
Notation .....	1
Assumptions .....	3
Events .....	3
Event Relationships .....	4
Directed Follow Graphs .....	6
Causal association between events.....	10
Directed Causal Graphs .....	12
<b>Problem Statement for Root Cause Analysis of Fulfillment Decisions.....</b>	<b>13</b>
<b>Algorithm For Root Cause Analysis.....</b>	<b>13</b>
<b>Appendix A: Probabilistic Temporal Logic .....</b>	<b>13</b>
Definitions and Review on Probabilistic Temporal Logic in (Kleinberg, Causality, Probability, and Time, 2012).....	13
Examples of PTL.....	14
Definitions and Review on Probabilistic Real Time Computation Tree Logic (PCTL) in (Hansson & Jonsson, 1994) .....	20
<b>Bibliography.....</b>	<b>26</b>
<b>Downloadable Links for the Bibliography .....</b>	<b>27</b>

## Preliminaries

Before we can formulate the problem statement and the algorithm providing a solution, we need to start with a set of notational conventions and definitions.

### Notation

$A, B, \dots, Z$  - with capital Latin letters we will denote *scalar quantities* which are either essential algorithm parameters or constants which will not change during the algorithm execution; for example, *number of feasible nodes for the current bundle* (scalar constant) will be denoted with  $N$  and *inventory for given SKU on given node* (algorithm parameter) will be denoted with  $I$ . Graphs will also be denoted with capital Latin letters for historical reasons.

$a, b, \dots, z$  - with small Latin letters we will denote *variable/unknown (integral or not) quantities*, not necessarily scalar. For example, with  $x$  we can denote the number of order-lines fulfilled at a given node.

$\alpha, \beta, \dots, \omega$  - with small Greek letters we will denote *variable/unknown (integral or not) quantities*, not necessarily scalar.

$\mathcal{A}, \mathcal{B}, \dots, \mathcal{Z}$  - with capital Script letters we will denote a *set* (ordered or unordered) of quantities of the same type; for example, with  $\mathcal{S}$  we will denote the set of SKUs in some bundle of some order

$A, B, \dots, \Omega$  - with capital Greek letters we will denote a *concept, logical statement* or a *logical expression* of *logical terms / statements* which is adorned with *semantic meaning*. In case of a logical statement, the latter can be either true or false depending on the context. The capital Epsilon letter  $E$  will be reserved to denote an event type or event of interest. For instance,  $E_0$  will denote the event of type “an order has been received”.

$\mathfrak{A}, \mathfrak{B}, \dots, \mathfrak{Z}$  - with capital Fraktur letters we will denote a *map* over several arguments where at least one of those arguments is of type logical expression, a logical statement or a set of logical statements. For example,  $\mathfrak{R}(\Delta, \mathcal{E})$  denotes graph representation of the concept  $\Delta$  by the set of events  $\mathcal{E}$ .

$\mathbb{A}, \mathbb{B}, \dots, \mathbb{Z}$  - with double struck Latin capital letters we will denote standard number sets. For example

$\mathbb{C}$  - the set of complex numbers

$\mathbb{N}$  - the set of natural numbers

$\mathbb{R}$  - the set of the real numbers

$\mathbb{Z}$  - the set of integer numbers

Reserved letters for quantities, sets and concepts:

$B_t$  – number of bundles in the order  $t$ .

$o_t$  – order received at moment  $t$ .

$b_i(o_t)$  – the  $i$ -th bundle of order  $o_t$ ; alternatively, denoted as  $b_{i,t}$ .

$\mathcal{S}_i(o_t)$  or  $\mathcal{S}_{i,t}$  - the set of SKUs for the  $i$ -th bundle will be denoted with  $\mathcal{S}_i$ .

$x_{i,t}$  - denotes some quantity  $x$  related to the  $i$ -th bundle of the  $t$ -th order.

$y_{s,j}$  – denotes some quantity  $y$  related to the SKU  $s$  at node  $j$  e.g., inventory for SKU  $s$  at node  $j$ .

$G$  - directed graph

$\mathcal{V}(G)$  - the vertex set of the directed graph  $G$

$\mathcal{A}(G)$  – the arc set of the directed graph  $G$

$\omega(E_a|E_b)|_{\mathcal{D}}$  – denotes the *relative frequency of occurrence* of the event  $E_a$  given event  $E_b$  with the dataset  $\mathcal{D}$

$S(E_b \rightsquigarrow E_a|\mathcal{K})$  – denotes *Average Degree Of Causal Significance (ADCS)* of event  $E_b$  for event  $E_a$  given the background contexts  $\mathcal{K}$

$E_a < E_b$  denotes the statement that event  $E_b$  *follows in time* event  $E_a$

$E_a \lesssim E_b$  denotes the statement that event  $E_b$  *generally follows* event  $E_a$  (either in time or via static association)

$E_a < E_b$  denotes the statement that event  $E_a$  *precedes in time* event  $E_b$

$E_a \leqslant E_b$  denotes the statement that event  $E_a$  is *generally reachable from* event  $E_b$  (either in time or via static association)

$E_a \leq E_b$  denotes the statement that event  $E_a$  is *reachable in time* from event  $E_b$

$\mathfrak{R}(\Delta)$  – denotes graph representation of the concept  $\Delta$

$\mathfrak{R}(\Delta, \mathcal{E})$  - denotes complete representation of the concept  $\Delta$  with the event set  $\mathcal{E}$

$\mathbb{G}: \mathcal{E} \times \mathcal{E} \rightarrow \{0,1\}$  – denotes static dependency map

$\mathfrak{A}: \mathcal{E} \times \mathcal{E} \rightarrow \{0,1\}$  – denotes static association map

Reserved symbols for relations and operations

$\wedge$  - denotes logical conjunction

$\vee$  - denotes logical disjunction

$\neg$  - denotes logical negation

$<$  - denotes *follows in time* relation between two concepts

$\lesssim$  - denotes *generally follows* relation between two concepts

$\leq$  - denotes *is reachable in time* relation between two concepts

$\leqslant$  - denotes *generally reachable* relation between two concepts

$\rightharpoonup$  - denotes *static dependency* between two concepts

$\rightsquigarrow$  - denotes *dynamic dependency* between two concepts

$\rightleftarrows$  - denotes *association (static, dynamic)* between two concepts

$\rightarrow^\epsilon$  – denotes  $\epsilon$ -*spurious cause* relation between two concepts

$\leftrightarrow$  - denotes *causal association* between two concepts

$\rightsquigarrow$  - *prima facie* causal relation between two concepts

$\rightsquigarrow$  - denotes Eells causal relation between two concepts

$\triangleleft$  - denotes *matching* between directed follow graph (DFG) and a concept

## Assumptions

All orders can be ordered in an increasing sequence of moments in time  $t_1, t_2, \dots, t_o, t_{o+1}, \dots$ . That is, we assume that no two orders will arrive at the same moment in time. Thus, the time  $t$  will take the form of a discrete variable on the natural numbers i.e.,  $t \in \mathbb{N}$ . Therefore, any order will be uniquely identified by a subscript  $t \in \mathbb{N}$ .

### Definition: Atomic Proposition

A basic proposition (or *atom*) which cannot be represented as a set of other atoms connected using conjunction  $\wedge$ , disjunction  $\vee$ , negation  $\neg$ , implication  $\therefore$  and equivalence  $\Leftrightarrow$ .

## Events

### Definition: Event

The word *Event* will be used to denote a *specific kind of an event* which is relevant for the causal analysis. *Event* can be viewed as a *template* from which a specific event can be *instantiated*. We will denote each event with capital Greek letter. Where it will be clear from the context, we will use interchangeably the word “*event*” to denote either *Event of specific kind* or an *Event instance*.

### Definition: Parameters of Event

Each event has a *set of parameters* which will be denoted with  $\mathcal{P}$ . The set of parameters  $\mathcal{P}$  of an event  $E$  together with the semantic description  $\mathcal{S}$  of the event uniquely identify the event. One can think of the semantic description  $\mathcal{S}$  as sort of “*semantic*” *template* (or *predicate* from some first order logic) identifying this event type. The template parameters will be given obviously with the parameter set  $\mathcal{P}$  which is an ordered set. Thus, each event is defined with the pair  $(\mathcal{S}, \mathcal{P})$ . An Event Instance additionally to  $\mathcal{S}$  and  $\mathcal{P}$  is given a specific value  $v$  for each  $p \in \mathcal{P}$ . We will denote the value space of an Event Instance with  $\mathcal{V}$ . Thus, an event instance is defined with the triplet  $(\mathcal{S}, \mathcal{P}, \mathcal{V})$ .

**Note:** Every event additionally to its standard parameter set  $\mathcal{P}$  will have an implicit timestamp parameter  $\tau$  which will always be present without regard of the nature of the event. We will not include explicitly the timestamp among the event parameters unless it is necessary in order to define uniquely the event instance.

We define the following *Events* which are relevant for the analysis of Fulfillment decisions causing splits:

### Set of events for analysis of the cause of splits in Fulfillment Decisions

$E_0$ - order  $O_t$  is received. Event parameters:  $t$

$E_1$ - the  $i$ -th bundle of order  $O_t$  is being processed. Event parameters:  $t, i$

$E_2$ - SKU  $s$  in the  $i$ -th bundle of order  $O_t$  is being processed. Event parameters:  $t, i, s$

$E_3$ - node  $j$  has sufficient inventory for SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_4$ - node  $j$  has sufficient capacity for SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_5$ - node  $j$  is shipping eligible for SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_6$ - node  $j$  is deprioritized; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_7$ - node  $j$  is turned on; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_8$ - node  $j$  is turned off; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_9$ - node  $j$  is soft capacity; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_{10}$ - service level  $sl$  for node  $j$  is overridden; node  $j$  has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j, sl$

$E_{11}$ - carrier  $c$  for node  $j$  is overridden; node  $j$  has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j, c$

$E_{12}$ - backlog days  $d$  for node  $j$  is overridden; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j, d$

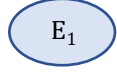
$E_{13}$ - node  $j$  is with depleted inventory; node  $j$  has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_{14}$ - node  $j$  is with depleted capacity; node has SKU  $s$  in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

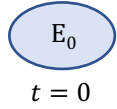
$E_{15}$ - node  $j$  contains an orphan for SKU  $s$  which is in  $\mathcal{S}_i$  with order  $O_t$ . Event parameters:  $t, i, s, j$

$E_{split}$  – splitting decision is made; that is, at least for one bundle  $i$  the SKUs in  $S_i$  are fulfilled by more than one node with order  $O_t$ . Event parameters:  $t, B, i_1, i_2, \dots, i_k$

We will visualize an event with an ellipse and a Capital Greek letter denoting the event. For example:



We will visualize an Event instance with an ellipsis and will use a Capital Greek letter to denote the specific kind of event which it is an instance of. We will attach a set of labels where each label will represent an *atomic proposition* with pertinent semantic information for this instance. For example, in case of  $E_0$  we can have:



Thus, the parameter set of  $E_0$  is given with  $\mathcal{P}_0 = \{t\}$ . Since  $t \in \mathbb{N}$  the value space for the parameters of the event instances of  $E_0$  is given with  $\mathcal{V}_0 = \mathbb{N}$ .

Let us consider the event  $E_1$ :  $\mathcal{P}_1 = \{t, i\}$ . Since  $t, i \in \mathbb{N}$  the value space for the parameters of the event instances of  $E_1$  is given with  $\mathcal{V}_1 = \mathbb{N} \times \mathbb{N}$ .

For  $E_2$  we have accordingly  $\mathcal{P}_2 = \{t, i, s\}$  and  $\mathcal{V}_2 = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ .

For  $E_3, \dots, E_9, E_{13}, E_{14}$  we have accordingly  $\mathcal{P}_m = \{t, i, s, j\}$  and  $\mathcal{V}_m = \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ . Here  $m \in \{3, 4, \dots, 9, 13, 14\}$

For

//TODO: finish this

## Event Relationships

**Definition:** Event  $E_b$  *follows in time* Event  $E_a$

We say that event  $E_b$  *follows in time*  $E_a$  (denoted with  $E_a < E_b$ ) if event  $E_b$  has occurred in time *after* event  $E_a$  and there does not exist a third event  $E_c$  which occurs *after*  $E_a$  and *before*  $E_b$ .

The *follows in time* relation implies the timestamp associated with  $E_b$  is newer than that associated with  $E_a$  i.e.  $\tau_b > \tau_a$ . See the **Note** on the event parameters.

The *follows in time* relation is a strong partial order which satisfies the conditions:

- Irreflexivity:  $E_a \not< E_a$
- Asymmetry: if  $E_a < E_b$  then  $E_b \not< E_a$
- Transitivity: if  $E_a < E_b$  and  $E_b < E_c$  then  $E_a < E_c$

Note that the *follows in time* relation is not guaranteed to hold for every pair of event instances – that is, if  $E_a$  and  $E_b$  are event instances then it can be true that both  $E_a \not< E_b$  and  $E_b \not< E_a$ . This would be the case if  $E_a$  and  $E_b$  have the same timestamp or at least one of them does not have timestamp specified in its parameter set.

**Definition:** Static (semantic) dependency between events  $E_a$  and  $E_b$

We say that event  $E_b$  is static dependent on  $E_a$  (denoted with  $E_a \rightarrow E_b$ ) if each instance of  $E_b$  can exist only *in the context* of some instance of  $E_a$  for *any* order data set  $\mathcal{D}$ . That is, removing an instance of  $E_a$  in  $\mathcal{D}$  will remove all instances of  $E_b$  underneath  $E_a$  from the event tree for any chosen  $\mathcal{D}$ .

if there is a static dependency then we can define a map (called *static dependency map*)  $\mathcal{S}: \mathcal{E} \times \mathcal{E} \rightarrow \{0,1\}$  such that  $\mathcal{S}(E_a, E_b) = 1$  when  $E_a \rightarrow E_b$ .

Example of static (semantic) dependency-

Let the event  $E_0$  denote the statement that an order  $O_t$  with two bundles was received. Let the event  $E_1$  denotes the statement that the current bundle being processed is the second bundle for order  $O_t$ . Then  $E_0 \rightarrow E_1$  for the order  $O_t$ .

Example of absence of static (semantic) dependency

Let the event  $E_0$  denote the statement that an order  $O_t$  with two bundles was received.

Let the event  $E_{split}$  denotes the statement that a splitting decision for both bundles in order  $O_t$  is made; that is, the SKUs in both bundles are fulfilled by more than one node with order  $O_t$ .

Let the event  $E_{15}$  denotes the statement that node  $j$  contains an orphan for SKU  $s$  which is in  $\mathcal{S}_i$  with order  $O_t$ .

Then we can write  $E_0 \rightarrow E_{split}$ ; that is, the splitting decision for order  $O_t$  can be reached only after order  $O_t$  has been received. However, there is an absence of static dependency between  $E_{15}$  and  $E_{split}$ . The relevant for  $E_{split}(O_t)$  instance of event  $E_{15}(t')$  could have been received in an earlier time  $t'$  than that of order  $O_t$ ; that is,  $t' < t$ . Removing the instance of  $E_0$  corresponding to order  $O_t$  has no impact on the existence of  $E_{15}(t')$ .

**Definition:** Static (semantic) descendant

We say that the event  $E_c$  is a static descendant of another event  $E_a$  if there exist a chain of events such that:

$E_a \rightarrow E_{b_1} \rightarrow E_{b_2} \rightarrow \dots \rightarrow E_{b_k} \rightarrow E_c$  for some  $k \in \mathbb{N}$  or if  $E_a \rightarrow E_c$ .

**Lemma:** *Static (semantic) dependency* defines a directed follow graph of statically associated events

Refer to the DFG shown for **Example 1** as an illustration.

**Definition:** *Dynamic dependency* between events  $E_a$  and  $E_b$

We say that event  $E_b$  is dynamic dependent on  $E_a$  (denoted with  $E_a \leftrightarrow E_b$ ) if all of the following is true:

- $E_a < E_b$
- removing an instance of  $E_a$  may trigger the removal of an event which is static dependent of  $E_b$  or removal of  $E_b$  itself.

//TODO: Finish this

Example of dynamic dependency-

In the previous Example of absence of static (semantic) dependency we considered three events -  $E_0$  (order has been received),  $E_{15}$  (node contains an orphan for SKU in the order),  $E_{split}$  (split fulfillment decision has been made). Clearly, there is some kind of dependency between event  $E_{15}$  and  $E_{split}$  as triggering of event  $E_{15}$  at a time  $t'$  earlier than the time the split decision is made can potentially impact the  $E_{split}$  instance. Clearly, this dependency is not static as event  $E_{15}$  at a time  $t'$  later than the time the split decision is made hence this is not a static dependency. Thus, the relation between  $E_{15}$  and  $E_{split}$  matches the definition of dynamic dependency.

**Definition:** Event  $E_b$  is *associated (statically, dynamically)* with Event  $E_a$

We say that event  $E_b$  is *associated (statically)* with  $E_a$  (denoted with  $E_a \rightleftharpoons E_b$ ) if each instance of  $E_b$  is (static, dynamic) descendant of some instance of  $E_a$  or vice versa. That is, if the

//TODO: Finish this

In order to find how a set of events are associated statically we can define a map (called *static association*)

$\mathcal{A}: \mathcal{E} \times \mathcal{E} \rightarrow \{0,1\}$  such that  $\mathcal{A}(E_a, E_b) = 1$  when  $E_a \rightleftharpoons E_b$ . The construction and depiction of static association map will be discussed in **Example 1** below.

**Definition:** Event  $E_c$  is (generally) reachable from Event  $E_a$

We say that event  $E_c$  is (generally) reachable from  $E_a$  (denoted with  $E_a \preceq E_c$ ) if event  $E_c$  has occurred in time *after* event  $E_b$  or if there is a static dependency between  $E_a$  and  $E_b$  **and**  $E_b$  is reachable from  $E_a$ . Any event is reachable from itself, i.e.,  $E_a \preceq E_a$ . Formally,

$$E_a \preceq E_c \Leftrightarrow \exists E_b \text{ s.t. } (E_b < E_c \vee E_b \rightarrow E_c) \wedge E_b \preceq E_c$$

Note that the (generally) reachable relation  $\preceq$  represents *weak partial order* obeying the reflexivity, asymmetry and transitivity conditions.

**Definition:** Event  $E_b$  is reachable in time from Event  $E_a$

*Is reachable from* relation (denoted with  $\leq$ ) implies that the timestamp associated with  $E_b$  is newer than or equal to that associated with  $E_a$  i.e.,  $\tau_b \geq \tau_a$ . See the **Note** on the event parameters.

Note that the *reachable in time* relation  $\leq$  represents *weak partial order* obeying the reflexivity, asymmetry and transitivity conditions.

**Definition:** (irreflexive) *reachable in time* relation (denoted with  $<$ ) can be defined similarly to its reflexive counterpart postulating that in order an event  $E_b$  to be (irreflexively) reachable in time from  $E_a$  the corresponding timestamp must be **newer** than the timestamp of the other event i.e.  $\tau_b > \tau_a$ . The *reachable in time* relation  $<$  represents *strong partial order* obeying the irreflexivity, asymmetry and transitivity conditions.

We will use both definitions in different occasions.

For example, let the event  $E_8$  denotes the node  $j$  being turned off, and event  $E_1$  denotes order  $O_t$  received at time  $t$ . Then  $E_8 < E_1$  can be interpreted as “node  $j$  was turned off prior to receiving order  $O_t$ ”.

**Lemma:** *reachable in time* relation is the transitive closure of the *follows in time* relation.

**Lemma:** Static (semantic) dependency implies *reachable* relation

That is,  $E_a \rightarrow E_b \therefore E_a < E_b$ . Note that the *reachable* relation between  $E_a$  and  $E_b$  will hold for all pairs of instances of those events.

## Directed Follow Graphs

We use *Directed Follow Graphs* (DFG) to depict order fulfillment scenarios which we are interested to capture.

Each node of the DFG will represent an *Event instance* of interest. We use an arc (or a directed edge) to denote a *follow-in-time* relation  $<$  between two Event instances  $E_a$  and  $E_b$  or static (semantic) dependency  $\rightarrow$  between  $E_a$  and  $E_b$ . Each arc is marked with label which indicates what kind of relation it represents. In this document we draw all arcs which represent *follow-in-time* relation with red color and all arcs which represent *static dependency* with blue color.

Each *follow-in-time* arc has a label with a counter which counts how many times the current arc connecting a pair of events has been seen in the data log given specific dataset.

**Definition:** *Labeled Directed Follow Graph (LDFG)*

We extend the concept of Directed Follow Graph (DFG) by introducing a set of labels to each node and to each arc. Each label represents an atomic proposition which is relevant to the specific node or to specific arc.

We use LDFGs to represent the follow relationships between event types and event instances for a given dataset of orders.

## Discussion on how DFG is constructed

Let us consider a given order data set  $\mathcal{D}$  and let us assume we have Fulfillment Optimization engine processing the set of orders  $\mathcal{D}$  sequentially thereby generating order metrics events. Let us assume we have a parsing engine which combs through the order metrics created after the Fulfillment Optimization engine run. This parsing engine parses the events which it is configured to recognize and assembles the DFG instance based on the parsed events data. Let us denote with  $E_l, l = 1..M$  the events which the parsing engine is configured to recognize. Per our definition of *Parameters of Event* given earlier each event type  $E_l$  is represented by the pair  $(\mathfrak{S}_l, \mathcal{P}_l)$  where  $\mathfrak{S}_l$  is the template of the event which together with the parameter set  $\mathcal{P}_l$  uniquely identifies this type of event. Let us denote with  $\mathcal{V}_l$  the ordered set of values which correspond to each parameter  $p_l \in \mathcal{P}_l$  for all instances of  $E_l$  generated using  $\mathcal{D}$ . Since each event instance has a timestamp, we can construct DFG from the parsed events. Each *follow-in-time* arc between two event types  $E_a$  and  $E_b$  will be labeled with the final count showing how many times this pair of events have been seen in a *follow-in-time* relation  $E_a < E_b$ .

For example:

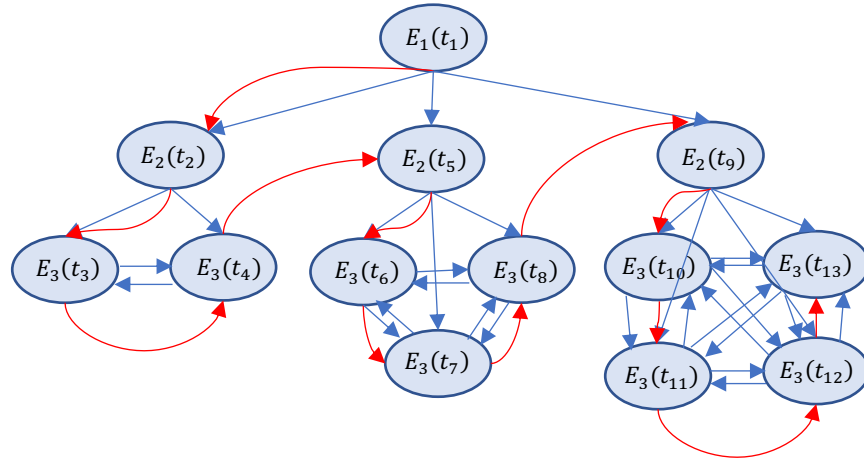
**Example 1:**

An order  $o_t$  for  $t = 0$  has 3 bundles. The first bundle has three SKUs –  $s_1, s_2, s_3$ , the second bundle has two SKUs –  $s_4$  and  $s_5$ , the third bundle has 4 SKUs –  $s_6, s_7, s_8, s_9$ .

Let us denote with  $E_1$  the Event that an order with 3 bundles have been received. Also, we will denote with an instance of event type  $E_2$  each of the bundles of the order  $o_t$ . We denote with an instance of event type  $E_3$  each of the SKUs in each bundle of the order  $O_t$ . We will assume the following time sequence of the event instances:

$$E_1(t_1) < E_2(t_2) < E_3(t_3) < E_3(t_4) < E_2(t_5) < E_3(t_6) < E_3(t_7) < E_3(t_8) < E_2(t_9) < E_3(t_{10}) < E_3(t_{11}) < E_3(t_{12}) < E_3(t_{13})$$

We depict this scenario with the following DFG:



## Discussion on static association map

How can we define the *static association* map  $\mathfrak{A}$ ? The answer can be found in the definition of *Parameters of Event* given earlier. We have the parameter spaces of the two events -  $\mathcal{P}_a$  and  $\mathcal{P}_b$  and the value sets  $\mathcal{V}_a$  and  $\mathcal{V}_b$  of the corresponding event instances. Note that  $\mathcal{P}_a = \{p_a(1), p_a(2), \dots, p_a(P_a)\}$  where  $P_a = |\mathcal{P}_a|$ . Here  $\mathcal{V}_a$  denotes the cartesian product of the value sets for each parameter  $p \in \mathcal{P}_a$  of event  $E_a$ ; thus, we have:

$$\mathcal{V}_a = \mathcal{V}_a(1) \times \mathcal{V}_a(2) \times \dots \times \mathcal{V}_a(P_a).$$

In general, the map  $\mathfrak{A}$  should be defined over the cartesian product of the event tuples  $(\mathfrak{S}_a, \mathcal{P}_a, \mathcal{V}_a) \times (\mathfrak{S}_b, \mathcal{P}_b, \mathcal{V}_b)$ . Let us consider this question from the context of our Fulfillment Decision **Example 1**.

Clearly, we expect that the  $E_0$  instance and all  $E_1$  instances under the parent  $E_0$  instance are statically associated. We expect that each  $E_1$  instance and all  $E_2$  instances which are children of the current  $E_1$  instance are statically associated as well. Let us denote the  $E_0$  instance in this example with  $E_0|_{t=0}$ . Let us denote the three instances of  $E_1$  with  $E_1|_{t=0,i=1}$ ,  $E_1|_{t=0,i=2}$ , and  $E_1|_{t=0,i=3}$ . Then we obviously we have:

$$\begin{cases} E_0|_{t=0} \rightarrow E_1|_{t=0,i=1} \\ E_0|_{t=0} \rightarrow E_1|_{t=0,i=2} \\ E_0|_{t=0} \rightarrow E_1|_{t=0,i=3} \end{cases} \quad (1)$$

The relation (1) represents the fact that each child is statically associated to its parent and is statically dependent on the parent event, namely an order with specified number of bundles has been received. This relation is depicted with arrow  $\rightarrow$  in (1).

Additionally, we can write:

$$\begin{cases} E_1|_{t=0,i=1} \rightleftharpoons E_1|_{t=0,i=2} \\ E_1|_{t=0,i=1} \rightleftharpoons E_1|_{t=0,i=3} \\ E_1|_{t=0,i=2} \rightleftharpoons E_1|_{t=0,i=3} \end{cases} \quad (2)$$

The relation (2) represents the fact that the children of the same parent are statically associated.

Similarly, we continue with writing the static association relations involving the instances of  $E_2$

$$\begin{cases} E_1|_{t=0,i=1} \rightarrow E_2|_{t=0,i=1,s=1} \\ E_1|_{t=0,i=1} \rightarrow E_2|_{t=0,i=1,s=2} \end{cases} \quad \begin{cases} E_1|_{t=0,i=2} \rightarrow E_2|_{t=0,i=2,s=1} \\ E_1|_{t=0,i=2} \rightarrow E_2|_{t=0,i=2,s=2} \\ E_1|_{t=0,i=2} \rightarrow E_2|_{t=0,i=2,s=3} \end{cases} \quad \begin{cases} E_1|_{t=0,i=3} \rightarrow E_2|_{t=0,i=3,s=1} \\ E_1|_{t=0,i=3} \rightarrow E_2|_{t=0,i=3,s=2} \\ E_1|_{t=0,i=3} \rightarrow E_2|_{t=0,i=3,s=3} \\ E_1|_{t=0,i=3} \rightarrow E_2|_{t=0,i=3,s=4} \end{cases} \quad (3)$$

Additionally, all  $E_2$  instances of the same parent are statically associated with each other - we write this as:

$$E_2|_{t=0,i=m,s=p} \rightleftharpoons E_2|_{t=0,i=m,s=q} \quad \forall m = 1,2,3 \text{ and } \forall p, q \in \mathcal{I}(\mathcal{S}_m) \quad (4)$$

Here  $\mathcal{I}(\mathcal{S}_m)$  denotes the index set of  $\mathcal{S}_m$ .

Also, all  $E_2$  instances are associated with their grandparent  $E_0|_{t=0}$  which is expressed with:

$$E_0|_{t=0} \rightleftharpoons E_1|_{t=0,i=m,s=p} \quad \forall m = 1,2,3 \text{ and } \forall p, q \in \mathcal{I}(\mathcal{S}_m) \quad (5)$$

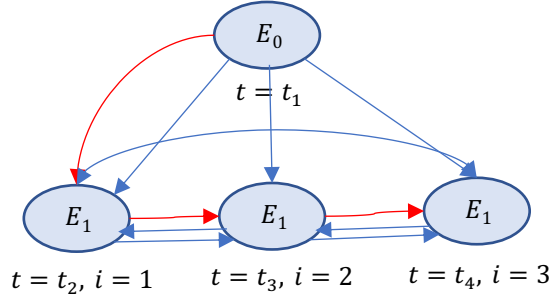
Let us construct the map  $\mathfrak{A}$  for the sets (1)-(5). We start with (1):

//TODO: finish this



**Definition: Directed Follow Graph Instance (DFGI)**

DFGI is a directed graph  $G_i$  in which each node is a specific *event instance (or a token)* of an event type and each **red** arc denotes *follow-in-time* relation  $<$ . Each **blue** arc denotes static (semantic) dependency  $\rightarrow$  between the event instances. Static association  $\rightleftharpoons$  between nodes is shown via a pair of **blue** arcs each pointing to the opposite node. Each node (event instance/token) is labeled with the value set of parameters for this event type. For example:



**Definition: Aggregated Directed Follow Graph (ADFG)**

The ADFG  $G$  corresponding to DFGI  $G_i$  can be obtained by replacing each event instance by its corresponding event type and replacing a multi-set of *follow-in-time* arcs leaving an event instance of type  $E_a$  and entering event instance of type  $E_b$  with a single arc labeled with the corresponding instance count.

**Definition: Frequency Count  $f(E_a, E_b)$**  of pair of events  $E_a$  and  $E_b$  – this is the number  $f$  of DFG instances  $D_i$  in which  $E_b$  directly follows in time  $E_a$  i.e.,  $E_a < E_b$ .

**Definition: DFG Representation** of a concept  $\Delta$  over an event set  $\mathcal{E}$

We say that DFG is a representation of  $\Delta$  if the graph constructed with the events in  $\mathcal{E}$  models *semantically* the internal structure of  $\Delta$ .

For example, the DFG shown in *Example 1* is DFG representation of the order  $O_t$ . The DFG  $G$  representing  $O_t$  will be denoted with  $G = \mathfrak{N}(O_t)$  or shortly  $G(O_t)$ .

**Definition: Complete Representation** of a concept  $\Delta$  over an event set  $\mathcal{E}$

We say that the DFG  $G$  is a *complete representation* of the concept  $\Delta$  (e.g., fulfillment order, fulfillment decision) from the event set  $\mathcal{E}$ , denoted with  $G = \mathfrak{N}(D, \mathcal{E})$ , iff there does not exist DFG  $G_1$  such that  $G_1 = \mathfrak{N}(D, \mathcal{E})$  with  $\mathcal{V}(G) \subset \mathcal{V}(G_1) \subseteq \mathcal{E}$  and  $\mathcal{A}(G) \subseteq \mathcal{A}(G_1)$ .

**Definition: Order Fulfillment Decision**

The process of fulfilling the order which can be viewed as a set of events relevant to the decision which was made.

The events are pairwise related by the *follow-in-time* ( $<$ ) relation which is depicted by red-colored arcs.

Additionally, we depict semantic dependencies ( $()$ ) using blue arcs. The events are represented by DFG over some set of events  $\mathcal{E}$ .

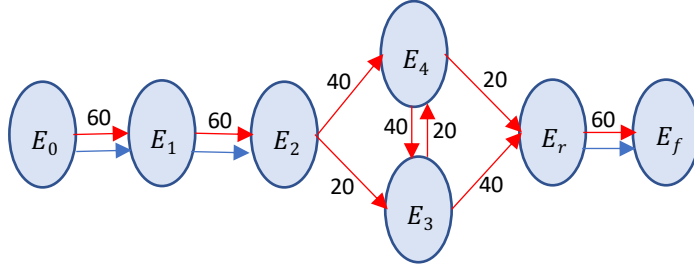
For example:

**Example 2**

Sixty orders  $O_t, t = 1, 2, \dots, 60$  with a single bundle  $b_1$  and single SKU  $s_1$  with unit quantity have been received. Let us define the following event set  $\mathcal{E} = \{E_0, E_1, E_2, E_3, E_4, E_r, E_f\}$ : The event “order  $O_t$  has been received” will be denoted with  $E_0$ . The event “The order bundle is being processed” will be denoted with  $E_1$ . The event “SKU  $s_1$  is being processed” will be denoted with  $E_2$ . The event “node  $n_j$  has inventory for SKU  $s_1$ ” will be denoted with  $E_3$ .

The event “node  $n_j$  has capacity for SKU  $s_1$ ” will be denoted with  $E_4$ . The event “Reward for node  $n_j$  has been calculated” will be denoted with  $E_r$ . The event “Fulfilling node has been chosen” will be denoted with  $E_f$ .

This is visualized as:



**Definition:** *DFG Matching* of an order fulfillment decision

Let  $\Delta_t$  denotes the fulfillment decision of order  $O_t$ . Let  $G$  denotes some DFG. We say that the DFG  $G$  matches  $\Delta_t$  (denoted with  $G \triangleleft \Delta_t$ ) if  $G$  is a representation of  $\Delta_t$  over some set of events  $\mathcal{E}$ .

//TODO: Finish this

### Causal association between events

What does it mean that certain event types can be associated causally with each other? Let us consider two event types -  $E_a$  and  $E_b$ . Per our definition of *Parameters of Event* given earlier the event  $E_a$  is characterized with the pair  $(\mathcal{E}_a, \mathcal{P}_a)$  where  $\mathcal{E}_a$  is the template of the event which together with the parameter set  $\mathcal{P}_a$  uniquely identifies this type of event. Similarly, we will consider another event type  $E_b$  represented by  $(\mathcal{E}_b, \mathcal{P}_b)$ . Now let us consider a given order data set  $\mathcal{D}$  and let us assume we have Fulfillment Optimization engine processing the set of orders  $\mathcal{D}$  sequentially thereby generating order metrics events. Let us denote with  $\mathcal{V}_a$  the ordered set of values which correspond to each parameter  $p_a \in \mathcal{P}_a$  for all instances of  $E_a$  generated using  $\mathcal{D}$ . Similarly, with  $\mathcal{V}_b$  we denote the ordered set of values which correspond to each  $E_b$  parameter and generated for all instances of  $E_a$  using order data set  $\mathcal{D}$ . For an instance of  $E_a$  we will denote the values of the instance parameters with  $v_a$ . Thus, for each instance of  $E_a$  in  $\mathcal{D}$  (denoted as  $E_a|_{\mathcal{D}}$ ) we have  $v_a \in \mathcal{V}_a$ . Similarly, for  $E_b|_{\mathcal{D}}$  we have  $v_b \in \mathcal{V}_b$ .

**Definition:** *Causal association between events*  $E_a$  and  $E_b$  – Given the dataset  $\mathcal{D}$  we say that  $E_a$  and  $E_b$  are associated (causally) if one of the following is true:

- both  $E_a$  and  $E_b$  are causes of another event  $E_c$  in  $\mathcal{D}$  //do we need this?
- both  $E_a$  and  $E_b$  are caused by another event  $E_c$  in  $\mathcal{D}$  //do we need this?
- either  $E_a$  causes  $E_b$  or  $E_b$  causes  $E_a$
- there is a semantic association between  $E_a$  and  $E_b$

Note: we denote causal association between the events  $E_a$  and  $E_b$  with the symbol  $\Leftrightarrow$  i.e.  $E_a \Leftrightarrow E_b$ .

For example, a *prima facie causal association* implies that all causal relationships in its definition are *prima facie causes* (defined in the paragraph below).

**Definition:** *Conditional probability of an event*

Let us consider the event type  $E_a$ . Per our definition of *Parameters of Event* given earlier the event  $E_a$  is characterized with the pair  $(\mathcal{E}_a, \mathcal{P}_a)$  where  $\mathcal{E}_a$  is the template of the event which together with the parameter set  $\mathcal{P}_a$  uniquely identifies this type of event. Similarly, we will consider another event type  $E_b$  represented by  $(\mathcal{E}_b, \mathcal{P}_b)$ . Now let us consider a given order data set  $\mathcal{D}$  and let us assume we have Fulfillment Optimization engine processing the set of orders  $\mathcal{D}$  sequentially thereby generating order metrics events.

Let us run the Fulfillment engine with the given order set  $\mathcal{D}$  and we find that in  $A$  out of the  $N$  instances in which event  $E_a$  has occurred there has been an instance of  $E_b$  associated with each instance of  $E_a$ .

Then given the data set  $\mathcal{D}$  the relative frequency of occurrences of  $E_a$  given  $E_b$  is obtained as:

$$\omega(E_a|E_b)|_{\mathcal{D}} = \frac{A}{N} \quad (6)$$

We say that the relative frequency given  $\mathcal{D}$  is an estimate for the conditional probability  $P(E_b|E_a)$  i.e.

$$P(E_b|E_a)|_{\mathcal{D}} \sim \omega(E_a|E_b)|_{\mathcal{D}} \quad (7)$$

**Definition:** Event  $E_a$  is *prima facie cause* of Event  $E_b$

Given the data set  $\mathcal{D}$  let us denote with  $E_b|_{\mathcal{D}}$  the set of instances of  $E_b$  which follow the set of instances of  $E_a$ , denoted with  $E_a|_{\mathcal{D}}$ . That is,  $\forall E_b|_{\mathcal{D}} \exists E_a|_{\mathcal{D}} s.t. E_a|_{\mathcal{D}} < E_b|_{\mathcal{D}}$ .

We say that event  $E_a$  is a *prima facie cause* of event  $E_b$  (denoted with  $E_a \rightsquigarrow E_b$ ) iff:

the sets  $E_a|_{\mathcal{D}}$  and  $E_b|_{\mathcal{D}}$  are non-empty

and

$$P(E_b|E_a)|_{\mathcal{D}} > P(E_b)|_{\mathcal{D}} \quad (8)$$

**Lemma:** *Prima facie* cause between event  $E_a$  and event  $E_b$  implies dynamic dependency between the two events  
That is,  $E_a \rightsquigarrow E_b \therefore E_a \leftrightarrow E_b$ .

**Definition:** Event  $E_b$  is  *$\epsilon$ -spurious cause* of an Event  $E_a$

Let us consider the event type  $E_a$  given with its template  $\mathfrak{S}_a$  and parameter space  $\mathcal{P}_a$ .

Let us consider another event type  $E_b$  given with its template  $\mathfrak{S}_b$  and parameter space  $\mathcal{P}_b$ .

Given the data set  $\mathcal{D}$  we denote with  $\mathcal{E}_c$  the set of all events with which  $E_a$  is associated such that  $E_c|_{\mathcal{E}_c(\mathcal{D})} < E_b|_{\mathcal{D}}$ .

Let  $E_b$  is an event such that:

- it is not necessarily in  $\mathcal{E}_c$  :  $E_b \notin \mathcal{E}_c$
- $E_a$  is reachable from  $E_b$  i.e.  $E_b|_{\mathcal{D}} < E_a|_{\mathcal{D}}$

Then we say that  $E_b$  is  *$\epsilon$ -spurious cause* of an Event  $E_a$  iff

- $P(E_b \wedge E_c)|_{\mathcal{D}} > 0$
- $|P(E_a|E_b \wedge E_c) - P(E_a|E_c)| < \epsilon$  over  $\mathcal{D}$
- $P(E_a|E_b \wedge E_c) \geq P(E_a|E_c)$  over  $\mathcal{D}$

We denote  *$\epsilon$ -spurious cause* with  $E_c \rightsquigarrow^\epsilon E_a$

**Definition:** Event  $E_b$  is *Suppe's cause* of an Event  $E_a$  (a.k.a. *Suppe's causality*)

We define  $\mathcal{E}_c$  and  $E_b$  as in the definition of  *$\epsilon$ -spurious cause*.

Given the data set  $\mathcal{D}$  we denote with  $\mathcal{E}_c$  the set of all events with which  $E_a$  is associated such that  $E_c|_{\mathcal{E}_c(\mathcal{D})} < E_b|_{\mathcal{D}}$ .

Let  $E_b$  is an event such that:

- it is not necessarily in  $\mathcal{E}_c$  :  $E_b \notin \mathcal{E}_c$
- $E_a$  follows  $E_b$  i.e.,  $E_b|_{\mathcal{D}} < E_a|_{\mathcal{D}}$

Then we say that  $E_b$  *Suppe's cause* of an Event  $E_a$  iff

- $P(E_b \wedge E_c)|_{\mathcal{D}} > 0$
- $P(E_a|E_b \wedge E_c) > P(E_a|E_c)$  over  $\mathcal{D}$  (*Suppe's causal relation hypothesis*)

We denote *Suppe's causality relation* with  $E_b \rightsquigarrow^s E_a$

**Definition:** Event  $E_b$  is *Eells* cause of an Event  $E_a$  (a.k.a. *Eells' causality*)

Let us consider the event type  $E_a$  given with its template  $\mathcal{S}_a$  and parameter space  $\mathcal{P}_a$ .

Let us consider another event type  $E_b$  given with its template  $\mathcal{S}_b$  and parameter space  $\mathcal{P}_b$ .

Given the data set  $\mathcal{D}$  we denote with  $\mathcal{E}_c$  the set of all events with which  $E_a$  is causally associated such that  $E_c|_{\mathcal{E}_c(\mathcal{D})} < E_a|_{\mathcal{D}}$ .

Additionally, we define the following causal *background contexts*  $\mathcal{K} = \{K_1, K_2, \dots, K_n\}$ . Those are formed by holding fixed the set of all factors causally associated with  $E_a$ . For instance, given a set of three associated with  $E_a$  events  $\{E_{c,1}, E_{c,2}, E_{c,3}\}$  one possible background context will be  $\{E_{c,1}, \neg E_{c,2}, E_{c,3}\}$

Let  $E_b$  is an event such that:

- it is not necessarily in  $\mathcal{E}_c$  :  $E_b \notin \mathcal{E}_c$
- $E_a$  is reachable from  $E_b$  i.e.,  $E_b < E_a$

Then we say that  $E_a$  is *Eells*-caused by Event  $E_b$  iff

$$P(E_a|K_i \wedge E_b) \neq P(E_a|K_i \wedge \neg E_b) \quad \forall K_i \in \mathcal{K} \text{ s.t. } K_i < E_a \text{ over } \mathcal{D} \quad ( )$$

We denote *Eells*-causality with  $E_b \rightsquigarrow E_a$

**Definition:** *Average Degree Of Causal Significance (ADCS)* of event  $E_b$  for event  $E_a$  in given context

The *Average Degree Of Causal Significance (ADCS)* of event  $E_b$  for event  $E_a$  given the background contexts  $\mathcal{K}$  is defined as:

$$S(E_b \rightsquigarrow E_a|\mathcal{K}) = \sum_{i=1}^n P(K_i)[P(E_a|K_i \wedge E_b) - P(E_a|K_i \wedge \neg E_b)]$$

We use the Latin capital letter  $S$  to denote *ADCS* from the Lat. *significatio* (significance).

**Lemma:**

Static dependency implies *Eells* causality. However, *Eells* causality does not imply static dependency.

That is, if  $E_a \rightarrow E_b$  then it is true  $E_a \rightsquigarrow E_b$ . However, if  $E_a \rightsquigarrow E_b$  it does not necessarily follow that  $E_a \rightarrow E_b$ .

Example of *Eells*-causality-

Let the event  $E_0$  denote the statement that an order  $O_t$  with one bundle was received. Let the event  $E_1$  denotes the statement that the capacity feasible nodes for order  $O_t$  are node  $i$  and node  $j$ .

//TODO: finish this

**Note:** the *caused by*  $\rightsquigarrow, \rightsquigarrow, \rightarrow, \rightarrow$  relations do **not** impose a total order; that is, for **every** pair of events  $E_a$  and  $E_b$  it does **not** follow that either  $E_a \rightsquigarrow E_b$  or  $E_b \rightsquigarrow E_a$  is true. Therefore, a set of events cannot be visualized as an ordered sequence; instead, we will use *Directed Causal Graph* for the purpose.

## Directed Causal Graphs

**Definition:** *Directed Causal Graph (DSG)*

A directed graph in which each node represents an Event Type, and each arc represents causal relation. Each arc is labeled with causal significant factor, a real number between 0 and 1, describing how significant is the causal relationship between the two event types.

## Problem Statement for Root Cause Analysis of Fulfillment Decisions

The goal of the RCA algorithm applied to Fulfillment Optimization events is to understand and analyze causal relationship between predefined set of events based on the order metrics payloads. Each detected causal relationship will be assigned a significance factor which will indicate based on the supplied dataset how significant was this causal relationship inferred from the dataset and the configured set of events  $\mathcal{E}$ . Thus, the result of a single RCA algorithm run with a given dataset  $\mathcal{D}$  will be a Directed Causal Graph instance  $G$ , where the vertex set will be a subset of the events set i.e.,  $\mathcal{V}(G) \subseteq \mathcal{E}$ . Each arc will represent a causal relationship between the connected events, and it will be labeled with a causal significance factor  $S(E_a, E_b) \in [0,1]$  (abbrev.  $S_{a,b}$ ). For instance, for the set of events shown earlier (see *Set of events for analysis of the cause of splits in Fulfillment Decisions*) we can have the following output of the RCA algorithm:

//TODO: finish this

## Algorithm For Root Cause Analysis

*Brief description of the RCA algorithm*

1. Choose a set  $\mathcal{E}$  of events of interest.  $E_\alpha \in \mathcal{E}, \alpha \in \mathcal{I}$
2. Compile order sequence  $o_1, o_2, \dots, o_t, \dots$  from the given events dataset  $\mathcal{D}$
3. Using the given dataset  $\mathcal{D}$  create Directed Follow Graph instances  $G_t$  for each order  $o_t$  in the dataset.
4. From the created  $G_t, t = 1, 2, \dots$  construct Aggregated Directed Follow Graph  $G$  with the set of events of interest  $E_\alpha \in \mathcal{E}, \alpha \in \mathcal{I}$
- 5.
6. Using Eell's definition of causality calculate the Average Degree of Causal Significance (ADCS)  $S_{a,b}$  for each pair of nodes in  $G$  using the already calculated in 3. frequency counts  $f_{a,b}$  for each pair of events  $(E_a, E_b)$  in  $G$ .
7. Given a minimum significance level  $S_{min}$  construct Directed Causal Graph (DCG)  $G_C$  using  $G$  and  $S_{a,b}$  for each pair of events in  $\mathcal{V}(G)$  such that every arc  $a - b$  in  $G_C$  will have significance factor  $S_{a,b}$  larger or equal to  $S_{min}$ .

//TODO: finish the algorithm

## Examples

//TODO: finish the examples

## Appendix A: Probabilistic Temporal Logic

Definitions and Review on Probabilistic Temporal Logic in (Kleinberg, Causality, Probability, and Time, 2012)

*Probabilistic Temporal Logic* (PTL) is a tool for state machine model checking which is a more complete alternative of the Labeled DFG defined earlier. A somewhat reduced subset of Probabilistic Temporal Logic is defined with the help of *Kripke* structures. With randomness introduced *Kripke* structure is roughly equivalent to a Discrete Time Markov Chain, and it is just another tool to validate specific first order logic statements relevant for RCA against our process model.

**Definition A1:** *Kripke structure*

Let  $\mathcal{P}$  be a set of atomic propositions. A *Kripke* structure  $M$  over  $\mathcal{P}$  is defined as the tuple  $M = \{\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{Q}\}$  where

- $\mathcal{S}$  is a finite set of states
- $\mathcal{S}_0 \subseteq \mathcal{S}$  is the set of initial states

- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$  is a total transition relation, such that  $\forall s \in \mathcal{S}, \exists s' \in \mathcal{S} \text{ s.t. } (s, s') \in \mathcal{R}$
- $\mathcal{L} : \mathcal{S} \rightarrow 2^{|\mathcal{P}|}$  is a function which labels each state with a set of atomic propositions that are true within it.

The function (relation)  $\mathcal{R}$  being a total transition function (relation) means that for every state, there is at least one transition from that state (to itself or to another state). The function (relation)  $\mathcal{L}$  maps states to the truth values of propositions at that state. Since there are  $|\mathcal{P}|$  propositions, there are  $2^{|\mathcal{P}|}$  possible truth values and  $\mathcal{L}$  maps each state to one of these.

A *path* in a *Kripke* structure is an infinite sequence of states. Precisely, a path  $\pi$  is a sequence of states ( $\pi = s_0, s_1, \dots$ ) such that for every  $i \geq 0$ ,  $(s_i, s_{i+1}) \in \mathcal{R}$ . That says that the series of transitions described in the sequence  $\pi$  is possible. The notation  $\pi^i$  is used to denote the *subpath suffix* of the path  $\pi$  starting with state  $s_i$ .

To find the properties that are true in such kind of structures we need a formal method for representing the properties to be tested. There are number of temporal logic systems which express (slightly) different sets of formula.

We are going to introduce *Computational Tree Logic* (CTL) system which will be used to build upon later and define PTL.

The formulas in CTL are composed of paired *path quantifiers* and *temporal operators*. Path quantifiers describe whether a property holds **for all paths** (denoted with the operator  $A$ ), or **for some path** (denoted with operator  $E$ ), starting at a given state. The temporal operators describe where along the path the properties will hold. For example, if  $f$  is some state, then  $AGf$  is a valid CTL formula, but  $AGFf$  is not, since  $F$  is not paired with one of  $A$  or  $E$ . More formally,

- *Finally* ( $F$ ) – at some state on the path the property will hold
- *Globally* ( $G$ ) – the property will hold along the entire path
- *Next* ( $X$ ) – the property will hold at the next state of the path
- *Until* ( $U$ ) – applies to two properties, the first one holds in every state along the path until at some state the second property holds
- *Weak Until aka Until or Release* ( $W$ )

//Finish this paragraph on CTL

As in CTL, in PTL there are two types of formulas: *path formulas* and *state formulas*. State formulas express properties that must hold within a state, such as being labeled with certain atomic propositions, while path formulas refer to sequences of states along which the formula must hold. The formulas are comprised of atomic propositions  $a \in \mathcal{P}$ , propositional logical connectives (such as  $\neg$ ,  $\vee$ ,  $\wedge$ ), and the modal operators denoting time and probability. The logic syntax tells how valid PTL formulas are constructed:

1. All atomic propositions are state formulas
2. If  $\neg f$  and  $g$  are state formulas, so are  $f$ ,  $f \wedge g$ ,  $f \vee g$ , and

Examples of PTL

$E_a \xrightarrow{\geq p, \leq s} E_b$  : Event  $E_b$  is reachable from event  $E_a$  with probability at least  $p$  after at least  $r$  steps and at most  $s$  steps

//Finish the paragraph on PTL Examples

**Definition A2:** *prima facie* cause expressed with PTL formulas

1.  $F_{>0}^{\leq \infty} c$
2.  $c <_{\geq p}^{\geq 1, \leq \infty} e$
3.  $F_{< p}^{\leq \infty} e$

These conditions mean that 1) a state where  $c$  is true will be reached with non-zero probability and 2) the probability of reaching a state where  $e$  is true (within the time bounds) is greater after being in a state where  $c$  is true (probability  $\geq p$ ) than 3) it is by simply starting from initial state of the system (probability  $< p$ ). When making inferences from data that means that  $c$  must occur at least once, and the conditional probability of  $e$  given  $c$  is greater than the marginal probability of  $e$  (usually calculated from frequencies). Since negative (probability lowering) causes can be defined in terms of their complement (so that if  $c$  lowers the probability of  $e$ ,  $\neg c$  raises its probability, the definition here is in terms of positive, probability raising causes.

### *A Review on Suppes' Causality framework discussed in (Otte, 1982)*

**Definition A3:** Suppes' definition of *prima facie* cause

An event  $B_{t'}$  is a *prima facie* cause of event  $A_t$  iff:

1.  $t' < t$
2.  $P(B_{t'}) > 0$
3.  $P(A_t|B_{t'}) > P(A_t)$

We should interpret this as being for all  $t$  and  $t'$  where  $t' < t$ . That is, the probability of  $A$  occurring at any time after  $B$  is greater than the marginal probability of  $A$  occurring at any time. Thus, the conditions 1-3 do not refer to specific values of  $t$  and  $t'$  but rather describe the relationship between  $t$  and  $t'$ . In some cases, these causes may turn out to be false. Even if something meets the criterion of being a *prima facie* cause, this may be due only to common cause of it and the effect. Suppes introduces two ways in which something may be a false, or spurious cause. In each, the idea is that there is some earlier event than the *prima facie* cause that accounts equally well for the effect, so that his other information is known, the spurious cause does not have any influence on the effect.

**Definition A4:** Suppes' first definition of *spurious cause*

An event  $B_{t'}$ , a *prima facie* cause of event  $A_t$ , is a *spurious cause* in sense one iff  $\exists t'' < t'$  and  $C_{t''}$  such that:

1.  $P(B_{t'} \wedge C_{t''}) > 0$
2.  $P(A_t|B_{t'} \wedge C_{t''}) = P(A_t|C_{t''})$
3.  $P(A_t|B_{t'} \wedge C_{t''}) \geq P(A_t|B_{t'})$

While  $B_{t'}$  is a possible cause of  $A_t$ , there may be another, earlier, event that has more explanatory relevance to  $A_t$ . However, condition 2 of the definition above is very strong and perhaps counterintuitive. It means that there exists an event that completely eliminates the effectiveness of the cause for predicting the effect. One way of relaxing this condition is to find not individual events but rather kinds of events. In Suppes' second definition of spurious causes there will be a set of nonempty sets that cover the full sample space, and which are mutually exclusive (pairwise disjoint). Thus, only one of these sets can be true *and* together they cover all possibilities.

**Definition A5:** Suppes' second definition of *spurious cause*

An event  $B_{t'}$ , a *prima facie* cause of event  $A_t$ , is a *spurious cause* in sense two iff there is a partition  $\pi_{t''}$  where  $t'' < t'$  and for every  $C_{t''}$  in  $\pi_{t''}$ :

1.  $P(B_{t'} \wedge C_{t''}) > 0$
2.  $P(A_t|B_{t'} \wedge C_{t''}) = P(A_t|B_{t'})$

Distinction between these two kinds of spuriousness is made with an example given by (Otte, 1982) on pp63:

*For now on I will abbreviate "spurious in sense two" by  $spurious_2$  and "spurious in sense one" by  $spurious_1$ . This definition makes an event  $spurious_2$  if the world can be partitioned in such a way that the above conditions are satisfied. Thus, if we can observe a certain kind of event given by the partition, the observation of the later event  $B_{t'}$  is uninformative, which makes it a  $spurious_2$  cause. Suppes proves that if an event is a  $spurious_2$  cause, then*

it is a *spurious<sub>1</sub>* cause. The converse of this theorem, however, is not necessarily true: it is possible for an event to be a *spurious<sub>1</sub>* cause and not be a *spurious<sub>2</sub>* cause.

As an example of a *spurious<sub>1</sub>* cause, let us take the case of decreasing air pressure causing not only rain but a falling barometer reading. The falling barometer reading is a *prima facie* cause of rain; given that the barometer reading is dropping, the probability that it will rain rises. Letting  $A$  denote rain,  $B$  denote a falling barometer reading, and  $C$  denote decreasing air pressure, the probability of rain given that the barometer reading, and the air pressure are decreasing,  $P(A|B \wedge C)$ , is equal to the probability of rain given that the air pressure is decreasing,  $P(A|C)$ ; thus the second condition of the second definition of spurious cause is satisfied. The third condition is likewise satisfied, since the probability of rain given decreasing air pressure and a falling barometer reading is at least as great as the probability of rain given a falling barometer reading,  $P(A|B \wedge C) \geq P(A|B)$ . Thus, by the second definition a falling barometer reading is a *spurious<sub>1</sub>* cause of rain. The falling barometer reading is a *spurious<sub>2</sub>* cause of rain. If we let  $\pi$  be our partition (decreasing air pressure, non-decreasing air pressure), then

1.  $P(BC) > 0$
2.  $P(A|BC) = P(A|C)$
3.  $P(A|B \neg C) = P(A|\neg C)$

So the falling barometer reading is a *spurious<sub>2</sub>* cause of the rain.

**Definition A6:** Suppes' definition of *genuine cause*

All non-spurious *prima facie* causes i.e., *prima facie* causes which do not meet the **Definition A4** and **Definition A5**.

Looking again at **Definition A4** and **Definition A5** for spurious causes, the stipulation that  $P(A_t|B_{t'} \wedge C_{t''}) = P(A_t|B_{t'})$  means that some causes may not be deemed spurious, despite meeting all the conditions, if there is a small difference in the probabilities on either side of this equality. To address this issue, Suppes introduced the concept of an  $\varepsilon$ -spurious cause.

**Definition A7:** Suppes' definition of  $\varepsilon$ -spurious cause

An event  $B_{t'}$  is an  $\varepsilon$ -spurious cause of event  $A_t$  iff  $\exists t'' < t'$  and a partition  $\pi_{t''}$  such that for every  $C_{t''}$  of  $\pi_{t''}$ :

1.  $t' < t$
2.  $P(B_{t'}) > 0$
3.  $P(A_t|B_{t'}) > P(A_t)$
4.  $P(B_{t'} \wedge C_{t''}) > 0$
5.  $|P(A_t|B_{t'} \wedge C_{t''}) - P(A_t|C_{t''})| < \varepsilon$

This definition means that a genuine cause that has a small effect on the probability of the event being caused will be ruled spurious. The partition  $\pi_{t''}$  separates off the past prior to the possibly spurious cause  $B_{t'}$ . Note that there is no set value for  $\varepsilon$  other than it being small //Can we improve on that?

One issue that arises when using these definitions to determine the true cause of an effect is that we may find an earlier and earlier causes that make the later ones spurious, and the cause may be quite removed from the effect in time (not to mention space). Suppes introduces the idea of *direct cause* to account for this issue. This is a concept very similar to screening off and spurious causes, except here we must consider whether there is some event coming temporarily between cause and effect. Note that there is no link between spurious and indirect causes. A direct cause may still be remote in space (and perhaps in time), but this can rule out indirect remote causes. One of the first problems which we encounter with these definitions is in handling of causal chains. As discussed by (Otte, 1982), pp64:

*Closely related to the notion of spurious cause is the idea of indirect cause. We will first define a direct cause:*

**Definition A8:** Otte's definition of *direct cause*

An event  $B_{t'}$  is a direct cause of  $A_t$  iff  $B_{t'}$  is a *prima facie* cause of  $A_t$  and there is no  $t''$  and no partition  $\pi_{t''}$  such that for every  $C_{t''}$  in  $\pi_{t''}$

1.  $t' < t'' < t$
2.  $P(B_{t'} \wedge C_{t''}) > 0$
3.  $P(A_t|B_{t'} \wedge C_{t''}) = P(A_t|C_{t''})$



We will then define an indirect cause to be a prima facie cause that is not direct. One immediately notices the similarity between **Definition A5** and **Definition A8**. The main difference is that  $t''$  falls between  $t$  and  $t'$  in **Definition A8**. Although Suppes does not do so, this similarity suggests that a definition of a direct cause could also be developed using the analysis of a *spurious*<sub>1</sub> cause.

**Definition A9:** Otte's first definition of direct cause

An event  $B_{t'}$  is a direct cause in sense one of  $A_t$  if and only if  $B_{t'}$  is a prima facie cause of  $A_t$  and for every  $t' < t'' < t$ , there is no  $C_{t''}$  such that

1.  $P(B_{t'} \wedge C_{t''}) > 0$
2.  $P(A_t | B_{t'} \wedge C_{t''}) = P(A_t | C_{t''})$
3.  $P(A_t | B_{t'} \wedge C_{t''}) \geq P(A_t | B_{t'})$

The conditions of **Definition A9** are similar to those of **Definition A4** with the difference that  $t' < t'' < t$ . We will call the definition of direct cause given by **Definition A8** direct cause in sense two. We abbreviate "direct cause in sense one" with *direct*<sub>1</sub> and direct cause in sense two by *direct*<sub>2</sub>. **Definitions A8** and **A9** say that a cause is a direct cause if and only if there is no later event (or kind of event) that will account for  $A_t$  as well as  $B_{t'}$  does. Whereas an event is *direct*<sub>2</sub> if a certain kind of event does not exist, an event is *direct*<sub>1</sub> if a certain event does not exist. This mirrors the difference between *spurious*<sub>1</sub> and *spurious*<sub>2</sub> causes. Recall, that Suppes proves that if a cause is a *spurious*<sub>2</sub> cause, then it is also a *spurious*<sub>1</sub> cause. A similar proof can be constructed to show that if a prima facie cause is *direct*<sub>1</sub> cause, then it is also a *direct*<sub>2</sub> cause, and if it is an *indirect*<sub>2</sub> cause then it is *indirect*<sub>1</sub> cause.

Additionally, Suppes defines supplementary causes:

**Definition A10:** Suppes' definition of supplementary cause

Events  $B_{t'}$  and  $C_{t''}$  are supplementary causes of  $A_t$  iff:

1.  $B_{t'}$  is prima facie cause of  $A_t$
2.  $C_{t''}$  is prima facie cause of  $A_t$
3.  $P(B_{t'} \wedge C_{t''}) > 0$
4.  $P(A_t | B_{t'} \wedge C_{t''}) \geq \max(P(A_t | B_{t'}), P(A_t | C_{t''}))$

Two causes are supplementary causes if the probability of an event occurring given both is higher than it would have been either one alone. Thus, consuming drugs and consuming alcohol are supplementary causes of death, because the probability of dying given one has consumed drugs and alcohol is greater than either the probability of dying given one has consumed drugs or the probability of dying given one has consumed alcohol.

**Theorem A1** no spurious cause of A can be a supplementary cause of A

Proof: If according to condition 2. of **Definition A4** or **Definition A5**  $P(A_t | B_{t'} \wedge C_{t''}) = P(A_t | C_{t''})$ , then it is not the case that condition 4. of **Definition A10** can be satisfied, so  $B$  and  $C$  will not be supplementary causes.

Sufficient causes are viewed as those limiting cases in which the conditional probability of an event reaches one:

**Definition A11:** Suppes' definition of *sufficient (or determining) cause*

An event  $B_{t'}$  is a sufficient (or determining) cause of  $A_t$  iff  $B_{t'}$  is a prima facie cause of  $A_t$  and  $P(A_t | B_{t'}) = 1$

Suppes' framework implies that the sufficient cause relation is transitive, that is if  $C_{t''}$  is a sufficient cause of  $B_{t'}$ , and if  $B_{t'}$  is a sufficient cause of  $A_t$ , then  $C_{t''}$  is a sufficient cause of  $A_t$ . This is captured by the following theorem

**Theorem A2:** transitivity of sufficient cause relation based on Suppes' causality framework

If  $P(A_t|B_{t'}) > P(A_t)$ ,  $P(B_{t'}|C_{t''}) > P(B_{t'})$ ,  $P(A_t|B_{t'}) = 1$ ,  $P(B_{t'}|C_{t''}) = 1$  then  $P(A_t|C_{t''}) = 1$

Proof:

If  $P(A_t|B_{t'}) = 1$  and  $P(B_{t'} \wedge C_{t''}) > 0$  then  $P(A_t|B_{t'} \wedge C_{t''}) = 1$  (Suppes' Theorem 1, 1970, p. 35)

Proof of Suppes' Theorem 1:

Obviously,  $P(A_t|B_{t'}) = 1 \therefore P(A_t \wedge B_{t'}) = P(B_{t'})$ . We have  $P(B_{t'}) = P(A_t \wedge B_{t'}) + P(\neg A_t \wedge B_{t'})$ .

Thus, we get:  $P(\neg A_t \wedge B_{t'}) = 0$

Assuming that  $P(B_{t'} \wedge C_{t''}) > 0$  then we can write:

$$P(A_t|B_{t'} \wedge C_{t''}) = \frac{P(A_t \wedge B_{t'} \wedge C_{t''})}{P(B_{t'} \wedge C_{t''})} = \frac{P(B_{t'} \wedge C_{t''}) - P(\neg A_t \wedge B_{t'} \wedge C_{t''})}{P(B_{t'} \wedge C_{t''})}$$

Finally,  $\neg A_t \wedge B_{t'} \wedge C_{t''}$  is a subset of event with probability zero, namely  $\neg A_t \wedge B_{t'}$  so we conclude that  $P(\neg A_t \wedge B_{t'} \wedge C_{t''}) = 0$ . Thus, we get  $P(A_t|B_{t'} \wedge C_{t''}) = 1$ .

Using

$$P(A_t|B_{t'} \wedge C_{t''}) = P(A_t|B_{t'} \wedge A_t|C_{t''}) = P(A_t|B_{t'}) + P(A_t|C_{t''}) - P(A_t|B_{t'} \vee A_t|C_{t''}) = P(A_t|C_{t''})$$

We find that  $P(A_t|C_{t''}) = 1$ .

Alternatively,

$$P(A_t|C_{t''}) = P(B_{t'}|C_{t''})P(A_t|B_{t'} \wedge C_{t''}) + P(\neg B_{t'}|C_{t''})P(A_t|\neg B_{t'} \wedge C_{t''}) \text{ (by total probability)}$$

Then  $P(\neg B_{t'}|C_{t''}) = 0$  from  $P(B_{t'}|C_{t''}) = 1$  from where it follows that  $P(A_t|C_{t''}) = 1$

Another important idea in causation is that of a necessary cause or condition.

**Definition A12:** Otte's definition of *necessary causes*

An event  $B_{t'}$  is a necessary cause (or condition) of  $A_t$  iff the probability of  $A_t$  given the absence of  $B_{t'}$  is equal to zero. That is,  $B_{t'}$  is a *prima facie* cause of  $A_t$  and  $P(A_t|\neg B_{t'}) = 0$ .

**Theorem A3:** transitivity of *necessary causes*

If  $P(A_t|B_{t'}) > P(A_t)$ ,  $P(B_{t'}|C_{t''}) > P(B_{t'})$ ,

### *Equivalence between the causality concepts based on PTL and Suppes' causal framework*

**Theorem A4:** Assume there is a *Kripke* structure  $M = \{\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{Q}\}$  representing the underlying system governing the occurrences of the events. Then the conditions for causality given in the **Definition A2** for *prima facie* cause earlier are satisfied if and only if the conditions for causality given by **Definition A3** (**Definition A3**) are satisfied.

Proof:

We begin by showing that **Definition A2**  $\rightarrow$  **Definition A3** and then show that **Definition A3**  $\rightarrow$  **Definition A2**.

**Proposition A1.1:** **Definition A2**  $\rightarrow$  **Definition A3**

Proof:

Assume that  $c = C$ ,  $e = E$  and there is a *Kripke* structure  $M = \{\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{Q}\}$ , representing the underlying system governing the occurrences of these events. Also assume that states in  $M$  that satisfy  $c$  and  $e$  are labeled as such. If  $t' < t$  in **Definition A3**, we assume that in  $M$  that satisfy  $c$  and  $e$  are labeled as such. If  $t' < t$  in **Definition A3**, we assume that in  $M$  there will be at least one transition between an event at  $t'$  and one at  $t$ . That is, the timescale of  $M$  is as fine as that of Suppes and vice versa. Further, we assume that the probabilities of Suppes's formulation and those in  $M$  come from the same source and this if represented correctly,  $P(E)$  in **Definition A3** is equal to  $P(e)$  in **Definition A2**.

Condition 1:  $P(E_t|C_{t'}) > P(E_t)$

By definition of  $F_{<p}^{\leq \infty} e$ , the probability  $P(E_t)$  of  $E$  occurring at any time  $t$  is less than  $p$ . Recall that the probability of a path is the product of the transition probabilities along the path, and the probability of a set of paths is the sum of their individual path probabilities. For a structure to satisfy this formula, the set of paths from the start state that reach a state where  $e$  holds must be less than  $p$ , and the probability of reaching a state where  $e$  holds in this system is less than  $p$ . Thus,

$$P(E_t) < p \quad (\text{A.1})$$

Now we must show  $P(E_t | C_{t'}) > p$ . We now show that this conditional probability is greater than or equal to  $p$  if:

$$c \leq_{\geq p}^{\geq 1, \leq \infty} e \quad (\text{A.2})$$

is satisfied.

The probability  $p_1$  of a transition from state  $s_1$  to state  $s_2$  that labels the edge between them,

$$s_1 \xrightarrow{p_1} s_2 ,$$

Is the conditional probability:

$$P(s_{2,t+1} | s_{1,t}) \quad (\text{A.3})$$

The probability of reaching  $s_2$  one time unit after  $s_1$ . Then, for a path:

$$s_1 \xrightarrow{p_1} s_2 \xrightarrow{p_2} s_3 ,$$

we can calculate the probability, given  $s_1$ , of reaching  $s_3$  (via  $s_2$ ) within two time units:

$$P(s_{3,t+2}, s_{2,t+1} | s_{1,t}) = P(s_{3,t+2} | s_{2,t+1}, s_{1,t}) \times P(s_{2,t+1} | s_{1,t}) \quad (\text{A.4})$$

and since  $s_3$  and  $s_1$  are independent conditioned on  $s_2$  this becomes:

$$P(s_{3,t+2}, s_{2,t+1} | s_{1,t}) = P(s_{3,t+2} | s_{2,t+1}) \times P(s_{2,t+1} | s_{1,t}) \quad (\text{A.5})$$

Note that the probabilities of the righthand side are simply the transition probabilities from  $s_1$  to  $s_2$ , and  $s_2$  to  $s_3$  (since there is one time unit between the states, they can only be reached via single transition).

Thus, the conditional probability is precisely the path probability:

$$P(s_{3,t+2}, s_{2,t+1} | s_{1,t}) = p_2 \times p_1 \quad (\text{A.6})$$

Then, if we have a set of paths from  $s_1$  to  $s_3$ , the conditional probability  $P(s_3 | s_1)$  is the sum of these path probabilities. For example, we may have the following paths:

$$s_1 \xrightarrow{p_1} s_2 \xrightarrow{p_2} s_3$$

$$s_1 \xrightarrow{p_3} s_4 \xrightarrow{p_4} s_3$$

In which case:

$$P(s_{3,t+2} | s_{1,t}) = P(s_{3,t+2}, s_{2,t+1} | s_{1,t}) + P(s_{3,t+2}, s_{4,t+1} | s_{1,t}) \quad (\text{A.7})$$

and from eq. (A.6) this becomes:

$$P(s_{3,t+2}|s_{1,t}) = p_2 \times p_1 + p_4 \times p_3 \quad (\text{A.8})$$

the sum of the individual path probabilities. Let us now assume that  $s_1$  is labeled with  $c$  and  $s_3$  is labeled with  $e$ , these are the only  $c$  and  $e$  states in the system, and there are no other paths between the states taking less than or equal to 2 time units. Then, this probability we have computed is in fact the probability of:

$$c \prec^{\geq 1, \leq 2} e \quad (\text{A.9})$$

since the probability of reaching  $s_3$ , during a window of time simply means looking at the set of paths reaching  $s_3$  during that window. Similarly, to find the probability of:

$$c \prec^{\geq 1, \leq \infty} e \quad (\text{A.10})$$

we must consider the set of paths from states labeled with  $c$  to those labeled with  $e$  that take at least 1 time unit. Since there can be cycles in our graph, calculating the probability associated with a leads-to formula with an infinite upper time bound requires a slightly different method.

//Finish the paragraph the leads-to formula with lower and upper bound

#### *Leads-to with Both Lower and Upper Time Bounds*

This paragraph deals with evaluation of *Leads-To* with applied window of time in which  $c$  leads to  $e$ . We assume a minimum time after  $c$  is true before which  $e$  is true. Here it is shown that it is possible to add such a lower bound. By Definition:

$$f \prec_{\geq p}^{\geq t_1, \leq t_2} g \equiv AG[f \rightarrow F_{\geq p}^{\geq t_1, \leq t_2} g] \quad (\text{A.11})$$

where  $t_1 \leq t_2$ . Thus, we are only adding a minimum time to the consequent of the conditional. If we can label states where  $F_{\geq p}^{\geq t_1, \leq t_2} g$  is true, then we can proceed as in the algorithm of (Hansson & Jonsson, 1994).

//Finish the paragraph on Leads-to with Both Lower and Upper Bounds taken from the B.2 of (Kleinberg, Causality, Probability, and Time, 2012)

Definitions and Review on Probabilistic Real Time Computation Tree Logic (PCTL) in (Hansson & Jonsson, 1994)

#### *Notation*

Assume a finite set  $A$  of *atomic propositions*. We use  $a, a_1$ , etc. to denote atomic propositions. Formulas in PCTL are built from atomic propositions, propositional logic connectives and operators for expressing time and probabilities. The set of PCTL formulas is divided into *path formulas* and *state formulas*. Their syntax is defined inductively as follows:

- Each atomic proposition is a state formula
- If  $f_1$  and  $f_2$  are state formulas, then so are  $\neg f_1$ ,  $(f_1 \wedge f_2)$ ,  $(f_1 \vee f_2)$ ,  $(f_1 \rightarrow f_2)$
- If  $f_1$  and  $f_2$  are state formulas, and  $t$  is a nonnegative integer or  $\infty$ , then  $(f_1 U^{\leq t} f_2)$  and  $(f_1 W^{\leq t} f_2)$  are path formulas,
- If  $f$  is a path formula and  $p$  is a real number with  $0 \leq p \leq 1$ , then  $[f]_{\geq p}$  and  $[f]_{> p}$  are state formulas.

We shall use  $f, f_1$ , etc. to range over PCTL formulas. Intuitively, state formulas represent properties of states and path formulas represent properties of paths (i.e., sequences of states). The propositional connectives  $\neg, \vee, \wedge$ , and  $\rightarrow$  have their usual meanings. The operator  $U$  is the (*strong*) *until* operator, and  $W$  is the *unless* (or *weak until*) operator. For a given state  $s$ , the formulas  $[f]_{\geq p}$  and  $[f]_{> p}$  express that  $f$  holds for a path from  $s$  with a probability of at least  $p$  and greater than  $p$ , respectively.

We shall use  $f_1 U_{\geq p}^{\leq t} f_2$  as a shorthand for  $[f_1 U^{\leq t} f_2]_{\geq p}$ , and use  $f_1 W_{\geq p}^{\leq t} f_2$  as a shorthand for  $[f_1 W^{\leq t} f_2]_{\geq p}$ . Intuitively,  $f_1 U_{\geq p}^{\leq t} f_2$  means that there is at least a probability  $p$  that both  $f_2$  will become true within  $t$  time units and that  $f_1$  will be true from now on until  $f_2$  becomes true.  $f_1 W_{\geq p}^{\leq t} f_2$  means that there is at least a probability  $p$  that either  $f_1$  will remain true for at least  $t$  time units, or that both  $f_2$  will become true within  $t$  time units and that  $f_1$  will be true from now on until  $f_2$  becomes true.

PCTL formulas are interpreted over structures that are discrete time Markov chains. A specified initial state is associated with the structure. In addition, for each state there is an assignment of truth values to atomic propositions appearing in a given formula. Formally, a structure is a quadruple  $\langle S, s^i, \mathcal{T}, L \rangle$ , where

$S$  is a finite set of states, ranged over by  $s, s_1$ , etc.,

$s^i \in S$  is an *initial state*,

$\mathcal{T}$  is a *transition probability function*,  $\mathcal{T}: S \times S \rightarrow [0,1]$ , such that for all  $s$  in  $S$  we have

$$\sum_{s' \in S} \mathcal{T}(s, s') = 1,$$

$L$  is a labeling function assigning atomic propositions to states, i.e.,

$$L: S \rightarrow 2^A$$

Intuitively, each transition is considered to require one *time unit*. We will display structures as transition diagrams, where states (circles) are labeled with atomic propositions and transitions with non-zero probability are represented as arrows labeled with their probabilities (e.g., the arrow going from state  $s_k$  to state  $s_l$  is labelled with  $\mathcal{T}(s_k, s_l)$ ). The initial state  $s^i$  is indicated with an extra arrow.

A path  $\sigma$  from a state  $s_0$  in a structure is an infinite sequence

$$s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n \rightarrow \cdots$$

of states with  $s_0$  as the first state. The  $n$ :th state ( $s_n$ ) of  $\sigma$  is denoted  $\sigma[n]$ , and the prefix of  $\sigma$  of length  $n$  is denoted  $\sigma \upharpoonright n$ , i.e.,

$$\sigma \upharpoonright n = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$$

For each structure and state  $s_0$  we define a probability measure  $\mu_m$  on the set of paths from  $s_0$ .  $\mu_m$  is defined on the probability space  $\langle X, \mathcal{A} \rangle$ , where  $X$  is the set of paths starting in  $s_0$  and  $\mathcal{A}$  is a sigma-algebra on  $X$  generated by sets

$$\{\sigma \in X : \sigma \upharpoonright n = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n\}$$

Of paths with a common finite prefix  $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$ . The measure  $\mu_m$  is defined as follows: for each finite sequence  $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n$  of states,

$$\mu_m(\{\sigma \in X : \sigma \upharpoonright n = s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_n\}) = \mathcal{T}(s_0, s_1) \times \cdots \times \mathcal{T}(s_{n-1}, s_n)$$

i.e., the measure of the set of paths  $\sigma$  for which  $\sigma \upharpoonright n = s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$  is equal to the product  $\mathcal{T}(s_0, s_1) \times \dots \times \mathcal{T}(s_{n-1}, s_n)$ . For  $n = 0$  we define  $\mu_m(\{\sigma \in X : \sigma \upharpoonright 0 = s_0\}) = 1$ . This uniquely defines the measure  $\mu_m$  on all sets of paths in the sigma-algebra  $\mathcal{A}$ .

We define the truth of PCTL formulas for a structure  $K$  by a satisfaction relation:

$$s \models_K f$$

which means that the state formula  $f$  is true at state  $s$  in the structure  $K$ . In order to define the satisfaction relation for states, it is helpful to use another relation

$$s \vdash_K f$$

which means that the path  $\sigma$  satisfies the path formula  $f$  in  $K$ . The relations  $s \models_K f$  and  $s \vdash_K f$  are inductively defined as follows:

$$\begin{aligned} s \models_K a & \text{ iff } a \in L(s) \\ s \models_K \neg f & \text{ iff not } s \models_K f \\ s \models_K f_1 \wedge f_2 & \text{ iff } s \models_K f_1 \text{ and } s \models_K f_2 \\ s \models_K f_1 \vee f_2 & \text{ iff } s \models_K f_1 \text{ or } s \models_K f_2 \\ s \models_K f_1 \rightarrow f_2 & \text{ iff } s \models_K \neg f_1 \text{ or } s \models_K f_2 \\ \sigma \vdash_K f_1 U^{\leq t} f_2 & \text{ iff there exist an } i \leq t \text{ such that } \sigma[i] \models_K f_2 \text{ and } \forall j : 0 \leq j < i : (\sigma[j] \models_K f_1) \\ \sigma \vdash_K f_1 W^{\leq t} f_2 & \text{ iff } \sigma \vdash_K f_1 U^{\leq t} f_2 \text{ or } \forall j : 0 \leq j < i : (\sigma[j] \models_K f_1) \\ s \models_K [f]_{\geq p} & \text{ iff the } \mu_m\text{-measure of the set of paths } \sigma \text{ starting in } s \text{ for which } \sigma \vdash_K f \text{ is at least } p. \\ s \models_K [f]_{> p} & \text{ iff the } \mu_m\text{-measure of the set of paths } \sigma \text{ starting in } s \text{ for which } \sigma \vdash_K f \text{ is greater than } p. \end{aligned}$$

We define

$$\models_K f \equiv s^i \models_K f$$

where  $s^i$  is the initial state of  $K$ .

### Properties expressible in PCTL

We will present examples of properties that can be expressed in PCTL. First, we discuss some of the facilities of PCTL which makes it suitable for specification of soft and hard deadlines.

The main difference between PCTL and branching time temporal logics such as CTL, is the quantification over paths and the ability to specify quantitative time. CTL allows universal ( $Af$ ) and existential ( $Ef$ ) quantification over paths, i.e., one can state that a property should hold for all computations (paths) or that it should hold for some computations (paths). It is not possible to state that a property should hold for a certain portion of the computations, e.g., for at least 50% of the computations. In PCTL, on the other hand, arbitrary probabilities can be assigned to path formulas, thus obtaining a more general quantification over paths. An analogy to universal and existential quantification can in PCTL be defined as:

$$\begin{aligned} Af & \equiv [f]_{\geq 1} \\ Ef & \equiv [f]_{> 0} \end{aligned}$$

Quantitative time allows us to specify time-critical properties that relate the occurrence of events of a system in real-time. In PCTL it is possible to state that a property will hold continuously during a specific time interval, or that a property will hold sometime during a time interval. Combining this with the above quantification we can define

$$G_{\geq p}^{\leq t} f \equiv f W_{\geq p}^{\leq t} \text{false}$$

$$F_{\geq p}^{\leq t} f \equiv \text{true} U_{\geq p}^{\leq t} f$$

$G_{\geq p}^{\leq t} f$  means that the formula  $f$  holds continuously for  $t$  time units with a probability of at least  $p$ , and  $F_{\geq p}^{\leq t} f$  means that the formula  $f$  holds within  $t$  time units with a probability of at least  $p$ .

An important requirement on most real-time and distributed systems is that they should be continuously operating, e.g., every time the controller receives an alarm signal from a sensor the controller should take the appropriate action. We can express such requirements with the following PCTL operators:

$$\begin{aligned} AGf &\equiv f W_{\geq 1}^{\leq \infty} \text{false} \\ AFf &\equiv \text{true} U_{\geq 1}^{\leq \infty} f \\ EGf &\equiv f W_{> 0}^{\leq \infty} \text{false} \\ EFf &\equiv \text{true} U_{> 0}^{\leq \infty} f \end{aligned}$$

$AGf$  means that  $f$  is always *true* (in all states that can be reached with non-zero probability),  $AFf$  means that a state where  $f$  is *true* will eventually be reached with probability 1,  $EGf$  means that there is a non-zero probability for  $f$  to be continuously true, and  $EFf$  means that there exists a state where  $f$  holds which can be reached with non-zero probability.

**Definition** (Owicki & Lamport, 1982)<sup>1</sup>: (unquantified) *leads-to* operator ( $a < b$ )  
Whenever  $a$  becomes true,  $b$  will eventually hold.

**Definition**: PCTL quantified *leads-to* operator ( $f_1 <_{\geq p}^{\leq t} f_2$ ):

$$\begin{aligned} f_1 <_{\geq p}^{\leq t} f_2 &\equiv AG[(f_1 \rightarrow F_{\geq p}^{\leq t} f_2)] \\ f_1 <_{\geq p}^{\leq t} f_2 &\text{ means that whenever } f_1 \text{ holds there is a probability of at least } p \text{ that } f_2 \text{ will hold within } t \text{ time units.} \end{aligned}$$

Many modal operators can be derived from the basic PCTL operators. We can for instance define an operator that corresponds to the CTL operator  $A[f_1 \cup f_2]$  (E.M. Clarke, 1983) as follows:

$$A[f_1 \cup f_2] \equiv f_1 U_{\geq 1}^{\leq \infty} f_2$$

As an example, we will specify a mutual exclusion property. Consider two processes ( $P_1$  and  $P_2$ ) using the same critical section. The atomic propositions  $N_i$ ,  $T_i$ , and  $C_i$  indicates that  $P_i$  is in its non-critical, trying, and critical regions, respectively. The mutual exclusion property can be expressed as:

$$AG[\neg(C_1 \wedge C_2)]$$

This is not sufficient for most *real-time systems* since the property only states that simultaneous access to the critical section must be avoided always under all circumstances. To capture a specific real-time behavior, we can specify that whenever  $P_1$  enters its trying region, it will enter its critical region within 4 time units. This can in PCTL be expressed as:

$$T_1 <_{\geq 1}^{\leq 4} C_1$$

For some systems, it might be sufficient that the deadline is almost always met (e.g. in 99% of the cases). The relaxed property can be expressed as:

$$T_1 <_{\geq 0.99}^{\leq 4} C_1$$

---

<sup>1</sup> In (Hansson & Jonsson, 1994) and (Owicki & Lamport, 1982) the symbol  $\leadsto$  is used to denote the *leads-to* operator. In this document we use  $\leadsto$  to denote *prima facie* cause.

Relaxing the timing requirement might enable a less costly implementation that still shows acceptable behavior. To be on the safe side we could add a strict upper limit to the relaxed property, combining the hard and soft deadlines above. If we assume that we want  $P_1$  to always enter its critical region within 10 time units, and almost always within 4 time units we get the property:

$$(T_1 <_{\geq 1}^{\leq 10} C_1) \wedge (T_1 <_{\geq 0.99}^{\leq 4} C_1)$$

### Model Checking in PCTL

In this section we present a model checking algorithm, which given a structure  $K = \langle S, s^i, \mathcal{T}, L \rangle$  and a PCTL formula  $f$  determines whether  $\models_K f$ . The algorithm is based on the algorithm for model checking CTL (E.M. Clarke, 1983). It is designed so that when it finishes each state will be labeled with the set of subformulas of  $f$  that are *true* in the state. One can then conclude that  $\models_K f$  if the initial state ( $s^i$ ) is labeled with  $f$ .

For each state of the structure, the algorithm uses a variable  $label(s)$  to indicate the subformulas that are *true* in state  $s$ . Initially, each state  $s$  is labeled with the atomic propositions that are *true* in  $s$ , i.e.,  $label(s) := L(s), \forall s \in S$ . The labeling is then performed starting with the smallest subformula of  $f$  that has not yet been labeled and ending with labeling states with  $f$  itself. Composite formulas are labeled based on the labeling of their parts. Assuming that we have performed the labeling of  $f_1$  and  $f_2$ , the labeling corresponding to negation ( $\neg f_1$ ) and propositional connectives ( $f_1 \wedge f_2, f_1 \vee f_2, f_1 \rightarrow f_2$ ) is straightforward, i.e.

$$\begin{aligned} label(s) &:= label(s) \cup \{\neg f_1\} \text{ if } f_1 \notin label(s), \\ label(s) &:= label(s) \cup \{f_1 \wedge f_2\} \text{ if } f_1, f_2 \in label(s), \\ label(s) &:= label(s) \cup \{f_1 \vee f_2\} \text{ if } f_1 \in label(s) \text{ or } f_2 \in label(s) \\ label(s) &:= label(s) \cup \{f_1 \rightarrow f_2\} \text{ if } f_1 \notin label(s) \text{ or } f_2 \in label(s) \end{aligned}$$

where in addition the new formula must be a subformula of  $f$ . The next section presents two algorithms for labeling states with the modal subformulas of PCTL. After that, in the subsequent section, we discuss labeling in cases with extreme parameter values (e.g.,  $p = 1, p = 0$ , and  $t = \infty$ ).

### Labeling states with the modal subformulas of PCTL

We shall give an algorithm for the labeling of states for the formula  $f_1 U_{\geq p}^{\leq t} f_2$  assuming that we have done the labeling for formulas  $f_1$  and  $f_2$ , and that  $t \neq \infty$ .

//Appendix: Finish the paragraph on PTL Theory

### Review on Computation Tree Logic and Specification Language in (E.M. Clarke, 1983)

In this Appendix section we will review an efficient procedure for verifying that a finite state concurrent system meets a specification expressed in a (propositional) branching-time temporal logic. The reviewed algorithm has linear complexity in both the size of the specification and the size of the global transition. The global state graph can be viewed as a finite Kripke structure and an efficient algorithm can be given to determine whether a given structure is a model of a particular formula. The algorithm, which we call a *model checker*, is similar to the global flow analysis algorithms used in compiler optimization and has complexity linear in both the size of the structure and the size of the specification.

### The Specification Language

The syntax for CTL is given below.  $AP$  is the underlying set of *atomic propositions*.

1. Every atomic proposition  $p \in AP$  is a CTL formula
2. If  $f_1$  and  $f_2$  are CTL formulae, the so are  $\neg f_1, f_1 \wedge f_2, AX f_1, EX f_1, A[f_1 U f_2], E[f_1 U f_2]$ .



The symbols  $\wedge$  and  $\neg$  have their usual meanings.  $X$  is the *nexttime* operator; the formulae  $AXf_1$  ( $EXf_1$ ) intuitively means that  $f_1$  holds in every (in some) immediate successor of the current program state.  $U$  is the *until* operator; The formula  $A[f_1Uf_2]$  ( $E[f_1Uf_2]$ ) intuitively means that for every computation path (for some computation path), there exists an initial prefix of the path such that  $f_2$  holds at the last state of the prefix and  $f_1$  holds at all other states along the prefix.

The semantics of CTL formulae with respect to a labeled state-transition graph is defined below. Formally, a CTL structure is a triple  $M = (S, R, P)$  where

1.  $S$  is a finite set of states
2.  $R$  is a binary relation on  $S$  i.e.,  $R \subseteq S \times S$ . It gives the possible transitions between states and must be total i.e.,  $\forall x \in S \exists y \in S \text{ s.t. } (x, y) \in R$ .
3.  $P$  is an assignment of atomic propositions to states i.e.,  $P : S \rightarrow 2^{AP}$

A *path* is an infinite sequence of states  $(s_0, s_1, s_2, \dots)$  such that  $\forall i (s_i, s_{i+1}) \in R$ . For any structure  $M = (S, R, P)$  and state  $s_0 \in S$ , there is an *infinite computation tree* with root labeled  $s_0$  such that  $s \rightarrow t$  is an arc in the tree iff  $(s, t) \in R$ .

We use the standard notation to indicate truth in a structure:  $M, s_0 \models f$  means that formula  $f$  holds at state  $s_0$  in structure  $M$ . When the structure  $M$  is understood, we simply write  $s_0 \models f$ . The relation  $\models$  is defined inductively as follows:

- $s_0 \models f$  iff  $p \in P(s_0)$
- $s_0 \models \neg f$  iff not( $s_0 \models f$ )
- $s_0 \models f_1 \wedge f_2$  iff  $s_0 \models f_1$  and  $s_0 \models f_2$
- $s_0 \models AXf_1$  iff for all states  $t$  such that  $(s_0, t) \in R$ ,  $t \models f_1$
- $s_0 \models EXf_1$  iff for all states  $t$  such that  $(s_0, t) \in R$ ,  $t \models f_1$
- $s_0 \models A[f_1Uf_2]$  iff for all paths  $(s_0, s_1, s_2, \dots) \exists i \text{ s.t. } i \geq 0 \wedge s_i \models f_2 \wedge \forall j, 0 \leq j < i : s_j \models f_1$
- $s_0 \models E[f_1Uf_2]$  iff for some path  $(s_0, s_1, s_2, \dots) \exists i \text{ s.t. } i \geq 0 \wedge s_i \models f_2 \wedge \forall j, 0 \leq j < i : s_j \models f_1$

## Model Checker

Assume that we wish to determine whether formula  $f_0$  is true in the finite structure  $M = (S, R, P)$ . When the algorithm finishes, each state will be labelled with the set of subformulae true in the state. We let  $label(s)$  denote this set for state  $s$ . Consequently,  $M, s \models f$  iff  $f \in label(s)$  at termination. We first consider the case in which each state is currently labelled with the *immediate* subformulae of  $f$  which are true in that state. We will use the following primitives for manipulating formulas and accessing the labels associated with states:

- $arg1(f)$  and  $arg2(f)$  give the first and second arguments of a two-argument formula  $f$  such as  $A[f_1Uf_2]$
- $labelled(s, f)$  will return true (false) if state  $s$  is (is not) labelled with formula  $f$ .
- $add\_label(s, f)$  adds formula  $f$  to the current label of state  $s$ .

The state labeling algorithm (procedure  $label\_graph(f)$ ) must be able to handle seven cases depending on whether  $f$  is atomic or has one of the following forms:  $\neg f_1, f_1 \wedge f_2, AXf_1, EXf_1, A[f_1Uf_2], E[f_1Uf_2]$ . We will only consider the case in which  $f = A[f_1Uf_2]$  here since all other cases are either straightforward or similar. For the case  $f = A[f_1Uf_2]$  the algorithm uses depth first search to explore the state graph. The bit array  $marked[1:nstates]$  is used to indicate which states have been visited by the search algorithm. The algorithm also uses stack  $ST$  to keep track of those states which require additional processing before the truth or falsity of  $f$  can be determined. The Boolean procedure  $stacked(s)$  will determine (in constant time) whether state  $s$  is currently on the stack  $ST$ .

```

def label_graph(f, b):
    """
    f: formula
    b (bool): result
    """
    ST = empty_stack
    for s in S:
        marked(s) = false
    for s in S:
        if not marked(s):
            au(f, s, b)

def au(f, s, b):
    """
    f: formula
    s: state
    b (bool): result
    """
    """
    If s is marked and stacked, return false (see lemma 3.1).
    If s is already labelled with f, then return true. Otherwise,
    If s is marked but nether stacked nor labelled, then return false

    if marked(s):
        if stacked(s):
            b = False
            return
        if labelled

```

//Appendix: Finish the paragraph on CTL algorithms

## Bibliography

- Clarke, E. M., & Schlingloff, B. H. (2001). Model Checking. In A. Robinson, & A. Voronkov, *Handbook of Automated Reasoning* (pp. 1369-1520). Elsevier Science Publishers B.V.
- E.M. Clarke, E. E. (1983). Automatic Verification Of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. *ACM*, 117-126.
- Eells, E. (1991). *Probabilistic Causality*. Cambridge UK: Cambridge University Press.
- G.E. Hughes, M. C. (1996). *A New Introduction To Modal Logic*. London: Routledge.
- Hansson, H., & Jonsson, B. (1994). *A Logic for Reasoning about Time and Reliability*. Uppsala, Sweden: SICS Research Report SICS/R90013.
- Houdth, G. V., Depaire, B., & Martin, N. (2022). Root Cause Analysis in Process Mining with Probabilistic Temporal Logic. *ICPM 2021 Workshops* (pp. pp. 73–84). Eindhoven, The Netherlands: LNBIP Volume 433.
- Kleinberg, S. (2012). *Causality, Probability, and Time*. Cambridge, UK: Cambridge University Press.
- Kleinberg, S., & Mishra, B. (2009). The Temporal Logic Of Causal Structures. *The Conference on Uncertainty in Artificial Intelligence*, (pp. 303-312). Montreal, Canada.
- Otte, R. E. (1982). *Probability and Causality, PhD Thesis*. Ann Arbor , MI, 48106: University of Arizona Graduate College, University Microfilm International.
- Spirtes, P., Glymour, C., & Sheines, R. (1993). *Causation, Prediction and Search*. New York: Springer Verlag.

## Downloadable Links for the Bibliography

(Clarke & Schlingloff, 2001): [here](#)

(Eells, 1991): [here](#)

(Hansson & Jonsson, 1994): [here](#)

(Houdth, Depaire, & Martin, 2022): [here](#)

(Kleinberg & Mishra, The Temporal Logic Of Causal Structures, 2009): [here](#)

(Spirtes, Glymour, & Sheines, 1993): [here](#)

(Otte, 1982): [here](#)

(G.E. Hughes, 1996): [here](#)

(E.M. Clarke, 1983): [here](#)