Hierarchical Simulation of Timed Behaviours of Structured Occurrence Nets

Salma Alharbi[1]

[1]*School of Computing, Newcastle University*
Science Square, Newcastle upon Tyne, NE4 5TG, United Kingdom

Abstract

The typical notation for recording the behaviour of an asynchronous system is some form of a directed acyclic graph. The formalism of structured occurrence nets (SO-nets) can play an important role in the representation of complex evolving system behaviours. SO-nets are sets of related occurrence nets, employing different types of formally defined relations and supporting various types of abstraction, which represent the detail of the concurrency and causality relations between executed events. The aim of this paper is to present theoretical underpinnings, new algorithms, and prototype software implementation for hierarchical and abstraction-based analyses and simulations of timed behaviours of complex evolving systems using SO-nets.

1. Introduction

An occurrence net is an acyclic net that provides a complete and unambiguous record of all the causal dependencies among the events involved, including both backward (more than one arrow incoming to a place) and forward non-determinism (more than one arrow outgoing from a place). It represents a singular 'causal history' and has a single final marking. A structured occurrence net (SO-net) is an extension of an occurrence net [1], and it illustrates causality and concurrency information for a single system execution involving a number of occurrence nets which interact by synchronous and asynchronous communication. SO-nets were created in order to characterise the behaviour of evolving systems of systems. It should be stressed that SO-nets are not meant to be models of concurrent systems, but rather models of their observed behaviours. In particular, an SO-net can be created and analysed without knowing the structure and components of the actual system which generated an observed behaviours.

The importance of structure in helping designers to cope with design complexity is well-accepted, especially in the software engineering domain and in the hardware design domain. The effective use of structuring notations greatly reduces the cognitive complexity of designs, and the resources involved in their representation and manipulation. The purpose of a system design is to define how a system will behave. Notations for recording actual or potential system behaviour have not attracted levels of interest comparable

PNSE'23, International Workshop on Petri Nets and Software Engineering, June, 2023, Lisbon, Portugal
s.alharbi2@newcastle.ac.uk (S. Alharbi)

to those for system design notations. This is probably because detailed records of the behaviour of complex systems are mainly used out of sight within tools for system visualisation, verification, synthesis and failure analysis, rather than in documents and user interfaces [2].

Time simulation is a powerful tool used in many fields to model and analyse the behaviour of complex systems over time, such as crimes and accidents. In criminal investigations, time simulation may be an effective technique since it enables investigators to piece together the sequence of events that led to a crime. Investigators may better grasp the circumstances leading up to a crime by simulating time, which can be useful in identifying suspects and acquiring evidence. Structured occurrence net (or SO-nets) is an extension of the notion of an occurrence net, which is a directed acyclic graph [1] that captures causality and concurrency information about a single execution of a system. SO-nets can play an important role in the representation of complex evolving systems (CE-systems) behaviours. Representing date-time information about such systems is important. Timed SO-nets are based on groups of associated timed occurrence nets and are designed for reasoning about related events and causality with time information that is uncertain or missing in evolving systems. When modelling a system based on time, such as accidents or crimes, it is important to identify the order in which events have happened and to identify the duration of the events. However, the temporal information that is known about an occurrence is often inaccurate or lacking. For instance, although it may not be able to pinpoint a certain date-time (such as [2022/11/04 05:05:05]) at which a robbery happened, it could be possible to provide temporal boundaries such as (earliest start and latest start).

The contribution of this paper is a novel tool-supported formalism (timed SO-nets and timed simulation) for modelling and reasoning about concurrent events with uncertain or missing time information in developing systems. It is built on collections of connected timed occurrence nets. Timed simulation tool for criminal investigations to help investigators reconstruct the timeline of events related to a crime.

This paper is aimed to address the theoretical underpinnings, which will be described in the subsequent sections. Section 2 gives a background about SO-nets, search description and research question. The notation of timed SO-net (based on discrete time intervals), and algorithms for estimating and increasing the precision of date-time intervals using default duration intervals and redundant time information are given in Section 3. A simulation of timed behaviour using duration intervals is given in Section 4. Support for date-timed SO-nets and simulation of the time is provided by the SONCRAFT tool, which is described in Section 5. Conditions for checking the consistency of date-time intervals and algorithms are defined in the Appendix.

2. Background

Occurrence nets (or O-nets) represent single executions of computing systems, providing details of the concurrency and causality relations between executed events.

An O-net is a triple $onet = (C, E, F)$, where C and E are finite disjoint sets of *conditions*

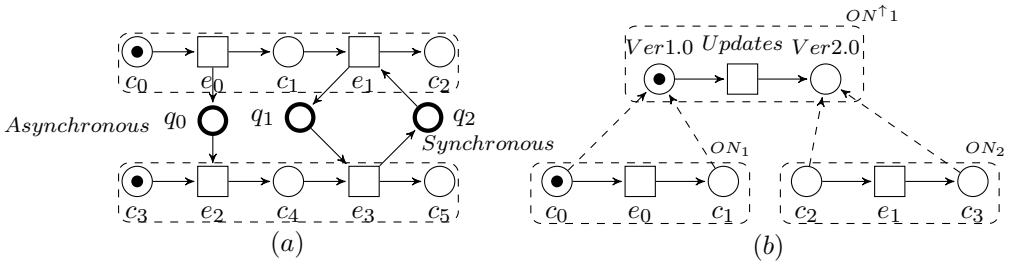

Figure 1: CSO-net (a); and BSON-net portraying a system (off-line) update (b).

and *events* (referred to as the nodes), and $F \subseteq (C \times E) \cup (E \times C)$ is the *flow relation*. The *inputs* and *outputs* of a node x are ${}^\bullet x = \{y \mid (y,x) \in F\}$ and $x^\bullet = \{y \mid (x,y) \in F\}$. For $X \subseteq C \cup E$, ${}^\bullet X$ and $X^\bullet$ are the sets of all inputs and outputs of the nodes in X. Also, we assume:

- For all $c \in C$: $|{}^\bullet c| \leq 1$ and $|c^\bullet| \leq 1$.
- For all $e \in E$: $|{}^\bullet e| \geq 1$ and $|e^\bullet| \geq 1$.
- The *causality relation* $\prec$ over E is acyclic, where $e \prec f$ if $e^\bullet \cap {}^\bullet f \neq \varnothing$.

In an O-net, a *marking* is any set of conditions. The initial and final markings are $M_0^{onet} = \{c \in C \mid {}^\bullet c = \varnothing\}$ and $M_{fin}^{onet} = \{c \in C \mid c^\bullet = \varnothing\}$, respectively. An O-net can be executed by firing sets of events; this execution follows the standard Petri net step semantics. (We also will omit details of execution of other SO-nets described in this section.)

The paper [3] used communication structured occurrence nets (or CSO-nets) to represent communication between different subsystems. A CSO-net is a set of O-nets connected by special nodes called *channel places*. Figure 1 shows a CSO-net with two O-nets communicating *synchronously* (events e_1 and e_3 are always executed simultaneously) and *asynchronously* (events e_0 and e_2 can be executed simultaneously, or e_2 is executed after e_0).

A *phase* of O-net is a non-empty set of conditions, each phase is a fragment of an O-net beginning with a cut and ending with a cut (a maximal set of causally unrelated conditions) which follows it in the causal sense, including all the conditions occurring between these cuts. In [3], a phase decomposition is a sequence of phases from the initial state to the final state, and whenever one phase ends, its maximal cut is the starting point of the successive one (minimal cut).

A CSO-net is a tuple $cson = (onet_1, \ldots, onet_k, Q, W)$, where each $onet_i = (C_i, E_i, F_i)$ is an O-net (we denote $\mathbf{C} = \bigcup_{i=1}^k C_i$, $\mathbf{E} = \bigcup_{i=1}^k E_i$ and $\mathbf{F} = \bigcup_{i=1}^k F_i$), Q is a set of channel places, and $W \subseteq (Q \times \mathbf{E}) \cup (\mathbf{E} \times Q)$. It is assumed that:

- The $onet_i$'s and Q are mutually disjoint.
- The inputs and outputs of $q \in Q$, ${}^\bullet q = \{e \in \mathbf{E} \mid (e,q) \in W\}$ and $q^\bullet = \{e \in \mathbf{E} \mid (q,e) \in W\}$, belong to distinct component $onet_i$; and $|{}^\bullet q| = 1$ and $|q^\bullet| \leq 1$.
- The relation $(\sqsubset \cup \prec)^* \circ \prec \circ (\prec \cup \sqsubset)^*$ over $\mathbf{E}$ is irreflexive, where: $e \prec f$ if there is $c \in \mathbf{C}$ with $c \in e^\bullet \cap {}^\bullet f$; and $e \sqsubset f$ if there is $q \in Q$ with $q \in e^\bullet \cap {}^\bullet f$.

Intuitively, the original causality relation $\prec$ represents the 'earlier than' relationship of the events, and $\sqsubset$ represents the 'not later than' relationship. The inputs and outputs of a node are extended to include channel places.

A marking of *cson* is a set of conditions and channel places. The initial marking is $M_0^{cson} = M_0^{onet_1} \cup \cdots \cup M_0^{onet_k}$ and the final marking is $M_{fin}^{cson} = M_{fin}^{onet_1} \cup \cdots \cup M_{fin}^{onet_k}$. In a CSO-net, a step is a set of events which may come from one of more component O-nets.

In Figure 1(a), event e_1 and e_3 communicate synchronously due to a cycle involving channel places q_1 and q_2. In diagrams, such a situation may also be represented by a single channel place with undirected edges between events, e.g., as for e_0 and e_1 in Figure 12.

Behavioural structured occurrence nets (or BSO-nets) model activities of evolving systems [3]. An execution history is represented on two different levels: the lower level, which is used to indicate behavioural details, and the upper level, which is used to represent the stages (phases) of system evolution. Thus a BSO-net provides details about the evolution of a system. Figure 1 shows an example of BSO-net representing a system update. A version change brought on by an update event is represented at the top level. The lower level reveals the behaviour of the system before and after the update [3].

Let *cson* be CSO-net as above, and $cson^{\uparrow} = (onet_1^{\uparrow}, \ldots, onet_m^{\uparrow}, Q^{\uparrow}, W^{\uparrow})$ be a disjoint CSO-net such that each $onet_i^{\uparrow} = (C_i^{\uparrow}, E_i^{\uparrow}, F_i^{\uparrow})$ is line-like (i.e., it can be represented as a chain $c_1 e_1 \ldots e_{l-1} c_l$). In addition, let $\mathbf{C}^{\uparrow} = \bigcup_{i=1}^{m} C_i^{\uparrow}$, $\mathbf{E}^{\uparrow} = \bigcup_{i=1}^{m} E_i^{\uparrow}$, and $\mathbf{F}^{\uparrow} = \bigcup_{i=1}^{m} F_i^{\uparrow}$. A *BSO-net* is a tuple $bson = (cson, cson^{\uparrow}, \beta)$ such that $\beta \subseteq \mathbf{C} \times \mathbf{C}^{\uparrow}$, and the following hold:

1. For every $onet_i$, there exists exactly one $onet_j^{\uparrow}$ satisfying $\beta(C_i) \cap C_j^{\uparrow} \neq \varnothing$.
2. For every $onet_j^{\uparrow}$ represented by a chain $\xi_{onet_j^{\uparrow}} = c_1 e_1 \ldots e_{l-1} c_l$, the sequence $\pi_1 \pi_2 \ldots \pi_l = \beta^{-1}(c_1) \ldots \beta^{-1}(c_l)$ is a concatenation of phase decompositions of different O-nets in *cson*. For all c_j and e_j occurring in the chain $\xi_{onet_j^{\uparrow}}$, $\pi(c_j) = \pi_j$ and $causal(e_j) = {}^{\bullet}(Max_{\beta^{-1}({}^{\bullet}(e_j))}) \times \{e_j\} \cup \{e_j\} \times ({}^{\bullet}Min_{\beta^{-1}(post(e_j))})$.
3. The relation $(\sqsubset \cup \prec \cup \lhd)^* \circ (\prec \cup \lhd) \circ (\prec \cup \sqsubset \cup \lhd)^*$ over $\mathbf{E} \cup \mathbf{E}^{\uparrow}$ is irreflexive, where: $e \prec f$ if there is $c \in \mathbf{C} \cup \mathbf{C}^{\uparrow}$ with $c \in e^{\bullet} \cap {}^{\bullet}f$; $e \sqsubset f$ if there is $q \in Q \cup Q^{\uparrow}$ with $q \in e^{\bullet} \cap {}^{\bullet}f$; and $e \lhd f$ if $(e, f) \in \bigcup_{e' \in E^{\uparrow}} causal(e')$.

The initial marking M_0^{bson} of BSO-net is $M_0^{cson^{\uparrow}}$ together with $M_0^{onet_i}$ for each i such that $\beta(M_0^{onet_i}) \cap M_0^{cson^{\uparrow}} \neq \varnothing$. The final marking M_{fin}^{bson} is defined similarly.

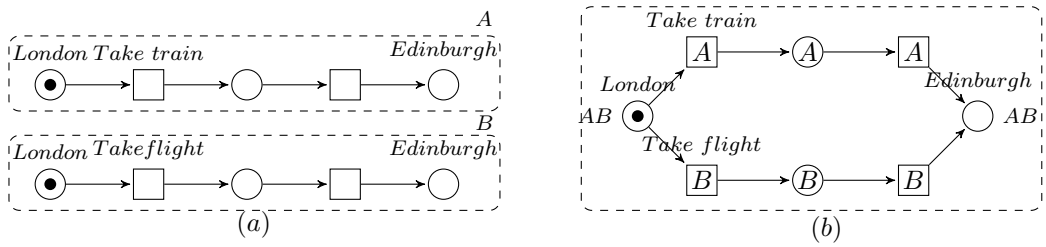

Figure 2: (a) Multiple scenarios modelled by two O-nets, and (b) AO-net.

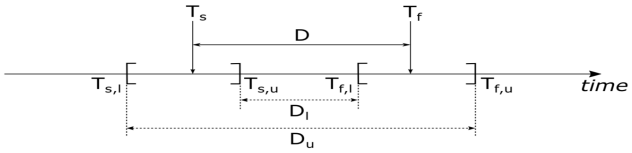

Figure 3: The relationship among unknown and known time values and duration [4].

A BSO-net consists of two CSO-nets connected by a behavioural relation β. In the above, (1) implies that each phase points to exactly one condition of the upper level, and (2) means that each upper level condition maps to a single phase of a lower level O-net . The ordering of the upper level conditions must match that of the phase decompositions of the lower level O-net.

In [3] an extension of SO-nets, called alternative structured occurrence nets (or AO-nets), was introduced to facilitate the modelling of alternative behaviours, as shown by the example in Figure 2(a). Using the SO-net concept, the activities of a person travelling to a destination in different ways would need to be represented by two distinct O-nets, (A and B), where each one represents a unique scenario. Since these two O-nets represent the same system, modelling in this way results in several duplicate states. An improvement for such a case is shown in Figure 2(b), where the two O-nets of Figure 2(a) are essentially combined into a single structure [3].

Let $AS = \{A_1, \ldots, A_\varphi\}$ be a set of alternative scenarios. A *tagged occurrence net (or TON-net)* is a tuple $tagon = (C, E, F, \vartheta)$ where C and E are disjoint sets of respectively conditions and events (collectively referred to as the nodes), $F \subseteq (C \times E) \cup (E \times C)$ is the flow relation, and $\vartheta : C \cup E \cup F \to 2^{AS} \setminus \{\varnothing\}$ is a mapping, such that, for each $A \in AS$, $tagon(A) = (C(A), E(A), F(A))$ is an O-net , where for $X \in \{C, E, F\}$, $X(A)$ is the set of all $x \in X$ such that $A \in \vartheta(x)$. The initial marking M_0^{tagon}, final marking M_{fin}^{tagon}, input and output of a node x, i.e., ${}^\bullet x$ and $x^\bullet$, in TON-nets are defined in the same way as in O-nets. It is further assumed that for all $A \in AS$, $M_0^{tagon} = M_0^{tagon(A)}$ and $M_{fin}^{tagon} = M_{fin}^{tagon(A)}$.

We define two nodes, x and y, as being *alternatively related (or conflicting)* if there are distinct events, e and f, such that ${}^\bullet e \cap {}^\bullet f \neq \varnothing$ and $(e, x) \in F^*$ and $(f, y) \in F^*$ (denoted by $x \# y$). They are causally related if $(x, y) \in F^+$ or $(y, x) \in F^+$. A *block* of the TON-net is a non-empty set $B \in (C \cup E)$ such that $B \cap C = {}^\bullet(B \cap E) \cap (B \cap E)^\bullet$, $({}^\bullet B \setminus B) \times (B^\bullet \setminus B) \subseteq \prec$, and for all $e, f \in (B \cap E)$, $\neg(e \# f)$.

Essentially, a TON-net can be regarded as an overlay of multiple O-nets, each being tagged by a symbol A in AS representing a unique scenario. This is specified by the mapping ϑ. Thus each element in a TON-net is tagged by one or more tags to indicate to which scenarios it belongs. The tagging is not completely arbitrary; all the elements with the same tag form a valid O-net (i.e., $tagon(A)$). This represents the behavioural report of what has happened from the point of view of one of the possible scenarios $A \in AS$, and thus a step sequence portrays a valid system execution with respect to a particular scenario.

In [4] timed structured occurrence nets (or TSO-nets) are based on groups of associated

Figure 4: The time information of two o-nets

timed o-nets and are designed for reasoning about related events and causality with time information that is uncertain or missing. When modelling a system based on time, such as accidents or crimes, it is important to identify the order in which events have happened and to identify the duration of the events. Each node (condition, event and channel place) has a start time (T_s), finish time (T_f) and duration (D). As shown in Figure 3, all time values have bounded uncertainty represented via specified times intervals ($[T_{s,l}, T_{s,u}]$) and $[T_{f,l}, T_{f,u}]$). In addition, the duration has bounded uncertainty represented via a specified duration interval ($[D_l, D_u]$).

As shown in Figure 4, the time interval information is specified in each arc and prefixed by 'T :'. The time information represents the finish time of the source of the node in addition to the start time of the destination node. The duration interval is prefixed by 'D :' ([3]).

3. Time model

In a time-based system, it is critical to establish the order in which events have fired and the duration intervals between them in order to determine, e.g., eliminate improbable hypothesised scenarios.

The way in which time is represented in our model (using dates, etc.) is much more practical for supporting, e.g., crime investigations than the previous number-based attempts. The notations and definitions below are adapted from [4].

We assume that each node (condition, event) in so-net has a start date (T_s) and finish date (T_f), and that each date and time event has a bounded uncertainty that is represented by a specified earliest and latest date-time interval. ($[T_{s,e}, T_{s,l}]$) and ($[T_{f,e}, T_{f,l}]$ respectively). Also, each node has a duration (D) with a bounded uncertainty represented by a specified early and late duration interval ($[D_e, D_l]$), and a calculated earliest (shortest) and latest (longest) duration between the start date and end date of each node ($[D_e, D_l]$).

Notation. Let n be a node in a so-net. The early and late date-time start interval of n is denoted by $I_s^n = [T_{s,e}^n, T_{s,l}^n]$, where $T_{s,e}^n \leq T_{s,l}^n$ are the earliest and latest date-time start, respectively. The early and late date-time finish interval of n is denoted by $I_f^n = [T_{f,e}^n, T_{f,l}^n]$, where $T_{f,e}^n \leq T_{f,l}^n$ are the earliest and latest date-time start, respectively. The constraint $(T_s^n) \leq (T_f^n)$ states that the start time of n must be at or earlier than the finish time of n. The start and finish time intervals of n should satisfy the following conditions in order to assure consistency with this constraint: $T_{s,e}^n \leq T_{f,e}^n \wedge T_{s,l}^n \leq T_{f,l}^n$. The early and late duration interval of n are denoted by $I_d^n = [D_e^n, D_l^n]$ where $0 \leq D_e^n \leq D_l^n$ are the earliest

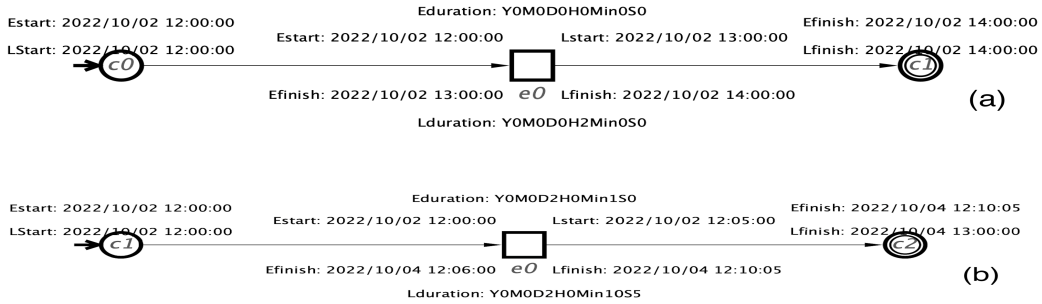

Figure 5: Line-like o-net with a date and time interval.

and latest duration intervals.

Time consistency in line-like o-nets. In line-like o-nets, each event has exactly one input condition and one output condition, and each condition has at most one input and output event. Then, for any two directly connected nodes (i.e., a condition that ends in an event or an event that starts a condition), we assume that the finish time of the source node is equal to the start time of the destination node. Consequently, we have

$$I_f^{n_1} = I_s^{n_2}, \quad for\ every\ \ (n_1, n_2) \in F \tag{1}$$

The information regarding node n is *node consistent* if and only if

$$\begin{aligned} [T_{s,e} + D_e, T_{s,l} + D_l] \cap [T_{f,e}, T_{f,l}] &\neq \varnothing \\ [T_{f,e} - D_l, T_{f,l} - D_e] \cap [T_{s,e}, T_{s,l}] &\neq \varnothing \\ [max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}] \cap [D_e, D_l] &\neq \varnothing \end{aligned} \tag{2}$$

For example, the first part of Condition (2) verifies the bounds are consistent (i.e., overlap) with the specified finish date and time interval of n. A line-like o-net is time consistent if and only if every node n, in the o-net is node consistent and the flow relation of the o-net satisfies Condition (1).

For example, consider the line-like o-nets shown in Figure 5. The date and time intervals are shown above each node (with the prefixes *Estart* and *Lstart* representing the early and late start date and time intervals, respectively), and the early, late finish date and time intervals (with *Efinish* and *Lfinish* representing the early and late finish date and time intervals). In addition, the duration interval of the event node is prefixed by *Eduration* and *Lduration* in the format (Y:*Year*, M:*Month*, D:*Day*, H:*Hour*, *Min*:$Minute$, S:*Second*). Using Condition (2) above, it can be observed that the time information in event $e0$ in Figure 5(a) is inconsistent. Its estimated finish time-interval is $[T_{s,e} + D_e, T_{s,l} + D_l] = [Efinish : 2022/10/02\ 12:00:00, Lfinish : 2022/10/02\ 15:00:00]$, and its specified finish time-interval is $[Efinish : 2022/10/02\ 13:00:00,\ Lfinish : 2022/10/02\ 14:00:00]$. In contrast, event $e0$ in Figure 5(b) is node consistent.

Time consistency in cso-nets. In [3], a cso-net captures communication between different subsystems. Communication between events is represented using special nodes

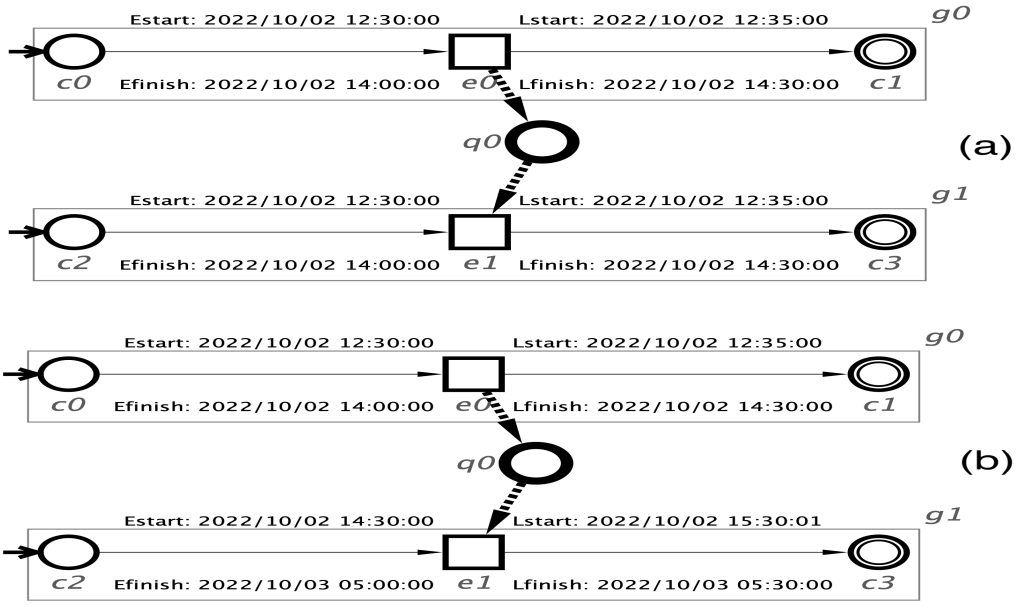

Figure 6: Two CSO-nets with different runs of e_0 and e_1.

called *channel place* that behave similarly to conditions. In asynchronous communication, the sending event e executes either before, or simultaneously with, the receiving event e', and the two events are linked by an *asynchronous channel place* that records information about the communication using a condition. In synchronous communication, the two communicating events execute simultaneously and are linked by two *synchronous channel places* as explained in section 2.

Figure 6 shows how date and time information in a CSO-net can reveal the behaviour of events during asynchronous communication. In Figure 6(a) events e_0 and e_1 have the same start and finish date and time intervals, which indicates that the two events are executed simultaneously. In Figure 6(b) the date and time intervals are different which indicates that e_0 executes earlier than e_1.

Time consistency in bso-nets. The verification of date and time consistency in BSO-nets involves verifying time consistency between O-nets at various levels of abstraction using the behavioural β and *causal* relationships. The assumption made, for the sake of simplicity, is that all abstraction levels have the same date, time origin and granularity.

Given a BSO-net, let *causalU* be the binary relation consisting of the *causally* related pairs of events: $causalU = \bigcup\{causal(e) \mid e \in \mathbf{E}\}$, where $\mathbf{E}$ is the set of events in the O-nets of the BSO-net. The time information of *causalU* is date and time consistent if and only if the following condition is satisfied:

$$\forall (g,h) \in causalU : (T^g_{s,e} \leq T^h_{s,e} \wedge T^g_{s,l} \leq T^h_{s,l}) \tag{3}$$

For all conditions $c_i, c'_i \in \mathbf{C}$ ($\mathbf{C}$ is the set of conditions in the O-nets of the BSO-net) with $(c_i, c'_i) \in \beta$ and c_i belonging to the initial condition of the lower level O-net of the

BSO-net, the following must be true:

$$I_s^{c_i} = I_s^{c'_i} \tag{4}$$

In addition, for all conditions c_t, c'_t with $(c_t, c'_t) \in \beta$ and c_t belonging to the final condition of the lower level O-net of the BSO-net, the following must hold:

$$I_f^{c_t} = I_f^{c'_t} \tag{5}$$

Condition (3) states that there are two events, g and h, the start date and time of event g must be the same as, or precede, the start date and time of event h. The initial and end states of the BSO-net are constrained by conditions (4) and (5): the start and (finish) date and time of a lower level O-net must coincide with the start and (finish) date and time of its corresponding upper level condition.

Figure 7 shows that the post-modified system's (c_4) start time interval and the pre-modified system's (c_3) completion time interval are the same as their respective higher level conditions.

Estimating date-time and duration intervals. Investigations of, e.g., crimes and accidents, typically encounter situations where time information is missing or unavailable. In such cases, it is often required to estimate the time information that would have filled the gaps. Therefore, a missing interval of the node can be estimated if the other two intervals are specified:

$$\begin{array}{lcl} [T_{s,e}, T_{s,l}] & = & [T_{f,e} - D_l, T_{f,l} - D_e] \\ [T_{f,e}, T_{f,l}] & = & [T_{s,e} + D_e, T_{s,l} + D_l] \\ [D_e, D_l] & = & [max(0, T_{f,e} - T_{s,l}, T_{f,l} - T_{s,e})]. \end{array} \tag{6}$$

In situations where both date and time intervals of a node are missing, it is necessary to use a specified time interval from another node. The next section describes algorithms that were adapted from [4] for estimating the missing time intervals of the nodes in SO-net using the following two approaches: the first approach is to estimate the intervals of an individual node, and the second approach is to estimate the intervals of all the nodes of the SO-net.

Estimating finish date-time intervals. Algorithms 1 and 2 adapted from [4] describe the structure of *estimatEfinish*, which computes the finish time interval of a node n using the *causal* functions and are outlined below.

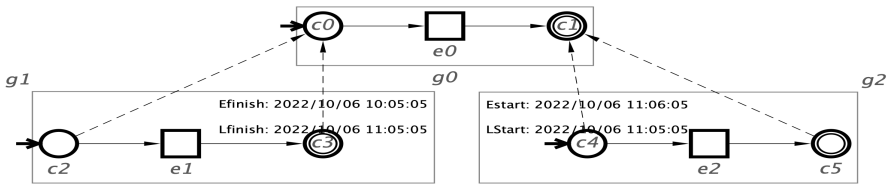

Figure 7: BSO-net portraying system (off-line) update.

Algorithm 1 Estimation finish time interval of a node using causal relation

1: **procedure** ESTIMATE EARLY, LATE FINISH(Node n)
2: $RBoundary := \emptyset$ ▷ nearest right nodes of n with specified early, late finish date and time intervals
3: $RSector := n$ ▷ nodes on paths from n to $RBoundary$ nodes
4: $findRightBoundary(n, RBoundary, RSector)$
5: $backwardBFSDates(n, RBoundary, RSector)$
6: **procedure** FINDRIGHTBOUNDARY($Node\ n, Set\ Boundary, Sector$)
7: $Working := n$ ▷ nodes used for forward boundary searching
8: **while** $Working \neq \emptyset$ **do**
9: $NextWorking := \emptyset$ ▷ nodes with unspecified early and late finish time intervals
10: **for all** $m \in Working$ **do**
11: **if** $causalPostset(m) = \emptyset$ **then**
12: add m to $Boundary$
13: **else**
14: **for all** $nd \in causalPostset(m)$ **do**
15: add nd to $Sector$
16: **if** $nd.Efinish.specified \wedge nd.Lfinish.specified$ **then**
17: add nd to $boundary$
18: **else**
19: add nd to $NextWorking$
20: remove m from $working$
21: $Working := NextWorking$

- Given a node n with an unspecified early and late finish date and time intervals perform forward breadth first search (BFS) using the *findRightBoundary* procedure to identify the nodes with a specified early and late finish date and time interval that are nearest to n, see Figure 8.
- Using the identified nodes, perform the *backwardBFSDates* (BFS) procedure to calculate the unspecified early and late date, time and duration intervals of the nodes causally-related to n (lines 6-7, 11-14). To increase the precision, recalculate the intervals (lines 15-17), until node n is reached.

Estimating finish time interval for ao-nets. In AO-nets, each condition has zero or more input and output events. A condition can have multiple input and output events that are different in different scenarios [5]. Algorithm 3 describes procedure *estimateEarlyFinish*, which computes the finish date-time interval of a node n using causal relations, as follows:

- Conduct a forward depth-first search (DFS) for a node n with an undetermined early finish date-time interval to find all paths that start at n and end to the closest node with a specified early finish date-time interval.
- For each path in AO-net, apply Equation (6) to compute a possible early finish date-time interval of n, where $I^n_{f,e}$ is the specified early finish date-time interval

Algorithm 2 Estimation finish time interval of a node using causal relation

```
1: procedure BACKWARDBFSDATES Node n, Set Boundary, Sector(
       )
2:     Working := Boundary   ▷ nodes used for backward estimation of time intervals
3:     while Working ≠ n do
4:         NextWorking := ∅ ▷ nodes with unspecified date, time and duration intervals
5:         for all m ∈ working do
6:             if ¬m.Eduration.specified ∧ ¬m.Lduration.specified then
7:                 m.Eduration, m.Lduration := defaultDuration
8:                 for all nd ∈ causalPreset(m) ∩ Sector do
9:                     add nd to NextWorking
10:                    nd.visits := nd.visits + 1
11:                    if ¬nd.Efinish.specified ∧ m.Efinish.specified then
12:                        nd.Efinish := m.Efinish − m.Lduration
13:                        if ¬nd.Lfinish.specified ∧ m.Lfinish.specified then
14:                            nd.Lfinish := m.Lfinish − m.Eduration
15:                        else if m.Efinish.specified ∧ m.Lfinish.specified then
16:                            nd.Efinish := nd.Efinish ∩ (m.Efinish − m.Lduration)
17:                            nd.Lfinish := nd.Lfinish ∩ (m.Lfinish − m.Eduration)
18:        for all nd ∈ NextWorking do
19:            if nd.visits = causalPostset(nd) then
20:                for all ndout ∈ causalPostset(nd) do
21:                    if ¬ndout.Estart.specified ∧ ndout.Efinish.specified then
22:                        ndout.Estart := nd.Efinish
23:                        ndout.Eduration := ndout.Eduration ∩ ndout.Efinish −
   ndout.Lstart
24:                        if ¬ndout.Lstart.specified ∧ ndout.Lfinish.specified then
25:                            ndout.Lstart := nd.Lfinish
26:                            ndout.Lduration := ndout.Lduration ∩ ndout.Lfinish −
   ndout.Estart
27:            else
28:                remove nd from NextWorking
29:                Working := NextWorking
```

of the last node in each path and I^n_{Dl} is the late duration of the last node in each path (default late duration interval is used for nodes with unspecified late duration interval).

Figure 9 shows that a forward DFS is used to find all paths from e_0 to the nearest node with a specified early finish date-time interval. More specifically, the first path is $(c_0, e_0, c_1, e_1, c_2, e_3)$ and the second path $(c_0, e_0, c_1, e_2, c_4, e_4)$, the AO-net contains two specified early finish date-time interval:

- The early finish date-time interval of the node e_3 in the first scenario is: $I^{e_3}_{f,e} =$

Algorithm 3 Estimation finish date interval of a node with alternative using causal relation

1: **procedure** ESTIMATE EARLY, LATE FINISH(Node n)
2: **Input:**$AlternativeON$ (AON)
3: **Output:**$AlternativeON$ (AON) with estimated finish date-time interval of all nodes in each scenario
4: $PossibleTimes := \emptyset$ ▷ possible finish date-time intervals from forward search
5: $visited :=<>$
6: add n to $visited$
7: $forwardDFSDate(visited)$
8: **procedure** FORWARDDFSDATE($(List\ visited)$)
9: **for all** $n \in causalPostset(visited.last)$ **do**
10: $I := null$ ▷ possible finish date intervals
11: **if** $I^n_{f,e}.specified \wedge I^n_{f,l}.specified$ **then**
12: $I := I^n_{f,e} - I^n_{Dl}, I^n_{f,l} - I^n_{De}$ ▷ Eqn 6
13: add I to $PossibleTimes$
14: **else if** $n \notin visited$ **then**
15: add n to $visited$
16: forwardDFSDates($visited$)
17: remove $visited.last$

[2022/11/04 05:05:05] and the late duration interval of the node e_3 in the first scenario is $I^{e3}_{D_l} = [Lduration: Y1M1D1H1Min1S1]$, which is subtracted from the specified early finish date-time interval of e_3 using Equation 6.

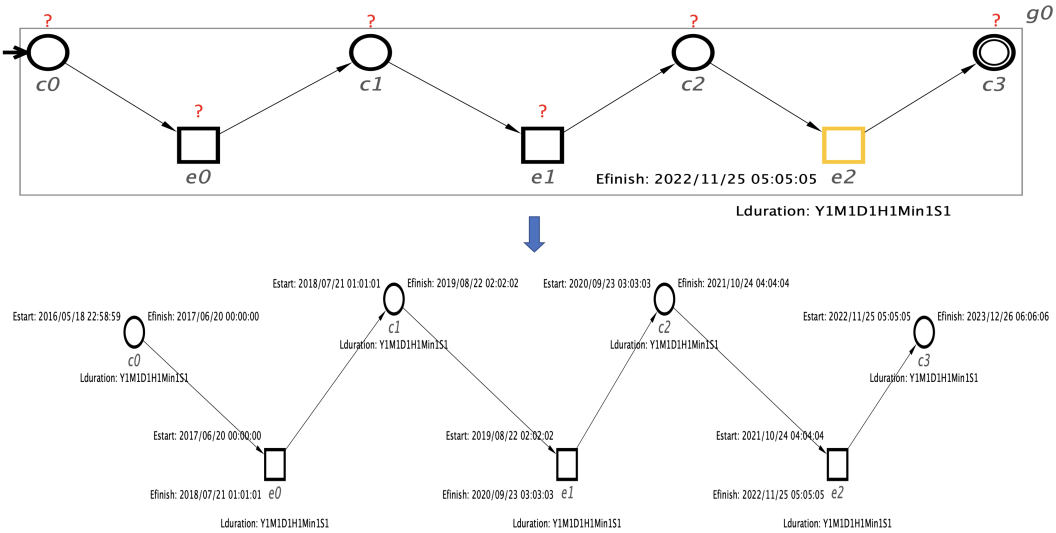

Figure 8: Estimation for early start

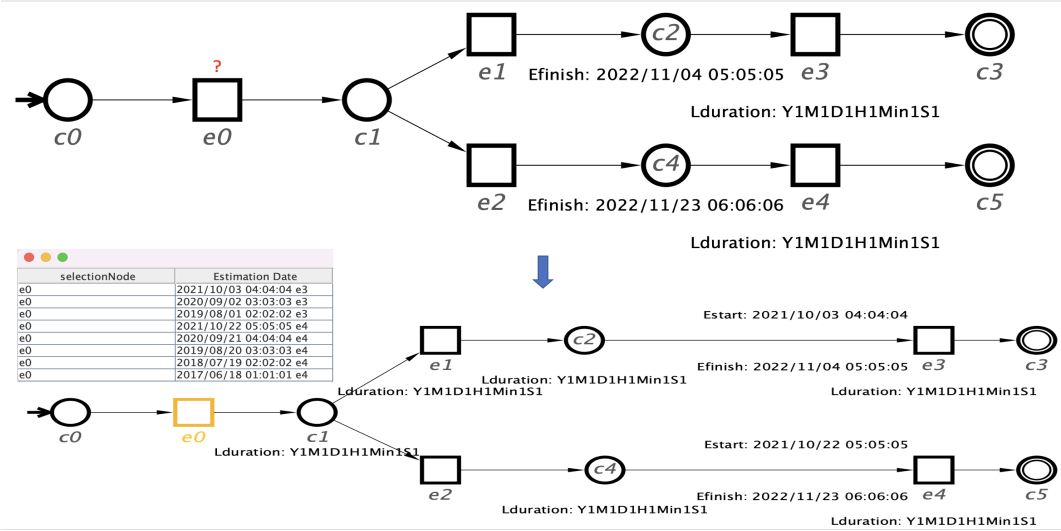

Figure 9: Estimating AO-net early start

- Another possible early finish date-time interval is calculated using the second scenario. The early finish date-time interval of the node e_4 in the second scenario is: $I^{e_4}_{f,e} = [2022/11/23\ 06{:}06{:}06]$ and the late duration interval of the node e_4 in the second scenario is $I^{e_4}_{D_l} = [Lduration{:}\ Y1M1D1H1Min1S1]$. Figure 9 shows the table for possible times estimations of the early finish date-time interval for e_0 using the first and second scenarios. Figure 10 shows the table for possible times estimations result for each scenario separately.

4. Hierarchical Simulation of Timed Behaviours

Simulation is a crucial problem-solving technique for tackling a variety of real-world problems, and can be used to analyse and explain the behaviour of a system. In a real-time simulation, the simulation is run in discrete time with constant steps, also known as fixed step simulation, as time moves forward in equal duration of time [2].

Time simulation in so-nets. Every activity in a system has a time duration interval which is different from zero, and we make the added assumption that all activities complete in a finite amount of time. We will assume that every transition takes a bounded, non-zero amount of time to fire. In the semantics of non-timed O-nets, transitions can fire at any time after they are enabled, removing input token and creating output token. When firing durations are included, the O-net semantic is changed. Each transition has a time associated with it. When a transition becomes enabled, it removes the input token immediately but does not create the output token until the firing duration has finished.

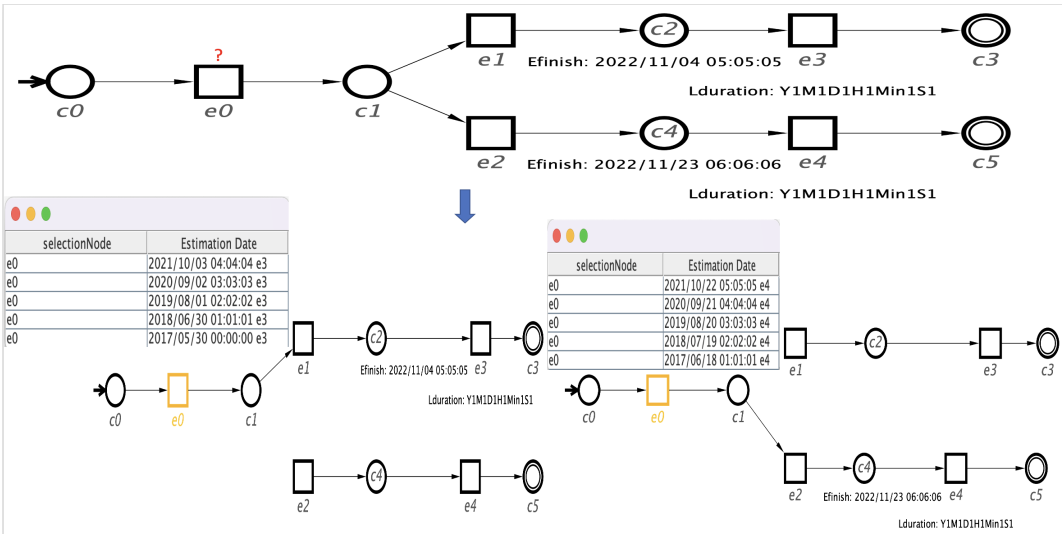

Figure 10: Estimating AO-net early start for each scenario.

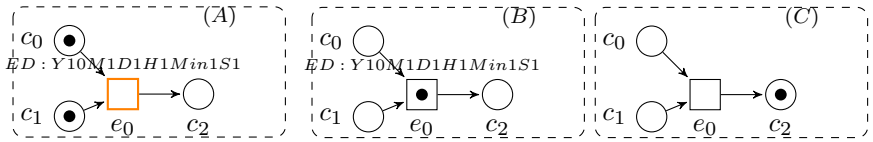

Figure 11: Time Transition firing.

Time transition firing. In TSO-nets, an enabled transition is fired in three 'conceptual' steps: the first (immediate) step removes tokens from the input places, the second (temporal) step holds the tokens for the duration of the firing time, and the third (immediate) step moves tokens to all of the transition's output places. For example, as shown in Figure 11 early duration interval is displayed in e_0 as ED:$Y10, M1, D1, H1, Min1, S1$ which represent $Year, Month, Day, Hour, Minute, Second$ respectively.

Time in synchronous so-nets. As shown in Figure 12, the CSO-net with two O-nets and synchronous communication. It shows that e_0 and e_1 are enabled and have the same duration intervals, which indicates that the two events are executed simultaneously. Therefore, after firing e_0 or e_1 the transitions will hold the tokens for duration of the firing time, and then moves tokens to all of the transition's output places.

Interval-timed acyclic net. The simulation of timed behaviours of CESs in meaningful ways so that, for example, an incident (crime) investigator can use them effectively. The main feature of interval-timed acyclic nets (described below) which are used for simulation after working out the timings of transitions, is that transitions which are enabled need to start immediately their enabling, and the firing lasts some time within an interval. Thus, the *startfiring* and the *endfiring* of a transition are considered as two distinct events [6].

The firing of an enabled transition in timed simulation is composed of two 'conceptual' steps; the first calculates duration of the firing time, and the second one starts to countdown the time units of the duration for the enabled transition [6]. Below $\mathbb{N}$ is the set of non-negative integers, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$.

Definition 1 (interval-timed acyclic net)**.** *An* interval-timed acyclic net *(or* ITA*-net) is a quadruple* $itanet = (P,T,F,I)$ *such that* (P,T,F) *is a well-formed acyclic net, and* $I : T \to \{[n,m] \mid n,m \in \mathbb{N}_+ \wedge n \leq m\}$ *is its* interval function*.*

For every $t \in T$, we denote $I(t) = [\mathrm{sfd}(t), \mathrm{lfd}(t)]$, calling $\mathrm{sfd}(t)$ and $\mathrm{lfd}(t)$ the *shortest firing duration* and the *longest firing duration* of t, respectively.

Definition 2 (state of ITA-net)**.** *A* state *of an* ITA*-net* $itanet = (P,T,F,I)$ *is a pair* $S = (M,h)$ *such that* $M \subseteq P$ *and* $h \subseteq T \times \mathbb{N}$ *is the* clock relation*. The* initial state *of itanet is* $M^{init}_{itanet} = (M^{init}_{(P,T,F)}, \varnothing)$*. All states of itanet are denoted by* states(*itanet*)*.*

The pair $(t,0) \in h$ means that t has just finished its *activity*, and $(t,k) \in h$ with $k > 0$ means that t has still k time units to finish. Moreover, if there is no k such that $(t,k) \in h$, then t is *inactive*.

Definition 3 (state change rules)**.** *Let* $S = (M,h)$ *be a state of an* ITA*-net* $itanet = (P,T,F,I)$*. The following are possible state-changing events:*

- ***Startfire events:*** $S \xrightarrow[+]{U} (M \setminus {}^\bullet U, h \cup \{(t,k_t) \mid t \in U\})$*, where* U *is a maximal step enabled at* M*, and* $\mathrm{sfd}(t) \leq k_t(t) \leq \mathrm{lfd}(t)$*, for every* $t \in U$*.*
- ***Endfire events:*** $S \xrightarrow[-]{V} (M \cup V^\bullet, h \setminus \{(t,0) \mid t \in V\})$*, where* $V = \{t \mid (t,0) \in h\}$*.*
- ***Tick events:*** $S \xrightarrow{\checkmark} (M, \{(t,k-1) \mid (t,k) \in h\})$*.*
- ***Global events:*** $S \xrightarrow[\pm]{U:V} S'$*, provided that:* $S \xrightarrow[-]{V} S'' \xrightarrow[+]{U} S''' \xrightarrow{\checkmark} S'$*, for some states* S'' *and* S'''*.*

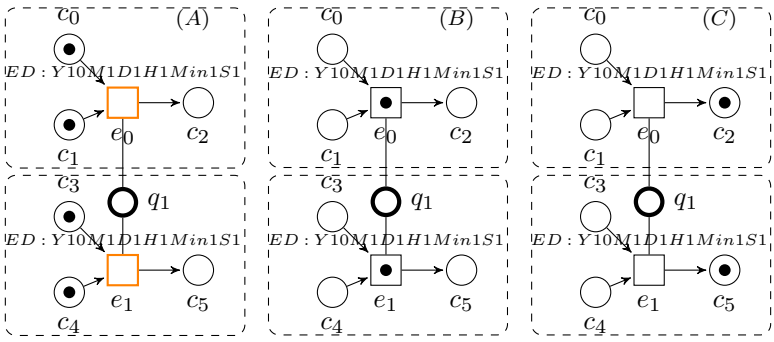

Figure 12: Time in synchronous communication.

Clearly, a global event $S \xrightarrow[\pm]{U:V} S'$ may be such that $U = V = \varnothing$, in which case only the tick event has an effect. If, in addition, $S = (M, \varnothing)$, then also the tick event has no effect and $S = S'$ and M is a terminal marking of *acnet*.

Definition 4 (time semantics)**.** *The* time step sequence semantics *of itanet is defined through sequences of global events:*

$$(M_{itanet}^{init} =) S_0 \xrightarrow[\pm]{U_1:V_1} S_1 \xrightarrow[\pm]{U_2:V_2} \dots \xrightarrow[\pm]{U_{n-1}:V_{n-1}} S_n.$$

Algorithm 4 allows to simulate the timed behaviours for o-net the (line 6 and 7) check if the intervals late start and early finish are specified for the transition node and then calculate the early duration interval for the event. Line 10 displays the duration above the event with colours for each time unit (*Year*:red, *Month*:magenta, *Day*:blue, *Hour*:green, *Minute*:purple, *Second*:cyan). Lines 11 and 12 start firing using the duration interval.

1. Check if *latest start time* and *earliest finish time* of t are specified.
2. Calculate the *shortest firing duration* then display the time units of t *Eduration*: Y, M, D, H, Min, S denoted by *Year*, *Month*, *Day*, *Hour*, *Minute* and *Second* respectively.
3. Remove the token from the *pre-place* to enabled transition t to holds the token for the duration of the firing time, then the timer will start by decrementing the time unit.
4. If the time units are finished then the token moves automatically to the transition *post-place*

Maximal events (firing steps). The time simulation for a maximal enabled events of multiple o-nets, the maximal sets of firable transitions are selected so that the enabled transitions in those sets must all fire simultaneously, and the firing of each transition takes a specific amount of time. Algorithm 5 implements maximal events (firing steps), line (6-8) check if the *late start* and *early finish* intervals are specified in the enabled transitions then calculate the early duration interval and associate it with each transition. Line (12-14) start firing simultaneously all the enabled transitions using the *early duration* interval.

In Figure 13, we have scenarios A, B and C with a set of enabled transitions that have different duration interval for each transition. The early duration interval ED is associated with each transition t, the firing of the transition t takes exactly ED time units $ED : Y, M, D, H, Min, S$. When a transition starts to fire then its time units start to countdown $I(t)$. If there is a conflict between several enabled maximal sets, the choice is arbitrarily solved.

Simulation of the o-nets and ao-nets is illustrated in Figure 13. We have two sequences of maximal steps: σ_1 can fire $\{a_{st}, c_{st}, g_{st}\}$, then $\{g_{fin}, \checkmark\}$ after that g finish firing and h can start firing, $\{h_{st}\}$; and σ_2 can fire $\{a_{st}, c_{st}, e_{st}\}$ another possible maximal steps can fire in Figure 13, $\{a_{fin}, \checkmark\}$, after that a finish firing and b can start firing $\{b_{st}\}$, where st indicates the start of firing and fin is the finish of firing.

Algorithm 4 Timed Simulation algorithm of ON

Inputs:
$ON-$ Occurrence Net with time intervals
$M-$ current marking
Output:ON with a step enabled and time units
$U :=$ *a step* of ON
for all $t \in$ **T do**
  if ${}^\bullet t \subseteq M$ **then** ▷ t is ON-enabled
  if $I^t_{l,s}.specified \wedge I^t_{e,f}.specified$ **then**
    $I^t_{e,d} := I^t_{e,f} - I^t_{l,s}$
    display duration in t
    add t to U
for all t $\in U$ **do**
  if $t \in U$ has time units **then**
    $t-firing$ according to its time units
    if t its time units $= 0$ **then**
      $t-$executed
$t-$calculate new state

5. Implementation

WORKCRAFT [7] is a tool that provides a flexible common underpinning for graph-based models. Its plug-in, SONCRAFT, provides some initial facilities for entering, editing,

Algorithm 5 Timed Simulation algorithm for maximal sets of enabled transitions

Inputs:
$ON-$ set of scenarios (ONs) with time intervals
$M-$ current marking
Output:ON set of maximal steps (ONs) and time units
array set of $maxenabledU := []$
for all $t \in$ **T do**
  for all ${}^\bullet t \subseteq M$ **do** ▷ t is ON-enabled
    if $I^t_{l,s}.specified \wedge I^t_{e,f}.specified$ **then**
      $I^t_{e,d} := I^t_{e,f} - I^t_{l,s}$
      display duration in t
      add t to set of $maxenabledU$
for all t $\in$ set of $maxenabledU$ **do**
  if $t \in$ set of $maxenabledU$ has time units **then**
    $t-startfiring$ according to its time units
    if t its time units $= 0$ **then**
      $t-$executed
$t-$calculate new state

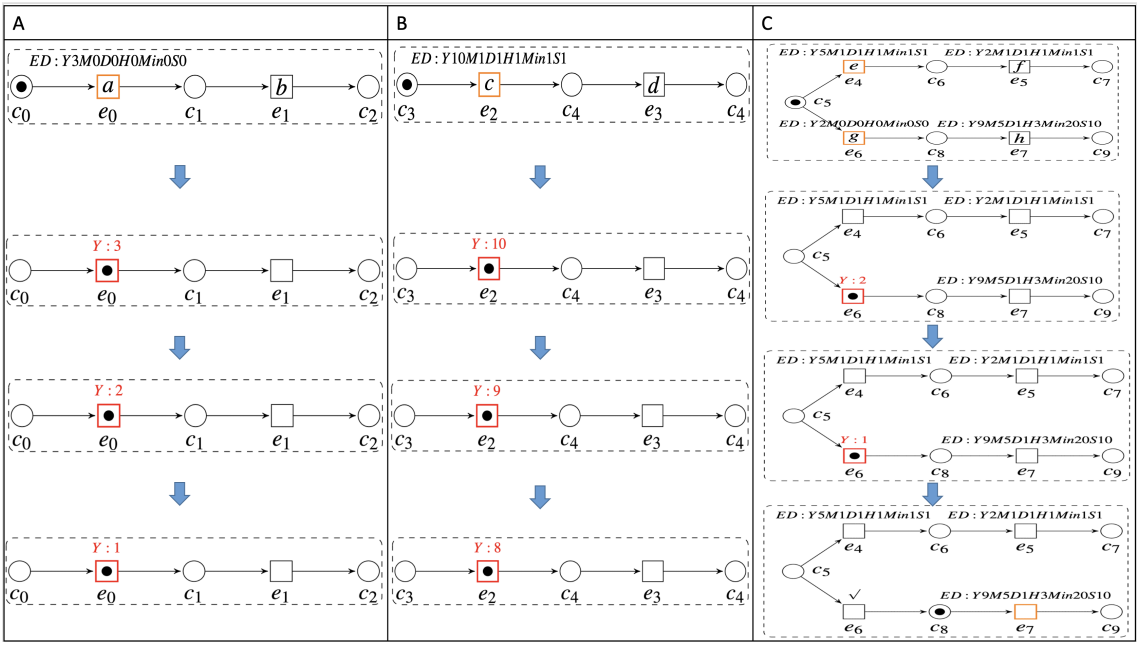

Figure 13: Firing maximal enabled events.

validating and simulating three types of SO-net structures, namely, CSO-nets, BSO-nets and TSO-nets abstractions. We have implemented date-time intervals for events and conditions and their analysis in SONCRAFT. In this section, we present the implementation of the date-based tools and the time simulation tool.

Time setting tool. The *time setting tool* shown in Figures 14 and 15 is an interface for specifying date-time and duration intervals of event and condition nodes in a SO-net model. The early and late duration is specifying as follows (*Y*:*Year*, *M*:*Month, D:Day, H:Hour, Min:Minute, S:Second*). Users can manually set the date information for a selected event in the date panel. For each manually input date, the tool verifies whether or not the date or time is well-defined for each event in the SO-net model. Moreover, for each node, the date is checked.

Date consistency checking tool. The *date consistency checking* is a tool that provides consistency checking for the time information specified for events in the SO-net model. Figure 16 shows the tool performing a consistency check for O-net models. The result displays the time information checking task when a node has complete, partial or empty time information, as in Figure 17. Moreover, nodes with a date information and inconsistency error from any O-net are shown. Figure 17 shows one event with a date inconsistency error. For example, the error message involves event e_0 and shows that its start date (*Estart*: 2022/10/04 06:20:02, *Lstart*: 2022/10/04 06:20:02) is greater than its end date (*Efinish*: 2022/10/03 00:00:00, *Lfinish*: 2022/10/03 00:00:00); in addition, the consistency verification also checks the consistency for CSO-net and BSO-net models.

Timed simulation tool. Each event in the SO-net has an early and late duration interval, which were used to simulate the time of the event. In the time simulation tool we have implemented two buttons to simulate the duration of the event: early time simulation button, and late time simulation button, see Figure 18.

The simulation function in the SO-net plugin can be activated by clicking on the early time simulation button in the editor tools panel. The initial marking is automatically set, and all enabled events are highlighted Figure 19. The timed simulation is then conducted manually by clicking the enabled event after which the calculation for the early event duration starts, displaying the result above each event. As shown in Figure 20, a countdown then begins; the durations have different colours for each time unit (*Year*:red, *Month*:magenta, *Day*:blue, *Hour*:green, *Minute*:purple, *Second*:cyan)

6. Concluding Remarks

The concept of a timed SO-nets has been introduced in this paper in order to represent and analyse causally connected events and concurrent occurrences in developing systems with ambiguous or lacking temporal information. In addition, we presented a timed simulation tool for, e.g., criminal investigations to help investigators reconstruct the timeline of events and analyse the behaviour of the systems.

The future work will add multi-level modelling and embedding time granularity. A key problem of handling complexity when large datasets are involved will be addressed by developing hierarchical approach to simulation based on behavioural abstractions.

References

[1] E. Best, R. Devillers, Sequential and concurrent behaviour in Petri net theory, Theoretical Computer Science 55 (1987) 87–136.

[2] J. Banks, Introduction to simulation, in: P. A. Farrington, H. B. Nembhard, D. T. Sturrock, G. W. Evans (Eds.), Proceedings of the 31st conference on Winter simulation: Simulation - a bridge to the future, WSC 1999, Phoenix, AZ, USA, December 05-8, 1999, Volume 1, WSC, 1999, pp. 7–13.

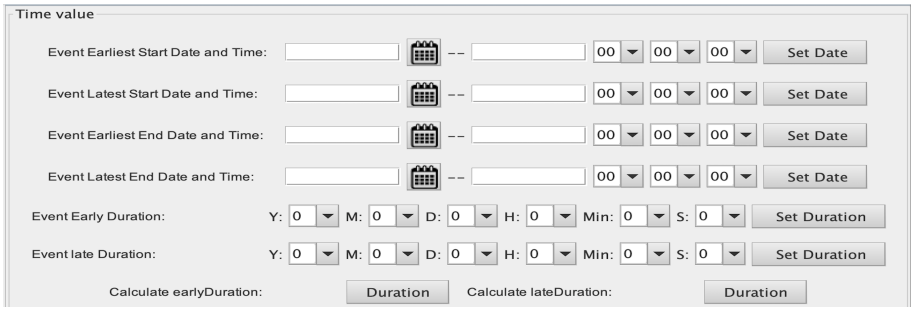

Figure 14: Time property setter.

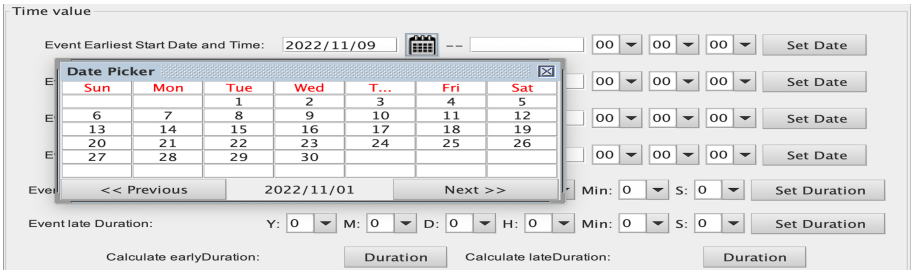

Figure 15: Calendar to set date.

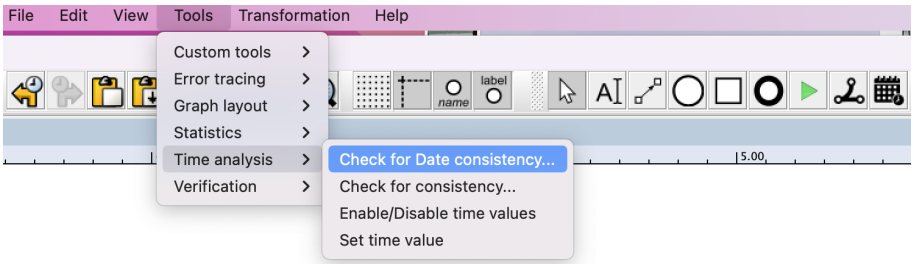

Figure 16: Date consistency.

[3] B. Li, Visualisation and Analysis of Complex Behaviours using Structured Occurrence Nets, Ph.D. thesis, School of Computing Science, Newcastle University, 2017.

[4] A. Bhattacharyya, B. Li, B. Randell, Time in structured occurrence nets, in: L. Cabac, L. M. Kristensen, H. Rölke (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering 2016, Toruń, Poland, June 20-21,

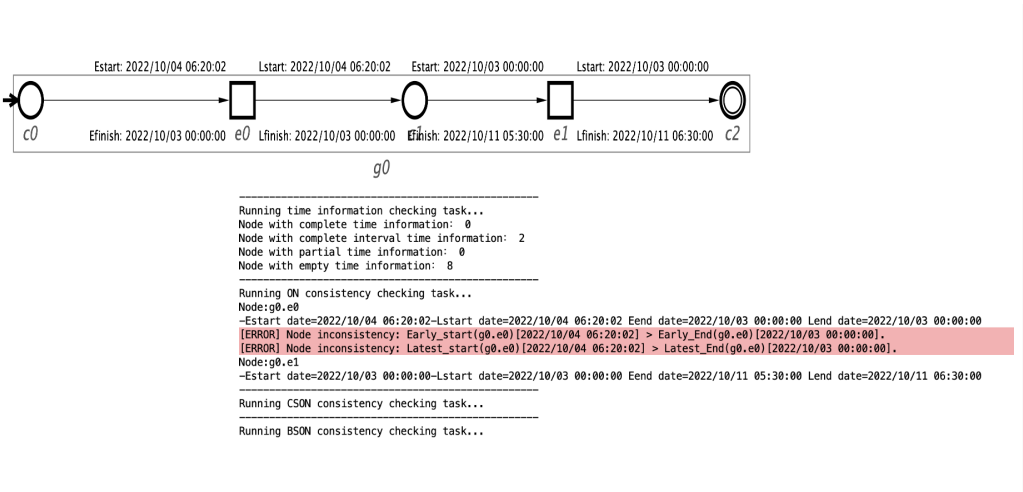

Figure 17: O-net with date consistency checking.

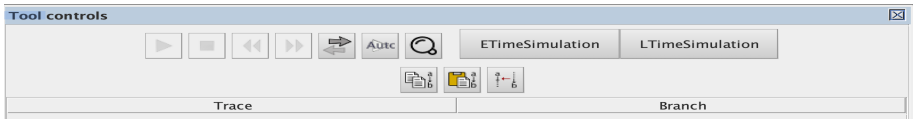

Figure 18: Buttons for time simulation.

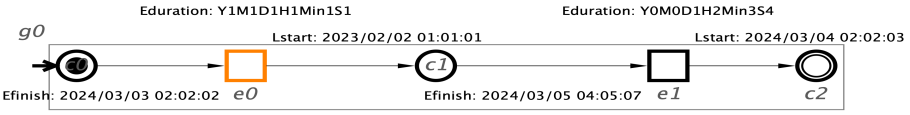

Figure 19: Enabled event to simulate the time.

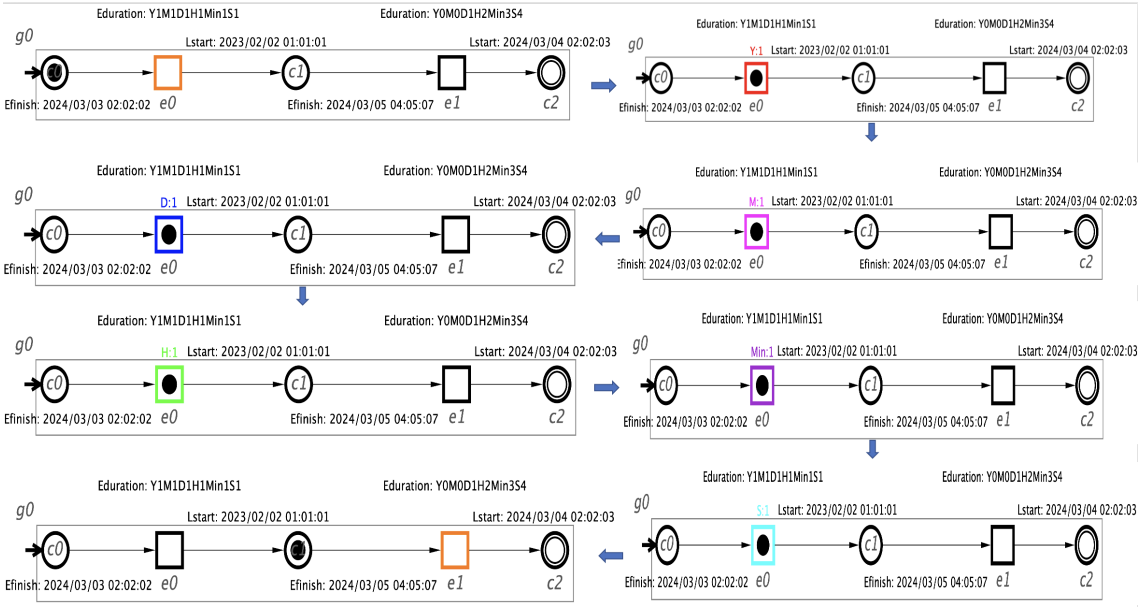

Figure 20: *Year*, *Month*, *Day*, *Hour*, *Minute* and *Second*.

2016, volume 1591 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 35–55.

[5] M. Koutny, B. Randell, Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques, Fundamenta Informaticae 97 (2009) 41–91.

[6] E. Pelz, Timed processes of interval-timed Petri nets, in: B. Schlingloff (Ed.), Proceedings of the 25th International Workshop on Concurrency, Specification and Programming, Rostock, Germany, September 28-30, 2016, volume 1698 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 13–24.

[7] I. Poliakov, V. Khomenko, A. Yakovlev, Workcraft - A framework for interpreted graph models, in: G. Franceschinis, K. Wolf (Eds.), Applications and Theory of Petri

Nets, 30th International Conference, PETRI NETS 2009, Paris, France, June 22-26, 2009. Proceedings, volume 5606 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 333–342.

[8] P. Missier, B. Randell, M. Koutny, Modelling provenance using structured occurrence networks, in: P. Groth, J. Frew (Eds.), Provenance and Annotation of Data and Processes - 4th International Provenance and Annotation Workshop, IPAW 2012, Santa Barbara, CA, USA, June 19-21, 2012, Revised Selected Papers, volume 7525 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 183–197.

[9] B. Randell, Incremental construction of structured occurrence nets, Technical Report, School of Computing, Newcastle University, 2013.

[10] B. Li, B. Randell, A. Bhattacharyya, T. Alharbi, M. Koutny, Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets, in: 18th International Conference on Application of Concurrency to System Design, ACSD 2018, Bratislava, Slovakia, June 25-29, 2018, IEEE Computer Society, 2018, pp. 70–74.

[11] J. Kawises, W. Vatanawood, Formalizing time Petri nets with metric temporal logic using promela, in: M. Nakamura, H. Hirata, T. Ito, T. Otsuka, S. Okuhara (Eds.), 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2019, Toyama, Japan, July 8-11, 2019, IEEE, 2019, pp. 162–166.

[12] J. Euzenat, A. Montanari, Time granularity, in: M. Fisher, D. M. Gabbay, L. Vila (Eds.), Handbook of Temporal Reasoning in Artificial Intelligence, volume 1 of *Foundations of Artificial Intelligence*, Elsevier, 2005, pp. 59–118.

[13] T. Alharbi, M. Koutny, Visualising data sets in structured occurrence nets, in: D. Moldt, E. Kindler, H. Rölke (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'18), co-located with the39th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2018 and the 18th International Conference on Application of Concurrency to System Design ACSD 2018, Bratislava, Slovakia, June 24-29, 2018, volume 2138 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 121–132.

[14] T. Alshammari, Towards automatic extraction of events for SON modelling, in: M. Köhler-Bussmeier, D. Moldt, H. Rölke (Eds.), Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022, volume 3170 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 188–201.

[15] M. Alahmadi, Master channel places for communication structured acyclic nets, in: M. Köhler-Bussmeier, E. Kindler, H. Rölke (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021), Paris, France, June 25th, 2021 (due to COVID-19: virtual conference), volume 2907 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 233–240.

[16] N. Almutairi, M. Koutny, Verification of communication structured acyclic nets using SAT, in: M. Köhler-Bussmeier, E. Kindler, H. Rölke (Eds.), Proceedings

of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021), Paris, France, June 25th, 2021 (due to COVID-19: virtual conference), volume 2907 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 175–194.

[17] N. Almutairi, Probabilistic communication structured acyclic nets, in: M. Köhler-Bussmeier, D. Moldt, H. Rölke (Eds.), Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022, volume 3170 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 168–187.

A. Checking Consistency

Algorithm 6 first verifies *Concurrent Consistency* with respect to the late finish date and time interval of the input places of the given event, then with respect to the early start date and time intervals of the output places of the event, then invokes *nodeConsistency* for the basic consistency checking of the event itself.

In Algorithm 7 asynchronous consistency function is invoked only if a SO-net contains a communication relation. The function applies two conditions for asynchronous and synchronous-based checking. Formally, let q be a channel place and let e, e' be the input and output events of q respectively. The date information of q is defined to be *a/synchronously consistent* if and only if the following conditions are satisfied: $I^{e}_{f,l} = I^{q}_{s,e}$ and $I^{e'}_{s,e} = I^{q}_{f,l}$.

In Algorithm 8, the *behavioural consistenc* function in line 4 verifies the consistency of all binary relations in *causalU*, lines 7 and 9 verify the restrictions on the initial and final places of the BSO-net. The return value of the function is all the nodes which are behaviourally inconsistent.

In AO-nets, each event has at least one input condition and at least one output condition, and each condition has zero or more input and output events. A condition in AO-net can have multiple input and output events that are in different scenarios. The verification of the consistency of the alternative O-nets ensures that each condition is in at least one scenario. Algorithm 9 displays the *alternativeConsistency* function in line 2. It states that the early start date and time interval of c is the same as the late finish date and time interval of an input event e, line 4 states that the late finish date and time intervals of c are the same as the early start date and time intervals of an output event e'.

Algorithm 6 Concurrent consistency

function: Boolean concurrent consistency (Event e)
for $c \in {}^\bullet e$ **do**
  if $c.Lfinish \neq e.Estart$ **then**
    return $FALSE$
for $c \in e^\bullet$ **do**
  if $c.Estart \neq e.Lfinish$ **then**
    return $FALSE$
 return $nodeConsistency(e)$

Algorithm 7 A/Synchronous consistency

function: Boolean asynchronous consistency(Channel place q)
if $q.input.Lfinish \neq q.Estart \vee q.output.Estart \neq q.Lfinish$ **then**
  return $FALSE$
return $nodeConsistency(q)$

Algorithm 8 behavioural consistency

function: Boolean behavioural consistency (Relation $causalU$)
$Result := \emptyset$ ▷ behaviourally inconsistent nodes
for $(e_1, e_2) \in causalU$ **do**
  if $e_1.Estart.lower > e_2.Estart.lower \vee e_1.Lstart.upper > e_2.Lstart.upper$ **then**
    add e_1, e_2 *to* $Result$
for $c \in$ **C do**
  if ${}^\bullet c = \emptyset \wedge c.Estart \neq \beta(c).Estart$ **then**
    add c to $Result$
  else if $c^\bullet = \emptyset \wedge c.Lfinish \neq \beta(c).Lfinish$ **then**
    add c to $Result$
return $Result$

Algorithm 9 Alternative consistency

function: Boolean alternative consistency (Condition c)
if $e.input.Lfinish \not\models c.Estart$ **then**
  return $FALSE$
if $e.output.Estart \not\models c.Lfinish$ **then**
  return $FALSE$