Mining Declarative Models Using Time Intervals

Jan Martijn van der Werf [⋆], Ronny S. Mans [⋆⋆], and Wil M.P. van der Aalst

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`{ j.m.e.m.v.d.werf, r.s.mans, w.m.p.v.d.aalst }@tue.nl`

Abstract. A common problem in process mining is the interpretation of the time stamp of events, e.g., whether it represents the moment of recording, or its occurrence. Often, this interpretation is left implicit. In this paper, we make this interpretation explicit using time intervals: an event occurs somewhere during a time window. The time window may be fine, e.g., a single point in time, or coarse, like a day. As each event is related to an activity within some process, we obtain for each activity a set of intervals in which the activity occurred. Based on these sets of intervals, we define ordering and simultaneousness relations. These relations form the basis of the discovery of a declarative process model describing the behavior in the event log.

Keywords: Process mining, time intervals, concurrency theory, declarative process models

1 Introduction

Information systems of today collect large amounts of data. For example, banks are saving information about the granting of mortgages and loans, insurance companies are saving information concerning the handling of claims, and hospitals are saving the actions taken to treat patients. Many of the recorded data concern events which have been performed in the context of a certain business process. For each event, different aspects are stored, for example, the activity and case for which the event is raised, its type, and when it has been raised. Process mining [1] aims to extract process knowledge from these recorded events to discover, monitor and improve the actual processes supported by these systems.

The information about when the event occurred, for example using the order in which events are recorded, or its recorded timestamp is used to discover, monitor and check the control flow of processes. The implicit assumption many of the process mining algorithms make is that if two events are recorded consecutively, e.g. one is recorded

[⋆] supported by the PoSecCo project (project no. 257129), partially co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7).

[⋆⋆] supported by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Ministry of Economic Affairs.

before the other, or the timestamp of the first is before the latter, they occurred consecutively. However, in many cases, this assumption may not hold, as systems implement log recording differently. So, although time information is recorded, it can be interpreted in many different ways.

One interpretation of the timestamp is that it is the time on which the event actually occurred. More likely, many systems implement this as the time on which the event is recorded. Other systems implement logging using a queue system, i.e., the event is placed in the queue, and then written. Thus, if two events occur at the same time, their timestamp may differ as they are written consecutively.

A second problem of timestamps is their scale. On the one hand, a too fine time scale introduces causality that in reality does not exist. For example, consider an information system consisting of many different components each with their own logging mechanism. To construct the process of the information system, the recordings of each component need to be combined in a single event log. As a result, one needs to ensure that all components have the same time. On the other hand, a too coarse time scale may falsely introduce concurrency. For example, if the time scale is in days, the order of activities executed on the same day cannot be discovered.

A third problem lies in the reliability of the time information. For example, the order based on timestamps of events is more reliable if the same timestamp generator is used. Thus, timestamps of events in the same component are more reliable than when the events are recorded by different components. Another source of unreliability is whether time information depends on user input, such as calendars, or if it is generated by the system.

As a result, one should always first check the order in which events occur. One way to resemble this is to use intervals for event occurrences instead of single timestamps. This allows to change the time scale from a very fine scale, such as single points, to very coarse time scales, such as days. For example, an event that occurred on timestamp '2013/04/12 12:24:36.3', can be seen as an interval of a single point, or, if the required time scale under consideration is in days, it can be seen as an event that occurred on April 12, 2013, i.e., in the interval '2013/04/12 0:00' - '2013/04/12 23:59:59'.

Process mining focuses on the extraction of process knowledge. Whereas process knowledge mainly focuses on the level of activities, systems recordings are on an event level, which is not necessarily the same level, as several events may be raised for the same activity, for example when the activity started or completed. Thus, to be able to reason on the level of activities, events should be combined into activities. Aggregration of events in activities can be done in many ways, such as the life-cycle of activities [1]. Depending on the aggregration, each activity may occur several times, each in its own time interval, resulting in a set of time intervals for each activity.

In this paper, we want to make the time intervals in which an activity occurs explicit. Based on a set of intervals for each activity, we reason about which relations can be inferred. For example, do two activities occur simultaneously or do they occur sequentially. As we use intervals instead of single time points, many activities may occur concurrently. Procedural languages, like Petri nets [3], model explicitly the order in which activities occur. For example, places in a Petri net are used to control choices and to reduce the degree of concurrency in a model. As a consequence, concurrency needs to be modelled explicitly, rather than being a language primitive. In a declarative approach, all events may be executed concurrently, unless it is prohibited by constraints. Therefore, we choose a declarative modelling language instead, called *Declare* [2], and show how a declarative model can be derived from the intervals induced by the timestamps of the events.

This paper is structured as follows. In Sec. 2 we introduce the basic notions used throughout the paper. Sec. 3 discusses the role of intervals within an event log. These events and their intervals can be mapped onto activities in many different ways, as shown in Sec. 4. Next, in Sec. 5, we define simultaneousness and causality relations on sets of intervals. Sec. 6 presents a method to build a declarative model based on these interval relations. Last, Sec. 7 concludes the paper.

2 Basic Notions

Let S be a set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. We use $|S|$ for the number of elements in S. Two sets U and V are *disjoint* if $U \cap V = \emptyset$. We denote the cartesian product of two sets S and T by $S \times T$. On a cartesian product we define two projection functions $\pi_1 : S \times T \rightarrow S$ and $\pi_2 : S \times T \rightarrow T$ such that $\pi_1((s,t)) = s$ and $\pi_2((s,t)) = t$ for all $(s,t) \in S \times T$. We lift the projection function to sets in the standard way.

A binary relation R from S to T is defined by $R \subseteq (S \times T)$. For $(x,y) \in R$, we also write $x\,R\,y$. For a relation $R \subseteq (S \times T)$, the inverse relation R^{-1} is defined as $R^{-1} = \{(y,x) \in (T \times S) \mid x\,R\,y\}$. A relation R is called a function if $x\,R\,y$ and $x\,R\,z$ implies $y = z$ for all $x \in S$ and $y,z \in T$. It is called a binary relation over S if $R \subseteq (S \times S)$. A binary relation R is reflexive if $x\,R\,x$ for all $x \in S$. It is transitive if $x\,R\,y$ and $y\,R\,z$ implies $x\,R\,z$ for all $x,y,z \in S$. It is reflexive if $(x,x) \in R$ for all $x \in S$, and irreflexive if $(x,x) \notin R$ for all $x \in S$. Relation R is symmetric if $x\,R\,y$ implies $y\,R\,x$ for all $x,y \in S$ and asymmetric if $x\,R\,y$ implies $\neg y\,R\,x$ for all $x,y \in S$. The relation is antisymmetric if $x\,R\,y$ and $y\,R\,x$ imply $x = y$ for all $x,y \in S$. The transitive closure of a binary relation R is defined as the smallest relation R^+ such that $x\,R^+\,y$ if either $x = y$, or $x\,R^+\,z$ and $z\,R\,y$ for some $z \in S$.

A binary relation R over some set S is an *equivalence relation* if it is reflexive, symmetric and transitive. A transitive, irreflexive binary relation is called a *strict order*. It is a *preorder*, denoted by (S,R), if R is reflexive and transitive. A preorder is a *partial order* if (S,R) is also antisymmetric. A partial order is called a *total order*, if in addition also $x\,R\,y$ or $y\,R\,x$ for all $x,y \in S$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1,\ldots,n\} \rightarrow S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1,\ldots,n\}$, we write $\sigma = \langle a_1,\ldots,a_n\rangle$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. Let $\nu,\gamma \in S^*$ be two sequences. *Concatenation*, denoted by $\sigma = \nu;\gamma$ is defined as $\sigma : \{1,\ldots,|\nu| + |\gamma|\} \rightarrow S$, such that for $1 \leq i \leq |\nu|$: $\sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$: $\sigma(i) = \gamma(i - |\nu|)$.

Given a set S and a, possibly infinite set $T \subseteq \mathbb{R}$, a function $f : S \rightarrow T \times T$ is called an *interval function* if $\pi_1(f(a)) \leq \pi_2(f(a))$ for all $a \in S$.

2.1 Event Logs

For each user action on the system, an *event* is raised. An event records its type, for which activity it has been raised, for which *case* or business process instance, when it was raised, by whom, and the data inserted by the user. Such a recording is called an *event log* [1]. The set of all possible events, i.e., the event universe is denoted by $\mathcal{E}$. Similarly, we denote the case, attribute and value universes by $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{V}$, respectively, such that $\mathcal{E}, \mathcal{C}, \mathcal{A} \subseteq \mathcal{V}$ and $\mathcal{E}$, $\mathcal{C}$ and $\mathcal{A}$ are pairwise disjoint. We assume $A \subseteq \mathcal{V}$ to be the (possibly infinite) set of activities.

Definition 1 (Event log). *An* event log *is a 3-tuple* $L = (C, E, \#)$ *where*

- $C \subseteq \mathcal{C}$ *is a set of* case identifiers *in the event log;*
- $E \subseteq \mathcal{E}$ *is a set of* event identifiers *in the log;*
- $\# : \mathcal{A} \times (C \cup E) \rightarrow \mathcal{P}(\mathcal{V})$ *is an attribute mapping.*

For an attribute $n \in \mathcal{A}$ *we write* $\#_n(\cdot)$ *as a shorthand for* $\#(n, \cdot)$. *The following attributes are always defined:*

- *Each event belongs to exactly one case and each case has at least one event, denoted by the mandatory attribute* $case \in \mathcal{A}$*, i.e., for all events* $e \in E$*, a case* $c \in C$ *exists with* $\#_{case}(e) = \{c\}$*, and for all* $c \in C$ *an event* $e \in E$ *exists with* $\#_{case}(e) = \{c\}$*;*
- *Each event belongs to some activity, denoted by the mandatory attribute* $act \in \mathcal{A}$*, i.e., for all events* $e \in E$ *an activity* $a \in A$ *exists such that* $\#_{act}(e) = \{a\}$*;*
- *An event may record the time it was recorded using the timestamp attribute* $time \in \mathcal{A}$*, i.e., for all events* $e \in E$ *we have* $\#_{time}(e) = \{t\}$ *for some timestamp* $t \in T$*, where* T *resembles the set of timestamps.*

3 Intervals in Event Logs

There are many techniques for discovering a process model out of an event log. An extensive overview of available process discovery techniques can be found in [24]. Some examples are the alpha miner [4], the ILP miner [27] and the declarative miner [17]. In many discovery methods, events are considered to be instantaneous: they occur at a single point in time. However, in many information systems, such as electronic patient records, or financial statements, only a date is recorded. Consequently, even if events are considered to occur instantaneously, if they are observed within the same interval, the only conclusion to be drawn is that these occurred simultaneously.

The more coarse the chosen time scale (e.g., days, weeks or months), the more events will occur concurrently. Another consequence of a more coarse time scale is that events occur in some time window, rather than occurring at a single moment in time. It is important to note that there are some techniques which do not consider events to be instantaneous. That is, the authors of [15], exploit the fact that activities take time, i.e. each activity has a start and complete event. As a result, parallelism can be detected explicitly. Two activities are considered to occur in parallel if there is at least one case in which the activities overlap in time. In [20], the authors consider the execution of an activity as a time interval based on a starting and ending event. Parallelism is detected

Table 1. Example event log, time scale in days

date	events
7-1-2013	(1, A)
8-1-2013	(1, B), (1, E)
9-1-2013	(2, A), (1, G)
10-1-2013	(2, E), (2, C), (3, A)
11-1-2013	(2, G), (3, D)
14-1-2013	(3, G) (4, A)
15-1-2013	(4, F), (5, A), (6, A)
16-1-2013	(4, G), (5, D)
17-1-2013	(5, G), (6, F)
18-1-2013	(6, G)

by identifying two executions in which one activity occurs before the other one, and the other way around. The work described in [21] is comparable to [20], which presents a different control-flow discovery algorithm based on the notion of time intervals. All the aforementioned techniques only use one notion for determining intervals for activities and whether they overlap. In this paper, we study the case where activities occur in multiple intervals within the same execution.

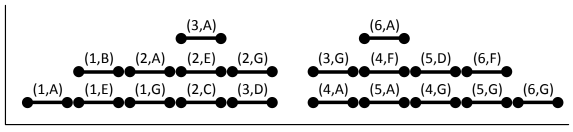
Consider the events presented in Tbl. 1 showing for each day the events that occurred. For each event, its case and activity are recorded. The time stamps of these events are in days, e.g., event $(1, B)$ occurred on January 1, 2013, as well as event $(1, E)$. Based on this information, we cannot infer any order between B and E, the only fact that can be inferred is that these events occurred simultaneously.

As the time scale is relatively coarse, a first analysis of this event log would be the degree of concurrency. We can build a graph that depicts the intervals on a time scale, as shown in Fig. 1(a). Based on this graph, we derive a *concurrency relation* $I \subseteq \mathcal{E} \times \mathcal{E}$, such that $a\, I\, b$ if and only if a and b occur within the same time interval. This results in a graph as depicted in Fig. 1(b), where the dashed and solid edges together represent the relation I. For readability, the self loops have been omitted. Note that (A, G) is an edge in the graph, while no case exists in which activities A and G occur simultaneously. Therefore, we can partition the relation I into two relations I_S and I_G such that $a\, I_S\, b$ if and only if $\#_{\text{case}}(a) = \#_{\text{case}}(b)$, and I_G analogously. In Fig. 1(b), the edges of relation I_S are solid, the edges of relation I_G are dashed. Similarly, the concurrency relation is not transitive with respect to the event log: even though (B, E) and (E, C) are edges in the graph, B, C and E never occur simultaneously in any case.

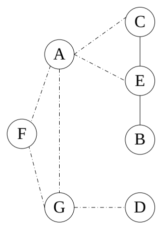
Whereas in Fig. 1(b) an absolute time window is taken, one could also choose to map each event to a relative interval, e.g. the respective day from the start of the day, as shown in Fig. 1(c). To allow such abstractions, we introduce the notion of an *event interval mapping function* that maps each event onto a time interval.

Definition 2 (Event interval mapping function). *Let $L = (C, E, \#)$ be an event log. A function $m_L : E \to T \times T$ is an* event interval mapping function *for L if it is an interval function. The* default interval mapping function *$D_L : E \to T \times T$ of L is defined by $D_L(e) = (\#_{time}(e), \#_{time}(e))$ for all $e \in E$.*

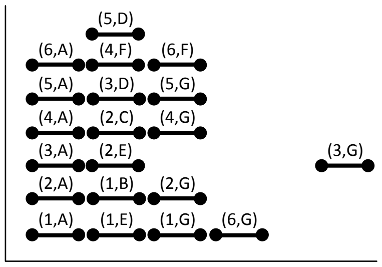
Based on the event interval mapping function, two notions of concurrency can be observed: one based on the whole event log, called the *concurrency relation*, and one based on the individual executions: the *simultaneousness relation*. Thus, the simultaneousness relation I for an event log L can be defined as the events that occur in the same interval defined by some interval mapping function.



(a) intervals



(b) concurrency graph



(c) Relative time, intervals

Fig. 1. Intervals of Tbl. 1

Definition 3 (Concurrency, simultaneousness relation). *Let L be an event log, and m a corresponding event interval mapping function. Its* concurrency relation $\bar{I}_m \subseteq E \times E$ *is defined by $a\,\bar{I}_m\,b$ iff $\pi_1(m(a)) \leq \pi_2(m(b))$ and $\pi_1(m(b)) \leq \pi_2(m(a))$ for $a, b \in E$. Its* simultaneousness relation $I_m \subseteq E \times E$ *is defined by $a\,I_m\,b$ iff both $a\,\bar{I}_m\,b$ and $\#_{case}(a) = \#_{case}(b)$ for $a, b \in E$.*

In the literature, the graph imposed by the concurrency relation is called the *interval graph* [11, 16]. Following [11], we can define an ordering relation $\succ$ that is defined by $a \succ b$ iff $\pi_2(m(a)) < \pi_1(m(b))$, stating that b "wholly occurs after" a. Relation $\succ$ is called an *interval order* [28, 29], as proven in [11].

Definition 4 (Interval order). *A binary relation R over some set S is an* interval order *if $a\,R\,b$ and $c\,R\,d$ imply $a\,R\,d$ or $c\,R\,b$ for all $a, b, c, d \in S$.*

Using intervals in concurrency is not new. For example, Janicki and Koutny [14] show that the notion of interval orders naturally follows from a basic assumption on concurrency: "the observer can state that one event preceded another event, or that two events occurred simultaneously". The authors show that for finite event logs, events can be interpreted as intervals on a discrete time scale. The authors introduce a model as a set of relations defining (weak) causalities, commutativity and synchronisation. An observation is called a *history* of a model if the relations induced by the observation coincide with the relations of the model.

In [6], Allen defines a set of assertions and properties based on time intervals: "before", "equal", "meets", "overlaps", "during", "starts" and "finishes". Based on these predicates, the authors introduce the assertion "occurs" with two variables: an event and an interval. This approach is often used in the area of artificial intelligence to reason over time using logic programming [7, 22].

4 Activities as Sets of Intervals

The interval mapping function on event logs introduced in the previous section induces an interval order on the events in the event log. In this way, approaches like in [9, 14] are directly applicable on this interval mapping function. These approaches mainly focus on a single run of a system: each event occurs exactly once. However, process mining mainly focuses on the analysis of the process implied by the activities for which the events in the event log occurred.

Different events for the same activity may indicate that the activity has been executed several times. Or, if an event represents the different stadia of some life cycle of activities, like a start and complete type, multiple events occur for the same activity. In [19] an approach is given for identifying pairs of events which denote the start and end of an activity. Thus, a single execution involves multiple occurrences of activities with some duration. Therefore, we search for new relations such that we can describe the relations on activity level, rather than on the level of events.

One way to lift the interval functions from events to activities is by defining a relation based on the interval order. Similar to the concurrency and simultaneousness relation, one would obtain two relations $\bar{R}$ and R such that

$$a \bar{R} b \Leftrightarrow \exists e_1, e_2 \in E : \#_{\text{act}}(e_1) = a \wedge \#_{\text{act}}(e_2) = b \wedge e_1 \succ e_2$$
$$a R b \Leftrightarrow \exists e_1, e_2 \in E : \#_{\text{case}}(e_1) = \#_{\text{case}}(e_2) \wedge \#_{\text{act}}(e_1) = a \wedge \#_{\text{act}}(e_2) = b \wedge e_1 \succ e_2$$

In fact, using the default event interval mapping of an event log, relation R coincides with the weak order relation of [25], which allows us to construct a relation set [26] based on intervals. In this paper, we will focus on behavioural relations based on the interval in which an activity is executed.

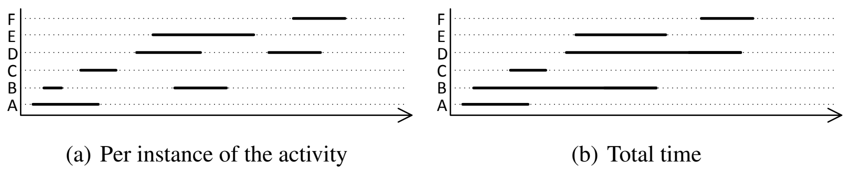
Although the above relation $\bar{R}$ is transitive, it abstracts away from the observed sequences in the event log. As activities may have multiple occurrence intervals, it is not an interval function. Therefore, we need to generalize the interval function to sets of intervals.

Definition 5 (Generalized interval function). *Given a set S and a, possibly infinite set $T \subseteq \mathbb{R}$, a function $f : S \to \mathcal{P}(T \times T)$ is called a* generalized interval function *if $x \leq y$ for all $(x, y) \in f(a)$ and $a \in S$.*

A generalized interval function can define a large set of small intervals, or a small set of large intervals. We call this the *granularity* of the interval function. Given any generalized interval function, we can define its most fine granular interval function, i.e., each point is its own interval, and the most coarse granular interval function, i.e., the conjunction of all intervals.

Table 2. Event log of a single case

Act.	Type	Time	Act.	Type	Time
A	start	1	B	start	9
B	start	2	D	complete	10
B	complete	3	B	complete	11
C	start	4	E	complete	12
A	complete	5	D	start	13
C	complete	6	F	start	14
D	start	7	D	complete	15
E	start	8	F	complete	16



(a) Per instance of the activity (b) Total time

Fig. 2. Possible occurrence intervals of Tbl. 2

Definition 6 (Finest and coarsest interval functions). *Let $f : S \to \mathcal{P}(T \times T)$ be a generalized interval function. Its* finest interval function*, denoted by $f\downarrow: S \to \mathcal{P}(T \times T)$, is defined by*

$$f\downarrow(s) = \{(t,t) \mid \exists (x,y) \in f(s) : x \leq t \leq y\}$$

The coarsest interval function *of f, denoted by $f\uparrow: S \to \mathcal{P}(T \times T)$, is defined by:*

$$f\uparrow(s) = \{\,(\min\{\pi_1(\,f(s)\,)\}, \max\{\pi_2(\,f(s)\,)\}\,)\,\}$$

Consider as an example the event log shown in Tbl. 2 representing the events of a single case. In this example, the time scale is defined as hours since the start of the execution. Many different ways exists to map these events to a generalized interval function on activities.

Two example mappings are given in Fig. 2. In the first example, the start and complete events of each activity are used to define the different intervals, whereas in the second example the very first start event of the activity defines the begin of the interval, and the very last complete event of the activity the end of the interval. Observe that in Fig. 2(a) activities B and C have no overlap, whereas in Fig. 2(a) these activities do have overlap.

In general, an event log records many different executions. Therefore, we map each execution to its own activity interval function. This results in an activity interval mapping for an event log.

As each event belongs to a single activity, we require that an activity interval mapping defines a unique interval for each event in the event log. On the other hand, as an activity may be represented by multiple occurrences, multiple events may be related to the same activity interval.

Definition 7 (Activity interval mapping). *Let $L = (C, E, \#)$ be an event log with corresponding event interval mapping m, let A be a set of activities of L, and let $T \subseteq \mathbb{R}$ be the time scale. The function $G : C \times A \to \mathcal{P}(T \times T)$ is called an* activity interval mapping *iff*

– *each event has a unique corresponding interval, i.e.,*

$$\begin{aligned}\forall e \in E \ : \ & (\ \exists I \in G(\#_{case}(e), \#_{act}(e)) : m(e) \subseteq I\ ) \\ & \wedge\ (\ \forall I, J \in G(\#_{case}(e), \#_{act}(e)) : (m(e) \subseteq I \wedge m(e) \subseteq J) \implies I = J\ )\end{aligned}$$

– *each interval has at least one event occurrence, i.e.,*

$$\forall a \in A, c \in C, I \in G(c, x) : \exists e \in E : \#_{case}(e) = c \wedge \#_{act}(e) = a \wedge m(e) \subseteq I$$

The default activity interval mapping *of an event log L, denoted by $\bar{L} : C \times A \to T \times T$, is defined by:*

$$\bar{L}(c, a) = \{m(e) \mid \exists e \in E : \#_{case}(e) = \ c \wedge \#_{act}(e) = a\}$$

Many different interval functions can be defined for an event log. As the next corollary shows, such activity interval mappings are related, as intervals may be combined into larger intervals, or split into several smaller intervals. It is simple to see that given some activity interval mapping, its coarsest interval function is also an activity interval mapping. Further, the finest interval function of the minimal activity interval mapping is contained in the finest interval function of the activity interval mapping.

Corollary 8. *Given an event log L with corresponding event interval mapping m and activity interval function G. Let A be the set of activities in L. Then (1) $G\!\uparrow$ is an activity interval mapping, (2) $\bar{L}\!\downarrow \subseteq G\!\downarrow$, and (3) $\pi_1(G\!\uparrow(a)) \leq \pi_1(\hat{L}\!\uparrow(a))$ and $\pi_2(G\!\uparrow(a)) \geq \pi_2(\bar{L}\!\uparrow(a))$ for all activities $a \in A$.*

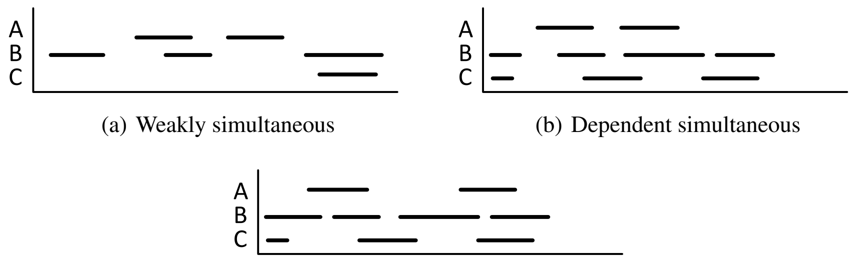
5 Relations on Interval Sets

In general, a generalized interval function does not define any interval order. Consequently, approaches like in [6, 14] cannot be used to determine causality and similarity relations. In this section, we derive such notions based on the generalized interval function.

5.1 Notions of Simultaneousness

In an interval order two intervals are unrelated if one does not wholly occur after the other, and vice versa. With sets of intervals, different degrees of simultaneousness can be defined.

The weakest form of simultaneousness is when two elements have some overlapping intervals. For example, in the intervals shown in Fig. 3(a), activities A and B have some intervals that overlap. Note that the relation is not transitive, as shown in the same figure. We say an element s is *dependent simultaneous* with some other element t if for every interval of s, an overlapping interval of t exists. Thus, everytime s is started, t will be started as well, whereas if t occurs, s does not have to occur. If s always overlaps with t, we say they are *strongly dependent*.



(a) Weakly simultaneous

(b) Dependent simultaneous

(c) Strongly simultaneous

Fig. 3. Simultaneousness relations

Definition 9 (Simultaneousness). *Let f be a generalized interval function over some set S. Let $s, t \in S$. Then:*

- *s and t are* weakly simultaneous*, denoted by $s \leftrightarrow t$, if s and t share some interval, i.e., $\exists I \in f(s), J \in f(t) : I \cap J \neq \emptyset$;*
- *s is* dependently simultaneous with *t, denoted by $s \rightrightarrows t$, if always if s occurs, then t occurs in the same interval, i.e., $\forall I \in f(s) : \exists J \in f(t) : I \cap J \neq \emptyset$;*
- *s and t are* strongly simultaneous*, denoted by $s \rightleftarrows t$, if s and t always overlap, i.e., $s \rightleftarrows t$ if and only if $s \rightrightarrows t$ and $t \rightrightarrows s$.*

Consider again Fig. 3. In Fig. 3(a) we have $A \leftrightarrow B$ and $B \leftrightarrow C$ but not $A \leftrightarrow C$, in Fig. 3(b) we have $A \rightrightarrows B$ as every interval of A overlaps with some interval of B, $B \rightrightarrows C$ as each interval of B overlaps with some interval of C and $A \leftrightarrow C$ but not $A \rightrightarrows C$ as not every interval of A overlaps with an interval of C. Last, in Fig. 3(c) we have $A \rightleftarrows B$, $B \rightleftarrows C$ and $A \rightrightarrows C$ but not $A \rightleftarrows C$, as every interval of A overlaps some interval of C but not vice versa.

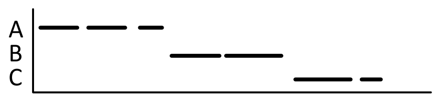
Based on their definitions, it is trivial to see that strong simultaneousness implies dependent simultaneousness which in turn implies weak simultaneousness.

Corollary 10. *Let f be a generalized interval function over some set S, and let $s, t \in S$. Then (1) $s \rightrightarrows t \wedge f(s) \neq \emptyset \implies s \leftrightarrow t$, and (2) $\rightleftarrows$ and $\leftrightarrow$ are symmetric and reflexive.*

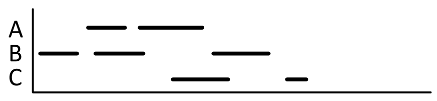
As shown in Fig. 3, none of these relations is transitive. Consequently, we cannot obtain equivalence classes based on the intervals. As the relation $\rightleftarrows$ is symmetric and reflexive, it can be used as a *dependence relation* over the set of activities, which allows us to use Mazurkiewicz trace theory [10] for e.g. synthesis and to check completeness of event logs.

5.2 Notions of Causality

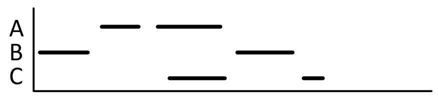
Fishburn showed in [11], that given an interval function f, any order $>$ with $x > y$ iff $\pi_2(f(x)) < \pi_1(f(y))$, i.e., that the interval of x is wholly after the interval of y, is an interval order. Similarly, the $>$ relation in relation sets [4,26] states that if $a > b$ but not



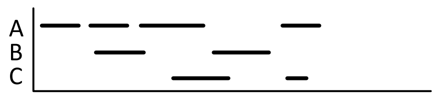
(a) Wholly succeeded by



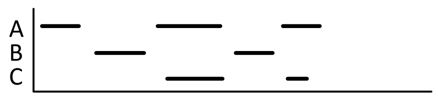
(b) Succeeded by



(c) Strictly succeeded by



(d) Preceeded by



(e) Strictly preceeded by

Fig. 4. Different causal relations based on the intervals

$b > a$, then a and b are causally ordered, i.e., a is followed by b, but b never followed by a. In terms of intervals, similar relations can be defined. Again, as an activity possibly has multiple intervals, we need to adapt the notion of causality to sets of intervals.

The first causality relation we introduce is if all intervals of some activity t occur after the intervals of s occurred, i.e., s is wholly succeeded by t. An example is depicted in Fig. 4(a), in which A is wholly succeeded by B and B is wholly succeeded by C. If for each interval of s some interval of t can be found that wholly succeeds the interval of s, we say that s is succeeded by t. In Fig. 4(b), A is always succeeded by B, and B is always succeeded by C. Note that this allows intervals of t to occur simultaneously with intervals of s, or even occurring before s, as shown in Fig. 4(b) where B occurs before A. If s is succeeded by t and they have no overlapping intervals, we say that s is *strictly* succeeded by t. An example is shown in Fig. 4(c), where A is strictly succeeded by B, and B strictly succeeded by C. Note that whereas the succeeded relation is transitive, the strictly succeeded is not, as A and C have overlap.

Symmetrically, if for each interval of t an interval of s can be found that wholly preceeds the interval of t, we say that t is preceeded by s. This allows intervals of s to occur after intervals of t, or even simultaneously, as shown in Fig. 4(d) where B is preceeded by A, and C by B. The relation is called *strict*, if s and t are not simultaneously. Again, as shown in Fig. 4(e), the strictly preceeded relation is not transitive, as B is strictly preceeded by A, and C by B, but A and C have overlap. This leads to the following notions of causality.

Definition 11 (Causality). *Let f be a generalized interval function over some set S. Let $s, t \in S$. Then:*

– *s is* wholly succeeded by *t, denoted by $s \blacktriangleright t$, if all intervals of t are after the intervals of s, i.e., $\pi_2(f\uparrow(s)) < \pi_1(f\uparrow(t))$;*

– *s is* succeeded by *t, denoted by $s \trianglerighteq t$, if each interval of s is followed by an interval of t, i.e., $\forall(a,b) \in f(s) : \exists(c,d) \in f(t) : b < c$;*
– *s is* strictly succeeded by *t, denoted by $s \triangleright t$, if $s \trianglerighteq t$ and not $s \leftrightarrow t$;*
– *t is* preceeded by *s, denoted by $s \sqsupseteq t$, if each interval of t is preceeded by an interval of s, i.e. $\forall(c,d) \in f(t) : \exists(a,b) \in f(s) : b < c$;*
– *t is* strictly preceeded by *s, denoted by $s \sqsupset t$, if $s \sqsupseteq t$ and not $s \leftrightarrow t$.*

It is easy to see that the wholly succeeded relation is a strict order. Similarly, the followed by and preceeded by relations are transitive. However, these relations are not irreflexive in general. Only if the set of intervals for some activity is finite, the relations are irreflexive as well, and thus a strict order. If an activity has an infinite set of intervals, then it is succeeded by itself.

If some activity is wholly succeeded by some other activity, then it is easy to show that the former activity is strictly succeeded by the latter, and the latter is strictly preceeded by the former.

Corollary 12. *Let f be a generalized interval function over some set S. Then (1) $\blacktriangleright$ is a strict order, (2) $\trianglerighteq$, and $\sqsupseteq$ are transitive, and (3) $x \blacktriangleright y \implies x \triangleright y \wedge x \sqsupset y$ for all $x, y \in S$.*

Further, the strictly succeeded by and strictly preceeded by relations are subsets of the succeeded by and preceeded by relations, respectively.

Corollary 13. *Let f be a generalized interval function over some set S. Then (1) $\triangleright \subseteq \trianglerighteq$, and (2) $\sqsupset \subseteq \sqsupseteq$.*

As for the interval order $>$ defined on events, the wholly succeeded by relation on activities is an interval order, which follows directly from the definitions.

Lemma 14 (Wholly succeeded is an interval order). *Let f be a generalized interval function over some set S. Then $\blacktriangleright$ is an interval order.*

Proof. Let $a, b, c, d \in S$ such that $a \blacktriangleright b$, and $c \blacktriangleright d$. We need to show that either $a \blacktriangleright d$ or $c \blacktriangleright b$ holds.

Suppose $a \blacktriangleright d$ does not hold, i.e., $\pi_2(f\uparrow(a)) \geq \pi_1(f\uparrow(d))$. Then $\pi_2(f\uparrow(c)) < \pi_1(f\uparrow(d)) \leq \pi_2(f\uparrow(a)) < \pi_1(f\uparrow(b))$. Hence, $c \blacktriangleright b$.

Similarly, suppose $c \blacktriangleright b$ does not hold, i.e., $\pi_2(f\uparrow(c)) \geq \pi_1(f\uparrow(b))$. Then $\pi_2(f\uparrow(a)) < \pi_1(f\uparrow(b)) \leq \pi_2(f\uparrow(c)) < \pi_1(f\uparrow(d))$. Hence, $a \blacktriangleright d$. □

5.3 Other Control-Flow Relations

The simultaneousness and causality relations form the basic building blocks of any process modelling language. Many other control-flow relations can be defined, depending on the needs within the process modelling notation. For example, one can define a *next-to* relation on activities, defining whether two activities are directly after one another, without any activitiy in between. As for simultaneousness, this can be a weak relation, i.e., for two activities there are intervals next to each other, or a strong relation, i.e., for all intervals.

Definition 15 (Next-to relation). *Let f be a generalized interval function over some set S. Let $s, t \in S$. We say s is* next to *t, denoted by $s \circ t$, if some interval of s is directly followed by an interval of t, without any occurrence of other activities in between, i.e.,*
$\exists (k,l) \in f(s), (o,p) \in f(t) : (\ l < o \wedge \neg(\exists u \in S\ : (m,n) \in f(u) : l < n \wedge m < o)\ )$

Similarly, s is followed by *t, denoted by $s \bullet t$, if all intervals of s are directly followed by an interval of t, without any occurrence of other activities in between, i.e.,*
$\forall (k,l) \in f(s) : (\ \exists (o,p) \in f(t) : l < o \wedge \neg(\exists u \in S\ : (m,n) \in f(u) : l < n \wedge m < o)\ )$

Naturally, if s is followed by t, then s is also succeeded by t.

Corollary 16 (Follows implies succeeded). *Let f be a generalized interval function over some set S, and let $s, t \in S$. Then if $s \bullet t$ then also $s \trianglerighteq t$.*

As activities are represented by sets of intervals, an activity can be enclosed by some other activity, i.e., some activity B always occurs between two intervals of A. We call this relation *betweenness*. Again, this can be a strong notion, requiring this for all intervals of B, or a weak notion, only requiring the existence of such an interval of B.

Definition 17 (Betweenness). *Let f be a generalized interval function over some set S. Let $s, t \in S$. We say t is* weakly in between *s, denoted by $s \hookrightarrow t$, if some interval of t is in between two intervals of s, i.e., $\exists (m,n) \in f(t), (k,l), (o,p) \in f(s) : l < m \wedge n < o$.*

Similarly, we say t is in between *s, denoted by $s \circlearrowleft t$, if all intervals of t are between two intervals of s, i.e., $\forall (m,n) \in f(t) : (\exists (k,l), (o,p) \in f(s) : l < m \wedge n < o)$.*

Altough betweenness seems a natural choice, it can be expressed in terms of the basic causality notions defined in Def. 11.

Corollary 18 (Betweenness implies basic causality). *Let f be a generalized interval function over some set S, and let $s, t \in S$. If $s \circlearrowleft t$ then $s \sqsupseteq t$ and $t \trianglerighteq s$.*

6 Discovering Declarative Models

The density of the time scale has a great impact on the level of concurrency in an event log, and hence in the model that describes the allowed behaviour of the executions in the event log. Procedural languages prescribe the order in which activities are supposed to occur. Consequently, concurrency needs to be modelled explicitly in such languages. Instead, we use a declarative approach that has concurrency as a language primitive: activities may occur simultaneous, unless constraints prohibit the execution of the activity.

6.1 Declare Language

In this paper, we use the declarative language Declare [2]. The language provides a graphical layout to visualize the activities and constraints in the model. It does not come with a predefined set of language constructs. Instead it offers a set of language constructs called *constraint templates*, which the user may adapt to its own needs. These constraint templates are based on Linear Time Logic (LTL) [8]. Declare comes with a

Table 3. Basic language constructs in Declare

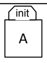
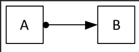
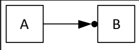
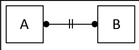
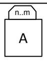
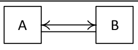
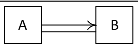
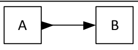
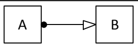
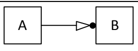
Constraint	Template	Graphically
init	$\sigma(1) = A$	
response	$\Box(A \implies \Diamond B)$	
precedence	$((\neg B)\ U\ A) \vee \Box\neg B$	
non coexistence	$\neg((\Diamond A) \wedge (\Diamond B))$	
$(n..m)$ occurrences	$\lvert\{i \mid \sigma(i) = A\}\rvert \in [n..m] \subseteq \mathbb{N}$	

Table 4. Newly introduced constraints in Declare

Constraint	Template	Graphically
strongly simultaneous	$A \rightleftarrows B$	
Dependently simultaneous	$A \rightrightarrows B$	
wholly succeeded	$A \blacktriangleright B$	
strict response	$A \rhd B$	
strict precedence	$A \sqsupset B$	

basic set of language constructs. Tbl. 3 depicts the language constructs from Declare used in this paper.

The first constraint template, init, states that the first activity of any sequence, represented by σ, should start with A, where A is a placeholder for the actual activity. Similarly, the response template states that every A should eventually be followed by some activity B. The precedence constraint template expresses that some activity B has to be preceeded by some activity A. With the non coexistence template, it is possible to express that two activities should not occur together in any sequence. Last, we allow to limit the number of times an activity can be executed using the $n..m$ occurrences template, where $n \leq m$ specifies the minimal and maximal number of times some activity A is executed.

6.2 Interval-Based Constraints

The constraints in Declare do not take activity duration into account. Consequently, we need to relate the constructs used in Declare with the simultaneousness and causality relations defined in the previous section.

First, consider the response constraint template. This constraint expresses that activity A is always eventually followed by B. This can be interpreted in many different ways, e.g., “once activity A is *started*, activity B will eventually start”, or “once activity A is *finished*, activity B will eventually start”. We choose the latter interpretation, i.e., after activity A finished, eventually activity B will start. A second consideration is whether the response and precedence templates should allow the activities in the constraint to occur simultaneously. As the response template is transitive in the Declare language, we allow the activities to overlap. Thus, we interpret the response template with the succeeded by relation introduced in the previous section. Similarly, we interpret the preceeds template as “before activity B starts, activity A should be finished”, which coincides with the preceeds by relation introduced in the previous section. Therefore, the strictly succeeded by and strictly preceeded by relations are added to the Declare language, as shown in Tbl. 4.

Although in Declare concurrency is a language primitive, each activity in the model is considered to be instantaneous. The language does not provide any constraint that limits concurrency without destroying it. Thus, the weak simultaneousness relation as presented in the previous section is directly supported in the language. The two stronger simultaneousness relations impose an order on the activities: although the activities may overlap, the other activity must be executed simultaneously. This is expressed by the strongly simultaneous template and dependent simultaneous template as depicted in Tbl. 4.

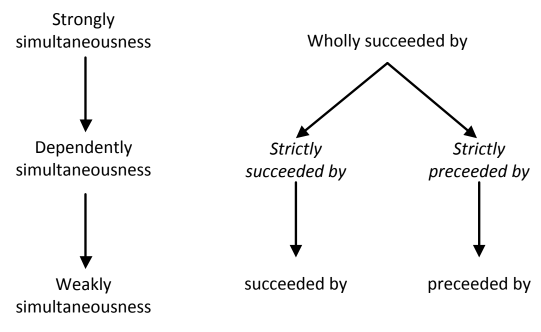
6.3 Discovery

In the previous section, we introduced several notions of simultaneousness and causality. Up to now, these relations only consider a single execution of the system. An event log contains a set of executions that are executed by, most likely, the same process. Hence, to come to a model that describes each of the executions in the event logs, we need to aggregate the relations over the different executions in the event log.

In what follows, we sketch a declarative discovery algorithm based on time intervals. Here, it is important to mention that the choice of the generalized interval functions for the activities breaks or makes the approach presented in this paper.

Events to interval First step in the approach is to map each event to an interval. In many cases the default event interval mapping, i.e., that maps each event to a single-point interval, can be used. In some cases, for example if event logs of multiple systems are combined, a reliability interval can be attached to each interval.

Activities to sets of intervals Next step is the construction or discovery of an accurate activity interval mapping. For example, one can fix the granularity of the time scale, make it relative or absolute, and then map each event to the corresponding time interval. Or, one can use the event types to determine the life cycle of an activity, and base the activity interval mapping on this information. Although at first sight this seems to be a trivial step, there are many pitfalls [12]. For example, if two instances of the same activity run simultaneously, which interval should be used to map the activity on?



(a) Simultaneousness (b) Causality

Fig. 5. Hierarchy of simultaneousness and causality relations

Derive relations Once the activity interval mapping has been established, we can start to derive the different relations. For example, the $(n..m)$ occurrences template can be easily constructed by analyzing the number of occurrences in each of the sequences in the event log. Similarly, the non-coexistence relation can be calculated by a single walk through the sequences of the event log.

Next step would be to derive the different simultaneousness relations and causality relations. For this, we use the relation hierarchy as depicted in Fig. 5, which follows directly from Cor. 10 and Cor. 12. An arrow from one relation to another relation means that the former is included in the latter. For example, the strongly simultaneousness relation is included in the dependently simultaneousness relation. The algorithm starts with assuming the strongest relation between each of the activities. By going through the different intervals, relations are weakened, until all intervals of all sequences in the event log have been inspected.

The algorithms to derive the simultaneousness and causality relations do not take transitivity into account, which results in models with many constraints, expressing the complete transitive closure of the respective relations. Therefore, these relations need to be reduced, such that the transitive closure of this reduced relations remain the same. For this, standard algorithms as described in [5] can be used. Although at first sight this seems a straightforward task, it is not, as one wants to take the hierarchy of relations into account during the reduction, which is closely related to the "minimum equivalent graph" problem [18], which is NP-hard.

Last, we sugar the models using nesting, as has been done in e.g. Dynamic Condition Response Graphs [13]. In this approach, set of nodes having the same constraints are nested in a so called "super nodes".

For the example event log of Tbl. 1, the discovery algorithm that has been sketched above results in the model depicted in Fig. 6. For the activities B, C, and E, we briefly illustrate the steps of the algorithm. In the first step, we take a relative time scale for the activity interval mapping. Moreover, a "start" event denotes the start time of an activity and a "complete" event denotes the end time of an activity. Secondly, all three

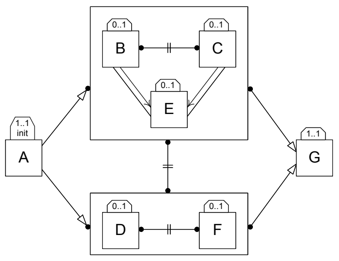


Fig. 6. Model discovered from the event log in Tbl. 1

activities occur at most once in each of the sequences of the event log resulting into a 0..1 occurrence relation for each of them. Also, activity *A* and *B* never occur together in each sequence. In the third step, it is discovered that activities *B* and *C* are strongly simultaneous. Also, the tree activities are all preceeded by activity *A* and succeeded by activity *G*. Finally, in the last step, activities *B*, *E* and *C* are nested, as these activities are all preceeded by *A*, do not coexist with *D* and *F*, and are all succeeded by *G*.

7 Conclusions

Timestamps in an event log play an essential role in process mining to determine the order in which events occur. A typical problem in process mining is the impreciseness of these timestamps. In this paper, we overcome this problem by assuming that each event occurs in some time window, i.e., in some interval. As the intervals in an event log are on the level of events, rather than on the level of activities, we have presented an approach based on sets of intervals to represent the occurrences of the activities in the model. On these sets of intervals new notions of simultaneousness and causalities are derived. These notions form the basis to discover declarative models.

The simultaneousness relation forms a natural candidate for the dependency relation in Mazurkiewicz traces. In this way, simultaneousness can be used to test the completeness of event logs, by exploring the Mazurkiewicz equivalent traces.

Although intervals are a natural choice to overcome the impreciseness of timestamps, choosing the right time window is a hard problem. The events and activities can be mapped to intervals in many different ways. The granularity of the time scale, like milliseconds, hours or days, can be used to define the intervals, the time scale can be relative or absolute. Or, if the event log contains a transition life cycle, like a start and complete event, then the first and last event of each execution can be used to determine the intervals in the activity interval mapping. Empirical research is needed to test, validate and compare the different alternatives.

As the proof of the pudding is in the eating, we will implement the presented approach in ProM [23] to perform more case studies to test and fine tune the resulting declarative models.

Acknowledgements The authors would like to thank Jetty Klein for the fruitful discussions about paradigms of concurrency and interval orders.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.
2. W.M.P. van der Aalst, M. Pesic, and M.H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113, 2009.
3. W.M.P. van der Aalst and C. Stahl. *Modeling Business Processes -Ű A Petri Net-Oriented Approach*. The MIT Press, 2011.
4. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *Knowledge & Data Engineering*, 16(9):1128–1142, 2004.
5. A.V. Aho, M. R. Garey, and J. D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, June 1972.
6. J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, July 1984.
7. J.F. Allen. Actions and Events in Interval Temporal Logic 1 Introduction. *Journal of Logic and Computation*, 4:531–579, 1994.
8. E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, Berlin, 1982.
9. P. Degano and U. Montanari. Concurrent Histories: A Basis for Observing Distributed Systems. *Journal of Computer and System Sciences*, 34(2-3):422–461, 1987.
10. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
11. P.C Fishburn. Interval graphs and interval orders. *Discrete Mathematics*, 55(2):135–149, July 1985.
12. T. Gschwandtner, J. Gärtner, W. Aigner, and S. Miksch. A Taxonomy of Dirty Time-Oriented Data. In G. Quirchmayr, J. Basl, I. You, and E. Weippl, editors, *CD-ARES 2012*, volume 7465 of *LNCS*, pages 58–72. Springer, 2012.
13. T. Hildebrandt, R. Mukkamala, and T. Slaats. Nested dynamic condition response graphs. *Fundamentals of Software Engineering*, pages 343–350, 2012.
14. R. Janicki and M. Koutny. Structure of concurrency. *Theoretical Computer Science*, 112(1):5–52, April 1993.
15. Wen. L., J. Wang, W.M.P. van der Aalst, B. Huang, and J. Sun. A Novel Approach for Process Mining based on Event Types. *Journal of Intelligent Information Systems*, 32:163–190, 2009.
16. R.D. Luce. Semiorders and a Theory of Utility Discrimination. *Econometrica*, 24(2):178–191, 1956.
17. F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *CAiSE*, volume 7328 of *LNCS*, pages 270–285. Springer-Verlag, Berlin, 2012.
18. D.M. Moyles and G.L. Thompson. An algorithm for finding the minimum equivalent graph of a digraph. *Journal of the ACM*, pages 455 – 460, 1969.
19. J. Nakatumba and W.M.P. van der Aalst. Analyzing Resource Behavior Using Process Mining. In S. et al. Rinderle-Ma, editor, *BPM 2009 Workshops*, volume 43 of *LNBIP*, pages 69–80. Springer, 2009.

20. S.S Pinter and M. Golani. Discovering Workflow Models from Activities' Lifespans. *Computers in Industry*, 53:283–296, 2004.
21. Y.-L. Qu and T.-S. Zhao. Building Process Models Based on Inverval Logs. In M. Ma, editor, *Communication Systems and Information Technology*, volume 100 of *LNEE*, pages 71–78. Springer, 2011.
22. G. Rosu and S. Bensalem. Allen Linear (Interval) Temporal Logic – Translation to LTL and Monitor. In *Computer Aided Verification*, pages 263–277. Springer, 2006.
23. H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In *Information System Evolution*, volume 72, pages 60–75. Springer, 2011.
24. J. de Weerdt, M. de Backer, J. Vanthienen, and B. Baesens. A Multi-dimensional Quality Assessment of State-of-the-Art Process Discovery Algorithms using Real-Life Event Logs. *Information Systems*, 37:654–676, 2012.
25. M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410 – 429, 2011.
26. M. Weidlich and J.M.E.M. van der Werf. On Profiles and Footprints – Relational Semantics for Petri Nets. In *Application and Theory of Petri Nets*, LNCS, pages 148–167. Springer, 2012.
27. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. *Fundamenta Informatica*, 94(3 – 4):387 – 412, 2009.
28. N. Wiener. A contribution to the theory of relative position. *Proceedings of the Cambridge Philosophical Society*, 17:441–449, 1914.
29. N. Wiener. A new theory of measurement: A study in the logic of mathematics. *Proceedings of the London Mathematical Soc.*, s2-19(1):181–205, 1921.