A Novel Approach for Process Mining Based on Event Types

Lijie Wen[1,2], Jianmin Wang[2], Wil M.P. van der Aalst[3], Zhe Wang[2], and Jiaguang Sun[1,2]

[1] Department of Computer Science & Technology,Tsinghua University,Beijing,China
`wenlj00@mails.tsinghua.edu.cn`
[2] School of Software, Tsinghua University, Beijing, China
`{jimwang, sunjg}@tsinghua.edu.cn, wang02@mails.tsinghua.edu.cn`
[3] Department of Technology Management, Eindhoven University of Technology, Eindhoven, The Netherlands.
`w.m.p.v.d.aalst@tm.tue.nl`

Abstract. Despite the omnipresence of event logs in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), historic information is rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs, i.e., the basic idea of process mining is to diagnose business processes by mining event logs for knowledge. Given its potential and challenges it is no surprise that recently process mining has become a vivid research area [5, 6]. In this paper, a novel approach for process mining based on two event types, i.e., START and COMPLETE, is proposed. Information about the start and completion of tasks can be used to explicitly detect parallelism. The algorithm presented in this paper overcomes some of the limitations of existing algorithms such as the α-algorithm (e.g., short-loops) and therefore enhances the applicability of process mining.

1 Introduction

During the last decade workflow management technology [3] has become readily available. Workflow management systems such as Staffware, IBM MQSeries, COSA, etc. offer generic modeling and enactment capabilities for structured business processes. By making process definitions, i.e., models describing the life-cycle of a typical case (workflow instance) in isolation, one can configure these systems to support business processes. These process definitions need to be executable and are typically graphical, e.g., in terms of Petri nets. Besides pure workflow management systems many other software systems have adopted workflow technology. Consider for example ERP (Enterprise Resource Planning) systems such as SAP, PeopleSoft, Baan and Oracle, CRM (Customer Relationship Management) software, SCM (Supply Chain Management) systems, B2B

(Business to Business) applications, etc. which embed workflow technology. Despite its promise, many problems are encountered when applying workflow technology. One of the problems is that these systems require a workflow design, i.e., a designer has to construct a detailed model accurately describing the routing of work. Modeling a workflow is far from trivial: It requires deep knowledge of the business process at hand (i.e., lengthy discussions with the workers and management are needed) and the workflow language being used.

In this paper, we do not focus on the design but instead we focus on techniques for *monitoring* enterprise information systems (i.e., WFM, ERP, CRM, SCM-like systems). Today, many enterprise information systems store relevant events in some structured form. For example, workflow management systems typically register the start and completion of activities [3]. ERP systems like SAP log all transactions, e.g., users filling out forms, changing documents, etc. Business-to-business (B2B) systems log the exchange of messages with other parties. Call center packages but also general-purpose CRM systems log interactions with customers. These examples show that many systems have some kind of *event log* often referred to as “history”, “audit trail”, “transaction log”, etc. [5, 8, 18, 39]. The event log typically contains information about events referring to an *task* and a *case*. The case (also named process instance) is the “thing” which is being handled, e.g., a customer order, a job application, an insurance claim, a building permit, etc. The task (also named activity, operation, action, or work-item) is some operation on the case. Typically, events have a *timestamp* indicating the time of occurrence. Moreover, when people are involved, event logs will typically contain information on the person executing or initiating the event, i.e., the *originator*. Based on this information several tools and techniques for process mining have been developed [2, 4, 5, 7, 8, 10, 19, 25, 35, 39, 50].

Process mining is useful for at least two reasons. First of all, it could be used as a tool to find out how people and/or procedures really work. Second, process mining could be used for *Delta analysis*, i.e., comparing the actual process with some predefined process (i.e., a descriptive or prescriptive process model).

In this paper, we present a new algorithm for process mining. This algorithm generates a Petri net based on some event log where both the start and completion of some event are logged. To illustrate the algorithm and its distinguishing features we use the event log shown in Table 1. The event log contains the audit trail of three cases. The first event is the start of task T1 for case 1. The second event is the completion of this task. The third event is the start of task T2 for case 1. The fourth event is the start of task T3 for case 1. Note that for case 1 the execution of T2 and T3 overlap. This suggests that T2 and T3 are in parallel. After the completion of T3 and T2 for case 1, the first event for case 2 is registered in the log. In total there are 36 events in the event log shown in Table 1: 18 events of type START and 18 events of type COMPLETE.

Using the algorithm presented in this paper, the log shown Table 1 can be used to generate the process model shown in Figure 1. This process model is expressed in terms of a Petri net. It is easy to see that the three cases can indeed

Case id	Task name	Event type	Case id	Task name	Event type	Case id	Task name	Event type
1	T1	START	1	T6	START	2	T5	START
1	T1	COMPLETE	1	T6	COMPLETE	2	T5	COMPLETE
1	T2	START	3	T2	START	2	T6	START
1	T3	START	3	T2	COMPLETE	2	T6	COMPLETE
1	T3	COMPLETE	2	T3	START	3	T4	START
1	T2	COMPLETE	2	T2	START	3	T4	COMPLETE
2	T1	START	2	T3	COMPLETE	3	T5	START
1	T4	START	2	T2	COMPLETE	3	T5	COMPLETE
2	T1	COMPLETE	2	T4	START	3	T5	START
1	T4	COMPLETE	2	T4	COMPLETE	3	T5	COMPLETE
3	T1	START	3	T3	START	3	T6	START
3	T1	COMPLETE	3	T3	COMPLETE	3	T6	COMPLETE

Table 1. An event log with START and COMPLETE events.

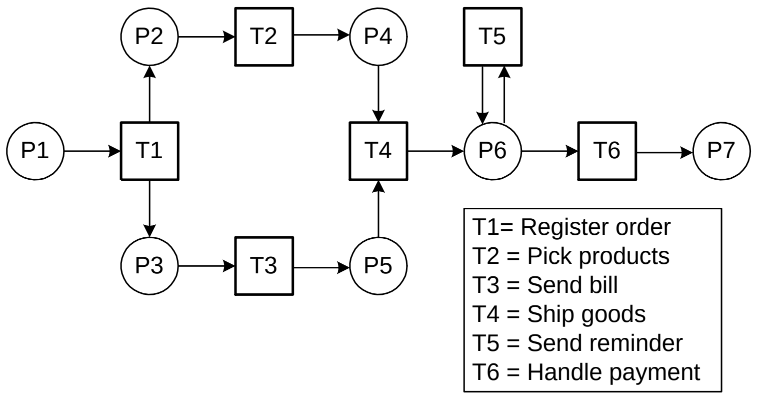

Figure 1. The Petri net corresponding to the event log shown in Table 1.

be handled by the Petri net. In Table 1 only task identifiers (T1, T2, etc.) are used. Figure 1 also shows the mapping of these identifiers onto task names.

Existing techniques for process mining do not consider event types, i.e., tasks are either considered to be atomic or only the completion of a task is considered (i.e., just event type COMPLETE) [2, 5, 7, 8, 10, 19, 50]. Note that the start and completion of a task can be considered as two atomic tasks when using the classical process mining techniques. Unfortunately, such an approach does not detect explicit parallelism. Moreover, the knowledge that the START and COMPLETE events are related is not exploited. As far as we know, the algorithm presented in this paper is the only algorithm explicitly detecting parallelism. It can be seen as a variant of the α-algorithm [7]. However, the causal relations and completeness notion are fundamentally different. Moreover, the new algorithm overcomes some of the problems of the basic α-algorithm, e.g., it is possible to correctly mine short loops. Note that Figure 1 contains a short loop, i.e., the construct involving T5 and P6 (sending 0, 1, or more reminders). This indicates that the basic α-algorithm is unable to correctly mine the process while the algorithm presented in this paper does.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces some preliminaries. In Section 4 a method for discovering characteristic relations between tasks is given. Based on these relations, in Section 5, a concrete algorithm for constructing process model is proposed. An experimental evaluation is outlined in Section 6. Finally, a conclusion is drawn in Section 7.

2 Related Work

The idea of process mining is not new [5, 7, 8, 10–12, 19–24, 29–31, 40–44, 47–49]. Cook and Wolf have investigated similar issues in the context of software engineering processes. In [10] they describe three methods for process discovery: one using neural networks, one using a purely algorithmic approach, and one Markovian approach. The authors consider the latter two the most promising approaches. The purely algorithmic approach builds a finite state machine where states are fused if their futures (in terms of possible behavior in the next k steps) are identical. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. Note that the results presented in [10] are limited to sequential behavior. Related, but in a different domain, is the work presented in [27, 28] also using a Markovian approach restricted to sequential processes. Cook and Wolf extend their work to concurrent processes in [11]. They propose specific metrics (entropy, event type counts, periodicity, and causality) and use these metrics to discover models out of event streams. However, they do not provide an approach to generate explicit process models. In [12] Cook and Wolf provide a measure to quantify discrepancies between a process model and the actual behavior as registered using event-based data. The idea of applying process mining in the context of workflow management was first introduced in [8]. This work is based on workflow graphs, which are inspired by workflow products such as IBM MQSeries workflow (formerly known as Flowmark) and InConcert. In this paper, two problems are defined. The first problem is to find a workflow graph generating events appearing in a given workflow log. The second problem is to find the definitions of edge conditions. A concrete algorithm is given for tackling the first problem. The approach is quite different from other approaches: Because the nature of workflow graphs there is no need to identify the nature (AND or OR) of joins and splits. As shown in [26], workflow graphs use true and false tokens which do not allow for cyclic graphs. Nevertheless, [8] partially deals with iteration by enumerating all occurrences of a given task and then folding the graph. However, the resulting conformal graph is not a complete model. In [31], a tool based on these algorithms is presented. Schimm [40, 41, 44] has developed a mining tool suitable for discovering hierarchically structured workflow processes. This requires all splits and joins to be balanced. Herbst and Karagiannis also address the issue of process mining in the context of workflow management [21, 19, 20, 23, 24, 22] using an inductive approach. The work presented in [22, 24] is limited to sequential models. The approach described in [21, 19, 20, 23] also allows for concurrency. It uses stochastic task graphs as an intermediate representation and it generates a workflow model described in the ADONIS modeling language. In the induction step task nodes are merged and split in order to discover the underlying process. A notable difference with other approaches is that the same task can appear multiple times in the workflow model, i.e., the approach allows for duplicate tasks. The graph generation technique is similar to the approach of [8, 31]. The nature of splits and joins (i.e., AND or OR) is discovered in the transformation step, where the stochastic task graph is transformed into an ADONIS workflow model with block-structured splits and joins. In contrast to the previous papers, the following papers are characterized by the focus on workflow processes with concurrent behavior (rather than adding ad-hoc mechanisms to capture parallelism).

The algorithm presented in this paper is most related to the α-algorithm presented in [2, 7, 47–50]. Based on an event log, the α-algorithm is able to construct a corresponding Petri net. In [47–50] a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs" as an intermediate step before constructing the corresponding Petri net. In [29] another variant of this technique is presented using examples from the health-care domain. The preliminary results presented in [29, 47–49] only provide heuristics and focus on issues such as noise. However, in [7] it is *proven* that the α-algorithm can find the proper process model for certain subclasses of Petri nets. In [2] the EMiT tool is presented which uses an extended version of α-algorithm to incorporate timing information. Note that EMiT also can handle START and COMPLETE events and use this to explicitly detect parallelism. However, this approach is different from the approach presented in this paper because the ordering relations are completely different. Moreover, the way EMiT deals with START and COMPLETE events is not proven to be correct. In fact, it is hardly documented.

Process mining can be seen as a tool in the context of Business (Process) Intelligence (BPI). In [18, 39] a BPI toolset on top of HP's Process Manager is described. The BPI tools set includes a so-called "BPI Process Mining Engine". However, this engine does not provide any techniques as discussed before. Instead it uses generic mining tools such as SAS Enterprise Miner for the generation of decision trees relating attributes of cases to information about execution paths (e.g., duration). In order to do process mining it is convenient to have a so-called "process data warehouse" to store audit trails. Such as data warehouse simplifies and speeds up the queries needed to derive causal relations. In [14, 33–35] the design of such warehouse and related issues are discussed in the context of workflow logs. Moreover, [35] describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [25]. The later tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [46] which is tailored towards mining Staffware logs. Note that none of the latter tools is extracting the process model. The main focus is on clustering and performance analysis rather than causal relations as in [8, 10–12, 19–24, 29–31, 40–44, 47–49].

More from a theoretical point of view, the rediscovery problem discussed in this paper is related to the work discussed in [9, 16, 17, 37]. In these papers the limits of inductive inference are explored. For example, in [17] it is shown that the computational problem of finding a minimum finite-state acceptor compatible with given data is NP-hard. Several of the more generic concepts discussed in these papers could be translated to the domain of process mining. It is possible to interpret the problem described in this paper as an inductive inference problem specified in terms of rules, a hypothesis space, examples, and criteria for successful inference. The comparison with literature in this domain raises interesting questions for process mining, e.g., how to deal with negative examples (i.e., suppose that besides log W there is a log V of traces that are not possible, e.g., added by a domain expert). However, despite the many relations with the work described in [9, 16, 17, 37] there are also many differences, e.g., we are mining at the net level rather than sequential or lower level representations (e.g., Markov chains, finite state machines, or regular expressions).

There is a long tradition of theoretical work dealing with the problem of inferring grammars out of examples: given a number of sentences (traces) out of a language, find the simplest model that can generate these sentences. There is a strong analogy with the process-mining problem: given a number of process traces, can we find the simplest process model that can generate these traces. Many issues important in the language-learning domain are also relevant for process mining (i.e. learning from only positive examples, how to deal with noise, measuring the quality of a model, etc.). However, an important difference between the grammar inference domain and the process-mining domain is the problem of concurrency in the traces: concurrency seems not relevant in the grammar inference domain. In spite of this important difference, it seems usefully to investigate which theoretical results, measurements, and mining techniques can be used or updated so that they become useful in process mining. A good overview of prominent computational approaches for learning different classes of formal languages is given in [36].

Additional related work is the seminal work on regions [15]. This work investigates which transition systems can be represented by (compact) Petri nets (i.e., the so-called synthesis problem). Although the setting is different and our notion of completeness is much weaker than knowing the transition system, there are related problems such as duplicate transitions, etc.

Most of the work mentioned thus far is primarily focusing on the process perspective. However, there are clear links with sociometry, and Social Network Analysis (SNA) in particular. Since the early work of Moreno [32] SNA has been an active research domain. There is a vast amount of textbooks, research papers, and tools available in this domain [45]. There have been many studies analyzing workflow processes based on insights from social network analysis. However, these studies typically have an ad-hoc character and sociograms are typically constructed based on questionnaires rather than using a structured and automated approach as described in this paper. Most tools in the SNA domain take sociograms as input. MiSoN is one of the few tools that generate sociograms as output. The only comparable tools are tools to analyze e-mail traffic, cf. BuddyGraph (http://www.buddygraph.com/) and MetaSight (http://www.metasight.co.uk/). However, these tools monitor unstructured messages and cannot distinguish between different activities (e.g., work-related interaction versus social interaction). One of the few approaches constructing sociograms from structured event logs is described in [4].

For more information on existing research, we also refer to special issue of Computers in Industry on process mining [6] and the survey paper [5].

3 Preliminaries: WF-nets

We assume some basic knowledge of Petri nets and WF-nets in particular. Readers not familiar with basic concepts such as (P, T, F) as a representation for a Petri net, the firing rule, firing sequences, preset $\bullet x$, postset $x\bullet$, boundedness, liveness, reachability, etc. are referred to [1, 13, 38]. Some basic definitions for WF-nets are provided in this section.

Before introducing the new algorithm we briefly discuss a subclass of Petri nets called a *WorkFlow nets* (WF-nets). This subclass is tailored towards modeling the control-flow dimension of a workflow[4] or any other case driven process, e.g., logging onto a system. It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation [1].

Definition 1 (Workflow nets). *Let $N = (P, T, F)$ be a Petri net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a* workflow net *(WF-net) iff:*

1. *object creation: P contains an input place i such that $\bullet i = \emptyset$,*

[4] Note that we use the words *workflow* and *process* interchangeably.

2. *object completion: P contains an output place o such that $o\bullet = \emptyset$,*
3. *connectedness: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected,*

The Petri net shown in Figure 1 is a WF-net. Note that although the net is not strongly connected, the *short-circuited* net with transition $\bar{t}$ is strongly connected. Even if a net meets all the syntactical requirements stated in Definition 1, the corresponding process may exhibit errors such as deadlocks, tasks which can never become active, livelocks, garbage being left in the process after termination, etc. Therefore, we define the following correctness criterion.

Definition 2 (Sound). *Let $N = (P, T, F)$ be a WF-net with input place i and output place o. N is* sound *iff:*

1. *safeness: $(N, [i])$ is safe,*[5]
2. *proper completion: for any marking $s \in [N, [i]\rangle$, $o \in s$ implies $s = [o]$,*
3. *option to complete: for any marking $s \in [N, [i]\rangle$, $[o] \in [N, s\rangle$, and*
4. *absence of dead tasks: $(N, [i])$ contains no dead transitions.*

The set of all sound WF-nets is denoted $\mathcal{W}$.

The WF-net shown in Figure 1 is sound. Soundness can be verified using standard Petri-net-based analysis techniques [1, 3].

Most process modeling languages offer standard building blocks such as the AND-split, AND-join, XOR-split, and XOR-join [3]. These are used to model sequential, conditional, parallel and iterative routing. Clearly, a WF-net can be used to specify the routing of cases, i.e., process instances. *Tasks*, also referred to as *activities*, are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-condition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. XOR-splits/XOR-joins correspond to places with multiple outgoing/ingoing arcs. Given the close relation between tasks and transitions we use the terms interchangeably.

Our process mining research aims at rediscovering WF-nets from event logs. However, not all places in sound WF-nets can be detected. For example places may be implicit which means that they do not affect the behavior of the process. These places remain undetected. Therefore, we limit our investigation to WF-nets without implicit places.

Definition 3 (Implicit place). *Let $N = (P, T, F)$ be a Petri net with initial marking s. A place $p \in P$ is called implicit in (N, s) if and only if, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in p\bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.*[6]

[5] $(N, [i])$ is the marked net with initial marking $[i]$, i.e., the marking with just one token in the source place i. Similarly, $[o]$ is used to denote the the marking with just one token in the sink place o.

[6] $[N, s\rangle$ is the set of reachable markings of net N when starting in marking s, $p\bullet$ is the set of output transitions of p, $\bullet t$ is the set of input places of t, and $\geq$ is the standard ordering relation on multisets.

Figure 1 contains no implicit places. However, adding a place p connecting transition $T1$ and $T4$ yields an implicit place. No mining algorithm is able to detect p since the addition of the place does not change the behavior of the net and therefore is not visible in the log.

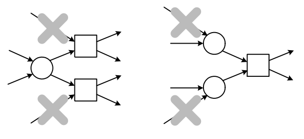

Figure 2. Constructs not allowed in SWF-nets.

For process mining it is very important that the structure of the WF-net clearly reflects its behavior. Therefore, we also rule out the constructs shown in Figure 2. The left construct illustrates the constraint that choice and synchronization should never meet. If two transitions share an input place, and therefore "fight" for the same token, they should not require synchronization. This means that choices (places with multiple output transitions) should not be mixed with synchronizations. The right-hand construct in Figure 2 illustrates the constraint that if there is a synchronization all preceding transitions should have fired, i.e., it is not allowed to have synchronizations directly preceded by an XOR-join. WF-nets which satisfy these requirements are named *structured workflow nets* and are defined as:

Definition 4 (SWF-net). *A WF-net $N = (P, T, F)$ is an* SWF-net *(Structured workflow net) if and only if:*

1. *For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|p\bullet| > 1$ implies $|\bullet t| = 1$.*
2. *For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|\bullet t| > 1$ implies $|\bullet p| = 1$.*
3. *There are no implicit places.*

The WF-net shown in Figure 1 is an example of an SWF-net. Note that all three requirements are satisfied.

Figure 3 gives another example of a process modelled in terms of an WF-net. This model is sound but it is not an SWF-net because the construct involving $P7$ and $P8$, i.e., $(P7, T11) \in F$ and $|\bullet T11| > 1$ but $|\bullet P7| > 1$. Nevertheless, the model will be used as the main example throughout the paper.

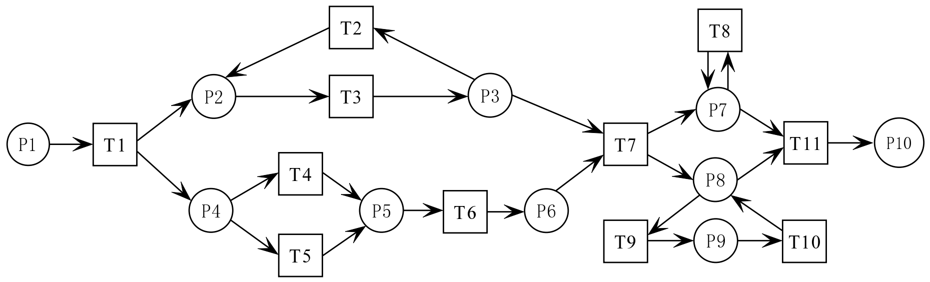

Figure 3. An example of process expressed in terms of a Petri net.

The transitions (drawn as rectangles) $T1$, $T2$, $\cdots$, $T11$ represent tasks and the places (drawn as circles) $P1$, $P2$, $\cdots$, $P10$ represent causal dependencies. A place can be used as pre-condition and/or post-condition for tasks. The arcs (drawn as directed edges) between transitions and places represent flow relations. In this process, sequential (from $T9$ to $T10$, etc.), alternative (from $P4$ to $T4$ and $T5$, etc.), parallel (from $T1$ to $P2$ and $P4$, etc.), synchronous (from $P7$ and $P8$ to $T11$, etc.) and iterative ($P2$-$T3$-$P3$-$T2$-$P2$, $P7$-$T8$-$P7$, etc.) routing are present. There are also three short loops (i.e., loops of length of one or two): the loop involving $T8$ (length 1), the loop involving $T2$ and $T3$ (length 2), and the loop involving $T9$ and $T10$ (also length 2). Also note the special parallel routing (splits from $T7$ and joins at $T11$).

The α-algorithm is unable to correctly mine WF-nets such as the one shown in Figure 3 (but also the model shown in the introduction), because of the presence of short loops. Moreover, tasks (i.e., transition firings) are considered to be atomic while in reality this is not the case.

4 Analyzing the event log

In this section, we focus on event logs with two event types. First, we define such event logs. Then, we define a new notion of completeness and ordering relations on tasks based on the two event types START and COMPLETE.

4.1 Event logs with two types of events

Existing approaches do not consider event types [2, 5, 7, 8, 10, 19, 50]. Tasks are either considered to be atomic or only the completion of a task is considered (i.e., just event type COMPLETE). One way to deal with this is to consider the start and completion of a task as two atomic tasks. EMiT uses some pre- and post-processing to incorporate multiple event types, but does not incorporate this in the mining algorithm and ordering relations.[7] In this paper, we propose a fundamentally different approach where parallelism is detected explicitly by registering overlapping activities.

As indicated in the introduction, there are two event types: START and COMPLETE. Therefore, each event is characterized by a task and an event type.

Definition 5 (Event). *Let T be a set of tasks. $E = T \times \{0, 1\}$ is a set of events over T. $(t, 0) \in E$ denotes the start of some task t and $(t, 1) \in E$ denotes the completion of t. For convenience, we also introduce the following notation for $e \in E$: $e.task$ refers to the task and $e.type$ refers to the event type. If $e = (t, 0)$, then $e.task = t$ and $e.type = START$. If $e = (t, 1)$, then $e.task = t$ and $e.type = COMPLETE$.*

[7] Note that EMiT allows for even more event types, e.g., there are also event types like SCHEDULE, ASSIGN, WITHDRAW, etc.

Note that Definition 5 abstracts from other information that may be present in the log, e.g., the timestamp of the event, the performer executing the task, and data linked to the event. An event always occurs in the context of a single case. The ordering of events corresponding to different cases is not important. Therefore, we consider a log to be a set of traces where each trace corresponds to a case.

Definition 6 (Event trace, Event log). *Let $E = T \times \{0,1\}$ be a set of events over T. $\sigma \in T^*$ is an* event trace *and $W \subseteq T^*$ is an* event log.[8]

Note that the log shown in Table 1 is consistent with this notation. For example, the event trace for the first case is $\sigma = (T1,0)(T1,1)(T2,0)(T3,0)(T3,1)(T2,1)$ $(T4,0)(T4,1)(T6,0)(T6,1)$.

Event traces are sequences. We use the following standard notation for sequences.

Definition 7. *Let $E = T \times \{0,1\}$, $\sigma \in T^*$ a sequence containing n elements, and $t \in T$ some task.*

1. *$dom(\sigma) = \{1,2,\ldots,n\}$ is the domain of σ,*
2. *σ_i is the i-th element, $i \in dom(\sigma)$,*
3. *$t \in \sigma$ iff there exists an $i \in dom(\sigma)$ such that $\sigma_i.task = t$,*
4. *$first(\sigma) = \sigma_1.task$ is the first task to start, and*
5. *$last(\sigma) = \sigma_n.task$ is the last task to complete.*

Note that Definition 6 allows for event traces like $(T1,1)$ $(T1,0)$ and $(T1,0)$ $(T2,1)$ (i.e., the COMPLETE event precedes the START event or there is not START/COMPLETE event at all). Therefore, we define the notion of consistency.

Definition 8 (Consistent). *Let $E = T \times \{0,1\}$ be a set of events over T and $\sigma \in T^*$ an event trace. σ is* consistent *if and only if*

1. *$\forall_{i \in dom(\sigma)} \sigma_i.type = 0 \Rightarrow (\exists_{j \in dom(\sigma)} j > i \ \wedge \ \sigma_j = (\sigma_i.task, 1) \ \wedge$ $\forall_{i<k<j} \sigma_i.task \neq \sigma_k.task)$, i.e., every START event has a corresponding COMPLETE event, and*
2. *$\forall_{i \in dom(\sigma)} \sigma_i.type = 1 \Rightarrow (\exists_{j \in dom(\sigma)} j < i \ \wedge \ \sigma_j = (\sigma_i.task, 0) \ \wedge$ $\forall_{j<k<i} \sigma_i.task \neq \sigma_k.task)$, i.e., every COMPLETE event has a corresponding START event.*

In the remainder we consider event traces to be consistent, i.e., any log W will hold only consistent traces. Note that in some situations this is not realistic, i.e., parts of the log may be missing or there may be some kind of noise. In [49] these issues are discussed and partially solved. We expect that the concepts presented in [49] can be transferred to the mining algorithm presented here.

[8] T^* is the set of all sequences that are composed of zero of more tasks from T.

4.2 Ordering relations

An essential prerequisite for process mining is the ordering of tasks. To define suitable ordering relations on tasks, we need to consider pairs of events, i.e., a START event and a corresponding COMPLETE event. Therefore, we define the notion of *task occurrence*.

Definition 9 (Task occurrence). *Let $\sigma \in E^*$ and $\sigma = e_1 e_2 \cdots e_n$. $t(e_i, e_j)$ is a task occurrence of t in σ iff*

1. $1 \leq i < j \leq n$,
2. $e_i.task = e_j.task = t$,
3. $e_i.type = 0$,
4. $e_j.type = 1$, *and*
5. $\forall_{i<k<j}\ \sigma_k.task \neq t$).

Note that every event in event trace corresponds to precisely one task occurrence. However, for one task there may be multiple task occurrences in the same event trace.

Intuitively, a task occurrence can be represented as a line segment. The left end is the START event and the right end is the COMPLETE event. These line segments represent the time the task is being executed and can be used to define succession (i.e., "directly" follows) and intersection (i.e., overlapping task occurrences).

Definition 10 (Succession). *Let $W \subseteq E^*$ an event log such that $E = T \times \{0,1\}$. Let $a, b \in T$ be two tasks. a is directly succeeded by b in W, notation $a >_W b$, iff there exists a $\sigma \in E^*$ such that $\sigma = e_1 e_2 \cdots e_n$ and two task occurrences $a(e_i, e_j)$ and $b(e_k, e_l)$ in σ such that $j < k$ and there is no task occurrence $c(e_p, e_q)$ in σ satisfying $j < p < q < k$.*

a is succeeded by b if and only if in at least one event trace a is "directly followed" by b, i.e., there is not another *complete* task occurrence in-between the two task occurrences $a(e_i, e_j)$ and $b(e_k, e_l)$.

Definition 11 (Intersection). *Let $W \subseteq E^*$ an event log such that $E = T \times \{0,1\}$. Let $a, b \in T$ be two tasks. a intersects with b in W, notation $a \times_W b$, iff there exists a $\sigma \in E^*$ such that $\sigma = e_1 e_2 \cdots e_n$ and two task occurrences $a(e_i, e_j)$ and $b(e_k, e_l)$ in σ such that $i < k < j$ or $k < i < l$.*

a intersects with b if and only if in at least one event trace where an occurrence of a overlaps with an occurrence of b. Note that the intersection relation is symmetric, i.e., $a \times_W b$ if and only if $b \times_W a$.

Both $a >_W b$ (a is succeeded by b) and $a \times_W b$ (a intersects with b) are illustrated in Figure 4.

Using the notation introduced in this section we can represent the finite set of tasks $T_W = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$, the finite set of initial tasks $T_I = \{t \in T | \exists_{\sigma \in W} t = first(\sigma)\}$ (the first tasks to start), and the finite set of final tasks

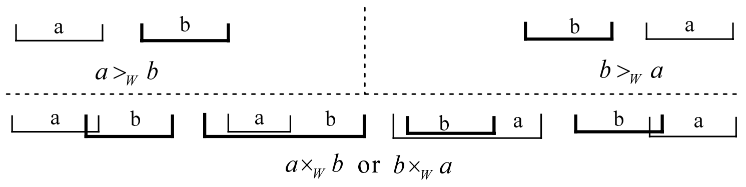

Figure 4. Illustration of $a >_W b$, $b >_W a$, $a \times_W b$, and $b \times_W a$.

$T_O = \{t \in T | \exists_{\sigma \in W} t = last(\sigma)\}$ (the last tasks to complete). It is also fairly straightforward to calculate the relations $>_W$ and $\times_W$. The complexity of an efficient algorithm to calculate these relations and sets is $O(n)$, where n is the number of total events in the corresponding traces.

The notions T_W, T_I, T_O, $>_W$, and $\times_W$ are the basic ingredients for the mining algorithm presented in this paper. To prove the correctness of the mining algorithm we need to assume some notion of completeness, i.e., for a complex process with many possible event traces we need a log that somehow reflects the possible behavior.

Definition 12 (Completeness of an event log). *Let N=(P,T,F) be a sound WF-net. W is an event log of N iff $W \subseteq E^*$ where $E = T \times \{0,1\}$ and every trace $\sigma \in W$ is a firing sequence of N starting in state $[i]$ and ending in state $[o]$. W is a complete event log of N iff 1) For any event log W' of N: $>_{W'} \subseteq >_W$ and $\times_{W'} \subseteq \times_W$, and 2) For any $t \in T$, there is a $\sigma \in W$ such that $t \in \sigma$.*

It is easy to check that the event log shown in Table 1 is complete, i.e., all tasks appear somewhere in the log and the relations $>_W$ and $\times_W$ are maximal.

$$
\begin{array}{c|ccccccccccc|}
>_W & T1 & T2 & T3 & T4 & T5 & T6 & T7 & T8 & T9 & T10 & T11 \\
\hline
T1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
T2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
T3 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
T4 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
T5 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
T6 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
T7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
T8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
T9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
T10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
T11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\quad
\begin{array}{c|ccccccccccc|}
\times_W & T1 & T2 & T3 & T4 & T5 & T6 & T7 & T8 & T9 & T10 & T11 \\
\hline
T1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
T3 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
T4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T5 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T6 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
T8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
T9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
T10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
T11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Figure 5. Matrices representing $>_W$ and $\times_W$ for the WF-net shown in Figure 3 based on some complete log W.

Assume that we have a complete event log for the WF-net shown in Figure 3. The resulting relations $>_W$ and $\times_W$ are shown in Figure 5. In this figure 0 denotes *false* and 1 denotes *true*.

4.3 Identifying the ordering relations between tasks

After establishing the basic relations $>_W$ and $\times_W$ we identify four derived relations. These derived ordering relations will be used to detect typical routings in the process model, such as sequential, parallel, alternative, iterative (i.e., loops) routing and their combination.

Definition 13 (Log-based ordering relations). *Let W be an event log over E where $E = T \times \{0,1\}$. For any $a, b \in T$:*

- $a \rightarrow_W b$ *iff* $a >_W b$ *and* $\neg(a \times_W b)$.
- $a \parallel_W b$ *iff* $a \times_W b$.
- $a \#_W b$ *iff* $\neg(a >_W b)$ *and* $\neg(a \times_W b)$.
- $a \nparallel_W b$ *iff* $\neg(a \times_W b)$.

Based on these definitions, it is obvious that relations $\parallel_W$ and $\nparallel_W$ satisfy commutativity while relations $\rightarrow_W$ and $\#_W$ do not. The two relations $\parallel_W$ and $\nparallel_W$ are mutually exclusive and complementary. From Definition 13, the following property can be inferred directly.

Property 1. Let W be an event log over E where $E = T \times \{0,1\}$. For any $a, b \in T$: $a \rightarrow_W b$, $a \#_W b$, or $a \parallel_W b$. Moreover, the relations $\rightarrow_W$, $\#_W$, and $\parallel_W$ are mutually exclusive and partition $T \times T$. Furthermore, the relation $\nparallel_W$ is the union of the relations $\rightarrow_W$ and $\#_W$.

After applying Definition 13 to the two matrices shown in Figure 5, we obtain the matrix shown in Figure 6.

$$
\begin{array}{c|ccccccccccc}
 & T1 & T2 & T3 & T4 & T5 & T6 & T7 & T8 & T9 & T10 & T11 \\
\hline
T1 & \#_W & \#_W & \rightarrow_W & \rightarrow_W & \rightarrow_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W \\
T2 & \#_W & \#_W & \rightarrow_W & //_W & //_W & //_W & \#_W & \#_W & \#_W & \#_W & \#_W \\
T3 & \#_W & \rightarrow_W & \#_W & //_W & //_W & //_W & \rightarrow_W & \#_W & \#_W & \#_W & \#_W \\
T4 & \#_W & //_W & //_W & \#_W & \#_W & \rightarrow_W & \#_W & \#_W & \#_W & \#_W & \#_W \\
T5 & \#_W & //_W & //_W & \#_W & \#_W & \rightarrow_W & \#_W & \#_W & \#_W & \#_W & \#_W \\
T6 & \#_W & //_W & //_W & \#_W & \#_W & \#_W & \rightarrow_W & \#_W & \#_W & \#_W & \#_W \\
T7 & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \rightarrow_W & \rightarrow_W & \#_W & \rightarrow_W \\
T8 & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \rightarrow_W & //_W & //_W & \rightarrow_W \\
T9 & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & //_W & \#_W & \rightarrow_W & \#_W \\
T10 & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & //_W & \rightarrow_W & \#_W & \rightarrow_W \\
T11 & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W & \#_W
\end{array}
$$

Figure 6. Matrix of the ordering relations for the WF-net shown in Figure 3 based on the two matrices shown in Figure 5.

The log-based relations shown in Figure 6 reflect the relations between the tasks in the WF-net shown in Figure 3 in an intuitive manner. For example, $T9$ and $T10$ are clearly in a sequence and indeed we obtain $T9 \rightarrow_W T10$ from the complete log. Another example is that $T3$ and $T4$ are in parallel and we indeed get $T3 \parallel_W T4$.

Note that it may appear to be strange that we compare the log-based relations (e.g., Figure 6) with a Petri net that is already known (e.g., Figure 3). However, please note that while building the relations we only consider the log and not the WF-net itself. Rediscovering a known WF-net based on a *complete log* is used for demonstrating the accuracy of the mining algorithm. The challenge is to derive Figure 3 from a complete log without any additional knowledge. Note that completeness is very important in this context. If the log is not complete, our mining algorithm will still be able to discover a process but this is likely to differ from the actual process because there are not enough observations.

5 Constructing a process model from ordering relations

In this section, we present the new algorithm which we have named the β-algorithm. However, first we investigate the relation between the ordering relations detected from the log and the presence of the connecting places in the corresponding process model. We will use this to prove the correctness of the β-algorithm. The proofs of all theorems presented in this section can be found in the appendix of this paper.

5.1 Ordering relations and connecting places

First we investigate the relation between $\rightarrow_W$ (i.e., the ordering relation indicating causality) and the existence of connecting places. If $\rightarrow_W$ relates two transitions (i.e., tasks), there will be a place connecting them.

Theorem 1. *Let $N = (P, T, F)$ be a sound WF-net and let W be a complete event log of N. For any $a, b \in T$: $a \rightarrow_W b$ implies $a \bullet \cap \bullet b \neq \emptyset$.*

Figures 6 and 3, can be used to illustrate the theorem. Since $T1 \rightarrow_W T3$ (cf. Figure 6), there has to be a place between $T1$ and $T3$. This place corresponds to $P2$ in the WF-net shown in Figure 3.

Theorem 1 holds for any WF-net. The other direction, does not hold for any WF-net. However, for SWF-nets we can show that if a place connects two successive transitions in an SWF-net, their corresponding tasks are related through $\rightarrow_W$.

Theorem 2. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b \in T$: $a \bullet \cap \bullet b \neq \emptyset$ implies $a \rightarrow_W b$.*

Based on figures 3 and 6, we can see that all connecting places between two successive transitions lead to $\rightarrow_W$ relations between the corresponding two tasks in the log, e.g., the presence of the place $P2$ connecting $T1$ and $T3$ indeed implies $T1 \rightarrow_W T3$, etc.

After showing the relation between $\rightarrow_W$ and places in the corresponding Petri net, we focus on parallelism. First, we show that two transitions cannot be in parallel according to $\|_W$ if they have common input or output places.

Theorem 3. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b \in T$:*

1. *If $a \bullet \cap b \bullet \neq \emptyset$, then $a \nparallel_W b$.*
2. *If $\bullet a \cap \bullet b \neq \emptyset$, then $a \nparallel_W b$.*

It is clear that $T4$ and $T5$ share one input place $P4$ and one output place $P5$ in Figure 3. The ordering relations between $T4$ and $T5$ are $T4 \#_W T5$ and $T5 \#_W T4$. Thus $T4 \nparallel_W T5$ holds, i.e., $T4$ and $T5$ can not occur concurrently.

To show that a similar relation holds in the other direction consider three tasks a, b, and c. If both a and b are causally related to c (i.e., a and c are connected by a place in the corresponding Petri net and so are b and c) and a and b are not in parallel (i.e., $a \nparallel_W b$ holds), then a and b are connected to c through a *common* place.

Theorem 4. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b, c \in T$:*

1. *If $a \rightarrow_W c$, $b \rightarrow_W c$ and $a \nparallel_W b$, then $a\bullet \cap b\bullet \cap \bullet c \neq \emptyset$.*
2. *If $c \rightarrow_W a$, $c \rightarrow_W b$ and $a \nparallel_W b$, then $c\bullet \cap \bullet a \cap \bullet b \neq \emptyset$.*

For example, $T1 \rightarrow_W T4$, $T1 \rightarrow_W T5$ and $T4 \nparallel_W T5$ hold in Figure 6. Therefore, as Theorem 4 points out, there is a place $P4$ connecting $T1$, $T4$ and $T5$ in Figure 3.[9] Another example is the fact that $T7 \rightarrow_W T8$, $T8 \rightarrow_W T8$ and $T7 \nparallel_W T8$ implies that $T7\bullet \cap T8\bullet \cap \bullet T8 \neq \emptyset$. As Figure 3 shows, the shared place is $P7$. Note that in terms of Theorem 4 $a = T7$, $b = T8$, and $c = T8$, i.e., $b = c$. This example shows that, unlike the classical relations used by the α-algorithm [7], the ordering relations can deal successfully with short loops.

The following theorem shows how to identify the connecting places.

Theorem 5. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any two task sets PS and SS, such that $PS \subseteq T$, $SS \subseteq T$: $\forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b$, $\forall_{a1, a2 \in PS} a1 \nparallel_W a2$ and $\forall_{b1, b2 \in SS} b1 \nparallel_W b2$ iff $\exists_{p \in P} \forall_{a \in PS} \forall_{b \in SS} a\bullet \cap \bullet b = \{p\}$.*

Theorem 5 illustrates the relation between the connecting places and the ordering relations among tasks. Considering an example from Figure 3, we get $PS = \{T4, T5\}$, $SS = \{T6\}$ and the unique connecting places is $p = P5$. Notably, although the net shown in Figure 3 is not an SWF-net, we can still get the correct relations. The connecting places $P7$ and $P8$ can be rediscovered successfully and efficiently, which indicates the power of the mining algorithm presented next.

5.2 Mining algorithm based on the ordering relations

Based on the theoretical results shown in the previous subsection, we now present the β-algorithm.

Mining algorithm β. *Let W be an event log over T. $\beta(W)$ is defined as follows*:

1. $T_W = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$,

[9] Note that Figure 3 is not an SWF-net. However, the part of the net considered does satisfy the requirements of an SWF-net. In fact, the applicability of the algorithm and therefore also the theorems are not limited to just SWF-nets.

2. $T_I = \{t \in T | \exists_{\sigma \in W} t = first(\sigma)\}$,
3. $T_O = \{t \in T | \exists_{\sigma \in W} t = last(\sigma)\}$,
4. $X_W = \{<PS, SS> | PS \subseteq T_W \wedge SS \subseteq T_W \wedge \forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b \wedge \forall_{a1,a2 \in PS} a1 \nparallel_W a2 \wedge \forall_{b1,b2 \in SS} b1 \nparallel_W b2\}$,
5. $Y_W = \{<PS, SS> \in X_W | \forall_{<PS', SS'> \in X_W} PS \subseteq PS' \wedge SS \subseteq SS' \Rightarrow <PS, SS> = <PS', SS'>\}$,
6. $P_W = \{p_{<PS,SS>} | <PS, SS> \in Y_W\} \cup \{i_W, o_W\}$,
7. $F_W = \{(a, p_{<PS,SS>}) | <PS, SS> \in Y_W \wedge a \in PS\} \cup \{(p_{<PS,SS>}, b) | <PS, SS> \in Y_W \wedge b \in SS\} \cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$, and
8. $\beta(W) = (P_W, T_W, F_W)$.

The mining algorithm constructs a Petri net (P_W, T_W, F_W) based on some event log W. Note that T_W, T_I and T_O can be obtained easily, i.e., the first three steps are self-explanatory and linear in the size of the log. The last three steps are also straightforward once Y_W has been obtained. In fact these three steps are linear in the size of the resulting model. It is important to see that Y_W corresponds to the set of internal places and that these places are discovered using the insights resulting from the theorems presented in Section 5.1. The most important and time-consuming steps are 4 and 5. Step 4 attempts to find all the pairs of task sets satisfying the specific conditions to generate X_W. Step 5 is used to find all the largest elements in X_W with respect to set inclusion to generate Y_W. To calculate Y_W, the complexity of these two steps is exponential in the number of tasks. In fact, the number of tasks in a practical process is less than 100. Therefore, the complexity is not a bottleneck for large-scale applications.

Now we will prove the correctness of the mining algorithm. Again the focus is on the connecting places.

Theorem 6. *Let N be a sound SWF-net and let W be a complete event log of N. $\beta(W) = N$ modulo renaming of places, i.e., the discovered model matches the original model after renaming places.*

The names of the corresponding places of N and N_W are different because the names of the places are not stored in the event log. However, the names of the places less relevant because they only serve as pre- and post-conditions for tasks. Let us demonstrate the algorithm using the results shown in Figure 6. We show the results in every step of the β-algorithm.

1. $T_W = \{T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11\}$,
2. $T_I = \{T1\}$,
3. $T_O = \{T11\}$,
4. $X_W = \{<\{T1\}, \{T3\}>, <\{T1\}, \{T4\}>, \ldots, <\{T7, T10\}, \{T9, T11\}>\}$,
5. $Y_W = \{<\{T1, T2\}, \{T3\}>, <\{T1\}, \{T4, T5\}>, <\{T3\}, \{T2, T7\}>, <\{T4, T5\}, \{T6\}>, <\{T7, T8\}, \{T8, T11\}>, <\{T7, T10\}, \{T9, T11\}>, <\{T6\}, \{T7\}>, <\{T9\}, \{T10\}>\}$,
6. $P_W = \{i_W, o_W, p_{<\{T1,T2\},\{T3\}>}, p_{<\{T1\},\{T4,T5\}>}, \ldots, p_{<\{T9\},\{T10\}>}\}$,
7. $F_W = \{(i_W, T1), (T1, p_{<\{T1,T2\},\{T3\}>}), (p_{<\{T1,T2\},\{T3\}>}, T3), \ldots, (T11, o_W)\}$,
8. $\beta(W) = (P_W, T_W, F_W)$.

The resulting net is indeed the WF-net shown in Figure 3. Although this net is not a SWF-net, the algorithm can still mine it successfully. There are no redundant nodes (i.e., transitions and places) or edges (i.e., arcs) and no information is lost except the names of places. Even the short loops and parallel routings are identified correctly. This example shows that the applicability of the algorithm is not limited to SWF-nets. It is applicable to a larger class of sound WF-nets.

Based on the log shown in Table 1 we can calculate the ordering relations and successfully discover the process model shown in Figure 1. Note that this net is an SWF net and therefore for any complete log, the β-algorithm will discover the SWF net modulo renaming of places, cf. Theorem 6. Note that the classical α-algorithm [7] is unable to successfully mine all SWF nets and will generate an incorrect model for a log shown in Table 1.

6 Experimental evaluation of the work

We have developed a mining tool based on the β-algorithm and integrated it into our workflow management system named *WebFlow*. This tool consists of three parts: a simulation component, a mining component and a process editor. The simulation component is used to generate an event log either manually or automatically. The mining component is used to mine a process model from a selected event log. The process editor is used to display the mined process model to the process designer for further editing.

In an experimental setting logs can be obtained in three ways: (1) as a download from an operation information system (i.e., a real log), (2) a manually created log, and (3) a log resulting from a simulation which records events in a simulation log. For evaluation of the β-algorithm, we have used all three possibilities. In this section, we show the results of our experimental evaluation of the β-algorithm.

Table 2 summarizes the execution time of the mining procedure for the process model shown in Figure 3 with logs having varying number of traces. Here $\#L$ is the number of traces, $\#T$ is the number of tasks, $\#E$ is the number of events, T_m is the execution time of the whole mining procedure and T_c is the execution time of the scanning step, i.e., loading the log and building the relations. The time unit used in Table 2 is seconds.

Tc Tm #L(#E) / #T	**110(4008)**		**220(7486)**		**440(15192)**		**880(30112)**	
11	0.19	0.21	0.41	0.43	0.802	0.822	1.632	1.652

Table 2. Execution time in seconds.

Note that T_m and T_c do not differ much, thus indicating that most of the time is spent on the scanning step. For clarity, we transformed the data shown in Table 2 to the two graphs shown in Figure 7. These graphs clearly show the linear relations between T_c, T_m and $\#L$, $\#E$.

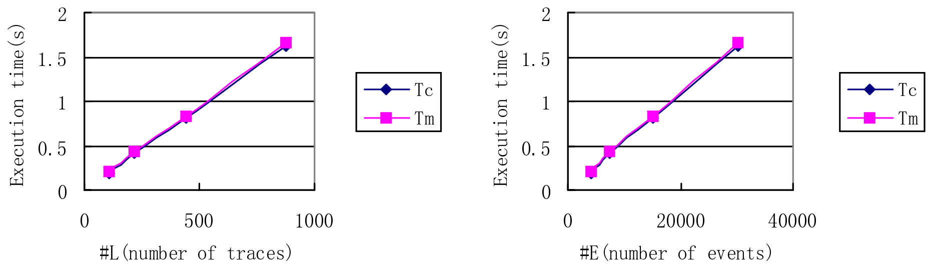

Figure 7. Relations between Tm, Tc and $\#L$, $\#E$.

To evaluate the β-algorithm fully, we change the range of $\#T$ from 10 to 100 and the range of $\#L$ from 10 to 10000. The physical size of the log is roughly proportional to $\#L$. For $\#L$=10000, the sizes of logs are 3MB, 7MB, 16MB and 36MB for process models with 10, 25, 50 and 100 tasks respectively. Table 3 summarizes the execution time of the mining procedure for these process models. Again the time unit is seconds.

#E / Tm / #T / #L	10		25		50		100	
10	270	0.019	332	0.025	756	0.110	1478	0.471
100	3176	0.171	3170	0.210	7170	0.451	16460	1.361
1000	31680	1.682	32000	2.073	72700	4.136	159560	9.814
10000	317720	16.694	318900	20.720	727200	38.896	1601648	91.061

Table 3. Execution time in seconds for different models.

To visualize the result presented in Table 3 we again show two graphs, see Figure 8. In practical process models, the number of tasks (i.e., $\#T$) is less than 100. The number of traces $\#L$ and also the number of events $\#E$ are typically much larger. Therefore the number of traces is the dominant factor in determining the execution time of the mining procedure. Table 3 and Figure 8 show that the mining procedure is fast enough and scales linearly with the input number of events for a given process model. It also scales well with the number of tasks in the practical process models.

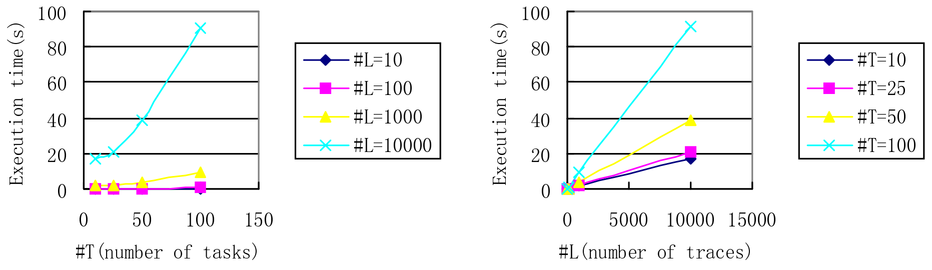

Figure 8. Execution time for different models using different traces.

From the experimental evaluation, it is clear that the mining procedure is suitable for practical situations. It runs fast and scales well for large-scale applications. As far as the quality of the mining algorithm is concerned, the β-algorithm can mine all of the sound SWF-nets successfully. In fact, in some cases sound WF-nets that do not satisfy all requirements of an SWF-net can still be rediscovered provided that the log is complete.

7 Conclusion and future work

In this paper, a new mining algorithm was presented: the β-algorithm. A distinguishing feature of the β-algorithm is that it exploits the fact that tasks take time and therefore parallelism can be detected explicitly. To do this, event logs with two kinds of event types, i.e., START and COMPLETE, are considered. Using these two types of events it is possible to see if occurrences of tasks overlap. Together with causality information, this is used to derive the ordering relations $\rightarrow_W$, $\#_W$, or $\|_W$. Based on these relations the β-algorithm constructs a Petri net. Assuming a complete log, it can be proven that the β-algorithm is able to correctly discover any SWF-net. In fact the application is not limited to SWF-nets, i.e., it can be applied to any event log with START and COMPLETE events. However, for some non-SWF-nets the result may be incorrect. Through experimental evaluation of the work, we demonstrated that the β-algorithm is simple, fast and powerful enough to be used in practical situations.

The β-algorithm can be seen as an extension of the α-algorithm. Some of the known problems of the α-algorithm, e.g., short-loops, are tackled by the β-algorithm using fundamentally different ordering relations. However, there is also a drawback. The α-algorithm can be applied in environments where tasks are considered to be atomic, e.g., just the COMPLETE events are logged. In such environments the α-algorithm will be unable to detect parallelism, while the α-algorithm is able to do this implicitly (assuming interleaving semantics).

Our future work will focus on the following three aspects. First of all, we plan to further evaluate and apply the mining algorithm in practical situations. Secondly, we plan to improve the storage structure of the event log and reduce the running time of the mining procedure even further. Finally, we will investigate which kind of sound non-SWF-nets (i.e., ordinary sound WF-nets) can be rediscovered by the β-algorithm.

Acknowledgements
The authors would like to thank Ton Weijters, Ana Karla Alves de Medeiros, Boudewijn van Dongen, Minseok Song, Laura Maruster, Eric Verbeek, Monique Jansen-Vullers, Hajo Reijers, Michael Rosemann, and Peter van den Brand for their on-going work on process mining techniques and tools at Eindhoven University of Technology.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

2. W.M.P. van der Aalst and B.F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer-Verlag, Berlin, 2002.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering interaction patterns in business processes. In M. Weske, B. Pernici, and J. Desel, editors, *International Conference on Business Process Management (BPM 2004)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2004.
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
7. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. QUT Technical report, FIT-TR-2003-03, Queensland University of Technology, Brisbane, 2003. (Accepted for publication in IEEE Transactions on Knowledge and Data Engineering.).
8. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
9. D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.
10. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
11. J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6)*, pages 35–45, 1998.
12. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
13. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 1995.
14. J. Eder, G.E. Olivotto, and Wolfgang Gruber. A Data Warehouse for Workflow Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin, 2002.
15. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.
16. E.M. Gold. Language Identfication in the Limit. *Information and Control*, 10(5):447–474, 1967.
17. E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.

18. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
19. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
20. J. Herbst. Dealing with Concurrency in Workflow Induction. In U. Baake, R. Zobel, and M. Al-Akaidi, editors, *European Concurrent Engineering Conference*. SCS Europe, 2000.
21. J. Herbst. *Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen*. PhD thesis, Universität Ulm, November 2001.
22. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998.
23. J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, Stockholm, Sweden, August 1999.
24. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000.
25. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.
26. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.workflowpatterns.com.
27. H. Mannila and D. Rusakov. Decomposing Event Sequences into Independent Components. In V. Kumar and R. Grossman, editors, *Proceedings of the First SIAM Conference on Data Mining*, pages 1–17. SIAM, 2001.
28. H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
29. L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, and W. Daelemans. Automated Discovery of Workflow Models from Hospital Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 183–190, 2001.
30. L. Maruster, A.J.M.M. Weijters, W.M.P. van der Aalst, and A. van den Bosch. Process Mining: Discovering Direct Successors in Process Logs. In *Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002)*, volume 2534 of *Lecture Notes in Artificial Intelligence*, pages 364–373. Springer-Verlag, Berlin, 2002.
31. M.K. Maxeiner, K. Küspert, and F. Leymann. Data Mining von Workflow-Protokollen zur teilautomatisierten Konstruktion von Prozemodellen. In *Proceedings of Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 75–84. Informatik Aktuell Springer, Berlin, Germany, 2001.

32. J.L. Moreno. *Who Shall Survive?* Nervous and Mental Disease Publishing Company, Washington, DC, 1934.
33. M. zur Mühlen. Process-driven Management Information Systems Combining Data Warehouses and Workflow Technology. In B. Gavish, editor, *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566. IEEE Computer Society Press, Los Alamitos, California, 2001.
34. M. zur Mühlen. Workflow-based Process Controlling-Or: What You Can Measure You Can Control. In L. Fischer, editor, *Workflow Handbook 2001, Workflow Management Coalition*, pages 61–77. Future Strategies, Lighthouse Point, Florida, 2001.
35. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
36. R. Parekh and V. Honavar. Automata Induction, Grammar Inference, and Language Acquisition. In Dale, Moisl, and Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 2000.
37. L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K.P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference (AII)*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44. Springer-Verlag, Berlin, 1889.
38. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
39. M. Sayal, F. Casati, and M.C. Shan U. Dayal. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
40. G. Schimm. Process Mining. http://www.processmining.de/.
41. G. Schimm. Generic Linear Business Process Modeling. In S.W. Liddle, H.C. Mayr, and B. Thalheim, editors, *Proceedings of the ER 2000 Workshop on Conceptual Approaches for E-Business and The World Wide Web and Conceptual Modeling*, volume 1921 of *Lecture Notes in Computer Science*, pages 31–39. Springer-Verlag, Berlin, 2000.
42. G. Schimm. Process Mining elektronischer Geschäftsprozesse. In *Proceedings Elektronische Geschäftsprozesse*, 2001.
43. G. Schimm. Process Mining linearer Prozessmodelle - Ein Ansatz zur automatisierten Akquisition von Prozesswissen. In *Proceedings 1. Konferenz Professionelles Wissensmanagement*, 2001.
44. G. Schimm. Process Miner - A Tool for Mining Process Schemes from Event-based Data. In S. Flesca and G. Ianni, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 525–528. Springer-Verlag, Berlin, 2002.
45. J. Scott. *Social Network Analysis*. Sage, Newbury Park CA, 1992.
46. Staffware. Staffware Process Monitor (SPM). http://www.staffware.com, 2002.
47. A.J.M.M. Weijters and W.M.P. van der Aalst. Process Mining: Discovering Workflow Models from Event-Based Data. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pages 283–290, 2001.
48. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data. In V. Hoste and G. de Pauw, editors, *Proceedings of the 11th Dutch-Belgian Conference on Machine Learning (Benelearn 2001)*, pages 93–100, 2001.

49. A.J.M.M. Weijters and W.M.P. van der Aalst. Workflow Mining: Discovering Workflow Models from Event-Based Data. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.
50. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

Appendix

Theorem 1. *Let $N = (P, T, F)$ be a sound WF-net and let W be a complete event log of N. For any $a, b \in T$: $a \rightarrow_W b$ implies $a \bullet \cap \bullet b \neq \emptyset$.*

Proof. Assume $a \rightarrow_W b$ and $a \bullet \cap \bullet b = \emptyset$. We will show that this assumption leads to a contradiction and thus prove the theorem. From Definition 13, we know that $a \rightarrow_W b$ implies $a >_W b$ and $\neg(a \times_W b)$. Since $a >_W b$ there exists at least one trace $\sigma = e_1 e_2 e_3 \cdots e_n \in W$ such that $\exists_{i,j} 2 \leq i \leq n-2 \wedge i < j < n$ such that $e_i.type$=COMPLETE, $e_i.task$=a, $e_j.type$=START, $e_j.task$=b and there is not any task occurrence between e_i and e_j. For $\forall_k i < k < j$ and $e_k.type$=COMPLETE, we know that e_k can occur before e_i in some traces. Similarly, for $\forall_m i < m < j$ and $e_m.type$=START, we know that e_m can wait until e_j occurs. Thus we can get a marking M of N, under which a can complete and after a completes, b can start immediately. Because $a \bullet \cap \bullet b$=$\emptyset$, a does not produce tokens for any input place of b. So under the marking M, b can start before a completes. Therefore, we can find $a \times_W b$ from the log and $a \parallel_W b$ holds. This result contradicts $a \rightarrow_W b$ and we conclude that $a \rightarrow_W b$ implies $a \bullet \cap \bullet b \neq \emptyset$.

Theorem 2. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b \in T$: $a \bullet \cap \bullet b \neq \emptyset$ implies $a \rightarrow_W b$.*

Proof. Because $a \bullet \cap \bullet b \neq \emptyset$, we assume a place $p \in a \bullet \cap \bullet b$. We should prove this theorem from the following two situations partitioned according to the properties of an SWF-net.

1. $|p \bullet|$>1. Thus $|\bullet b|$=1, b can start after a completes and $a >_W b$ holds in the log. Remains to prove $\neg(a \times_W b)$. If $|\bullet p|$=1, b cannot start before a completes. If $|\bullet p|$>1, then b might start before a completes and $a \times_W b$ might hold. If this assumption is true, there should be one token in p under some marking M. If a completes under M, a will produce one token for p and there would be two tokens in p. We get a contradiction, thus $\neg(a \times_W b)$ holds. Since $a >_W b$ and $\neg(a \times_W b)$, we conclude $a \rightarrow_W b$.
2. $|p \bullet|$=1. If $|\bullet b|$=1, the proof is as before. If $|\bullet b|$>1, then $|\bullet p|$=1. b cannot start before a completes and $\neg(a \times_W b)$. Before a completes, there should be a marking M such that M covers all other input places of b except p. If not, there should be one path leading from a to the remainder input places of b. Thus p becomes an implicit place connecting a and b, which violates the SWF-net requirement. Under the marking M, when a completes, b can start immediately. So $a >_W b$ holds and we conclude $a \rightarrow_W b$.

Theorem 3. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b \in T$:*

1. *If $a \bullet \cap b \bullet \neq \emptyset$, then $a \nparallel_W b$.*
2. *If $\bullet a \cap \bullet b \neq \emptyset$, then $a \nparallel_W b$.*

Proof. Assume $a \parallel_W b$ in both situations, we will show that this can lead to a contradiction respectively for the following two parts.

1. Assume a place $p \in a \bullet \cap b \bullet$. For $a \parallel_W b$, there should at least be one "overlapping sequence" in the log, i.e., $a \times_W b$. Since the COMPLETE event of a task may occur at any time after the corresponding START event, there may be a snippet $(a, 1)(b, 1)$ or $(b, 1)(a, 1)$ in some trace. In this case, there will be a marking M that does not cover p, under which both a and b have started and can complete immediately. Thus p will contain at least two tokens after a and b complete and the net is not safe. So we get a contradiction and $a \nparallel_W b$ holds.
2. Assume a place $p \in \bullet a \cap \bullet b$. For $a \parallel_W b$, there should be at least a sequence $a \times_W b$ in the log. There will be a marking M of the net under which p is covered and a or b can start. (Note that $| \bullet a| = | \bullet b| = 1$.) But after a or b starts, the only token in p is consumed and the other transition can not start. The other one will wait until there is a token in p again. So a snippet of $(a, 0)(c, 1)(b, 0)$ or $(b, 0)(c, 1)(a, 0)$ will appear in some trace in the log. However, the COMPLETE event of c may start before the START event of a or b. Under the marking M, $(c, 1)$ may occur just before $(a, 0)$ or $(b, 0)$ and thus there will be two tokens in the place p, i.e., the net is not safe. So we get a contradiction and $a \nparallel_W b$ holds.

Theorem 4. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any $a, b, c \in T$:*

1. *If $a \rightarrow_W c$, $b \rightarrow_W c$ and $a \nparallel_W b$, then $a \bullet \cap b \bullet \cap \bullet c \neq \emptyset$.*
2. *If $c \rightarrow_W a$, $c \rightarrow_W b$ and $a \nparallel_W b$, then $c \bullet \cap \bullet a \cap \bullet b \neq \emptyset$.*

Proof. Now we should prove the above two sub theorems respectively.

1. From Theorem 1 and $a \rightarrow_W c$, we deduce $a \bullet \cap \bullet c \neq \emptyset$. We assume a place $p_1 \in a \bullet \cap \bullet c$. Similarly, we assume a place $p_2 \in b \bullet \cap \bullet c$. To prove the theorem, we make an assumption that $a \bullet \cap b \bullet \cap \bullet c = \emptyset$, i.e., for any p_1 and p_2, $p_1 \neq p_2$. Because the net N is an SWF-net, $|p_1 \bullet| = |\bullet p_1| = |p_2 \bullet| = |\bullet p_2| = 1$ holds. Thus a, b and c will execute the same number of times. If a starts before b starts, a always completes before b starts, i.e., b can only start after a completes in this situation ($a \nparallel_W b$). There should be at least one path L_{ab} leading from a to b on the net. After a completes, a will produce tokens for the first place on L_{ab}. Similarly, one path L_{ba} leads from b to a and after b completes, b will produce tokens for the first places on L_{ba}. After a and b complete successively, some tokens are left on L_{ba}. There should be some transitions not on L_{ba} which consume the left tokens. Since N is an SWF-net, these

transitions all have one input place on L_{ba} and then these transitions cannot include c (c has at least two input places p_1 and p_2). Once the only input place contains a token, they may consume it (free choice). Similarly, there should be some transitions not on L_{ab} which consume the left tokens too. Thus there will be a marking M of N, which covers p_1 only (a executes first and b still has to execute next) but does not cover p_2 because the tokens on L_{ab} have been consumed by other transitions (neither b nor c) and b, c could not execute this time, i.e., a deadlock occurs. Therefore, the net is not sound and we get a contradiction. Thus the assumption is wrong and p_1 is the same as p_2, i.e., $a \bullet \cap b \bullet \cap \bullet c \neq \emptyset$.

2. From Theorem 1 and $c \rightarrow_W a$, we derive $c \bullet \cap \bullet a \neq \emptyset$. We assume a place $p_1 \in c\bullet \cap \bullet a$. Similarly, we assume a place $p_2 \in c\bullet \cap \bullet b$. To prove the theorem, we make an assumption that $c\bullet \cap \bullet a \cap \bullet b = \emptyset$, i.e., for any p_1 and p_2, $p_1 \neq p_2$. According to $c \rightarrow_W a$, after c completes, there should be a marking M under which a can start and both p_1 and p_2 contain one token. In this situation, $p_1 \notin \bullet b$ and $p_2 \notin \bullet a$ hold. After a starts, a consumes the only token in p_1 and there is still a token in p_2. Now let us investigate what will happen to b and for this purpose we distinguish two situations: (i) $|p_2 \bullet| > 1$ and (ii) $|p_2 \bullet| = 1$.
Assume $|p_2 \bullet| > 1$. Thus we get $|\bullet b| = 1$. Because the only input place of b contains one token, b can start immediately. There will be at least one sequence in the log containing the snippet $(a, 0)(b, 0)$ and $a \times_W b$. We get the relation $a \parallel_W b$, which is conflicting with the premise $a \nparallel_W b$. Therefore we get a contradiction.
Assume $|p_2 \bullet| = 1$. If $|\bullet b| = 1$, the situation is similar to (i) and we can get a contradiction. If $|\bullet b| > 1$, b must start finally because there is a token in one of its input places which can only be consumed by b. Because $a \nparallel_W b$ and a has started, b can only start after a completes, i.e., the start of b is dependent on the completion of a. Thus there is a path L_{ab} leading from a to b. Similarly, there is a path L_{ba} leading from b to a. The remainder of the proof is similar to the situation considered before. Again, we get a contradiction.
As a consequence, p_1 and p_2 must be the same place and $c \bullet \cap \bullet a \cap \bullet b \neq \emptyset$.

Theorem 5. *Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. For any two task sets PS and SS, such that $PS \subseteq T$, $SS \subseteq T$: $\forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b$, $\forall_{a1, a2 \in PS} a1 \nparallel_W a2$ and $\forall_{b1, b2 \in SS} b1 \nparallel_W b2$ iff $\exists_{p \in P} \forall_{a \in PS} \forall_{b \in SS} a \bullet \cap \bullet b = \{p\}$.*

Proof. We should prove the theorem in both directions.

1. Assume $\exists_{p \in P} \forall_{a \in PS} \forall_{b \in SS} a \bullet \cap \bullet b = \{p\}$. Using Theorem 2, it is easy to see that $\forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b$. Using Theorem 3 we can also show that the elements of PS and the elements in SS cannot be in parallel.
2. Assume $\forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b$, $\forall_{a1, a2 \in PS} a1 \nparallel_W a2$ and $\forall_{b1, b2 \in SS} b1 \nparallel_W b2$. From $\forall_{a \in PS} \forall_{b \in SS} a \rightarrow_W b$ and Theorem 1, we derive that $a \bullet \cap \bullet b \neq \emptyset$ for any $a \in PS$ and any $b \in SS$. From the property of an SWF-net, we get $|a \bullet \cap \bullet b| = 1$. From $\forall_{a1, a2 \in PS} a1 \nparallel_W a2$ and $a1 \rightarrow_W b$ and $a2 \rightarrow_W b$, we get

$a1\bullet \cap a2\bullet \cap \bullet b \neq \emptyset$ (Theorem 4) where b can be any task in SS. Note that $|a1\bullet \cap a2\bullet \cap \bullet b| = 1$. Hence we can deduce that $a1\bullet \cap \bullet b = a2\bullet \cap \bullet b = \{p\}$ for some p and complete the proof.

Theorem 6. *Let N be a sound SWF-net and let W be a complete event log of N. $\beta(W) = N$ modulo renaming of places, i.e., the discovered model matches the original model after renaming places.*

Proof. Let $N = (P, T, F)$ and $\beta(W) = N_W = (P_W, T_W, F_W)$. Based on the completeness of W and mining step 1 of the β algorithm, we get $T_W = T$. For the source and sink places (i.e., i and o) of N, there are the source and sink places i_W and o_W of N_W such that $i_W\bullet = i\bullet \wedge \bullet o_W = \bullet o$ and vice versa. Remains to prove that the "internal places" of the two Petri nets N and N_W match.

1. First we prove that $\forall_{p \in P \setminus \{i,o\}} \exists_{p_W \in P_W \setminus \{i_W, o_W\}} \ \bullet p_W = \bullet p \wedge p_W\bullet = p\bullet$. According to the β-algorithm and Theorem 5, we know that $<\bullet p, p\bullet> \in X_W$ and $\exists_{p_W \in P_W} \ \bullet p \subseteq \bullet p_W \wedge p\bullet \subseteq p_W\bullet$, i.e., $<\bullet p_W, p_W\bullet> \in Y_W$. Assume that $\exists_{t' \in T} t' \in \bullet p_W \wedge t' \notin \bullet p$. From Theorem 4, we get that $\forall_{t_P \in \bullet p} \forall_{t_S \in p\bullet} t_P\bullet \cap t'\bullet \cap \bullet t_S \neq \emptyset$. Assume that $p' \in t_P\bullet \cap t'\bullet \cap \bullet t_S$. Because $t' \notin \bullet p$, we know that $p' \neq p$. Therefore for $\forall_{t_S \in p\bullet} \exists_{p' \in \bullet t_S} \ |\bullet t_S| > 1 \ \wedge \ |\bullet p'| > 1$. This violates the second requirement of an SWF-net and we get a contradiction. If we assume that $\exists_{t' \in T} \ t' \in p_W\bullet \wedge t' \notin p\bullet$, we can still get a similar contradiction. Therefore we prove the result in one direction.
2. Finally, we prove $\forall_{p_W \in P_W \setminus \{i_W, o_W\}} \exists_{p \in P \setminus \{i,o\}} \ \bullet p = \bullet p_W \wedge p\bullet = p_W\bullet$. According to the β-algorithm and $p_W \in P_W$, we know that $<\bullet p_W, p_W\bullet> \in Y_W$. Theorem 5 can be used to show that $\exists_{p \in P} \forall_{a \in \bullet p_W} \forall_{b \in p_W\bullet} \ a\bullet \cap \bullet b = \{p\}$. Therefore we deduce $\bullet p_W \subseteq \bullet p \wedge p_W\bullet \subseteq p\bullet$. Assume that $\exists_{t' \in T} t' \in \bullet p \wedge t' \notin \bullet p_W$. Using Theorem 2 and Theorem 3 we can show that $\forall_{t \in \bullet p_W} \ t' \nparallel_W t$ and $\forall_{t \in p_W\bullet} \ t' \rightarrow_W t$. Therefore we can prove that $<\bullet p_W \cup \{t'\}, p_W\bullet> \in Y_W$ and thus obtain a contradiction. If we assume that $\exists_{t' \in T} t' \in p\bullet \wedge t' \notin p_W\bullet$, we can also get a contradiction, thus complete the proof.