Math. Struct. in Comp. Science (2009), *vol.* 19, *pp.* 1091–1124.
doi:10.1017/S0960129509990132

Process discovery and Petri nets[†]

NADIA BUSI[‡] and G. MICHELE PINNA[§]

[‡]*Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy*
[§]*Dipartimento di Matematica e Informatica, Università di Cagliari, Italy*
Email: gmpinna@unica.it

Received 17 Mar 2008; revised 19 March 2009

In memory of Nadia Busi

The aim of the research domain known as process mining is to use process discovery to construct a process model as an abstract representation of event logs. The goal is to build a model (in terms of a Petri net) that can reproduce the logs under consideration, and does not allow different behaviours compared with those shown in the logs. In particular, process mining aims to verify the accuracy of the model design (represented as a Petri net), basically checking whether the same net can be rediscovered. However, the main mining methods proposed in the literature have some drawbacks: the classical α-algorithm is unable to rediscover various nets, while the region-based approach, which can mine them correctly, is too complex.
In this paper, we compare different approaches and propose some ideas to counter the weaknesses of the region-based approach.

1. Introduction

Process mining is a relatively new and rather active research domain[¶]. The goal of process mining is to extract information about processes from transaction logs. It assumes that it is possible to record events in such a way that:

(i) each event refers to an *activity* (that is, a well-defined step in the process);
(ii) each event refers to a *case* (that is, a process instance);
(iii) each event can have a *performer* also referred to as the *originator* (the actor executing or initiating the activity).

Table 1 shows an example of a log involving 19 events, 5 activities and 6 originators. The information contained in process logs (transaction logs) is rarely complete, but, for the purposes of this paper, we assume that we have complete information. Thus, in the set of action sequences we will consider, all the possible sequences are represented *somehow*.

[†] This paper is based on a number of papers Nadia and I wrote together, and on various discussions we had. To be precise, the papers are Busi and Pinna (2006b; 2006a; 1998) and van Dongen *et al.* (2007), the last of these being coauthored with Boudewijn van Dongen and Wil M. P. van der Aalst.

[¶] A complete overview of recent process mining research is beyond the scope of this paper: for an introduction to this topic, consult the web pages at http://www.processmining.org for a complete overview and van der Aalst *et al.* (2003) and van der Aalst and Weijters (2004), or the proceedings of the *Business Process Managment Conference*.

case id	activity id	originator	case id	activity id	originator
case 1	activity A	John	case 5	activity A	Sue
case 2	activity A	John	case 4	activity C	Carol
case 3	activity A	Sue	case 1	activity D	Pete
case 3	activity B	Carol	case 3	activity C	Sue
case 1	activity B	Mike	case 3	activity D	Pete
case 1	activity C	John	case 4	activity B	Sue
case 2	activity C	Mike	case 5	activity E	Clare
case 4	activity A	Sue	case 5	activity D	Clare
case 2	activity B	John	case 4	activity D	Pete
case 2	activity D	Pete			

Table 1. *An event log (audit trail)*

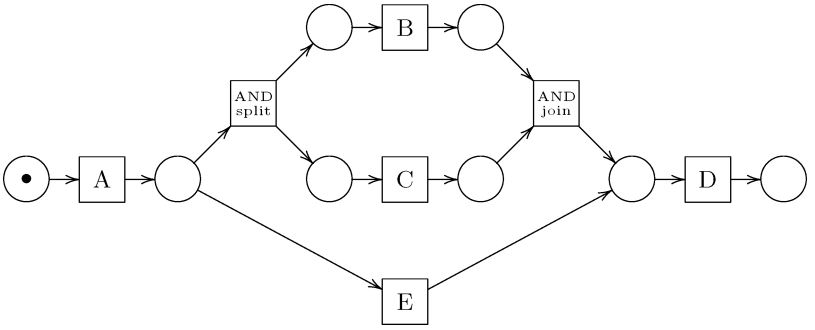


Fig. 1. The control flow structure of Table 1 expressed as a Petri net. The two transitions 'AND-split' and 'AND-join' are added for *routing purposes* and are originated using a top-down development of the net. They are not considered in the logs in Table 1

Event logs, such as the one shown in Table 1, are used as the starting point for mining. They can be looked at from different perspectives. We will focus on the so-called *process perspective* (Wohed *et al.* 2007; van Dongen *et al.* 2007), which considers the control flow (that is, the ordering of activities) shown in terms of a Petri net (*cf.* Reisig and Rozenberg (1998)), as in Figure 1. The goal of mining from this perspective is to find a good characterisation of all possible paths, for example, expressed in terms of a Petri net or Event-driven Process Chain (EPC) (IDS Scheer 2002; Keller and Teufel 1998), possibly exploiting the parallelism and choices that are flattened in the paths. Information concerning things like originators is not considered from the process perspective. Hence, we start from the logs in Figure 1, which were obtained from Table 1 (we indicate the maximal traces): `ABCD, ACBD, AED` (as we will see later, prefixes of these traces are also considered).

Mining a net from a suitable representation of its behaviour is not a new issue, and several approaches have been developed to this end (some of them, which can be related to process mining, will be reviewed briefly in Section 6). What is new here is the perspective we adopt, and the suitable class of nets we consider, that is, the so-called *workflow nets*, which we will also try to characterise through the nets we can mine from suitable logs.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

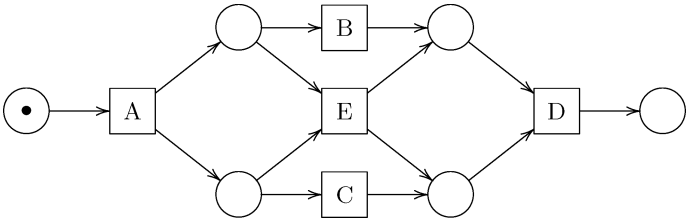


Fig. 2. The mined net of Figure 1. The transitions 'AND-split' and 'AND-join' are not found as they are not registered in the logs

Our starting point has been the weaknesses of the classical mining method (the so-called α-algorithm, which was developed within the business process community (van der Aalst *et al.* 2003; van der Aalst *et al.* 2004)). To provide cures for these weaknesses, we considered a region-based approach in Busi and Pinna (2006b). We began by associating a suitable *transition system* to the logs. Though some other approaches to workflow mining, which were based on regions over languages (the ones defined by the logs), have been proposed since (we will review them briefly in Section 6), we still consider the intermediate step (the construction of the transition system) quite useful because:

(i) It may help us understand why the desired result is not achieved in some cases (for example, the case of loops of length one, reviewed in Section 6).
(ii) The construction of the transition system can lead to better results when we have incomplete or noisy information, as it allows us to sort out problems *locally*, as suggested in van der Aalst *et al.* (2008) from a business process perspective and in Carmona *et al.* (2008) from a net perspective.
(iii) It is particularly suited to the incremental mining we will discuss in this paper.

Figure 2 shows the net we will be mining using virtually all the mining methods (including the α-algorithm) discussed here.

The theory of regions (Ehrenfeucht and Rozenberg 1989; Badouel and Darondeau 1998) establishes a connection between Petri nets and representations of their behaviour, like transition systems or languages, making the so-called *net synthesis* possible. The goal of the theory of regions applied to nets is to find a Petri net that exhibits an isomorphic behaviour to the one we are considering, possibly pointing out a characterisation of such behaviour. When it is represented as a transition system, the idea is to find a net whose marking graph is isomorphic or bisimilar to the given transition system; when it is represented as a language, the search is for a net exhibiting the same traces.

The intuition behind the *mining of a net* from a transition system is that states of a transition system, which should represent the markings of a net, embody enough information about which places are actually marked at that state. Hence, the property of a condition holding can be discovered simply by grouping all the states where this condition holds in a suitable way. Such a set of states is called a *region*, and it displays uniform *behaviour* with respect to the events of a transition system, that is, if an event e originates from a state that is not in the region and then *enters* the region, then none of the states where e can occur belong to that region, but all the states that are a result of

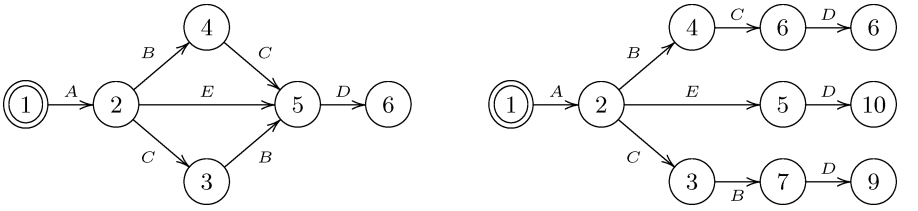


Fig. 3. Two transition systems corresponding to the net in Figure 2

event e do belong to that region. Furthermore, if an event e originates from a state in a region and *leaves* it, then all the states where e can occur belong to that region, but none of the states that are a result of event e belong to that region. Thus a region represents a condition that is the result of the events entering the region and that triggers those events leaving the region. A region can be alternatively defined as a pair of mappings (from the states and from the events of the transition system into $\mathbb{N}$ and $\mathbb{Z}$, respectively), the first representing the number of tokens of the place corresponding to regions assigned to each reachable marking (the states of the transition system), and the second characterising the transitions consuming, producing or leaving the tokens intact. This different view of regions can be lifted easily to words (and hence to the language generated by a Petri net), as has been done in Darondeau (1998) and Lorenz *et al.* (2007) among others.

In order to find the proper net, that is, a net whose behaviour is *described* by the transition system, the regions of the transition system should fulfill some properties. First, there should be enough regions to guarantee that two different states of the transition system can be distinguished by a region (representing a condition holding in one state but not in the other), and second, if an event does not leave a state, then there is a reason for this, and this reason can effectively be found by examining the regions. However, as we will see when discussing the weaknesses of other mining approaches, the inspection of the reasons hindering the fulfillment of these properties can be helpful in mining the correct net.

To see how the region-based approach works, consider the transition system on the left-hand side of Figure 3 and the associated net of Figure 2. States 2 and 4 can be grouped to form a region, as can 3 and 5, 4 and 5, and 2 and 3. Finally, two other regions can be obtained by considering states 1 and 6 individually. These regions satisfy the two properties mentioned earlier, and the net obtained using these regions is the one shown in Figure 2. If we consider the transition system on the right-hand side of Figure 3, the separation properties do not hold, for instance states 5 and 7 cannot be separated, but the net can still be mined correctly.

The notion of region can also be applied to other kinds of transition systems (for example, Step Transition Systems (Mukund 1992)). Furthermore, it should be observed that it is always possible to synthesise a net from a transition system using a set of regions. However, the result is *useful* only if the synthesis is done using a *complete* description of the net, that is, in terms of process mining, a transition system describing all possible behaviours of a process.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Process mining is useful for several reasons, and we will now briefly review those we consider most relevant:

(i) It could be used as a tool for finding out how people and/or procedures really work. Consider, for example, processes supported by an ERP system like SAP (such as, a procurement process). In such a system, all the transactions appear in logs, but in many cases they do not enforce a specific way of working. In such an environment, process mining could be used to gain an insight into the actual process (van der Aalst *et al.* 2004).

(ii) Process mining could be used to check the accuracy of the design by comparing the actual process with some predefined process. In many situations there is a descriptive (and sometimes prescriptive) process model specifying (and sometimes prescribing) how people and organisations are assumed/expected to work. By comparing the descriptive or prescriptive process model with the discovered model, discrepancies between the two can be detected and used to improve the process.

(iii) Additional benefits of process mining are that information about the way people and/or procedures really work and differences between actual processes and predefined processes can be used to trigger Business Process Re-engineering (BPR) efforts or to configure *process-aware information systems* (for example, workflow, ERP and CRM systems).

With all this is in mind, the accuracy of the mining methods and their extendibility are quite crucial issues, and region-based mining provides these advantages.

In this paper we mine nets out of transition systems obtained by the logs of workflow nets. This intermediate step (translating the logs into a transition system) could be avoided, as previously pointed out. However, we believe that more information can be gained using transition systems. Consider, for instance, logs that are produced by a system where the same activity is done in two different places. This information cannot be mined using a language approach, whereas it can be mined if we look for a way that splits the action into two (equally labelled) transition systems. Furthermore, inspecting the counterexamples preventing the satisfaction of the separation properties can cast light on how to solve them, thereby gaining insight into the processes to be discovered.

The paper is organised as follows. In the next Section, we will review the main notions concerning transition systems and Petri nets. In Section 3, we will clarify what workflow nets are and what the mining problem is formally. In Section 4, we will illustrate the region-based approach to mining, which is then fully developed and applied in Section 5 to the case of workflow nets. The approach is compared with other mining methods (including some based on regions) in Section 6, while in Section 7, an incremental method for mining workflow nets is proposed. Finally, Section 8 provides conclusions and points out some possible further developments of the results and ideas presented in this paper.

2. Background

In this section we recall the basic notions that we will use in the paper concerning transition systems, (safe) Place/Transition Petri nets and languages.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

We use $\mathbb{N}$ to denote the set of natural numbers including 0. Given a finite set S, a *multiset over* S is a function $m : S \to \mathbb{N}$. The set of all multisets over S is denoted by $\mathcal{M}(S)$. The *multiplicity* of an element s in m is the natural number $m(s)$. We write $m \subseteq m'$ if $m(s) \leqslant m'(s)$ for all $s \in S$. The operator $\oplus$ denotes *multiset union*:

$$(m \oplus m')(s) = m(s) + m'(s) \text{ for all } s \in S.$$

The operator $\setminus$ denotes *multiset difference*:

$$(m \setminus m')(s) = max\{m(s) - m'(s), 0\}.$$

We say that $s \in m$ if $m(s) > 0$. If $X \subseteq S$, with an abuse of notation, we use X to denote the multiset $X(s) = 1$ if $s \in X$ and $X(s) = 0$ otherwise.

2.1. *Transition systems*

We assume that a finite set of *actions* $\mathcal{A}ct$ is given. A *(finite) labelled graph* over an alphabet $\mathcal{A}ct$ is the triple $G = (V, \mathcal{A}ct, \rightarrow)$ where V is a (finite) set of *vertices* and $\rightarrow \subseteq V \times \mathcal{A}ct \times V$ is a (finite) set of *labelled edges*. A *rooted* labelled graph is a labelled graph with a distinguished initial state v_0.

A path between two vertices v and v' is a sequence of vertices $v_0, v_1, \ldots, v_n$ such that either $v = v_0 = v' = v_n$ or $v = v_0$, $v' = v_n$ and $(v_i, a_i, v_{i+1}) \in \rightarrow$ for each $i \geqslant 0$; if the graph is rooted, we say that a vertex v is reachable if and only if there exists a path from the initial vertex to v.

Definition 2.1. Let $G = (V, \mathcal{A}ct, \rightarrow)$ and $G' = (V', \mathcal{A}ct, \rightarrow')$ be two graphs. A *morphism* f from G to G' is a mapping $f : V \to V'$ such that $(v, a, v') \in \rightarrow$ and

$$f(v) \neq f(v') \Rightarrow (f(v), a, f(v')) \in \rightarrow' .$$

If the graphs are rooted, then $f(v_0) = v'_0$.

Two graphs are *isomorphic* if and only if the morphism f is an isomorphism and

$$(v, a, v') \in \rightarrow \Leftrightarrow (f(v), a, f(v')) \in \rightarrow' .$$

We will consider labelled graphs that satisfy some additional constraints.

Definition 2.2. A *transition system* is the triple $A = (St, E, \rightarrow)$, where St is a set of *states*, E is a non-empty set of *events* and $\rightarrow \subseteq St \times E \times St$ is a *transition relation* such that:

— $\forall (s, e, s') \in \rightarrow, s \neq s'$

— $\forall e \in E \ \exists (s, e, s') \in \rightarrow$.

A *rooted* transition system is a tuple $A = (St, E, \rightarrow, s_0)$, where $(St, E, \rightarrow)$ is a transition system and $s_0 \in St$ is a distinguished *initial* state. Moreover, each state is reachable from the initial one, that is, for all $s \in St$ there exists a path from s_0 to s.

The first condition says that the transition system is *self loop free* and the second that each event determines at least a change of state. We will omit the word rooted when it is clear from the context.

The notion of graph morphisms can be specialised to transition systems as follows.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Definition 2.3. Let $A = (St, E, \rightarrow, s_0)$ and $A' = (St', E, \rightarrow', s'_0)$ be two transition systems. Then $f : St \rightarrow St'$ is a *G-morphism* if and only if it is a pointed graph morphism, that is, $f(s_0) = s'_0$ and if $(s, e, s') \in \rightarrow$ and $(f(s), e, f(s')) \in \rightarrow'$, then $(f(s_1), e, f(s'_1)) \in \rightarrow'$ for every $(s_1, e, s'_1) \in \rightarrow$. If f is a bijection and $(s, e, s') \in \rightarrow \Leftrightarrow (f(s), e, f(s')) \in \rightarrow'$, then A and A' are isomorphic ($A \cong A'$).

A G-morphism represents the fact that A' can *partially* simulate A, that is, whenever A can evolve, A' can also, provided the move is defined (and if it is not, then the two states become indistinguishable). The third requirement is to ensure that the simulation is uniform on an event.

We can now state the notion of bisimulation between transition systems.

Definition 2.4. A bisimulation $\mathscr{R}$ between two transition systems $A = (St, E, \rightarrow, s_0)$ and $A' = (St', E, \rightarrow', s'_0)$ is a relation $\mathscr{R} \subseteq St \times St'$ such that:

— $(s_0, s'_0) \in \mathscr{R}$;
— if $(s, s') \in \mathscr{R}$ and $(s, e, s_1) \in \rightarrow$, then $(s', e, s'_1) \in \rightarrow'$ and $(s_1, s'_1) \in \mathscr{R}$;
— if $(s, s') \in \mathscr{R}$ and $(s', e, s'_1) \in \rightarrow'$, then $(s, e, s_1) \in \rightarrow$ and $(s_1, s'_1) \in \mathscr{R}$.

A bisimulation $\mathscr{R}$ is said to be *functional* if and only if it is a function with respect to the first argument.

Finally, we can point out a useful property of functional bisimulation.

Proposition 2.1. A functional bisimulation is a *G*-morphism.

2.2. *Petri nets*

We now recall some definitions and notation used with Petri nets.

Definition 2.5. A net is a tuple $N = (S, T, F)$, where:

— S and T are the (finite) sets of *places* and *transitions*, such that $S \cap T = \varnothing$; and
— $F \subseteq (S \times T) \cup (T \times S)$ is the flow relation.

A multiset over the set S of places is called a *marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ tokens. Let $x \in S \cup T$. The *preset* of x is the set ${}^\bullet x = \{y \mid F(y, x)\}$. The *postset* of x is the set $x^\bullet = \{y \mid F(x, y)\}$. The preset and postset functions are generalised in the obvious way to a set of elements:

$$\text{if } X \subseteq S \cup T \text{ then } {}^\bullet X = \bigoplus_{x \in X} {}^\bullet x \text{ and } X^\bullet = \bigoplus_{x \in X} x^\bullet.$$

A transition t is enabled at marking m if ${}^\bullet t \subseteq m$. The firing (execution) of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is usually written as $m[t\rangle m'$.

Definition 2.6. A *net system* is a pair (N, m_0), where N is a net and m_0 is a marking of N, called the *initial marking*.

With an abuse of notation, we use (S, T, F, m_0) to denote the net system $((S, T, F), m_0)$, and, furthermore, we will often just refer to it using the underlying net N.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

The set of *markings reachable from m*, denoted by $[m\rangle$, is defined as the least set of markings such that:

— $m \in [m\rangle$
— if $m' \in [m\rangle$ and there exists a transition t such that $m'[t\rangle m''$, then $m'' \in [m\rangle$.

The set of *firing sequences* is defined inductively as follows:

— m_0 is a firing sequence;
— if $m_0[t_1\rangle m_1 \dots [t_n\rangle m_n$ is a firing sequence and $m_n[t_{n+1}\rangle m_{n+1}$, then

$$m_0[t_1\rangle m_1 \dots [t_n\rangle m_n[t_{n+1}\rangle m_{n+1}$$

is a firing sequence.

Given a firing sequence $m_0[t_1\rangle m_1 \dots [t_n\rangle m_n$, we say $t_1 \dots t_n$ is a *transition sequence*. We use ε to denote the *empty transition sequence*.

Definition 2.7. Let (N, m_0) be a net system. We use $TrSeq(N, m_0)$ to denote the set of transition sequences of a net system (N, m_0).

We will omit the initial marking unless explicitly needed, so, for example, instead of writing $TrSeq(N, m_0)$, we will just write $TrSeq(N)$. We use σ to range over it. Let $\sigma = t_1 \dots t_n$; we use $m_0[\sigma\rangle m_n$ as an abbreviation for $m_0[t_1\rangle m_1 \dots [t_n\rangle m_n$.

Definition 2.8. The *marking graph* of a net system N is

$$MG(N) = ([m_0\rangle, T, \{(m, t, m') \mid m \in [m_0\rangle \wedge t \in T \wedge m[t\rangle m'\}, m_0).$$

It is obvious that a marking graph is a rooted transition system.

A net is *pure* if ${}^\bullet t \cap t^\bullet = \varnothing$ for all transitions $t \in T$. A net is *simple* if for all $x, y \in S \cup T$, we have if ${}^\bullet x = {}^\bullet y$ and $x^\bullet = y^\bullet$, then $x = y$. A net is *acyclic* if F^+, the transitive closure of the flow relation F, is an ordering relation. A net system is *safe* if each place contains at most one token in any marking reachable from the initial marking, that is, $m(s) \leqslant 1$ for all $s \in S$ and for all $m \in [m_0\rangle$. A net system is *reduced* if each transition can occur at least once: for all $t \in T$ there exists $m \in [m_0\rangle$ such that $m[t\rangle$. Given a net $N = (S, T, F)$, a place $s \in S$ is *implicit* if and only if the net $N' = (S \setminus \{s\}, T, F \cap ((S \setminus \{s\}) \times T) \cup (T \times S \setminus \{s\})))$ and N have isomorphic marking graphs. This means that any implicit place can be safely omitted.

In this paper we will only consider safe net systems that are pure, simple, reduced and without implicit places, unless we explicitly state otherwise.

A net $N = (S, T, F, m_0)$ is a state-machine if and only if $\forall t \in, |{}^\bullet t| = 1 = |t^\bullet|$. Let $N = (S, T, F, m_0)$ be net. N is decomposable if and only if there exist nets $N_i = (S_i, T_i, F_i, m_{0i})$, $i \in \{1, \dots, m\}$, such that $\forall i$ N_i is a state-machine net, $S = \bigcup S_i$, $T = \bigcup T_i$, F_i is the restriction of F to $(S_i \times T_i) \cup (T_i \times S_i)$ and m_{0i} is the restriction of m to S_i.

Finally, we introduce the notion of free-choice nets and illustrate some of their properties. A net (S, T, F) is *free choice* if and only if $\forall t, t' \in T$, either ${}^\bullet t \cap {}^\bullet t' = \varnothing$ or ${}^\bullet t = {}^\bullet t'$. This static requirement enforces the requirement that if two transitions t, t' are enabled at a marking m, then for all the markings m of the net, we have $m[t\rangle \Leftrightarrow m[t'\rangle$.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

A well-known result states that a free-choice net can be decomposed into suitable state-machines, called S-components. An S-component of a free-choice net (S, T, F) is a state-machine net (S', T', F') such that

— $S' \subseteq S$, $T' \subseteq T$ and $F' = F \cap ((S' \times T') \cup (T' \times S'))$, and
— ${}^\bullet s \cup s^\bullet \subseteq T'$,

where ${}^\bullet s$ and $s^\bullet$ are defined with respect to F and not to F' (the restriction of F to S' and T'). Each S-component can be constructed as follows: start from a place s and add all the transitions in ${}^\bullet s \cup s^\bullet$, then, for each added transition t, add a place in $t^\bullet$ guaranteeing that the net remains a state-machine one. This procedure converges and generates all the S-components. If the free-choice net has no implicit places, the decomposition into S-components is unique. The interested reader may consult Desel and Esparza (1995) for further details.

2.3. *Languages*

Let T be a finite alphabet. Then T^* is the set of all the words over T. We use ε to denote the empty word, and, given a word σ, a prefix of σ is the word $u \in T^*$ such that there exists a $v \in T^*$ and $\sigma = u{\cdot}v$. Given $L \subseteq T^*$, we say that L is *prefix-closed* if, given $\sigma \in L$, all prefixes u of σ belong to L.

The following obvious proposition relates net systems and languages.

Proposition 2.2. Let $N = ((S, T, F), m_0)$ be a safe net system. Then $TrSeq(N)$ is a prefix closed language.

3. Workflow nets

In this paper we consider a class of Petri nets, the so-called *workflow nets*. The key concept of workflow management is that of a task. As stated in van der Aalst (2004), 'A task is a piece of work to be done by one or more resources in a pre-determined time interval'. As we have already said, the control flow of tasks is the dimension we are most interested in, because the core of any workflow system is formed by the processes it supports. Clearly, a Petri net can be used to specify the routing of cases. Tasks are modelled by transitions and causal dependencies are modelled by places and arcs. In fact, a place corresponds to a condition that can be used as a pre- and/or post-condition for tasks. A Petri net that models the control-flow dimension of a workflow is called a workflow net (WF-net).

The following definition captures the structural constraints that a workflow net has to fulfill.

Definition 3.1. A Petri net $N = (S, T, F)$ is a *workflow net* (WF-net) if and only if:

— there is one *source* place $i \in S$ such that ${}^\bullet i = \varnothing$;
— there is one *sink* place $o \in S$ such that $o^\bullet = \varnothing$;
— every node $x \in S \cup T$ is on a path from i to o;

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

— it has no implicit places; and
— $(N, \{i\})$ is a safe net system.

A WF-net has one input place i and one output place o because any case handled by the procedure represented by the WF-net is created when it enters the workflow management system and it is deleted after its completion. The third requirement is added to ensure that there are no *dangling* conditions or tasks. It is easy to verify these requirements statically.

We do not need to explicitly specify the initial state of a workflow net, that is, its initial marking, as only the place i is assumed to be marked. We will often indicate the initial marking using the name of the source place i and the final marking using the name of the sink place o. Markings will also be called states.

The dynamic of a WF-net N should obey the following constraints:

— for every marking m reachable from the initial marking i there is a firing sequence starting at m and ending with o;
— there is at most one reachable marking with the place o marked; and
— the net system (N, i) is reduced.

If this is the case, we say that the WF-net N is *sound*, and here we will only consider sound workflow nets (SWF).

From a firing sequence of such net, if we look at the transition sequence only, we get an event trace. Let $\{w \in T^*\}$ be a finite set of words over an alphabet T – we call it a *workflow log* (intuitively a workflow (event) log represents a set of possible observation of the workflow management systems). We will assume that the workflow (event) log contains enough information to understand the relations among the tasks. We will also assume that we have a mining algorithm, that is, an algorithm that, starting from workflow logs (or a suitable representation of them) is able to find the net that originated the log. We can state the problem addressed by process mining as follows.

Definition 3.2. Let $N = (S, T, F)$ be a sound WF-net and β be a mining algorithm that maps workflow logs of N onto sound WF-nets. If for any complete workflow log W of N the mining algorithm returns a net N' such that $TrSeq(N') = TrSeq(N)$, then the β algorithm is able to rediscover N.

The following more restrictive definition of process mining is given in van der Aalst *et al.* (2003; 2004).

Definition 3.3. Let $N = (S, T, F)$ be a sound WF-net and β be a mining algorithm that maps workflow logs of N onto sound WF-nets. If for any complete workflow log W of N the mining algorithm returns N (modulo renaming of places), then the β algorithm is able to rediscover N.

When developing our approach, we will briefly discuss the conditions under which this more restricted mining view may be obtained.

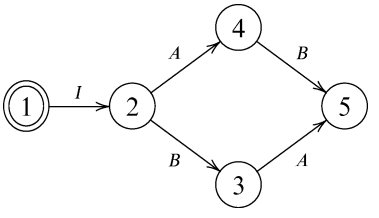
https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

4. Region-based synthesis of safe net systems

In this section we briefly recall the basic notions related to the theory of regions and the synthesis of (safe) net systems[†]. We will adapt the separation properties to the specific case of a safe net.

Given the marking graph G of a safe net system N, a region of G is basically a set of markings corresponding to the states where a real or potential place of N is marked. In other words, a region r groups together all the states of the graph in which a place r contains a token. Let r be a region of $MG(N)$. Consider a place s and a transition t such that $s \in {}^\bullet t$. Let $m \in r$ and assume that $m[t\rangle$: so s is marked in m, and after the firing of t, as we consider pure nets, s is no longer marked. Thus, we have $m \xrightarrow{t} m'$ in the marking graph, and $m'(s) = 0$, so $m' \notin r$. Moreover, if a state m is outside r, then t cannot happen in m, and the place s in ${}^\bullet t$ is empty, so we do not have t-labelled transitions exiting from s. To summarise, if $s \in {}^\bullet t$, then each t-labelled transition of the graph starts inside r and ends outside r. Analogously, if a transition t produces a token in s, that is, $s \in t^\bullet$, then each t-labelled transition in the graph has source outside r and target inside r. Suppose now that place s is unrelated to transition t, that is, $s \notin {}^\bullet t \cup t^\bullet$. If t fires in a state where s is marked, then place s is also marked after the firing of t, that is, if a t-labelled transition starts inside r, then it also ends inside r. Analogously, if t happens in a state where s is empty, then s still remains empty after the firing of t, that is, t-labelled transitions that start outside r also end outside r.

We can illustrate this notion with a small example. Consider the following transition system:



The set of states $\{1, 2, 4\}$ is such that each B-labelled transition exits it, whereas the set of states $\{4, 5\}$ is such that each A-labelled transition enters it, and, furthermore, each B-labelled transition is either inside the set or outside it. Thus t-labelled transitions have a uniform behaviour with respect to all the states in r: either all of them cross r exiting, or all of them cross r entering, or none of them cross r. This is the content of the following definition.

Definition 4.1. Let $TS = (St, E, \rightarrow)$ be a transition system. A set $r \subseteq St$ is said to be a *region* if and only if $\forall s_1 \xrightarrow{e} s_1', s_2 \xrightarrow{e} s_2'$ the following conditions hold:

— If $s_1 \in r$ and $s_1' \notin r$, then $s_2 \in r$ and $s_2' \notin r$.
— If $s_1 \notin r$ and $s_1' \in r$, then $s_2 \notin r$ and $s_2' \in r$.

[†] For a more detailed account, see Badouel and Darondeau (1998), Desel and Reisig (1996) and Ehrenfeucht and Rozenberg (1989).

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

It is easy to see that both St and $\varnothing$ are regions – they are called the *trivial* regions. We will use $Reg(TS)$ to denote the set of *non-trivial* regions of a transition system TS. The complementary set of a region is itself a region.

We now define the analogues of presets and postsets for events and regions.

Definition 4.2. Let $TS = (St, E, \rightarrow)$ be a transition system and $e \in E$. The preregionset and the postregionset of e are the sets of regions defined as follows:

$$\begin{aligned} {}^{\circ}e &= \{r \in Reg(TS) \mid \forall (s, e, s') \in \rightarrow : s \in r \wedge s' \notin r\} \\ e^{\circ} &= \{r \in Reg(TS) \mid \forall (s, e, s') \in \rightarrow : s \notin r \wedge s' \in r\}. \end{aligned}$$

Given a region r of TS, ${}^{\circ}r = \{e \in E \mid r \in e^{\circ}\}$ and $r^{\circ} = \{e \in E \mid r \in {}^{\circ}e\}$.

Definition 4.3. Let $N = (S, T, F, m_0)$ be a net system and let $s \in S$. We use r_s to denote the set of states of $MG(N)$ where s is marked:

$$r_s = \{m \in [m_0\rangle \mid m(s) = 1\}.$$

The following proposition explains the relation between the places of a net system and the regions of its marking graph.

Proposition 4.1. Let $N = (S, T, F, m_0)$ be a net system and let $s \in S$. The set r_s is a region of $MG(N)$. Furthermore, we have that ${}^{\bullet}s = {}^{\circ}r_s$ and $s^{\bullet} = r_s{}^{\circ}$.

A region does not always correspond to a place of the net, but may represent a potential place. The addition of such a potential place to the net system has no influence on its behaviour. In fact, consider a net system $N = (S, T, F, m_0)$, and let r be a region of $MG(N)$ such that

$$\forall s \in S : {}^{\circ}r \neq {}^{\bullet}s \text{ or } r^{\circ} \neq s^{\bullet}.$$

Let s_r be a place such that $s_r \notin S$. The net system $N^{+r} = (S', T, F', m_0')$ defined by

$$\begin{aligned} S' &= S \cup \{s_r\} \\ F' &= F \cup \{(s_r, t) \mid r \in {}^{\circ}t\} \cup \{(t, s_r) \mid r \in t^{\circ}\} \\ m_0' &= \begin{cases} m_0 \oplus \{s_r\} & \text{if } m_0 \in r \\ m_0 & \text{otherwise} \end{cases} \end{aligned}$$

is such that $MG(N)$ is isomorphic to $MG(N^{+r})$.

Thus, given a net sytem N, we can construct the *saturated* version of (the marking graph of) N, which is obtained by using all the non-trivial regions of $MG(N)$ as places. Note that the set $Reg(MG(N))$ is finite, as the set of non-trivial regions of a transition system is a subset of the powerset of the set of states of the transition system, and the set of states of the marking graph of a safe Petri net is finite.

Definition 4.4. Let $TS = (St, E, \rightarrow, s_0)$ be the marking graph of a net system. The net system $Sat(TS) = (S, T, F, m_0)$ is defined as follows:

— $S = Reg(TS)$, $T = E$ and $F = \{(r, e) \mid r \in {}^{\circ}e\} \cup \{(e, r) \mid r \in e^{\circ}\}$

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

— $m_0(r) = \begin{cases} 1 & \text{if } s_0 \in r \\ 0 & \text{otherwise.} \end{cases}$

The following proposition holds (Desel and Reisig 1996).

Proposition 4.2. Let N be a net system. The marking graph $MG(N)$ is isomorphic to $MG(Sat(MG(N)))$.

Definition 4.4 produces the 'largest' net system whose marking graph is isomorphic to a given transition system TS.

We now provide two conditions ensuring that the net system N constructed by taking the subset $R \subset Reg(TS)$ as set of places has a marking graph isomorphic to TS. The two conditions also allow us to characterise the set of transition systems that are isomorphic to the marking graph of a net system. The first condition ensures that two distinct states of TS will not be identified in the marking graph of the net system.

Definition 4.5. Let $TS = (St, E, \rightarrow, s_0)$ be a transition system and $R \subseteq Reg(TS)$. We say that R satisfies the state separation property (SSP) in TS if and only if

$$\forall s, s' \in St : s \neq s' \Rightarrow \exists r \in R : s \in r \Leftrightarrow s' \notin r.$$

We say that TS satisfies the state separation property if $Reg(TS)$ satisfies the state separation property in TS.

This property captures the intuition that for two markings to be different in a safe net, there must exist a place that is marked with a token in one of the two markings and empty in the other. Regions become the places of the synthesised nets, and if a state s belongs to a region, then the place corresponding to such a region is marked in s. Hence, if two states cannot be separated by any region, then they will be identified in the marking graph of the synthesised net; in other words, the net cannot generate all the states of the transition system.

The second condition ensures that if an event e cannot occur in a state s of TS, then e also cannot occur in the corresponding state in the marking graph of the net system. Suppose that the marking graph does not contain an e-labelled arc exiting from s. We use R_s to denote the marking such that $R_s(r) = 1$ whenever $s \in r$. Then we need that e is not enabled in the marking corresponding to s, R_s . For e not to be enabled at R_s, the following must hold:

— ${}^\bullet e \not\subseteq R_s$,

which means that there exists a region r such that $r \in {}^\bullet e$ and $r \notin R_s$, or, equivalently, $r \in {}^\bullet e$ and $s \notin r$.

Definition 4.6. Let $TS = (St, E, \rightarrow, s_0)$ be a transition system and $R \subseteq Reg(TS)$. We say that R satisfies the safe event state separation property (S-ESSP) in TS if and only if

$$\forall s \in St \ \forall e \in E : \text{ if } s \overset{e}{\not\rightarrow} \text{ then } \exists r \in R : r \in {}^\bullet e \wedge s \notin r$$

We say that TS satisfies the safe event state separation property if $Reg(TS)$ satisfies the safe event state separation property in TS.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

As the places of the constructed net correspond to the regions in R, and $s \in r$ means that place r is marked in state s, the isomorphism from TS to $MG(N)$ maps a state s on the set R_s of regions containing s. Note that the S-ESSP is a slight adaptation of the standard event state separation property for elementary net systems (see, for example, Desel and Reisig (1996)) to the context of safe net systems: the other condition in the original definition is related to the firing rule of elementary net systems, and is omitted here.

If a transition system TS satisfies SSP and S-ESSP, then it is isomorphic to the marking graph of the saturated net generated from TS (Badouel and Darondeau 1998).

Theorem 4.1. Let $TS = (St, E, \rightarrow, s_0)$ be a transition system. If TS satisfies SSP and S-ESSP, then TS is isomorphic to $MG(Sat(TS))$.

Regions are candidates for representing the places of the net we are synthesising. However, it can happen that not all the regions are necessary for the construction of a net with a given behaviour. For example, the trivial regions are useless. Now we show how to construct a net from an arbitrary set of regions.

Definition 4.7. Let $TS = (St, E, \rightarrow, s_0)$ be the marking graph of a net system and $R \subseteq Reg(TS)$. The net system $Gen(TS, R) = (S, T, F, m_0)$ is defined as follows:

— $S = R$, $T = E$ and $F = \{(r, e) \mid r \in {}^\circ e\} \cup \{(e, r) \mid r \in e^\circ\}$

— $m_0(r) = \begin{cases} 1 & \text{if } s_0 \in r \\ 0 & \text{otherwise.} \end{cases}$

As for the saturated version, we have the following result (see Badouel and Darondeau (1998), Bernardinello (1993) and Busi and Pinna (1997), among many others):

Theorem 4.2. Let TS be a transition system and $R \subseteq Reg(TS)$. If R satisfies SSP and S-ESSP in TS, then TS is isomorphic to $MG(Gen(TS, R))$.

The above definition allows us to construct all the safe and simple net systems whose marking graph is isomorphic to a given transition system, by taking all the subsets of regions that satisfy SSP and S-ESSP.

We now state a proposition connecting regions of transition systems that are related by a morphism (see Bernardinello (1993) for a proof).

Proposition 4.3. Let $TS = (St, E, \rightarrow, s_0)$ and $TS' = (St', E, \rightarrow', s_0')$ be two transition systems and f be a G-morphism between them. If r is a region of TS', then $f^{-1}(r)$ is a region of TS.

4.1. *Minimal regions*

The choice of which set of regions of a given transition systems should be used is rather crucial. We use $Min(Reg(TS))$ to denote the set of *minimal* regions with respect to set inclusion, that is, the set $\{r \mid r \in Reg(TS) \text{ and } \forall r' \in Reg(TS), r \neq r' \Rightarrow r' \not\subseteq r\}$. As any other region can be obtained using these regions (Bernardinello 1993; Busi and

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Pinna 1997), we have that $Min(Reg(TS))$ satisfies SSP and S-ESSP in TS and it can be used to synthesise a net.

4.2. *Transition systems bisimilar to marking graphs*

As we will see in the following, the transition system we will construct from logs is not a marking graph. Hence, we need to find out when the separation conditions we have seen so far are enough to guarantee the result.

Given a transition system $A = (St, E, \rightarrow, s_0)$ and two states x and y, the two states are *bisimilar* if and only if the subgraphs of A formed by the states that are reachable from x and y, respectively (and rooted in x and y), are bisimilar. We use $x \sim y$ to denote this fact.

Proposition 4.4. Given a transition system $A = (St, E, \rightarrow, s_0)$, if $x \not\sim y$ and x, y are not separated, then S-ESSP does not hold.

Proof. Assume $x \not\sim y$, then there exist $e_1, \ldots, e_n, e_{n+1}$ such that

(a) $x \xrightarrow{e_1} x_1 \ldots x_{n-1} \xrightarrow{e_n} x_n \xrightarrow{e_{n+1}} x_{n+1}$; and
(b) $y \xrightarrow{e_1} y_1 \ldots y_{n-1} \xrightarrow{e_n} y_n \not\xrightarrow{e_{n+1}}$.

We show that S-ESSP does not hold. Suppose the S-ESSP property holds. Then, as $y_n \not\xrightarrow{e_{n+1}}$, there must be a region r such that $r \in {}^\bullet e_{n+1}$ and $y_n \notin r$. As $x_n \xrightarrow{e_{n+1}} x_{n+1}$, we have $x_n \in r$. We now show that $x_{n-1} \in r$ also. Suppose $x_{n-1} \notin r$. As $x_{n-1} \xrightarrow{e_n} x_n$, we have $r \in e_n{}^\bullet$. As $y_{n-1} \xrightarrow{e_n} y_n$, by the definition of a region, we have $y_n \in r$. But as we are assuming that $y_n \notin r$, it follows that $x_{n-1} \in r$. Since

(a) $x_n, x_{n-1} \in r$,
(b) the arrow e_n does not cross r, and
(c) $y_n \notin r$,

we also have that $y_{n-1} \notin r$. Proceeding in the same way for $e_{n-1}, \ldots, e_1$, we get that $x \in r$ and $y \notin r$. Recalling that two states are separated if and only if there exists a region $\hat{r}$ *separating* them, we have that this is in contradiction with the assumption that x and y are not separated, so S-ESSP does not hold. □

From this proposition it follows that if S-ESSP holds, each pair of non-separated states are bisimilar.

Proposition 4.5. Let $TS = (St, E, \rightarrow, s_0)$ be a rooted transition system. If S-ESSP holds, the relation $\mathscr{R} = \{(s, R_s) \mid s \in St\}$ is a bisimulation from TS to $MG(Sat(TS))$.

Proof. Consider $Sat(TS) = (Reg(TS), E, F, m_0)$ where

$$F = \{(r, e) \mid r \in {}^\circ e\} \cup \{(e, r) \mid r \in e^\circ\}$$

$$m_0(r) = \begin{cases} 1 & \text{if } s_0 \in r \\ 0 & \text{otherwise.} \end{cases}$$

Let us check that $\mathscr{R}$ is a bisimulation. Clearly, $(s_0, R_{s_0}) \in \mathscr{R}$. Now we consider $s \xrightarrow{e} s'$ and assume that $(s, R_s) \in \mathscr{R}$. R_s is a marking and ${}^\bullet e \subseteq R_s$. So we have $R_s[e\rangle m'$ and

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

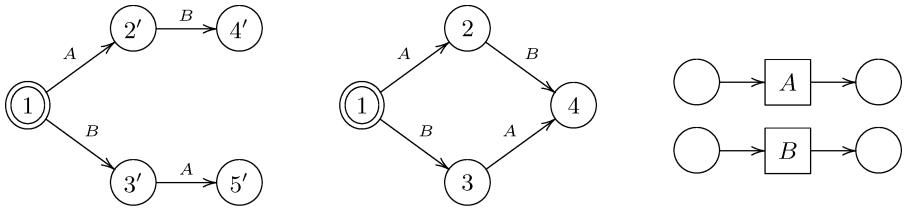


Fig. 4. A transition system TS, the marking graph of $Sat(TS)$ and the net $N = Sat(TS)$

$m' = (R_s \setminus {}^\bullet e) \oplus e^\bullet$, which is $R_{s'}$ and $(s', R_{s'}) \in \mathscr{R}$. The converse also holds, so $\mathscr{R}$ is a bisimulation □

Thus it is enough to verify the S-ESSP property on transition systems to guarantee that the synthesised net has a marking graph bisimilar to the original one. We can illustrate this result with a simple example. Consider the two transition systems in Figure 4. The bisimulation is as follows: $\{(1',1),(2',2),(3',3),(4',4),(5',4)\}$, which is easy to see with $Sat(TS)$ being the net N in the figure.

5. Region-based workflow mining

We begin this section by presenting a possible construction that produces a safe net system from a workflow log, and then show that such a technique works correctly on SWF nets. The motivation, as we have already said, is twofold: on the one hand, we want to apply the classical region-based approach to synthesise a net, adapting the technique to the case of workflow net, and, on the other hand, we want to achieve a characterisation of the nets that can be mined correctly.

5.1. *Synthesising a net system from a workflow log*

The first problem one has to face up to is related to the fact that the starting point for net synthesis (namely a transition system) is different from the starting point for process mining, which extracts a net system from a set of traces. As a first step, given a workflow log, we construct a corresponding transition system by gluing the common prefixes of the traces.

Given a workflow log $W \subseteq T^*$, we use $prefix(W)$ to denote the set of prefixes of the traces in W, that is, $prefix(W) = \{\sigma \mid \exists \tau : \sigma\tau \in W\}$. The *log graph* of a workflow log W, denoted $LG(W)$, is obtained by gluing together the greatest common prefixes of the set of traces in W.

Definition 5.1. Let $W \subseteq T^*$ be a workflow log. The *log graph* of W is defined as follows:

$$LG(W) = \{prefix(W), T', \{(\sigma, \sigma t) \mid \sigma t \in prefix(W)\}, \varepsilon\}$$

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

with

$$T' = \{t \in T \mid \exists \sigma, \tau : \sigma t \tau \in W\}.$$

Clearly, in most cases the transition system $LG(W)$ will not be isomorphic to the marking graph of the net exhibiting the set of logs W, as the transition systems generated by Definition 5.1 are basically rooted trees. However, it is an *unfolding* of this marking graph, where the paths are taken into account, and it is bisimilar to the marking graph.

Proposition 5.1. Let $N = ((S, T, F), m_0)$ be a safe net system and $TrSeq(N)$ be the associated log. Assume that each transition $t \in T$ appears at least in a sequence $\sigma \in LG(TrSeq(N))$. Then $LG(TrSeq(N))$ and $MG(N)$ are bisimilar.

Proof. Take any log $\sigma \in LG(TrSeq(N))$. Then $m_0[\sigma > m$ for some marking $m \in [m_0 >$. The relation $\{(\varepsilon, m_0)\} \cup \{(\sigma, m) \mid \sigma \in LG(TrSeq(N))$ and $m_0[\sigma > m\}$ is a bisimulation. Furthermore, it is a functional bisimulation. □

In any case, we are interested in constructing a net system whose transition sequences 'correspond' to the workflow log.

Definition 5.2. Let $W \subseteq T^*$ be a workflow log and $R \subseteq Reg(LG(W))$. The net system $LN(W, R) = (S, T', F, m_0)$ is defined as $Gen(LG(W), R)$.

As we are dealing with the unfolding of the marking graph of a net, we have to ignore the SSP as, when we consider traces, two states (that is, trace) might be indistinguishable with respect to regions. We first show that ignoring the SSP is not a limitation.

Proposition 5.2. Let N be a workflow net and $TrSeq(N)$ be the associated set of logs. Then the set of regions $Reg(LG(TrSeq(N)))$ satisfies the S-ESSP in $LG(TrSeq(N))$.

Proof. By Proposition 5.1, we know that there is a G-morphism from $LG(TrSeq(N))$ to $MG(N)$. Let us call it $f : LG(TrSeq(N)) \to MG(N)$. We know that $Reg(MG(N))$ satisfies the S-ESSP property. We prove now that $f^{-1}(Reg(MG(N)))$ satisfies the S-ESSP property in $LG(TrSeq(N))$, which means we will be done since

$$f^{-1}(Reg(MG(N))) \subseteq Reg(LG(TrSeq(N))).$$

Suppose there exists a state σ in $LG(TrSeq(N))$ and a transition t such that $\sigma t \notin TrSeq(N)$. Hence $\sigma \not\xrightarrow{t}$, but there is no region $r \in f^{-1}(Reg(MG(N)))$ such that $r \in {}^\bullet t$ and $\sigma \in r$. Now consider the marking m in $MG(N)$ that is reached firing σ. Clearly, $m \not\xrightarrow{t}$, and, as $Reg(MG(N))$ satisfies the S-ESSP property, there exists a region r' such that $r' \in {}^\bullet t$ and $m \notin r'$. But then we also have that $f^{-1}(r')$, which is a region of $LG(TrSeq(N))$, is such that $f^{-1}(r') \in {}^\bullet t$ and $\sigma \in f^{-1}(r')$, which contradicts the assumption. □

Proposition 5.3. Let N be a workflow net and $R \subseteq Reg(LG(TrSeq(N)))$ be a set of regions. If R satisfies S-ESSP in $LG(TrSeq(N))$, then $TrSeq(LN(TrSeq(N), R)) = TrSeq(N)$.

Proof. The proof is quite simple, since it is enough to observe that $LG(TrSeq(N))$ is bisimilar to $MG(N)$, hence also to $MG(LN(TrSeq(N), R)))$. The consequence of this is that the languages are the same. □

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

The fact that the mining based on minimal regions gives the proper result is stated in the following theorem, which is easy to prove using the propositions above. Before stating the first result of the paper, whose proof is an easy consequence of Proposition 5.3, we formalise the notion of completeness for the logs of a net.

Definition 5.3. Let $N = ((S, T, F), m_0)$ be a safe net and $W \subseteq TrSeq(N)$ be a set of logs. We say that W is *complete* if and only if $LG(W)$ and $MG(N)$ are bisimilar.

We are now ready to state the first result, which simply says that the region-based construction works well in the process mining setting by just looking at the safe event state separation property.

Theorem 5.1. Let N be a WF-net and W be a complete workflow log of N. Let $R \subseteq Min(Reg(LG(W)))$ be a set of minimal regions of $LG(W)$ satisfying the S-ESSP. Then $LN(W, R)$ is a net such that $TrSeq(N) = TrSeq(LN(W, R))$.

To see how our approach works, we will look again at the example in the introduction. Consider the WF-net in Figure 1, which is a sound WF-net, and let $W = \{\mathtt{ABCD}, \mathtt{ACBD}, \mathtt{AED}\}$ be a complete log. The log graph of W is

$$\begin{aligned} LG(W) \;=\; \{&\{\epsilon, \mathtt{A}, \mathtt{AB}, \mathtt{AC}, \mathtt{AE}, \mathtt{ABC}, \mathtt{ACB}, \mathtt{AED}, \mathtt{ABCD}, \mathtt{ACBD}\},\\ &\{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{AE}, \mathtt{D}, \mathtt{E}\},\\ &\{(\epsilon, \mathtt{A}), (\mathtt{A}, \mathtt{AB}), (\mathtt{A}, \mathtt{AC}), (\mathtt{A}, \mathtt{AE}), (\mathtt{AB}, \mathtt{ABC}), (\mathtt{AC}, \mathtt{ACB}), (\mathtt{AE}, \mathtt{AED}), (\mathtt{ABC}, \mathtt{ABCD}),\\ &\qquad\qquad (\mathtt{ACB}, \mathtt{ACBD})\},\\ &\epsilon\}. \end{aligned}$$

A set of minimal regions for this graph is

$$\begin{aligned} r_{\bullet\mathtt{A}} &= \{\epsilon\}\\ r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{B}, {}^\bullet\mathtt{E})} &= \{\mathtt{A}, \mathtt{AC}\}\\ r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{C}, {}^\bullet\mathtt{E})} &= \{\mathtt{A}, \mathtt{AB}\}\\ r_{(\mathtt{B}^\bullet, \mathtt{E}^\bullet, {}^\bullet\mathtt{D})} &= \{\mathtt{AE}, \mathtt{AB}, \mathtt{ACB}, \mathtt{ABC}\}\\ r_{(\mathtt{C}^\bullet, \mathtt{E}^\bullet, {}^\bullet\mathtt{D})} &= \{\mathtt{AE}, \mathtt{AC}, \mathtt{ABC}, \mathtt{ACB}\}\\ r_{\mathtt{D}\bullet} &= \{\mathtt{ABCD}, \mathtt{ACBD}, \mathtt{AED}\} \end{aligned}$$

(the region indexes show which activity they are related to and how).

It is easy to check that the regions

$$\{r_{\bullet\mathtt{A}}, r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{B}, {}^\bullet\mathtt{E})}, r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{C}, {}^\bullet\mathtt{E})}, r_{(\mathtt{B}^\bullet, \mathtt{E}^\bullet, {}^\bullet\mathtt{D})}, r_{(\mathtt{B}^\bullet, \mathtt{E}^\bullet, {}^\bullet\mathtt{D})}, r_{\mathtt{D}\bullet}\}$$

satisfy the S-ESSP property. Take, for instance, the state $\mathtt{AB}$. There is no arc labelled $\mathtt{E}$ exiting it, but $\mathtt{AB} \notin r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{B}, {}^\bullet\mathtt{E})}$ and $\mathtt{E} \in {}^\bullet r_{(\mathtt{A}^\bullet, {}^\bullet\mathtt{B}, {}^\bullet\mathtt{E})}$. The net constructed according to Definition 4.7 is the one shown in Figure 2.

Let us now consider the more restrictive case of mining given by Definition 3.3. The following proposition relates the regions of $LG(TrSeq(N))$ to those of the marking graph, and hence to the places of the net.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Proposition 5.4. Let $N = (S, T, F, m_0)$ be a WF-net, $MG(N)$ be its marking graph and W be a complete workflow log of N. Let $LG(W)$ be the transition system obtained by W, and $h = \{(\epsilon, m_0)\} \cup \{(\sigma, m) |\ m_0[\sigma > m\}$ be a relation on $LG(W)$ and $MG(N)$. Then h is a bisimulation and it is a functional bisimulation. Furthermore, if r is a region of $LG(W)$, then $h(r)$ is a region of $MG(N)$.

Proof. h is clearly a bisimulation, and, moreover, it is a functional bisimulation. Let r be a region of $LG(W)$. We show that $h(r)$ is a region of $MG(N)$. Consider $e \in^\bullet r$. Then $\forall s \xrightarrow{e} s'$ we have that $s \notin r$ and $s' \in r$. Now consider $h(r)$. Clearly, $h(s) \notin h(r)$ and $h(s') \in h(r)$ otherwise h would not be a bisimulation. We can apply similar resoning for $e \in r^\bullet$ to complete the proof. □

We can now state the following theorem, which establishes the adequateness of the region-based mining approach according to the more restrictive Definition 3.3 for a suitable class of WF-net.

Theorem 5.2. Let N be a sound and free-choice WF-net and W be a complete workflow log of N. Let $Min(Reg(LG(W)))$ be the set of minimal regions of $LG(W)$. Then $Gen(LG(TrSeq(N)), Min(LG(TrSeq(N)))) = N$ modulo renaming of places.

Proof. First we note that soundness guarantees that the set of transitions of N and of the synthesised net $Gen(LG(TrSeq(N)), Min(LG(TrSeq(N))))$ are the same.

Second, it is clear that to each place of the net N there corresponds a region in $MG(N)$, which is minimal (as the net N has no superfluous places). Due to the functional bisimilarity of $LG(TrSeq(N))$ and $MG(N)$, we have that each minimal region of $LG(TrSeq(N))$ is also a minimal region of $MG(N)$ and hence is a place of N. We now assume that there is a set of regions smaller than the set of minimal regions. Clearly, there must be a region, corresponding to a place, that is obtained by combining some minimal regions. But as the net is safe and free choice, this would mean that the net has several decompositions, which is not the case for free-choice nets. □

Consider again the net in Figure 1, which is a free-choice net. Its minimal regions are exactly those we are looking for to mine exactly the *same* net.

The main problem with this approach is knowing how to calculate regions, and, in particular, minimal regions. However, if we can calculate all the minimal regions of a transition system obtained as in Definition 5.1, we do not have to worry about the S-ESSP property, as it holds, provided the logs are complete. It is easy to adapt symbolic algorithms, like those presented in Cortadella *et al.* (1998) and Gorgônio *et al.* (2007), to this setting.

We conclude this section by observing that it is always possible to mine the net according to Definition 3.3.

Theorem 5.3. Let $N = (S, T, F)$ be any sound WF-net and W be a complete workflow log of N. Then it is possible to mine the net according to Definition 3.3, that is, there exists $R \subseteq Reg(LG(W))$, a set of regions of $LG(W)$, that satisfy the S-ESSP property and such that $Gen(LG(W), R) = N$ modulo renaming of places.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Proof. Soundness guarantees that each transition in N appears in W. It is then enough to observe that there is the set of regions R corresponding to the places of N (by Proposition 4.1, we know that to each place $s \in S$ there is a corresponding region r_s in the marking graph, and that $h : LG(W) \to MG(N)$ is a morphism. Hence $h^{-1}(r_s)$ is a region in $LG(W)$), which obviously satisfies the S-ESSP in $LG(W)$. □

6. Comparison with other approaches

In this section we compare our approach with other approaches to process mining, and briefly discuss the relative advantages.

We first review the classical mining method, the so-called α-algorithm.

6.1. *α-algorithm*

The α-algorithm is based on four relations that can be derived from the log. These are $>_w$, $\to_W$, $\#_W$ and $\|_W$. Let W be a workflow log over T, that is, $W \subseteq T^*$. Let $a, b \in T$. We define the four relations as follows:

— $a >_W b$ if and only if there is a trace $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ and $i \in \{1, \dots, n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$.
— $a \to_W b$ if and only if $a >_W b$ and $b \not>_W a$.
— $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$.
— $a \|_W b$ if and only if $a >_W b$ and $b >_W a$.

As pointed out in van der Aalst *et al.* (2004), the relation $\to_W$ suggests causality, and the relations $\|_W$ and $\#_W$ are used to differentiate between parallelism and choice. Since all relations can be derived from $>_W$, it is assumed that the logs are complete with respect to this relation (that is, if one task can follow another task directly, the log should have registered this potential behaviour).

In order to define the α-algorithm, we need some more terminology. Let T be a set of tasks. Let $\sigma = a_1 a_2 \dots a_n \in T^*$ be a sequence over T of length n. We define $\in$, *first* and *last* as follows: $a \in \sigma$ if and only if $a \in \{a_1, a_2, \dots a_n\}$, if $n \geqslant 1$, then $first(\sigma) = a_1$ and $last(\sigma) = a_n$.

The α algorithm associates a WF-net to a workflow log through the following definition.

Definition 6.1. Let W be a workflow log over T. Then $\alpha(W) = (S_W, T_W, F_W)$ is a WF-net defined as follows:

— $T_W = \{t \in T \mid \exists \sigma \in W.\ t \in \sigma\}$.
— $S = S_W \cup \{i, o\}$ where S_W is obtained as follows:

$$X_W = \{(A, B) \mid A \subseteq T_W \ \wedge\ B \subseteq T_W \ \wedge\ \forall a \in A\ \forall b \in B.\ a \to_W b \ \wedge\ \forall a_1, a_2 \in A.\ a_1 \#_W a_2 \ \wedge\ \forall b_1, b_2 \in B.\ b_1 \#_W b_2\}$$

$$S_W = \{(A, B) \in X_W \mid \forall (A', B') \in X_W.\ A \subseteq A' \ \wedge\ B \subseteq B' \ \Rightarrow (A, B) = (A', B')\},$$

and i and o are the source and the sink place, respectively.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

— $F_W = \{(a,(A,B)) \mid (A,B) \in S_W \wedge a \in A\} \cup \{((A,B),b) \mid (A,B) \in S_W \wedge b \in B\} \cup \{(i,t) | t \in T_I\} \cup \{(t,o) | t \in T_O\}$, where T_I and T_O are the following subsets of T_W:

$$T_I = \{t \in T \mid \exists \sigma \in W.\ t = first(\sigma)\}$$
$$T_O = \{t \in T \mid \exists \sigma \in W.\ t = last(\sigma)\}.$$

The α-algorithm will give a net where the transitions are the subset of T consisting of names appearing in at least one workflow log, and the places are pairs of maximal subsets of conflicting transitions such that all the transitions in the first component of the pair causally precede all the conflicting transitions in the second components. The definition of the flow relation is fairly obvious. Finally, initial and final transitions are identified using the *first* and *last* functions, and these are connected to the source and the sink places, respectively.

The main result proved in van der Aalst *et al.* (2004) shows that if a suitable class of workflow nets is considered, namely the class of *structured* WF-net, the α-algorithm is able to rediscover the net starting from a complete workflow log. A WF-net $N = (S, T, F)$ is called a *structured* workflow net (SWF-net) if and only if:

— for all $s \in S$ and $t \in T$ with $(s,t) \in F$, we have $|s^\bullet| > 1$ implies $|{}^\bullet t| = 1$;
— for all $s \in S$ and $t \in T$ with $(s,t) \in F$, we have $|{}^\bullet t| > 1$ implies $|{}^\bullet s| = 1$; and
— there are no implicit places, that is, places that can be deleted without changing the firing sequences of the net.

Theorem 6.1. Let $N = (S, T, F)$ be an SWF-net and W be a complete workflow log of N. If for all $a, b \in T$, we have $a^\bullet \cap {}^\bullet b = \varnothing$ or $b^\bullet \cap {}^\bullet a = \varnothing$, then $\alpha(W) = N$ modulo renaming of places.

The class of nets that can be discovered using the region-based mining approach does not suffer the limitation imposed on mining by the above theorem. Furthermore, it can mine nets that are not discovered by the α-algorithm.

In many cases the result of the region-based approach is the same as for the α-algorithm. However, the region-based approach mines the dependencies among workflow activities in a more precise way, so that with our approach we can mine nets that are not discovered by the α-algorithm. Consider the log {`IABCO, ICABO`}. A simple inspection shows that task `A` always precedes task `B`, but the α-algorithm simply fails to synthesise a WF-net, whereas using the region-based approach we can obtain the net in Figure 5 (which also has the trace `IACBO`).

In fact, the α-algorithm cannot capture dependencies that are immediately evident with the region-based approach, since it just records dependencies that arise from the fact that the two tasks follow each other. Furthermore, if we use the fully fledged theory of regions with inhibiting arcs, we can mine the net in Figure 5 from the log graph of {`IABCO, ICABO`}, with an inhibiting arc connecting the transition `C` and the place between `A` and `B`, which means that the behaviour `IACBO` would no longer be available. This can be achieved easily by inspecting the transition system obtained by the logs where bisimilar states are glued.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

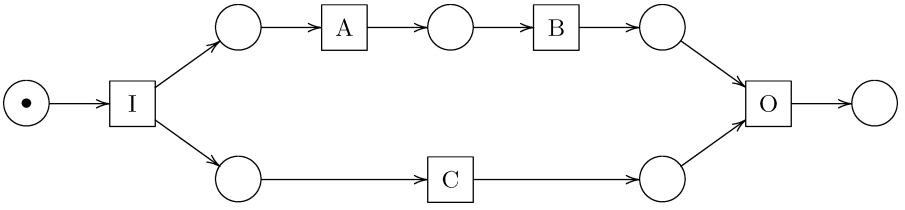


Fig. 5. A net synthesised with the regions but not with the α-algorithm

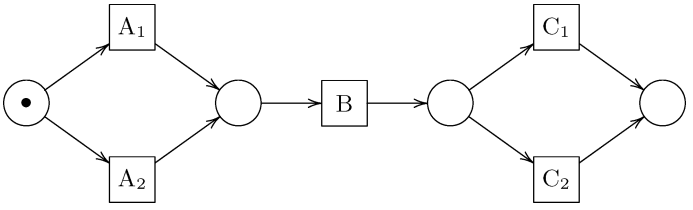


Fig. 6. A net generating the workflow log $\{\mathtt{A_1BC_1}, \mathtt{A_2BC_2}\}$

Though the region-based approach is more general, in some cases, the α-algorithm is able to rediscover a net starting from a workflow log that is smaller than the log required by the region-based approach. For example, consider the case of a sequential process $a_1, \ldots, a_n$, where there exist two tasks, say a_i and a_j with $i+1 < j$, composed of k alternatives (that is, $a_{i,1}, \ldots, a_{i,k}$ and $a_{j,1}, \ldots, a_{j,k}$). The α-algorithm only needs k traces (for example, the log $\{a_1 \ldots a_{i,1} \ldots a_{j,1} \ldots a_n, a_1 \ldots a_{i,2} \ldots a_{j,2} \ldots a_n, a_1 \ldots a_{i,k} \ldots a_{j,k} \ldots a_n\}$) to reconstruct the process. To reconstruct the same process using regions, we need all the possible traces (which is the usual requirement for completeness). As an instance of the example discussed above, consider the net N_1 in Figure 6; the log $\{\mathtt{A_1BC_1}, \mathtt{A_2BC_2}\}$ is complete and hence sufficient to rediscover the net, which is not a WF-net as it is not simple. On the other hand, if we apply the region-based construction to the above log, we obtain the net N_2 in Figure 7 (more precisely, this is a net obtained from a set of regions that satisfies S-ESSP). The two additional places s_1 and s_2 in system N_2 highlight two causal dependencies: one between actions $\mathtt{A_1}$ and $\mathtt{C_1}$ and another between actions $\mathtt{A_2}$ and $\mathtt{C_2}$. This difference is due to the fact that regions also capture causal dependencies between events that cannot appear immediately after each other in an event log (because

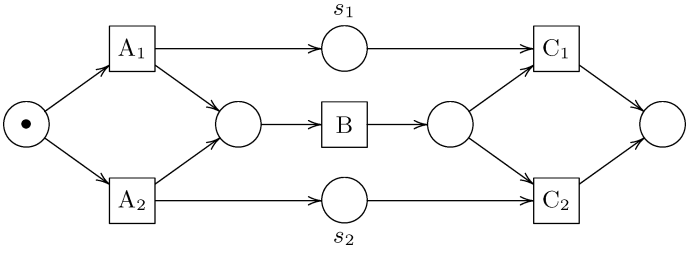


Fig. 7. A net synthesised with the regions from the workflow log $\{\mathtt{A_1BC_1}, \mathtt{A_2BC_2}\}$

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

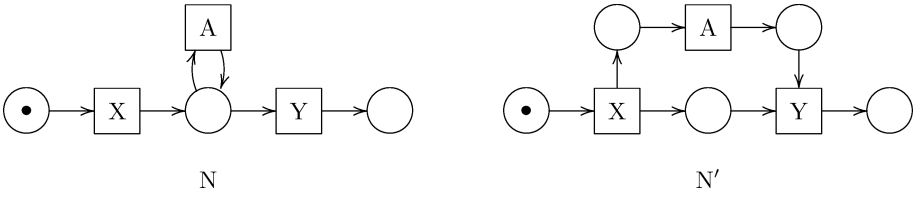


Fig. 8. A net with a one-self loop (N) and the net mined by the α-algorithm (N′)

of the existence of other causal dependencies), whereas the α-algorithm does not capture dependencies like these. In fact, if we look at the workflow log $\{\mathtt{A_1BC_1}, \mathtt{A_2BC_2}\}$, we could deduce that action $\mathtt{C_1}$ (respectively, $\mathtt{C_2}$) occurs only if action $\mathtt{A_1}$ (respectively, $\mathtt{A_2}$) has already occurred.

De Medeiros *et al.* (2003) presented a set of limitations of the α-algorithm, along with a sketch of some approaches to overcome the problems. The following subsections analyse problematic cases for the α-algorithm like these that could benefit from the adoption of a region-based technique.

6.1.1. *Length-one loops.* A length-one loop allows the repeated execution of a task for an unbounded number of times. This is not a simple net, so the region-based approach also fails.

The α-algorithm cannot reconstruct length-one loops, as shown by the example in Figure 8 (Net N_5 of Figure 7 in de Medeiros *et al.* (2003)). Similarly, the classical region-based synthesis cannot deal with length-one loops either – even if the exact transition system is provided. The problem is that if we consider the marking graph of, for example, the net N in Figure 8, there exists no region such that event A enters such a region. Hence, in the synthesised net system, there exists no place in the preset of A, so A is enabled in any reachable marking.

A solution to this problem can be obtained by adopting the technique used in Badouel and Darondeau (1995) for the synthesis of nets with read arcs, that is, arcs permitting a test to check that a place is marked in order for a transition to fire, but leaving the contents of the place unchanged after firing the transition. A place of the synthesised net, corresponding to a region r, is connected with a read arc to an event e if, for all s, s' such that $s \xrightarrow{e} s'$, both s and s' belong to r. As read arcs and length-one loops are indistinguishable in an interleaving semantics, we can simply replace each read arc (r, e) by the pair of flow arcs $(r, e), (e, r)$ to reconstruct the length-one loops. When moving to net synthesis from a workflow log, the problem becomes more difficult, because the set of transition sequences is infinite. Hence, in order to provide a solution to the problem, we need to characterise a finite subset of the transition sequences that allows us to reconstruct the net system. However, the technique sketched above seems promising, provided the workflow log contains enough information. For example, a workflow log $W_n = \{\mathtt{XY}, \mathtt{XAY}, \mathtt{XAAY}, \ldots, \mathtt{XA}^n\mathtt{Y}\}$, with $n > 1$, allows us to reconstruct the correct net for the example in Figure 8. A similar result also holds for net N_1 of Figure 4 in de Medeiros *et al.* (2003), which is shown here in Figure 9.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

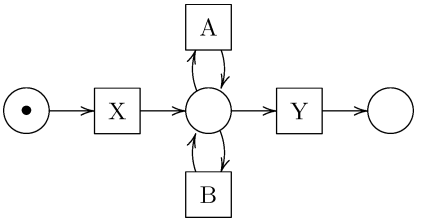


Fig. 9. A net with two one-self loops

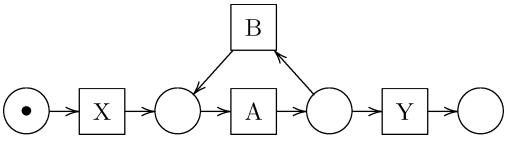


Fig. 10. A net with a length-two loop

6.1.2. *Length-two loops.* The α-algorithm infers that the two tasks involved in the loop are in parallel, and no place is created between them. Consider the log $W = \{$`XAY, XABAY, XABABAY`$\}$. The tasks `A` and `B` are in a two-length loop (this is net N_3 of Figure 4 in de Medeiros *et al.* (2003), which is shown here in Figure 10).

If we apply the α-algorithm, `A` and `B` are in parallel, whereas the regions $\{$`X, XAB, XABAB`$\}$ and $\{$`XA, XABA`$\}$ are the places connecting these tasks, in particular $\{$`X, XAB, XABAB`$\}$ is the place where `X` and `B` put a token, consumed by `A`, and $\{$`XA, XAY, XABA, XABAY`$\}$ is the place where `A` puts a token, consumed either by `B` or by `Y`. Hence the region-based approach can solve the problem easily.

However, when we consider the synthesis of a net from a workflow log, we have to face up to the fact that the set of transition sequences is infinite. If the exact transition system is provided, the classical region-based synthesis infers the correct net system, but obtaining an exact and finite transition system from an infinite set of transition sequences may not be straightforward. In any case, the net synthesis applied to nets N_3 and N_4 of Figure 4 in de Medeiros *et al.* (2003) – which are problematic for the α-algorithm – gives the correct nets, assuming a constraint on the workflow log similar to the one for the example of length-one loops.

6.1.3. *Implicit places.* A place is implicit if its presence or absence does not affect the possible log traces of a workflow. As implicit places do not influence the causal relations between tasks, they cannot be captured by the α-algorithm. Hence, the α-algorithm produces a 'smaller' version of the net, which is obtained by removing the useless implicit places. The region-based synthesis technique allows us to produce all the nets whose marking graph is isomorphic to a given transition system, and from these we can choose the net that best represents the situation we intend to model. Thus, when we are interested in mining exactly the same net, the region-based approach can be used fruitfully.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

6.1.4. *Non-free-choice.* Non-free-choice nets are not always correctly mined by the α-algoritm, as witnessed by net N_2 of Figure 8 in de Medeiros *et al.* (2003) (the same problem arises with the net in Figure 7). The problem is due to the fact that some of the causal relations between events are not inferred. If the transiton system is provided, the region-based technique also works correctly for non-free-choice nets.

As an example, consider the net N_2 in Figure 7. As we discussed earlier in this section, the α-algorithm cannot rediscover the net N_2: either starting from the workflow log $\{\mathtt{A_1BC_1}, \mathtt{A_2BC_2}\}$, or from the larger workflow log $\{\mathtt{A_1BC_1}, \mathtt{A_1BC_2}, \mathtt{A_2BC_1}, \mathtt{A_2BC_2}\}$. In fact, the net N_2 is non-free-choice. The unique place in the postset of $\mathtt{B}$ belongs to both the preset of $\mathtt{C_1}$ and the preset of $\mathtt{C_2}$, but, for example, the place s_1 only belongs to the preset of $\mathtt{C_1}$. The net discovered by the α-algorithm is N_1 in Figure 6. The problem arising in non-free-choice nets is connected with the existence of causal dependencies between two events that cannot be executed one immediately after the other. For example, place s_1 of net N_2 represents a causal dependency between $\mathtt{A_1}$ and $\mathtt{C_1}$, but no trace like … $\mathtt{A_1C_1}$ … can be generated. Thus, causal dependencies like these are not captured by the α-algorithm.

6.2. *Language-based mining*

In the region-based approach of Darondeau, given a non-empty and prefix-closed language L over an alphabet T, a *region* is a pair of maps (β, η), with $\beta : L \to \mathbb{N}$ and $\eta : T \to \mathbb{Z}$, such that $w = vu$ implies $\beta(w) = \beta(v) + \eta(u)$ for all $w \in L$, by letting η be the unique morphism on T^* that extends the map $\eta : T \to \mathbb{Z}$ (hence $\eta : T^* \to \mathbb{Z}$ such that $\eta(uv) = \eta(u) + \eta(v)$, with $u, v \in T^*$). We use $\mathscr{R}(L)$ to denote the set of regions. Given a prefix-closed language L over T, and the set of regions $\mathscr{R}(L)$, the *synthesised* net system is $\mathscr{N}(L) = (\mathscr{R}(L), T, F, m_0)$ where for all $(\beta, \eta) \in \mathscr{R}(L)$, for all $t \in T$, $F(t, (\beta, \eta) - F((\beta, \eta), t) = \eta(t)$, and for all $(\beta, \eta) \in \mathscr{R}(L)$, we have $m_0((\beta, \eta)) = \beta(\varepsilon)$. As observed in Darondeau (1998), the inclusion $L \subseteq TrSeq(\mathscr{N}(L))$ is always valid, whereas the equality can be characterised as $\forall w \in L$, $\forall t \in T$, $wt \notin L$ implies $\eta(t) < 0$ and $\beta(\varepsilon) + \eta(w) + \eta(e) < 0$. This characterisation only holds for nets with a possibly infinite set of places. When the net is unbounded, regions can be substituted with *abstract* regions, which are based on the observation that, given a prefix-closed language L over T and $\eta : T \to \mathbb{Z}$, the set $\{\eta(w) \mid w \in L\}$ has a minimum, denoted μ_η, which is necessarily non-positive as $\eta(\varepsilon) = 0$. A fixed map η gives rise to a collection of regions (β, η), each of them such that $\beta(\varepsilon) = J - \mu_\eta$ for some $J \geqslant 0$. The most significant of these regions is the one where $J = 0$ (as also observed in Bernardinello *et al.* (1996)), as it is the one that is most likely to produce a negative value when $wt \notin L$, and thus

$$\beta(wt) = \beta(w) + \eta(t) = \beta(\varepsilon) + \eta(w) + \eta(t) = (\eta(w) - \mu_\eta) + (J + \eta(t)).$$

So regions can be identified with abstract regions, which are characterised by η alone, for all η where the set $\{\eta(w) \mid w \in L\}$ has a minimum.

Thus a net can be obtained without first constructing a transition system. In this case, the following theorem can be proved.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Theorem 6.2. Let N be a sound WF-net and let $TrSeq(N)$ be its log. Then there exists a set of abstract regions $\mathscr{R}(TrSeq(N))$ such that

$$TrSeq(\mathscr{R}(TrSeq(N)), T, F, m_0) = TrSeq(N).$$

We can even state the stronger version of mining in this setting: there exists a set of abstract regions $\mathscr{R}(TrSeq(N))$ such that $\mathscr{R}(TrSeq(N)), T, F, m_0) = N$ modulo renaming of places.

A slightly different approach, which can be led back to the one using abstract regions, was surveyed in Lorenz *et al.* (2007) and Lorenz and Juhás (2007).

However, these approaches have a drawback: the logs to be considered for mining the correct net must be the whole language generated by the net (as words that do not belong to the languages are used to synthesise the net), whereas our approach can mine the correct net using finite subsets of infinite logs of sound workflow nets.

Though the method has some limitations, it does have the advantage of being much simpler to implement – for instance, see the results in Giua and Seatzu (2005), Cabasino *et al.* (2006b) and Cabasino *et al.* (2006a).

7. Incremental mining

The region-based approach to workflow mining that we have illustrated so far can be criticised in two respects. The first criticism is that there is already an approach based on languages that does not construct any transition system. As we hinted earlier, constructing a transition system allows us to mine nets in a more natural way, if the logs are enough, but are not the whole language generated by the net. Furthermore, the construction of the transition systems can help in *guessing* how the logs can be *completed* to mine a workflow net, as suggested in van der Aalst *et al.* (2008).

The second criticism concerns the complexity of the actual mining methods. To overcome this criticism, we will now present a way to construct the regions corresponding to a workflow log in an incremental way: we start by constructing the set of regions corresponding to each single trace, then we define a way to pairwise combine the set of regions of two transition systems.

We start by defining how to construct the transition system corresponding to a workflow log. We obtain such a transition system simply as the union of the transition systems corresponding to the single traces belonging to the log.

We will begin with some preliminary definitions.

Definition 7.1. Let $TS = (St, E, \rightarrow)$ be a transition system. We use $States(TS)$ to denote the set of states of the transition system (that is, $States(TS) = St$), and $Events(TS)$ to denote the set of events of the transition system (that is, $Events(TS) = E$).

We can now define the union of two transition systems.

Definition 7.2. Let $TS_1 = (St_1, E_1, \rightarrow_1)$ and $TS_2 = (St_2, E_2, \rightarrow_2)$ be two transition systems. The *union* of TS_1 and TS_2 is defined as $TS_1 \cup TS_2 = (St_1 \cup St_2, E_1 \cup E_2, \rightarrow_1 \cup \rightarrow_2)$.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Let $TS_i = (St_i, E_i, \rightarrow_i)$, $i \in I$, be a set of transition systems. The binary union operation is lifted to the n-ary union operation by

$$\bigcup_{i \in I} TS_i = \left(\bigcup_{i=1}^{n} St_i, \bigcup_{i=1}^{n} E_i, \bigcup_{i=1}^{n} \rightarrow_i \right).$$

We now show how to construct $LG(W)$ incrementally. Given a log σ (that is, a word over T^*), the transition system associated to it, denoted $LG(\sigma)$, is defined as follows.

Definition 7.3. Let $\sigma \in T^*$ be a trace. The transition system corresponding to σ is $LG(\sigma) = \{prefix(\sigma), E, \{(\rho, t, \rho t) \mid \rho t \in prefix(\sigma)\}$, with $E = \{t \in T \mid \exists \rho\tau : \sigma = \rho t \tau\}$.

Starting from the transition systems corresponding to each log, we can construct a transition system as follows.

Definition 7.4. Let $W \subseteq T^*$ be a workflow log. The transition system corresponding to W is $TS(W) = \bigcup_{\sigma \in W} LG(\sigma)$.

Clearly, in most cases the transition system $TS(W)$ will not be isomorphic to the marking graph of the net exhibiting the set of logs W, as the transition systems generated by Definition 7.4 are basically disconnected graphs composed of a set of chains. However, such chains are essentially paths in the marking graph of the net. In any case, we are interested in constructing a net system whose transition sequences 'correspond' to the workflow log.

The following proposition relates the construction of Definition 7.4 to that of Definition 5.1.

Proposition 7.1. Let $W \subseteq T^*$ be a workflow log, and $TS(W)$ and $LG(W)$ be the associated transition systems. Then there exists a morphism $h : TS(W) \rightarrow LG(W)$.

We are now ready to prove the results that allow us to construct the set of regions of a transition system TS (that is, the union of two transition systems TS_1 and TS_2), starting from the sets of regions of TS_1 and of TS_2. To this end, we need to define a notion of compatibility between regions.

Two regions of two different transition systems are said to be compatible if they can be unified to construct a region of the union of the two transition systems. The basic idea is that each event either crosses the two regions in the same way, or it does not belong to one of the two transition systems. The crucial point is that if a common state of the two transition systems belongs to one region, then it must also belong to the other one.

Definition 7.5. Let $TS_1 = (St_1, E_1, \rightarrow_1)$ and $TS_2 = (St_2, E_2, \rightarrow_2)$ be two disjoint transition systems. Let $r_1 \in Reg(TS_1)$ and $r_2 \in Reg(TS_2)$. We say that r_1 is *compatible* with r_2 if and only if:

— $\forall e \in {}^\circ r_1 : e \in {}^\circ r_2 \ \vee \ e \notin Events(TS_2)$;
— $\forall e \in r_1{}^\circ : e \in r_2{}^\circ \ \vee \ e \notin Events(TS_2)$;
— $\forall e \in {}^\circ r_2 : e \in {}^\circ r_1 \ \vee \ e \notin Events(TS_1)$;
— $\forall e \in r_2{}^\circ : e \in r_1{}^\circ \ \vee \ e \notin Events(TS_1)$;
— $\forall s \in States(TS_1) \cap States(TS_2) : s \in r_1$ if and only if $s \in r_2$.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

We are now ready to prove our main result: the set of regions of the union of two transition systems TS_1 and TS_2 can be constructed by taking the union of the pairs of compatible regions of TS_1 and TS_2, respectively.

Proposition 7.2. Let TS_1, TS_2 be two transition systems. Let $r_1 \in Reg(TS_1)$ and $r_2 \in Reg(TS_2)$. If r_1 is compatible with region r_2, then $r_1 \cup r_2 \in Reg(TS_1 \cup TS_2)$.

Proof. Suppose r_1 is compatible with r_2. We show that $r_1 \cup r_2 \in Reg(TS_1 \cup TS_2)$.
To show that $r_1 \cup r_2$ is a region, we need to prove that $\forall e \; \forall s_1 \xrightarrow{e} s_1', s_2 \xrightarrow{e} s_2'$:

— if $s_1 \in r_1 \cup r_2$ and $s_1' \notin r_1 \cup r_2$, then $s_2 \in r_1 \cup r_2$ and $s_2' \notin r_1 \cup r_2$;
— if $s_1 \in r_1 \cup r_2$ and $s_1' \notin r_1 \cup r_2$, then $s_2 \in r_1 \cup r_2$ and $s_2' \notin r_1 \cup r_2$.

We will only show the first case, as the other is similar. Suppose there exist $s_1 \xrightarrow{e} s_1'$ and $s_2 \xrightarrow{e} s_2'$ such that $s_1 \in r_1 \cup r_2$ and $s_1' \notin r_1 \cup r_2$. We show that $s_2 \in r_1 \cup r_2$ and $s_2' \notin r_1 \cup r_2$.

First let us assume that $s_1, s_2, s_1', s_2' \notin States(TS_1) \cap States(TS_2)$. From $s_1 \xrightarrow{e} s_1'$ and $s_2 \xrightarrow{e} s_2'$, we deduce that the following cases may happen:

— $s_2, s_2' \in States(TS_1)$
— $s_2, s_2' \in States(TS_2)$

(the other cases are not interesting as r_1 and r_2 are regions in TS_1 and TS_2, respectively; furthermore, by the construction of $TS_1 \cup TS_2$, we cannot have that $s \xrightarrow{e} s'$ and $s \in States(TS_1)$ and $s' \in States(TS_2)$ or $s \in States(TS_2)$ and $s' \in States(TS_1)$ unless $s, s' \in States(TS_1) \cap States(TS_2)$).

Without loss of generality, we can suppose that $s_2, s_2' \in States(TS_1)$. Again, two cases may happen:

(i) $s_1, s_1' \in States(TS_1)$.
As $s_1, s_2, s_1', s_2' \notin States(TS_1) \cap States(TS_2)$ and r_2 is a region of TS_2, and since $s_1 \in States(TS_1)$ and $s_1 \in r_1 \cup r_2$, we deduce $s_1 \in r_1$. Since $s_1' \notin r_1 \cup r_2$, we deduce $s_1' \notin r_1$. Hence, we have that $s_1 \xrightarrow{e} s_1'$, $s_1 \in r_1$ and $s_1' \notin r_1$. As r_1 is a region, from the above, and since $s_2 \xrightarrow{e} s_2'$, we deduce that $s_2 \in r_1$ and $s_2' \notin r_1$. Since $s_2 \in r_1$, we have $s_2 \in r_1 \cup r_2$. Now, we have assumed that $s_1, s_2, s_1', s_2' \notin States(TS_1) \cap States(TS_2)$, and since $s_2 \in States(TS_1)$, we get $s_2 \notin States(TS_2)$, and as r_2 is a region of TS_2, we get $s_2 \notin r_2$. Hence, $s_2 \notin r_1 \cup r_2$.
Summing up, we have $s_2 \in r_1 \cup r_2$ and $s_2 \notin r_1 \cup r_2$.

(ii) $s_1, s_1' \in States(TS_2)$.
As $s_1, s_2, s_1', s_2' \notin States(TS_1) \cap States(TS_2)$ and r_1 is a region of TS_1, and since $s_1 \in States(TS_2)$ and $s_1 \in r_1 \cup r_2$, we deduce $s_1 \in r_2$. Since $s_1' \notin r_1 \cup r_2$, we deduce $s_1' \notin r_2$. As r_2 is a region, and since $s_1 \xrightarrow{e} s_1'$, $s_1 \in r_2$ and $s_1' \notin r_2$, we get $e \in r_2^\circ$. By the definition of compatibility, two cases may happen: either $e \in r_1^\circ$ or $e \notin Events(TS_2)$. As $s_1 \in States(TS_2)$ and $s_1 \xrightarrow{e} s_2$, we have that $e \in Events(TS_2)$. Hence, we know that $e \in r_1^\circ$. As we assumed that $s_2, s_2' \in States(TS_1)$, we have from $s_2 \xrightarrow{e} s_2'$ and $e \in r_1^\circ$ that $s_2 \in r_1$ and $s_2' \notin r_1$. From $s_2 \in r_1$, we deduce $s_2 \in r_1 \cup r_2$. Now, $s_1, s_2, s_1', s_2' \notin States(TS_1) \cap States(TS_2)$, $s_2' \in States(TS_1)$ and r_2 is a region of TS_2, so we get that $s_2' \notin r_2$, and since $s_2' \notin r_1$, we deduce $s_2' \notin r_1 \cup r_2$.

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

Summing up, we have $s_2 \in r_1 \cup r_2$ and $s_2 \notin r_1 \cup r_2$.

Assume now that some of the states s_1, s_2, s_1' and s_2' are in $States(TS_1) \cap States(TS_2)$. If all of them are in the intersection, we are done.

Assume first that only $s_1 \in States(TS_1) \cap States(TS_2)$. As $s_1 \in r_1 \cup r_2$ by the definition of compatibility $s_1 \in r_1$ and $s_1 \in r_2$. Take $s_1' \in r_1 \cup r_2$. Clearly, $s_1' \in r_1$ or $s_1' \in r_2$, but not in both. Assume $s_1' \in r_1$ (the other case being similar). Now consider $s_2 \xrightarrow{e} s_2'$. The only possibility is that $s_2 \in r_2$. But as $s_2' \notin r_2$ would mean that $e \in r_2{}^\circ$, which would mean that $e \in r_1{}^\circ$, we have that $s_2' \in r_2$ and thus $r_1 \cup r_2$ is a region.

Assume now that s_1, s_2 are in $States(TS_1) \cap States(TS_2)$ and at least one of s_1' and s_2' are not. Then $s_1, s_2 \in r_1$ and $s_1, s_2 \in r_2$. If $s_1' \in r_1$, then $s_2' \in r_1$ also, since otherwise r_1 would not be a region.

As no other cases apply, we have that $r_1 \cup r_2$ is a region of $TS_1 \cup TS_2$. □

Proposition 7.3. Let TS_1, TS_2 be two transition systems. Let $r_1 \in Reg(TS_1)$ and $r_2 \in Reg(TS_2)$. If $r_1 \cup r_2 \in Reg(TS_1 \cup TS_2)$, then r_1 is compatible with region r_2.

Proof. Suppose $r_1 \cup r_2$ is a region of $TS_1 \cup TS_2$.

First we observe that $h_i : TS_i \to TS_1 \cup TS_2$ are morphisms, and $r_i = h_i^{-1}(r_1 \cup r_2)$. Clearly, it should be the case that $\forall s \in States(TS_1) \cap States(TS_2)$ we have $s \in h_1^{-1}(r)$ if and only if $s \in h_2^{-1}(r)$, as h_i are morphisms.

So we have to show that the other 4 conditions also hold. We will only consider the first condition in the definition of compatibility since the other three conditions can be verified in the same way.

Let $e \in {}^\circ r_1$. We show that either $e \in {}^\circ r_2$ or $e \notin Events(TS_2)$. Suppose $e \in Events(TS_2)$. We will show that $e \in {}^\circ r_2$. As $e \in {}^\circ r_1$, there exists $s_1, s_1' \in States(TS_1)$ such that $s_1 \xrightarrow{e} s_1'$, with $s_1 \notin r_1$ and $s_1' \in r_1$. Since $s_1 \in States(TS_1)$ and r_2 is a region of TS_2, we have $s_1 \notin r_2$. Thus, $s_1 \notin r_1 \cup r_2$. Since $s_1' \in r_1$, we have $s_1' \in r_1 \cup r_2$. Since $s_1 \xrightarrow{e} s_1'$, and since $s_1 \notin r_1 \cup r_2$ and $s_1' \in r_1 \cup r_2$, we have $e \in {}^\circ(r_1 \cup r_2)$. As we know that $e \in Events(TS_2)$, there exist $s_2, s_2' \in States(TS_2)$ such that $s_2 \xrightarrow{e} s_2'$. Since $e \in {}^\circ(r_1 \cup r_2)$ and $s_2 \xrightarrow{e} s_2'$, we have that $s_2 \notin r_1 \cup r_2$ and $s_2' \in r_1 \cup r_2$. From $s_2 \notin r_1 \cup r_2$, we deduce $s_2 \notin r_2$. Now, $s_2' \in States(TS_2)$ and r_1 is a region of TS_1, so $s_2' \in r_1 \cup r_2$ gives $s_2' \in r_2$.

Hence, we have $s_2 \xrightarrow{e} s_2'$, $s_2 \notin r_2$ and $s_2' \in r_2$. As r_2 is a region, we get $e \in {}^\circ r_2$. □

Proposition 7.4. Let TS_1, TS_2 be two transition systems. Let $r \in Reg(TS_1 \cup TS_2)$. Then there exists two compatible regions $r_1 \in Reg(TS_1)$ and $r_2 \in Reg(TS_2)$ such that $r = r_1 \cup r_2$

Proof. Consider $r \in Reg(TS_1 \cup TS_2)$. Take $r_1 = r \cap St_1$ and $r_2 = r \cap St_2$. Clearly, these are two regions belonging to $Reg(TS_1)$ and $Reg(TS_2)$, respectively.

We now show that they are compatible. The only interesting case is when $e \in {}^\circ r \cup r^\circ$ is such that $e \in E_1 \cap E_2$. Assume that $e \in {}^\circ r$ and $e \in {}^\circ r_1$. But then $e \in {}^\circ r_2$, otherwise r would not be a region.

The other conditions for compatibility can be checked in the same way, and since the last condition is satisfied too, we are done. □

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

We can now state the main result of this section – the proof is obvious using Proposition 7.4.

Theorem 7.1. Let TS_1, TS_2 be two transition systems and $Reg(TS_1)$, $Reg(TS_2)$ be their sets of regions. Then $Reg(TS_1 \cup TS_2) = \{r \mid \exists\ r_1 \in Reg(TS_1)$ and $\exists\ r_2 \in Reg(TS_2)$ such that $r = r_1 \cup r_2$ and r_1 is compatible with $r_2\}$.

Applying this method to the example in the introduction, we have three traces (`AED`, `ABCD`, `ACBD`), and the transition systems obtained by each trace have obvious minimal regions. It is easy to see that the regions with the state corresponding to `A` are all compatible, as are the ones with `D`. The regions with `B` and `C` are again compatible, so the construction gives the desired result.

Using the previous result, we can state and prove the following theorem.

Theorem 7.2. Let N be a WF-net and W be a complete workflow log of N. For each $\sigma \in W$, let $LG(\sigma)$ be the associated transition system. Then R, the set of minimal regions of $\bigcup_{\sigma \in W} LG(\sigma)$, satisfies the S-ESSP property. Furthermore, $TrSeq(N) = TrSeq(Gen(\bigcup_{\sigma \in W} LG(\sigma), R))$.

Proof. By Theorem 7.1, we have that all the regions are obtained as combinations of compatible regions in $LG(\sigma)$, with $\sigma \in W$.

Now consider the mapping $h : \bigcup_{\sigma \in W} LG(\sigma) \to LG(W)$ defined as $h(\sigma')) = \sigma'$, with σ' a prefix of $\sigma \in W$. This is a well-defined transition system morphism, which is also a functional bisimulation.

Consider the set R of minimal regions of $\bigcup_{\sigma \in W} LG(\sigma)$. For each $r \in R$, we have that $h(r)$ is a minimal region of $LG(W)$. Furthermore, for each minimal region r' of $LG(W)$, the associated $h^{-1}(r')$ is minimal since otherwise the minimal region $\hat{r} \subseteq h^{-1}(r')$ would be such that $h(\hat{r}) \subset r'$, which contradicts the minimality of r'. So $h(R)$ is the set of minimal regions of $LG(W)$. Clearly, R satisfies the S-ESSP property (as $h(R)$ does) and $TrSeq(N) = TrSeq(Gen(\bigcup_{\sigma \in W} LG(\sigma), R))$. □

The construction we have developed has a drawback: incrementally considering only minimal regions does not always give a correct answer. This means that we can still construct the proper set of regions by considering the compatible ones, but we cannot simply always use the minimal regions of the components of the transition system. Take the net in Figure 11, and consider first the traces `AB`, `CD` and the associated transition systems $TS_{\{\texttt{AB}\}}$ and $TS_{\{\texttt{CD}\}}$. When composing these transition systems, it is easy to see that the minimal regions of each are compatible with the empty region of the other, hence the minimal regions of the compound transition system are simply the union of the two sets of minimal regions. Now consider the traces `AD` or `CB`. When checking for compatibility of the regions of the compound transition system $TS_{\{\texttt{AB,CD}\}}$ and of $TS_{\{\texttt{AD}\}}$, we find that we can no longer just consider minimal regions, but also have to consider compound regions (in this case the region where `A` and `C` enter and `B` and `D` exit, as the net with these four traces is the one in Figure 11).

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

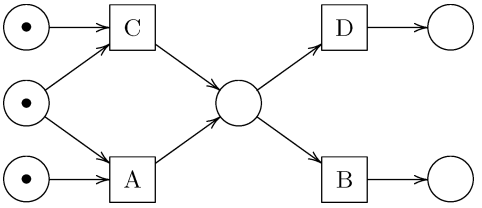


Fig. 11. A net where using minimal regions on the transition systems associated to traces fails

8. Conclusions and future work

In this paper we have collected some ideas on the discovery of processes (business process oriented) and Petri nets. The main results can be summarised as follows:

— Region-based mining of workflow nets is much more effective than other approaches. The approach based on languages can be easily implemented, but from the point of view of the re-engineering of workflow nets, which is the main application of process mining, we believe that the approach we have presented is much more useful.
— The incremental approach can be applied fruitfully. In fact, van Dongen *et al.* (2007) has reported an implementation of an incremental approach as a plug-in of the ProM framework (van Dongen *et al.* 2005), which is a flexible environment for process mining developed at the Eindhoven University of Technology. This approach can be refined, for example, by adding transition systems in suitable orders.

Though some steps have been already taken, there are still many open issues, which we have planned to work on. Here is a list, in no particular order, of some of the ongoing ideas we have had on this subject:

(i) Workflow nets are usually obtained via a top-down design, which makes them rather *regular*. We have pointed out that by looking at the situation hindering the separation properties, we can identify or split states of the transition system: in the first case making two states equivalent, while in the second we discover that a state holds more information, which can only be made available by splitting it (thus making the model more concrete). A clear classification of nets with respect to the property they violate can be helpful in understanding and characterising these subclasses.
(ii) The incremental methods do not always work if we just consider minimal regions (which is desirable). It may be useful to find a method for constructing the minimal set of regions of $TS_1 \cup TS_2$ from the minimal regions of TS_1 and TS_2.
(iii) Along the same line as the previous item, it would be interesting to define a property on a set R of regions allowing the construction without using all the regions, but just a proper subset.
(iv) Workflow nets are a suitable subclass of Bruni and Montanari's Zero-Safe nets (Bruni and Montanari 2001). Recently, Darondeau has investigated the synthesis of Zero-Safe nets (Darondeau 2008), and, with others, the synthesis of nets from step-firing policies (Darondeau *et al.* 2008). It would be interesting to investigate the impact of these recent developments on the mining of workflow nets, bearing in mind that the synthesis from the workflow log, as we have considered in this paper, is rather

https://doi.org/10.1017/S0960129509990132 Published online by Cambridge University Press

meager, and the logs have a much richer structure, which can be used to obtain the correct net.

Acknowledgments

It may seem strange for the second author (G. Michele Pinna) to acknowledge the first (Nadia Busi), but the loss of Nadia is still unbearable to me, so I would like to say *grazie Nadia*. I should add that the good ideas in the paper are hers and any poor development of them is down to me.

I would also like to thank the reviewers for their useful suggestions and criticisms, and I hope that Nadia would have agreed with the changes I have made to the paper following their recommendations.

References

van der Aalst, W. M. P. (2004) Business process management demystified: A tutorial on models, systems and standards for workflow management. In: Desel, J., Reisig, W. and Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **3098** 1–65.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G. and Weijters, A. J. M. M. (2003) Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering* **47** (2) 237–267.

van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B. F., Kindler, E. and Günther, C. W. (2008) Process mining: A two-step approach to balance between underfitting and overfitting. Technical Report BPM-08-01, BPM Center.

van der Aalst, W. M. P. and Weijters, A. J. M. M. (2004) Process Mining. *Computers in Industry* (Special Issue) **53** (3).

van der Aalst, W. M. P., Weijters, T. and Maruster, L. (2004) Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering* **16** (9) 1128–1142.

Badouel, E. and Darondeau, P. (1995) Trace nets and process automata. *Acta Informatica* **32** 647–679.

Badouel, E. and Darondeau, P. (1998) Theory of regions. In: Reisig, W. and Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models, Advances in Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **1491** 529–586.

Bernardinello, L. (1993) Synthesis of net systems. In: Ajmone Marsan, M. (ed.) Proceedings of the 14th Conference on the Application and Theory of Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **691** 89–105.

Bernardinello, L., De Michelis, G., Petruni, K. and Vigna, S. (1996) On the synchronic structure of transition systems. In: Desel, J. (ed.) *Structure in Concurrency Theory*, Springer Verlag 11–31.

Bruni, R. and Montanari, U. (2001) Transactions and zero-safe nets. In: Ehrig, H., Juhásm, G., Padberg, J. and Rozenberg, G. (eds.) Unifying Petri Nets, Advances in Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **2128** 380–426.

Busi, N. and Pinna, G. M. (1997) Synthesis of nets with inhibitor arcs. In: Mazurkiewicz, A. and Winkowski, J. (eds.) CONCUR'97 Conference Proceedings. *Springer-Verlag Lecture Notes in Computer Science* **1243** 151–165.

Busi, N. and Pinna, G. M. (1998) Transition systems and elementary net systems (manuscript).

Busi, N. and Pinna, G. M. (2006a) Characterizing workflow nets using regions. In: *SYNASC 2006*, IEEE Computer Society 399–406.

Busi, N. and Pinna, G. M. (2006b) Region-based workflow mining: a feasibility analysis (manuscript).

Cabasino, M. P., Giua, A. and Seatzu, C. (2006a) Identification of deterministic Petri nets. In: *WODES '06: 8th Workshop on Discrete Event Systems* 32–331.

Cabasino, M. P., Giua, A. and Seatzu, C. (2006b) Identification of unbounded Petri nets from their coverability graph. In: *CDC06: 45th IEEE Conference on Decision and Control* 434–440.

Carmona, J., Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, K. and Yakovlev, A. (2008) A symbolic algorithm for the synthesis of bounded Petri nets. In: van Hee, K. and Valk, R. (eds.) Proceedings of 29th International Conference on the Applications and Theory of Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **5062** 92–111.

Cortadella, J., Kishinevsky, M., Lavagno, K. and Yakovlev, A. (1998) Deriving Petri nets for finite transition systems. *IEEE Trans. Computers* **47** (8) 859–882.

Darondeau, P. (1998) Deriving unbounded Petri nets from formal languages. In: Sangiorgi, D. and de Simone, R. (eds.) *CONCUR '98. Springer-Verlag Lecture Notes in Computer Science* **1466** 533–548.

Darondeau, P. (2008) On the synthesis of zero-safe nets. In: Degano, P., De Nicola, R. and Meseguer, J. (eds.) Concurrency, Graphs and Models. *Springer-Verlag Lecture Notes in Computer Science* **5065** 409–426.

Darondeau, P., Koutny, M., Pietkiewicz-Koutny, M. and Yakovlev, A. (2008) Synthesis of nets with step firing policies. In: van Hee, K. and Valk, R. (eds.) Proceedings of 29th International Conference on the Applications and Theory of Petri Nets. *Springer-Verlag Lecture Notes in Computer Science* **5062** 112–131.

Desel, J. and Esparza, J. (1995) *Free Choice Petri Nets*, Cambridge Tracts in Theoretical Computer Science **40**, Cambridge University Press.

Desel, J. and Reisig, W. (1996) The synthesis problem of Petri nets. *Acta Informatica* **33** 296–315.

van Dongen, B. F., Busi, N., Pinna, G. M. and van der Aalst, W. M. P. (2007) An iterative algorithm for applying the theory of regions in process mining. In: Reisig, W., van Hee, K. and Wolf, K. (eds.) *Proceedings of the Workshop on Formal Approaches to Business Processes and Web Services (FABPWS'07)* 36–55.

van Dongen, B. F., de Medeiros, A. K. A., Verbeeki, H. M. W., Weijters, A. J. M. M. and van der Aalst, W. M. P. (2005) The ProM framework: A new era in process mining tool support. In: Ciardo, G. and Darondeau, P. (eds.) Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005. *Springer-Verlag Lecture Notes in Computer Science* **3536** 444–454.

Ehrenfeucht, A. and Rozenberg, G. (1989) Partial (set) 2-structures. Part I: Basic notions and the representation problem. Part II: State spaces of concurrent systems. *Acta Informatica* **27** (4) 315–368.

Giua, A. and Seatzu, C. (2005) Identification of free-labeled Petri nets via integer programming. In: *CDC05: 44th IEEE Conference on Decision and Control* 7639–7644.

Gorgônio, K. C., Cortadella, J., Xia, F. and Yakovlev, A. (2007) Automating synthesis of asynchronous communication mechanisms. *Fundamenta Informaticae* **78** (1) 75–100.

IDS Scheer (2002) ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper), IDS Scheer, Saarbruecken, Gemany.

Keller, G. and Teufel, T. (1998) *SAP R/3 Process Oriented Implementation*, Addison-Wesley.

Lorenz, R., Bergenthum, R., Desel, J. and Mauser, S. (2007) Synthesis of Petri nets from finite partial languages. In: Basten, T., Juhás, G. and Shukla, S. K. (eds.) *Proceedings of ACSD 2007* 157–166.

Lorenz, R. and Juhás, G. (2007) How to synthesize nets from languages – a survey. In: *Proceedings of the 2007 Winter Simulation Conference* 637–647.

de Medeiros, A. K. A., van der Aalst, W. M. P. and Weijters, A. J. M. M. (2003) Workflow mining: Current status and future directions. In: Meersman, R., Tari, Z. and Schmidt, D. C. (eds.) On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops. *Springer-Verlag Lecture Notes in Computer Science* **2888** 389–406.

Mukund, M. (1992) Petri nets and step transition systems. *International Journal on Foundation of Computing Science* **3** (4) 443–478.

Reisig, W. and Rozenberg, G. (eds.) (1998) Lectures on Petri Nets I: Basic Models. *Springer-Verlag Lecture Notes in Computer Science* **1491**.

Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M. and Russell, N. (2005) Pattern-based analysis of the control-flow perspective of uml activity diagrams. In: Delcambre, L. M. L., Kop, C., Mayr, H. C., Mylopoulos, J. and Pastor, O. (eds.) ER 2005. *Springer-Verlag Lecture Notes in Computer Science* **3716** 63–78.