Structural Patterns for Soundness of Business Process Models

B.F. van Dongen
Department of Technology Management, Eindhoven University of Technology, The Netherlands.
b.f.v.dongen@tm.tue.nl

J. Mendling
Vienna University of Economics and Business Administration, Austria,
jan.mendling@wu-wien.ac.at

W.M.P. van der Aalst
Department of Technology Management, Eindhoven University of Technology, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl

Abstract

The correctness of business process models is of paramount importance for the application on an enterprise level. A severe problem is that several languages for business process modelling do not have formal execution semantics which is a prerequisite to check correctness criteria. In this context, soundness defines a minimum correctness criterion that a process model should fulfil. In this paper we present a novel approach to reason about soundness based on so-called causal footprints*. A causal footprint represents a set of conditions on the order of activities that holds for every case of a process model. We identify three kinds of error patterns that affect the soundness of a process model, namely the deadlock pattern, the multiple termination pattern, and the trap pattern. We use Event-driven Process Chains (EPCs) and Petri nets to demonstrate the applicability of our approach for both conceptual as for formal process modelling languages. Furthermore, it can easily be applied to other languages, such as UML activity diagrams or BPEL. Based on the trap pattern, we prove that the "vicious circle", that is heavily discussed in EPC literature, is unsound.*

1. Introduction

Business process modelling is gaining increasing attention as a basis for the development of large-scale enterprise information systems. In this context, business process models can either be used as a formalization of requirements that guide the implementation, as input for code generation in a model-driven architecture, or as executable templates on a dedicated process engine defined e.g. with BPEL. All these three scenarios have in common that the correctness of the business process models is of paramount importance for the application on an enterprise level. Formal errors such as deadlocks can easily have a severe impact on the profits of an enterprise if a key business process is affected.

The detection of errors in a business process model is complicated by two problems. First, there is a plethora of languages for business process modelling (cf. [21]). Even though there are similarities concerning what workflow patterns these languages supports (cf. [4]), analysis techniques have to be defined for each language individually in order to capture the specifics of the semantics. Second, and even worse, there are several business process modelling languages for which no formal semantics have been defined or that have only formal semantics when certain structural conditions are fulfilled. Still, any business process model that is used in the development of enterprise systems needs to be checked in order to prevent ambiguities in the design phase and errors at run-time. In this paper, we address these two problems in a novel way.

While several available analysis techniques (e.g. for Petri nets) work on the state space that can be calculated for a process model, we abstract from the actual behaviour by introducing a concept that is called *causal footprint*.

The idea of a causal footprint is to derive a set of conditions on the order of activities that holds for the process model and to utilize it for reasoning on the correctness. Soundness is an important correctness criterion in this context. In essence, a process is sound if and only if (a) any case terminates in one of some pre-defined termination states and (b) for all activities in the process there is at least one case in which they can be executed. Figure 1 illustrates that the causal footprint gives an approximation of the process be-

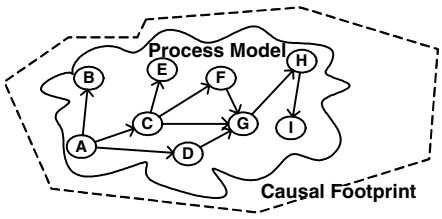

Figure 1. Process model and causal footprint.

haviour in terms of conditions that every process instance has to obey to. Throughout this paper, we use Event-driven Process Chains (EPCs) and Petri nets to show the applicability of causal footprints for both formal and conceptual business process modelling languages. This choice is motivated by the fact that EPCs are widely used for the documentation of business process e.g. in the SAP reference model. Furthermore, Petri nets are a well-understood formalism for the modelling of business processes that are used for the formalization of a variety of languages and standards (e.g. BPMN, BPEL, XPDL, UML activity diagrams).

The remainder of this paper is structured as follows. In Section 2, we introduce EPCs and Petri nets. Section 3 provides the definition of causal footprints and a respective mapping from EPCs and from Petri nets. Section 4 continues with two patterns (deadlock and multiple termination) that are typically found in a causal footprint that has errors. We use an example taken from the SAP reference model to illustrate our "pattern-based error detection" approach. Furthermore, we provide a third pattern (trap pattern) that represents a necessary condition for soundness. We apply this technique to the famous "vicious circle" (cf. [3, 17, 18]) to show that it is not sound under *any* semantics where blocking connectors are assumed to be unsound. In Section 5, we introduce the implementation of our technique as a plug-in for the ProM framework. Section 6 gives an overview of related research on business process verification. We conclude the paper with some conclusions and an outlook on future research.

2. Preliminaries

In the introduction, we mentioned two process modelling languages, namely Petri nets and EPCs. In this section, we introduce these modelling languages in the mathematical sense (that is, we formalize their structure). Both of these languages, and also the formalism we introduce later, are graph based. Therefore, we start by introducing some notation specifically for directed graphs.

Definition 2.1. (Pre-set and Post-set) Let $G = (N, E)$ be a directed graph and let $n \in N$. We define $\overset{G}{\bullet} n = \{m \in N \mid (m, n) \in E\}$ as the pre-set and $n \overset{G}{\bullet} = \{m \in N \mid (n, m) \in E\}$ as the post-set of n with respect to the graph G. If the context is clear, the superscript G may be omitted, resulting in $\bullet n$ and $n\bullet$.

Petri nets are a formal language that can be used to specify processes. Since the language has formal and executable semantics, processes modelled in terms of a Petri net can be executed by an information system. For an elaborate introduction to Petri nets, the reader is referred to [9, 22, 23]. A Petri net consists of two types of node elements (cf. Figure 2):

Transitions, which typically correspond to either an activity which needs to be executed, or to a "silent" step (cf. a routing step) that takes care of routing. A transition is drawn as a rectangle.

Places, which are used to define the preconditions and postconditions of transitions. A place is drawn as a circle.

Transitions and places are connected through directed arcs in such a way that (i) places and transitions have at least one incident edge and (ii) in every path, transitions and places alternate (no place is connected to a place and no transition is connected to a transition). For completeness sake, we mention that the Petri nets we use in this paper correspond to a classic subclass of Petri nets, namely workflow nets (WF-nets, [1]), which are tailored towards workflow modeling and analysis.

Definition 2.2. (Workflow net) $\omega = (P, T, F)$ is a workflow net (or WF-net [1]) if:

- P is a finite set of places,
- T is a finite, non empty set of transitions, such that $P \cap T = \emptyset$ and $T \neq \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation of the net,
- there exists exactly one $p_i \in P$, such that $| \bullet p_i| = 0$,
- there exists exactly one $p_f \in P$, such that $|p_f \bullet | = 0$,
- all places and transitions are covered by the paths from p_i to p_f.

Except for Petri nets, we also use EPCs to illustrate our ideas. EPCs provide an intuitive modeling language to model business processes. EPCs were introduced by Keller, Nüttgens, and Scheer in 1992 [15]. It is important to realize that the language is not intended to be a *formal* specification of a business process. Instead, it serves mainly as a means of communication. EPCs are extensively used in large-scale enterprise modelling projects. One prominent example of a publicly available model is the SAP reference model [7, 16]. An EPC consists of three types of node elements (cf. Figure 2):

Functions, which are the basic building blocks. A function corresponds to an activity (task, process step) which

needs to be executed. A function is drawn as a box with rounded corners.

Events, which describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the position of one function and act as a precondition of another function. Events are drawn as hexagons.

Connectors, which can be used to connect functions and events to specify the flow of control. There are three types of connectors: $\wedge$ (AND), $\times$ (XOR) and $\vee$ (OR). Connectors are drawn as circles, showing the type in the centre.

Functions, events, and connectors can be connected with edges in such a way that (i) events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming or an outgoing edge), (ii) functions have precisely one incoming edge and precisely one outgoing edge, (iii) connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and (iv) in every path, functions and events alternate (no two functions are connected to one-another and no two events are connected to one-another, not even when there are connectors in between). Furthermore, as a guideline, an event should never be followed by a choice connector. The latter relates to the implementation where all components of an information system that can handle a certain event, should handle it and no selection is made between those components.

Definition 2.3. (Event-driven Process Chain)
$\varepsilon = (F, E, C_{and}, C_{xor}, C_{or}, A)$ is an EPC if:

- F is a finite set of functions,
- E is a finite set of events,
- $C = C_{and} \cup C_{xor} \cup C_{or}$ is a finite set of connectors, such that $|C| = |C_{and}| + |C_{xor}| + |C_{or}|$,
- $A \subseteq ((F \cup E \cup C) \times (F \cup E \cup C))$ is the flow relation of the net, such that:
  - for all $f \in F$ there is one $(f, x) \in A$ and one $(x, f) \in A$,
  - for all $e \in E$ there is at most one $(e, x) \in A$ and at most one $(x, e) \in A$,
  - for all $c \in C$ there is either one $(c, x) \in A$ and more than one $(x, c) \in A$, or one $(x, c) \in A$ and more than one $(c, x) \in A$,
  - for all $f_1, f_2 \in F$ holds $(f_1, f_2) \notin A$,
  - for all $e_1, e_2 \in E$ holds $(e_1, e_2) \notin A$,
  - on all paths, functions and events alternate.

Note that in this section, we only provide an abstract syntax of Petri nets and EPCs and do not give any semantics. The reason is that we are not interested in the precise semantics. As illustrated by Figure 1 we aim for a causal footprint which is independent of specific semantical interpretations (e.g., the OR-join in EPCs).

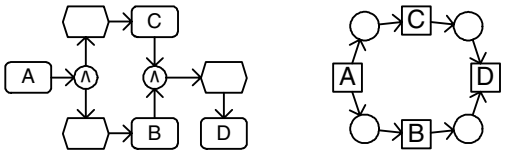

Figure 2. EPC and Petri net for parallelism.

3. Causality graphs

As we stated in the introduction, we will look at the structure of EPCs and Petri nets to provide conditions for soundness. We do so, by deriving a higher level specification, called a *causality graph*, that captures the intention of the control flow described in the process model. It is important to realize that these causality graphs do not capture the entire process model as such, in fact they are merely a footprint of the control flow in the given process model, i.e. they describe the approximate behaviour of a system at a very high level.

Let us look at process modelling in general. The intention of any process model is to capture the behaviour of a process in an understandable way, where the behaviour is formed by the execution of activities within a case. [1]

Definition 3.1. (Process behaviour/case)
Let T be a set of activities, and let Φ_T be a process containing these activities. The behaviour of the process Φ_T is defined as the set $W \subseteq T^*$, where T^* is the set of all sequences that are composed of zero of more tasks from T. A $\sigma \in W$ is called a *case*, i.e. a possible execution of the process. To denote an activity at a specific index in σ, we use $\sigma[i]$, where i is the index ranging from 1 to $|\sigma|$.

We have now formalized the behaviour of a process and each of the modelling languages of Section 2 is intended to capture this behaviour in a structured way. As we stated before, we will look at processes at a very high level. Therefore, we introduce the *causality graph*, representing a footprint of the process.

Definition 3.2. (Causality Graph) Let N be a set of activities. We define a causality graph $G = (N, F_{lb}, F_{la})$, where:

- N is a finite set of *nodes* (activities),
- $F_{lb} \subseteq (\mathcal{P}(N) \times N)$ is a set of *look-back links*[2],
- $F_{la} \subseteq (N \times \mathcal{P}(N))$ is a set of *look-ahead links*.

[1] Note that we assume trace semantics here rather than more refined notions such as (branching/weak) bisimulation [14].

[2] With $\mathcal{P}(N)$, we denote the powerset of N, i.e. $N' \in \mathcal{P}(N)$ if and only if $N' \subseteq N$ and $N' \neq \emptyset$.

3.1. Causal Footprint

When a causality graph is used to describe a process, it should be interpreted in the following way. For each *look-ahead link*, we say that the execution of the source of that link leads to the execution of at least one of the targets of that link. A look-ahead link is denoted as a bullet with one or more outgoing arrows. Furthermore, for each *look-back link*, the execution of the target is preceded by at least one of the sources of that link. The notation of a look-back link is a bullet with one or more incoming arrows. Note that we do not give any information about when in the future or past executions took place, but only that they are there. This way of describing a process is similar to the work presented in [12]. However, by splitting up the semantics in the two different directions (i.e. forward and backward), causal footprints are more expressive. With footprints you can for example express the fact that task `A` is always succeeded by `B`, but that `B` can also occur before `A`, which is typically hard to express in other languages.

If a causality graph indeed describes a process like this, we call it a *causal footprint*. We formalize this concept using the notion of cases.

Definition 3.3. (Causal Footprint) Let T be a set of activities, Φ_T be a process with behaviour W. Furthermore, let t_i and t_f be such that $t_i, t_f \notin T$ and $t_i \neq t_f$. Furthermore, let $G = (T \cup \{t_i, t_f\}, F_{lb}, F_{la})$ be a causality graph. (For notational purposes, we say that for all $\sigma \in W$ with $n = |\sigma|$,holds that $\sigma[0] = t_i$ and $\sigma[n+1] = t_f$, i.e. we add artificial starts and ends to each trace). We say that G is a *causal footprint* graph of Φ_T, denoted by $G \in \mathcal{F}_{\Phi_T}$, if and only if:

1. For all $(a, B) \in F_{la}$ holds that for each $\sigma \in W$ with $n = |\sigma|$, such that there is a $0 \leq i \leq n+1$ with $\sigma[i] = a$, there is a $j : i < j \leq n+1$, such that $\sigma[j] \in B$,
2. For all $(A, b) \in F_{lb}$ holds that for each $\sigma \in W$ with $n = |\sigma|$, such that there is a $0 \leq i \leq n+1$ with $\sigma[i] = b$, there is a $j : 0 \leq j < i$, such that $\sigma[j] \in A$,

It is clear from Definition 3.3 that a causal footprint is not unique, i.e., different processes can have common footprints. For example, $G = (T \cup \{t_i, t_f\}, \emptyset, \emptyset)$ is the causal footprint of any process having activities T. Therefore, we aim at footprints that are more informative without trying to capture detailed semantics. Moreover, by using a transitive closure, a causal footprint can be extended to be a more informative causal footprint.

In order to derive conditions for soundness of processes, we need to consider the transitive closure of a causal footprint. To do so, we first give a general way to transitively close a causality graph.

Definition 3.4. (Causal Closure) Let $G = (N, F_{lb}, F_{la})$ be a causality graph. We define $G^* = (N, F^*_{lb}, F^*_{la})$ to be the causal closure of G, where F^*_{lb} and F^*_{la} are the smallest possible sets, such that:

1. $(a, B) \in F_{la}$ implies that $(a, B) \in F^*_{la}$,
2. $(A, b) \in F_{lb}$ implies that $(A, b) \in F^*_{lb}$,
3. $(a, B) \in F^*_{la}$ implies that for all $N' \subseteq N$ holds that $(a, B \cup N') \in F^*_{la}$,
4. $(A, b) \in F^*_{lb}$ implies that for all $N' \subseteq N$ holds that $(A \cup N', b) \in F^*_{lb}$,
5. $(a, B) \in F^*_{la}$, $b \in B$ and $(b, C) \in F^*_{la}$, implies that $(a, (B \setminus \{b\}) \cup C) \in F^*_{la}$,
6. $(B, c) \in F^*_{lb}$, $b \in B$ and $(A, b) \in F^*_{lb}$, implies that $((B \setminus \{b\}) \cup A, c) \in F^*_{lb}$,

The rules for causally closing a causal graph obviously apply to a causal footprint as well. More importantly, the causal closure of a causal footprint is a causal footprint again and we refer to it as a *footprint closure*.

To illustrate that the causal closure of a causal footprint is indeed also a causal footprint, assume that an activity a in some process is always followed by at least one element of the set B. We can then extend the set B by any element, which corresponds to rules 3 and 4 of Definition 3.4. Now consider rule 5 and again assume a is always follows by some element of the set B (i.e. $(a, B) \in F^*_{la}$). Furthermore, we know that there is an activity $b \in B$, which is always followed by one element of the set C (i.e. $(b, C) \in F^*_{la}$). Then, it is easy to see that the activity $b \in B$ can be substituted by C, i.e. a is always followed by an element of C, or an element of B, except b, i.e. $(B \setminus \{b\}) \cup C$. Note that is it important to first remove b from B and then add all elements of C, since C might contain b.

It is important that the rules for making a causal closure are valid under the assumption that the process under consideration has sound behaviour. Obviously, when investigating soundness, soundness cannot be assumed right from the beginning. However, in this paper, we only investigate *conditions* for soundness using the following way of reasoning. If a process is indeed sound, it is safe to make this assumption. If the process is not sound and we use the causal closure, some of the derivations may not be correct. However, if we find a error using the causal closure it may result from a correct derivation showing a problem or the initial assumption (the process is sound) was wrong. In both cases, we can conclude that there is indeed a problem and the process is not sound.

The latter property will be used in this paper, where we actually do not have the full behaviour of a process, but only the process model, describing the behaviour in a more or less formal way.

The fact that a causal footprint of a process is not unique is irrelevant for the work presented in this paper. Any prop-

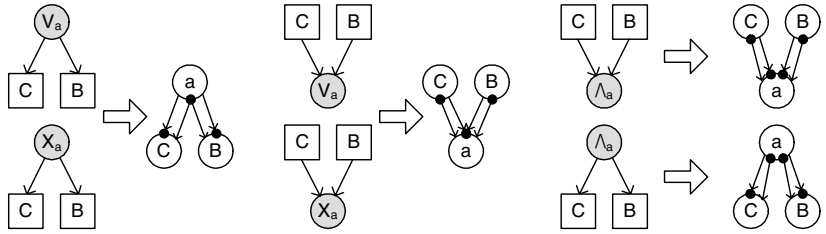

Figure 3. Mapping of EPC to causal footprint.

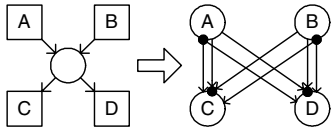

Figure 4. Mapping for Petri net.

erty that we derive that holds on a causal footprint of a process, holds on the process itself. The better the causal footprint of a process is, i.e. the more information it contains, the more properties we are able to deduce. As an example, again consider the graph that only has the activities of a process as nodes and no edges. This graph is a causal footprint of any process, but it does not contain any information about the process.

3.2. Deriving Causal Footprints

In the introduction, we stated that, as an example, we would focus on soundness properties using EPCs and Petri nets. Therefore, in this section, we present algorithms to derive causal footprints of EPCs and Petri nets. This is done in two stages. First, the models are translated to causality graphs in such a way that these graphs contain all elements of the modelling languages and only causalities that can be derived locally (i.e., immediate predecessor and successor relationships). These graphs are then transitively closed and projected onto the subset of interesting elements, thus leading to causal footprints.

Definition 3.5. (EPC to causal graph) Let $\varepsilon = (F, E, C_{and}, C_{xor}, C_{or}, A)$ be an EPC, with the set of modelling elements $N' = F \cup E \cup C_{and} \cup C_{xor} \cup C_{or}$, the set of initial elements $N_i = \{n \in N' \mid \overset{\varepsilon}{\bullet} n = \emptyset\}$ and the set of final elements $N_f = \{n \in N' \mid n \overset{\varepsilon}{\bullet} = \emptyset\}$. We define a causality graph $G_\varepsilon = (N, F_{lb}, F_{la})$ as follows:

- $N = N' \cup \{n_i, n_f\}$, where $n_i, n_f \notin N'$ and $n_i \neq n_f$,
- $F_{la} =$
  $\{(a, \{b\}) \in N \times \mathcal{P}(N) \mid a \in F \cup E \cup C_{and} \wedge b \in a \overset{\varepsilon}{\bullet}\} \cup$
  $\{(a, B) \in N \times \mathcal{P}(N) \mid a \in C_{or} \cup C_{xor} \wedge B = a \overset{\varepsilon}{\bullet}\} \cup$
  $\{(n_i, N_i)\} \cup \{(a, \{n_f\}) \mid a \in N_f\}$,
- $F_{lb} =$
  $\{(\{a\}, b) \in \mathcal{P}(N) \times N \mid b \in F \cup E \cup C_{and} \wedge a \in \overset{\varepsilon}{\bullet} b\} \cup$
  $\{(A, b) \in \mathcal{P}(N) \times N \mid b \in C_{or} \cup C_{xor} \wedge A = \overset{\varepsilon}{\bullet} b\} \cup$
  $\{(N_f, n_f)\} \cup \{(\{n_i\}, a) \mid a \in N_i\}$.

Definition 3.5 gives an algorithm to derive a causality graph from an EPC and Figure 3 a respective illustration. However, since we want to talk about processes, we are only interested in the functions and connectors of this EPC and not its events. By taking the causal closure of the causality graph and projecting it onto the functions and connectors (i.e. removing all events and all edges related to these events), we obtain a causal footprint of the process modelled by the EPC. For this, we first define a projection of a causal graph.

Definition 3.6. (Causal Graph Projection) Let $G = (N, F_{lb}, F_{la})$ be a causal graph and let N' be a set of nodes, such that $N' \subseteq N$. We define the projection $G' = (N', F'_{lb}, F'_{la})$ of G onto N', such that:

- ${F_{lb}}' = F_{lb} \cap (\mathcal{P}(N') \times N')$,
- ${F_{la}}' = F_{la} \cap (N' \times \mathcal{P}(N'))$

Property 3.7. (Causal graph gives a causal footprint for EPC)

Let T be a set of activities and Φ_T a process over T. Let $\varepsilon = (F, E, C_{and}, C_{xor}, C_{or}, A)$ be an EPC that models the process Φ_T. Note that $F = T$. Furthermore, let $G_\varepsilon = (N, F_{lb}, F_{la})$ be a causality graph following Definition 3.5. The causal closure of G_ε, $G^*_\varepsilon = (N, F^*_{lb}, F^*_{la})$, projected onto $F \cup C \cup \{n_i, n_f\}$, i.e. $G^{*\prime}_\varepsilon = (F \cup C \cup \{n_i, n_f\}, {F^*_{lb}}', {F^*_{la}}')$, is a causal footprint of the process defined by the EPC, i.e. $G^{*\prime}_\varepsilon \in \mathcal{F}(\Phi_T)$.

As we stated before, the translation rules from an EPC to a causal graph are given under the assumption that the EPC is sound. However, soundness would imply that there are clear executable semantics, which is not always the case. Therefore, our translation rules do not require explicit semantics. The only requirement for our rules to hold is that soundness should be defined in such a way that a blocking connector is considered to be unsound. Under this assumption, the proof of Property 3.7 is rather intuitive, i.e. any element in the EPC is always followed by all its successors, unless it is an XOR-split or OR-split in which case the element is followed by at least one of its successors. The same applies for the join connectors.

Similar to EPCs, we derive a translation for WF-nets to causality graphs and show that the result is a causal footprint for the modelled process (cf. Figure 4).

Definition 3.8. (Workflow net to causal footprint) Let $\omega = (P, T, F)$ be a WF-net, with the set of modelling elements $N' = P \cup T$, the initial place $p_i \in P$ and final place

$p_f \in P$. We derive a causality graph $G_\omega = (N, F_{lb}, F_{la})$ as follows:

- $N = T \cup \{n_i, n_f\}$, where $n_i, n_f \notin N'$and $n_i \neq n_f$,
- $F_{la} = \{(n, N') \in N \times \mathcal{P}(N) \mid \exists_{p \in P \setminus \{p_f\}}\ n \in \overset{\omega}{\bullet} p \wedge N' = p\overset{\omega}{\bullet}\} \cup \{(n_i, p_i\overset{\omega}{\bullet})\} \cup \{(t, \{n_f\}) \mid t \in \overset{\omega}{\bullet} p_f\}$,
- $F_{lb} = \{(N', n) \in \mathcal{P}(N) \times N \mid \exists_{p \in P \setminus \{p_i\}}\ n \in p\overset{\omega}{\bullet} \wedge N' = \overset{\omega}{\bullet} p\} \cup \{(\overset{\omega}{\bullet} p_f, n_f)\} \cup \{(\{n_i\}, t) \mid t \in p_i\overset{\omega}{\bullet}\}$,

Property 3.9. (Causal graph gives a causal footprint for workflow net) Let T be a set of activities and Φ_T a process over T. Let $\omega = (P, T', F)$ be a WF-net that models the process Φ_T. Note that $T' = T$. Furthermore, let $G_\varepsilon = (N, F_{lb}, F_{la})$ be a causality graph following Definition 3.8. The causal closure of G_ω, $G^*_\omega = (N, F^*_{lb}, F^*_{la})$, is a causal footprint of the process defined by the WF-net, i.e. $G^*_\omega \in \mathcal{F}(\Phi_T)$.

WF-nets have a clear executable semantics and therefore a clear definition of soundness. For this definition, we know that a WF-net is not considered to be sound if it contains deadlocks. Therefore, the proof of Property 3.9 is again rather intuitive, i.e. a transition t is always followed by one of the transitions that consumes a token that was produced by t and vice versa.

Both for WF-nets and EPCs, we have defined a translation to causal graphs, and we have shown that when these causal graphs are projected onto the interesting elements, they are causal footprints of the process under consideration. In the next section, we show that if these causal footprints have certain properties, our assumption that the EPC or WF-net was sound is violated and therefore the process is not sound.

4. Erroneous patterns

Typically, processes are considered to be sound if and only if the process fulfills the following conditions:

- Any case that is started terminates in one of some pre-defined termination states,
- For each activity in the process there is at least one possible occurrence sequence (i.e., case execution) going from the initial state to a pre-defined termination state that contains this activity.

For WF-nets, the definition of soundness is such that there is exactly one final state, i.e. the state where there is one token in the output place and no tokens in any other place. Furthermore, there can be no dead transitions.

The definition of soundness helps to identify three typical error patterns. Two of them indicate potential errors that affect the soundness of a business process model: deadlocks (Sect. 4.1) and multiple termination (Sect. 4.2). Section 4.3 illustrates the deadlock pattern with an example from the SAP reference model. Furthermore, the trap pattern gives rise to a sufficient condition for showing that a process is not sound (Sect. 4.4).

4.1. Deadlock Pattern

Figure 5 shows a process having a simple deadlock caused by an exclusive choice followed by a synchronizing join (AND-join). The process is depicted, both as an EPC and as a Petri net. In this case, the AND-join remains waiting for both b and c to complete while only one of them is activated. Accordingly, the process cannot terminate properly and is therefore not sound. The respective Petri net contains a so-called "Place-Transition handle" (PT-handle, [13]) as there are alternative paths from one place to a transition that waits for all paths to complete.

Definition 4.1. (Deadlock Pattern) Let T be a set of activities and Φ_T a process over T. Let $G = (N, F_{lb}, F_{la})$ be a causal footprint of the process defined, i.e. $G \in \mathcal{F}(\Phi_T)$. If there is a $(a, B) \in F_{la}$ with $|B| > 1$ and there exists a $(a, \{d\}) \in F_{la}$ such that for all $b \in B$ holds that $(\{b\}, d) \in F_{lb}$ and $(a, \{b\}) \notin F_{la}$ then we say that (a, B, d) is a *Deadlock Pattern*.

In causality footprints such PT-handles always map to a set of activities B (in our example $B = \{b, c\}$) that all share one look-ahead link from a that is in the causal graph of the model and one common successor d in the look-back link set plus a look-ahead link from a to d. Furthermore, there is no *single* look-ahead link from a to B (i.e. $(a, \{b\})$ and $(a, \{c\})$ do not exist). It can be proven that a PT-handle always produces a deadlock pattern. The idea of the proof is that there can be no predecessors of a that have a look-ahead link to successors of a. Therefore, there is no simple look-ahead link from the start to activities that are successors of a and predecessors of d. Following the soundness assumption of the mapping of the AND-join, d can always be reached from any activity in B. According to the look-back links, every predecessor of d is executed which finally contradicts that the successors of a are exclusive. Yet, it is still an open question whether models without errors can

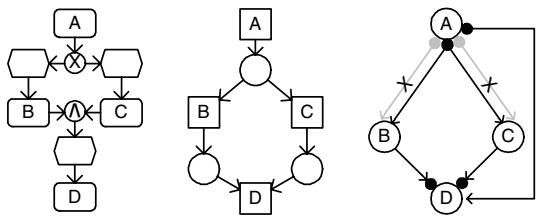

Figure 5. Deadlock in EPC and Petri net and the respective pattern in the causal footprint.

produce a deadlock pattern. The reason is that the deadlock pattern does not show that the paths from a to b and a to c are mutually exclusive. Thus, if we find a deadlock pattern between a and d in the causal footprint, it is *likely* that this really indicates a deadlock.

4.2. Multiple Termination Pattern

Figure 6 shows a process having an improper multiple instantiation both as an EPC and as a Petri net. In this case, the successors of d are executed each time a path to the XOR-join is completed. This leads to multiple terminations of the process which is not sound. The respective Petri net contains a so-called "Transition-Place handle" (TP-handle, [13]) as there are parallel paths from one transition to a set of places that propagate each token.

Definition 4.2. (Multiple-Termination Pattern) Let T be a set of activities and Φ_T a process over T. Let $G = (N, F_{lb}, F_{la})$ be a causal footprint of the process defined, i.e. $G \in \mathcal{F}(\Phi_T)$. If there is a $(B, d) \in F_{lb}$ with $|B| > 1$ and there exists a $(\{a\}, d) \in F_{lb}$ such that for all $b \in B$ holds that $(d, \{b\}) \in F_{la}$ and $(\{a\}, b) \notin F_{lb}$ then we say that (a, B, d) is a *Multiple Termination Pattern*.

In causal footprints such TP-handles always map to a set of activities B (in our example $B = \{b, c\}$) that all share one look-back link to d that is in the causal graph of the model and one common predecessor a in the look-ahead link set plus a look-back link from a to d. Furthermore, there is no single look-back link from B to d. It can be proven that a TP-handle always produces a multiple termination pattern. The idea of the proof is analogous to the proof for the deadlock pattern. Furthermore, it is still an open question, whether models without errors can produce a multiple termination pattern. The reason is that the multiple termination pattern does not enforce that the paths from b to d and from c to d are mutually exclusive paths. Thus, if we find a multiple termination pattern between a and d in the causal footprint, it is *likely* that this really indicates a multiple termination. However, in terms of an EPC for example, if the XOR-join is replaced by an OR-join, the pattern can still be found, but one might argue that the EPC is sound. This however depends on the semantics of the OR-join.

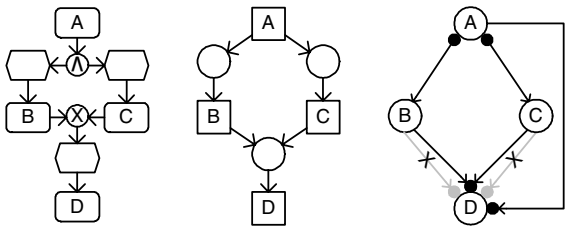

Figure 6. Multiple termination in EPC and Petri net and the respective pattern in the causal footprint.

4.3. SAP reference model example

Figure 7 gives a part of the "Release and Implementation of Measure" EPC that is part of the asset accounting branch of the SAP reference model [7, 16]. The corresponding causal footprint includes the deadlock pattern. This process part indeed contains a deadlock, e.g., taking the path starting with event "Reserved funds to reduce" leads to a deadlock before the last AND-join.

4.4. Trap Pattern

So far, we have shown that we can take a process model in terms of an EPC or WF-net and that we can describe the behaviour of the underlying process using a causal footprint. A causal footprint is declarative in nature, i.e. it describes what happens, but it does not describe how it happens. This property can be used to give conditions for soundness.

Definition 4.3. (Singular Trap Pattern) Let T be a set of activities and Φ_T a process over T. Let $G = (N, F_{lb}, F_{la})$ be a causal footprint of the process defined, i.e. $G \in \mathcal{F}(\Phi_T)$. If there is a $n \in N$, such that $(n, \{n\}) \in F_{la}$, or $(\{n\}, n) \in F_{la}$ then we say that (n) is a *Singular Trap Pattern*.

The singular trap pattern is stronger than the previous patterns, since we can say that if a process contains singular trap pattern, the process is not sound. The proof of this property is trivial. The presence of a look-ahead link $(n, \{n\})$ means that if activity n appears once in a case,

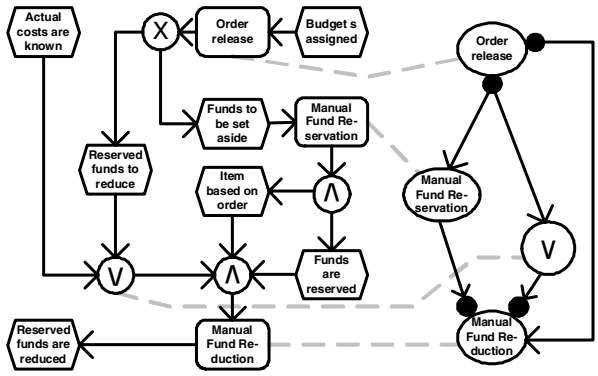

Figure 7. EPC from the SAP reference model and part of its causal footprint.

then it appears infinitely often in that case. Therefore, either the activity can never be performed, which means that the activity is dead, or there is a livelock. Similarly, if there is a look-back link $(\{n\}, n)$ then every occurrence of activity n has to be preceded by activity n. Therefore, activity n can never be performed, thus indicating that it is dead.

Definition 4.3 can easily be extended to a more general pattern.

Definition 4.4. (Generalized Trap Pattern) Let T be a set of activities and Φ_T a process over T. Let $G = (N, F_{lb}, F_{la})$ be a causal footprint of the process defined, i.e. $G \in \mathcal{F}(\Phi_T)$. If there is a non-empty set of nodes $N' \subset N$, such that for all $n' \in N'$ there is a $N'' \subseteq N'$ with $(n', N'') \in F_{la}$ or for all $n' \in N'$ there is a $N'' \subseteq N'$ with $(N'', n') \in F_{lb}$ then we say that (N) is a *Generalized Trap Pattern*.

The idea behind the generalized trap pattern is similar to the singular one and they have the same property that the process is not sound if the pattern occurs. Again the proof is straightforward, since as soon as one of the activities of the subset N' occurs, it will always lead to the execution of another element of that set, implying that there is a livelock, or the set is dead. For the look-back links the set is always dead. Note that these potential problems are related to the concepts of traps and siphons in Petri nets [9].

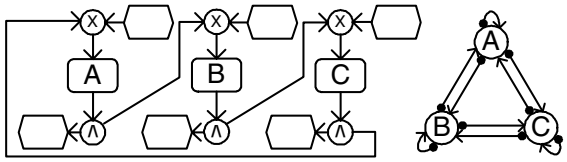

Figure 8. Vicious circle and causal footprint.

Figure 8 gives the famous EPC for the vicious circle. The EPC shows that there is a possible paradox when using multiple OR-joins and has been extensively discussed in literature(cf. [3, 17, 18]). The calculated causal footprint indicates that the model is not sound, *regardless of any semantics*, as long as a blocking connector is assumed to be unsound.

5. Tool support

The (Pro)cess (M)ining framework *ProM* has been developed as a completely plug-able environment for process mining and related topics in the context of business process analysis. It can be extended by simply adding plug-ins, i.e., there is no need to know or to recompile the source code. Currently, more than 80 plug-ins have been added. The ProM framework has been described before in [10] and can be obtained via www.processmining.org. The architecture of ProM allows for five different types of plug-ins:

Mining plug-ins which implement some mining algorithm, e.g., mining algorithms that construct a Petri net or EPC based on some event log.

Export plug-ins which implement some "save as" functionality for some objects. For example, there are plug-ins to save EPCs, Petri nets, spreadsheets, etc.

Import plug-ins which implement an "open" functionality for exported objects, e.g., load instance-EPCs from the Aris Process Performance Manager.

Analysis plug-ins which typically implement some property analysis on some mining result. For example, for Petri nets there is a plug-in which constructs place invariants, transition invariants, and a coverability graph.

Conversion plug-ins which implement conversions between different data formats, e.g., from EPCs to Petri nets.

In the context of this paper, we developed two conversion plug-ins and one analysis plug-in. First, we implemented two plug-ins to convert an EPC or a Petri net to a causal footprint. The two conversion plug-ins follow the rules presented in this paper and after conversion, they both calculate the causal closure of the result. The visualization is done in such a way that only informative relations are shown, i.e. only the relations for which there are no "larger" relations. The look-ahead links are shown in blue, the look-back links are shown in red. However, even with these restrictions, the resulting graph may still have many connections and appears to be "spaghetti-like". . Therefore, we implemented an analysis plug-in to find the erroneous patterns as described in Section 4. Figure 9 shows the result of the analysis plug-in, when applied on the EPC containing the vicious circle from Figure 8. The left-hand side shows the vicious circle in a generated layout. The right-hand side shows a selected part of the generated causal footprint. We chose to search for "Singular Trap Patterns" and the plug-in returned three results. Then, we asked it to show the pattern involving function "C", at which point the function "C" was highlighted in the EPC as well as in the causal footprint.

For the practical application of our results it is important to note that the ProM framework is currently capable of reading and writing EPCs to AML (native to the Aris Toolset), EPML (a standard for storing EPCs) and the Aris graph format used by Aris PPM. For Petri nets, multiple formats are available, including the standard PNML format. The framework is open-source and the latest release is available for download from www.processmining.org.

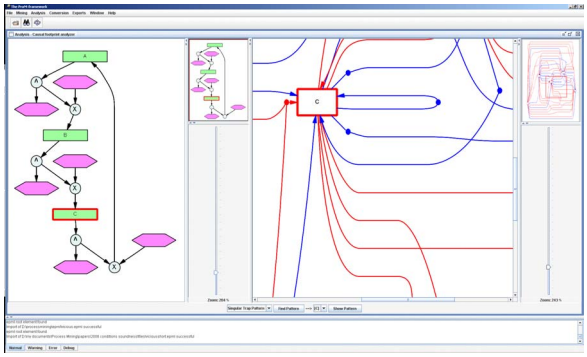

Figure 9. The vicious circle in ProM

6. Related Work

The synchronization semantics of OR-joins have been one of the focal points of research on EPCs. In [8] the so-called relaxed soundness criterion is presented to guide the modeller towards the specification of a sound WF-net from an EPC. The WofYAWL approach introduced in [24] extends this work to YAWL. The WofYAWL tool has been used to check the correctness of the EPCs in the SAP reference model [20] revealing that at least 5 % of the models have errors. In [11] an interactive verification approach is presented that builds on reduction rules. The possibility to provide executable semantics for EPCs has been investigated in [18], where executable semantics are proven to exist for a large sub-class of all EPCs. The vicious circle is an example of an EPC for which no suitable semantics exist. In [6] an approach is presented to efficiently calculate the state space of an EPC, thereby providing executable semantics for the EPC. The authors mainly motivate this work from the viewpoint of simulation/execution although their approach can also be used for verification purposes. Because of the semantical problems in some EPCs the algorithm does not always provide a result [18]. Our approach based on causal footprints identifies the vicious circle at least to be unsound, but with a much simpler calculation.

In a way our approach is related to classical approaches in Petri nets where traps, siphons, handles, and invariants are used to identify problems [1, 9, 13, 22]. In many cases these structural properties are used to derive statements on the behaviour of the Petri net. However, instead of working directly on a Petri net representation we use a more abstract representation that is language independent. Moreover, we use the causal closure to obtain our results.

Causal graphs can be seen as a declarative language, i.e., instead of using explicit control-flow operators like sequence, iterations, etc., constraints are given. In this paper we consider two types of constraints (look-back links and look-ahead links). These can easily be translated into a temporal logic [19]. In [5], DecSerFlow, the Declarative Service Flow Language, is defined which includes the two types of constraints used in this paper but also many others. This illustrates that the approach presented in this paper could be extended to include other types of constraints in the causal footprint. In [5] its is shown how these can be represented in Linear Temporal Logic (LTL). Moreover, ProM already contains an LTL checker which can check arbitrary LTL formulas on the basis of events logs [2].

7. Conclusion and Future work

In this paper, we have presented a novel approach for checking soundness of business process models. We capture the intent of a process model by deriving a *causal footprint*. Such a footprint should be seen as an abstraction of the process behaviour. As a causal footprint does not require an executable semantics of the process modelling language, we can apply our analysis technique both to formal languages such as Petri nets and to conceptual/informal languages such as EPCs. Even though we use these two languages throughout the paper to demonstrate the applicability of our technique, it can be easily adapted to other languages such as UML activity diagrams, BPMN, or BPEL. This is especially helpful regarding the heterogeneity of business process modelling languages. Based on causal footprints, we are able to identify three error patterns, namely deadlocks, multiple termination, and traps. For the latter we provide a proof that a trap pattern always implies that the model is unsound.

The work presented here scratches the surface of a new way of describing processes. Instead of taking a process model describing which activities have to be performed in which order, we derive a description of what can and cannot be done. In this paper, we only looked at two types of relations, i.e. one activity is always followed by at least one of a set of other activities (look-ahead links) and the counterpart in the other direction (look-back links). Obviously, this can be extended to more complex relations or relations of a different nature. More importantly, the causal footprint may be a way to bridge the gap between business process modelling and business rules. In future research, we aim to identify further relations and derivation rules and to provide further conditions for a process model to be sound or not.

References

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

[2] W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. In R. Meersman and Z. Tari et al., editors, *OTM Conferences 2005*,

volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, Berlin, 2005.

[3] W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In M. Nüttgens and F. J. Rump, editor, *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*, pages 71–79, 2002.

[4] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[5] W.M.P. van der Aalst and M. Pesic. Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware. BPM Center Report BPM-06-09, BPMcenter.org, 2006.

[6] N. Cuntz, J. Freiheit, and E. Kindler. On the Semantics of EPCs: Faster Calculation for EPCs with Small State Spaces. In M. Nuettgens and F.J. Rump, editors, *Proceedings of Fourth Workshop on Event-Driven Process Chains (WI-EPK 2005)*, pages 7–23, Hamburg, Germany, December 2005. Gesellschaft fuer Informatik, Bonn.

[7] T. Curran and G. Keller A. Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Enterprise Resource Planning Series. Prentice Hall PTR, Upper Saddle River, 1997.

[8] J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.

[9] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[10] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.

[11] B.F. van Dongen, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of EPCs: Using reduction rules and Petri nets. In *Conference on Advanced Information Systems Engineering (CAiSE 2005)*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Berlin, 2005.

[12] H. Eertink, W. Janssen, P. Oude Luttighuis, W. B. Teeuw, and C. A. Vissers. A business process design language. In *FM '99: Proceedings of the World Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 76–95, London, UK, 1999. Springer-Verlag.

[13] J. Esparza and M. Silva. Circuits, Handles, Bridges and Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 210–242. Springer-Verlag, Berlin, 1990.

[14] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[15] G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[16] G. Keller and T. Teufel. *SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998.

[17] E. Kindler. On the semantics of EPCs: Resolving the vicious circle. In J. Desel and B. Pernici and M. Weske, editor, *Business Process Management, 2nd International Conference, BPM 2004*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97, 2004.

[18] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.

[19] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[20] J. Mendling, M. Moser, G. Neumann, H.M.W. Verbeek, B.F. van Dongen, and W.M.P. van der Aalst. A quantitative analysis of faulty epcs in the sap reference model. BPM Center Report BPM-06-08, BPMcenter.org, 2006.

[21] J. Mendling, G. Neumann, and M. Nüttgens. *The Workflow Handbook 2005*, chapter A Comparison of XML Interchange Formats for Business Process Modelling, pages 185–198. Future Strategies Inc., 2005.

[22] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[23] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[24] H.M.W. Verbeek, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Verifying workflows with cancellation regions and OR-joins: An approach based on invariants. BETA Working Paper Series, WP 156, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.