Causal Nets: A Modeling Language Tailored Towards Process Discovery

W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
{W.M.P.v.d.Aalst,A.Adriansyah,B.F.v.Dongen}@tue.nl

Abstract. Process discovery—discovering a process model from example behavior recorded in an event log—is one of the most challenging tasks in process mining. The primary reason is that conventional modeling languages (e.g., Petri nets, BPMN, EPCs, and ULM ADs) have difficulties representing the observed behavior properly and/or succinctly. Moreover, discovered process models tend to have deadlocks and livelocks. Therefore, we advocate a new representation more suitable for process discovery: *causal nets*. Causal nets are related to the representations used by several process discovery techniques (e.g., heuristic mining, fuzzy mining, and genetic mining). However, unlike existing approaches, we provide declarative semantics more suitable for process mining. To clarify these semantics and to illustrate the non-local nature of this new representation, we relate causal nets to Petri nets.

1 Motivation

In this paper, we advocate the use of *Causal-nets* (*C-nets*) in process mining. C-nets were introduced in [2] and, in our view, provide a better representational bias for process discovery than conventional design-oriented languages such as Petri nets, BPMN, BPEL, EPCs, YAWL, and UML activity diagrams.

Figure 1 shows a C-net modeling the booking of a trip. After activity a (*start booking*) there are three possible activities: b (*book flight*), c (*book car*), and d (*book hotel*). The process ends with activity e (*complete booking*). Each activity has sets of potential input and output bindings (indicated by the black dots). Every connected set of dots on the output arcs of an activity is an output binding. For example, a has four output bindings modeling that a may be followed by (1) just b, (2) just c, (3) b and d, or (4) b, c, and d. Hence, it is not possible to book just a hotel or a hotel and a car. Activity c has two input bindings modeling that it is preceded by (1) just a or (2) a and b. This construct is used to model that when both a flight and a car are booked, the flight is booked first. Output bindings create *obligations* whereas input bindings remove obligations. For example, the occurrence of a with output binding $\{b, d\}$ creates two obligations: both b and d need to be executed while referring to the obligations created by a.

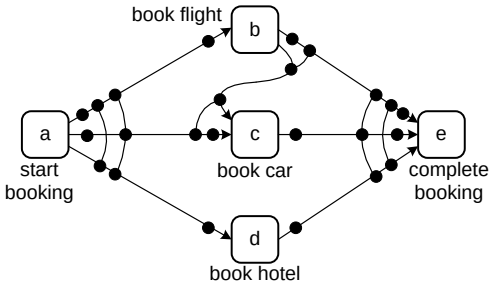


Fig. 1. Causal net C_{travel}

In a C-net there is one start activity (a in Fig. 1) and one end activity (e in Fig. 1). A *valid binding sequence* models an execution path starting with a and ending with e while removing all obligations created during execution. The behavior of a C-net is *restricted* to valid binding sequences. Hence, unlike conventional modeling languages, the semantics are non-local. Section 2 explains the semantics of C-nets in more detail and provides additional examples.

C-nets address important limitations of conventional languages in the context of *process mining* [2]. Process mining is an emerging research discipline focusing on the interplay between *event logs* (observed behavior) and process models. *Process discovery* is the process mining task that aims to learn process models based on example behavior recorded in events logs, e.g., based on a multi-set of activity sequences (process instances) a Petri net that models the observed behavior is discovered. *Conformance checking* is the process mining task that compares the example behavior in a events log with the modeled behavior. Based on such a comparison it is possible to highlight and quantify commonalities and differences.

In the last decade dozens of new process discovery techniques have been proposed, typically aiming at the creation of a conventional process model (e.g., a Petri net or EPC). This means that the search space that is implied by such a design-oriented language—often referred to as the "representational bias"—is not tailored towards process mining. This creates various problems. In this paper, we focus on two of them:

- The discovered process model is *unable to represent the underlying process well*, e.g., a significant proportion of the behavior seen in the log is not possible in the model (non-fitting model), the model allows for behavior not related to the event log (underfitting), the model is overfitting (no generalization), or the model is overly complex because all kinds of model elements need to be introduced without a direct relation to the event log (e.g., places, gateways, and events).
- Most of the process models in the search space determined by conventional languages are *internally inconsistent* (deadlocks, livelocks, etc.), i.e., there are more inconsistent models than consistent ones. Process discovery tech-

niques need to "guess" the underlying model based on example behavior. If almost all of these guesses result in models that are obviously incorrect (even without considering the event log), then the results are of little value.

Consider for example an algorithm producing a Petri net (e.g., the various region-based approaches [11] and variants of the α-algorithm [2]). The behavior in a Petri net can be restricted by adding places. However, places have no direct meaning in terms of the behavior seen in the event log. Moreover, the addition or removal of places may introduce deadlocks, livelocks, etc.

This is the reason why the more useful process discovery techniques use alternative representations: *fuzzy models* [7], *heuristic nets* [9], *flexible heuristic nets* [10], *causal matrices* [8], etc. Also for conformance checking one can find similar representations, e.g., *flexible models* [4]. On the one hand, these representations are similar to C-nets (i.e., activities can model XOR/OR/AND-splits and joins without introducing separate model elements). On the other hand, the semantics of such models are very different from the semantics we use for C-nets. The distinguishing feature is that we *limit the possible behavior to valid binding sequences*, thus excluding a variety of anomalies.

This paper introduces C-nets while focusing on their semantics (Sect. 2). We believe that our formalization sheds new light on the representations used in [4,7,8,9,10]. We also provide two mappings: one from C-nets to Petri nets and one from Petri nets to C-nets (Sect. 3). These mappings help to clarify the semantics and highlight the distinguishing features of C-nets. Moreover, to illustrate the practical relevance of C-nets, we describe how the ProM framework is supporting/using C-nets (Sect. 4).

2 Causal Nets

This section introduces *causal nets* – a representation tailored towards process mining – and their semantics.

2.1 Definition

A Causal-net (C-net) is a graph where nodes represent activities and arcs represent causal dependencies. Each activity has a set of possible *input bindings* and a set of possible *output bindings*. Consider, for example, the C-net shown in Fig. 2. Activity a has only an empty input binding as this is the start activity. There are two possible output bindings: $\{b, d\}$ and $\{c, d\}$. This means that a is followed by either b and d, or c and d. Activity e has two possible input bindings ($\{b, d\}$ and $\{c, d\}$) and three possible output bindings ($\{g\}$, $\{h\}$, and $\{f\}$). Hence, e is preceded by either b and d, or c and d, and is succeeded by just g, h or f. Activity z is the end activity having two input bindings and one output binding (the empty binding). This activity has been added to create a unique end point. All executions commence with start activity a and finish with end activity z. Note that unlike, Petri nets, there are no places in the C-net; the routing logic is solely represented by the possible input and output bindings.

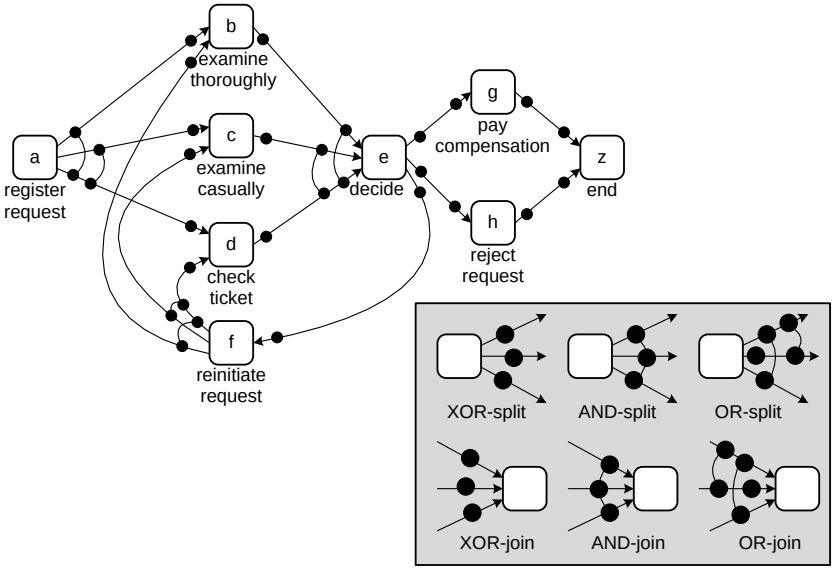


Fig. 2. C-net C_{rfc} modeling a "Request For Compensation" (RFC) process

Definition 1 (Causal net [2]). *A* Causal net *(C-net) is a tuple* $C = (A, a_i, a_o, D, I, O)$ *where:*

- A *is a finite set of* activities*;*
- $a_i \in A$ *is the* start activity*;*
- $a_o \in A$ *is the* end activity*;*
- $D \subseteq A \times A$ *is the* dependency relation*,*
- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \ \vee \ \emptyset \notin X\}$*;*[1]
- $I \in A \to AS$ *defines the set of possible* input bindings *per activity; and*
- $O \in A \to AS$ *defines the set of possible* output bindings *per activity,*

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\}$*;*
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in O(a_1)} as\}$*;*
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$*;*
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$*; and*
- *all activities in the graph* (A, D) *are on a path from* a_i *to* a_o*.*

The C-net of Fig. 2 can be described as follows. $A = \{a, b, c, d, e, f, g, h, z\}$ is the set of activities, $a = a_i$ is the unique start activity, and $z = a_o$ is the unique end activity. The arcs shown in Fig. 2 visualize the dependency relation $D = \{(a, b), (a, c), (a, d), (b, e), \ldots, (g, z), (h, z)\}$. Functions I and O describe the

[1] $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of A. Hence, elements of AS are *sets of sets* of activities.

sets of possible input and output bindings. $I(a) = \{\emptyset\}$ is the set of possible input bindings of a, i.e., the only input binding is the empty set of activities. $O(a) = \{\{b, d\}, \{c, d\}\}$ is the set of possible output bindings of a, i.e., activity a is followed by d and either b or c. $I(b) = \{\{a\}, \{f\}\}$, $O(b) = \{\{e\}\}$, ..., $I(z) = \{\{g\}, \{h\}\}$, $O(z) = \{\emptyset\}$. Note that any element of AS is a set of sets of activities, e.g., $\{\{b, d\}, \{c, d\}\} \in AS$. If one of the elements is the empty set, then there cannot be any other elements, i.e., for any any $X \in AS$: $X = \{\emptyset\}$ or $\emptyset \notin X$. This implies that only the unique start activity a_i has the empty binding as (only) possible input binding. Similarly, only the unique end activity a_o has the empty binding as (only) possible output binding.

An *activity binding* is a tuple (a, as^I, as^O) denoting the occurrence of activity a with input binding as^I and output binding as^O. For example, $(e, \{b, d\}, \{f\})$ denotes the occurrence of activity e in Fig. 2 while being preceded by b and d, and succeeded by f.

Definition 2 (Binding). *Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $B = \{(a, as^I, as^O) \in A \times \mathcal{P}(A) \times \mathcal{P}(A) \mid as^I \in I(a) \ \wedge \ as^O \in O(a)\}$ is the set of* activity bindings. *A* binding sequence *σ is a sequence of activity bindings, i.e., $\sigma \in B^*$.*

Note that sequences are denoted using angle brackets, e.g., $\langle\rangle$ denotes the empty sequence. B^* is the set of all sequences over B (including $\langle\rangle$). A possible binding sequence for the C-net of Fig. 2 is $\sigma_{ex} = \langle(a, \emptyset, \{b, d\}), (b, \{a\}, \{e\}), (d, \{a\}, \{e\}), (e, \{b, d\}, \{g\}), (g, \{e\}, \{z\}), (z, \{g\}, \emptyset)\rangle$.

Function $\alpha \in B^* \rightarrow A^*$ projects binding sequences onto *activity sequences*, i.e., the input and output bindings are abstracted from and only the activity names are retained. For instance, $\alpha(\sigma_{ex}) = \langle a, b, d, e, g, z\rangle$.

Consider C-net C_{travel} shown in Figure 1. The possible input and output bindings of C_{travel} are defined as follows: $O(a) = I(e) = \{\{b\}, \{c\}, \{b, d\}, \{b, c, d\}\}$, $I(a) = O(e) = \{\emptyset\}$, $I(b) = I(d) = \{\{a\}\}$, $O(c) = O(d) = \{\{e\}\}$, $I(c) = \{\{a\}, \{a, b\}\}$, and $O(b) = \{\{e\}, \{c, e\}\}$. A possible binding sequence for the C-net shown in Fig. 1 is $\sigma = \langle(a, \emptyset, \{b, c, d\}), (d, \{a\}, \{e\}), (b, \{a\}, \{c, e\}), (c, \{a, b\}, \{e\}), (e, \{b, c, d\}, \emptyset)\rangle$, i.e., the scenario in which a hotel, a flight, and a car are booked. $\alpha(\sigma) = \langle a, d, b, c, e\rangle$ is the corresponding activity sequence. Note that in Fig. 1 a hotel can only be booked if a flight is booked. Moreover, when both a car and a flight are booked, then first the flight needs to be booked.

2.2 Valid Sequences

A binding sequence is *valid* if a predecessor activity and successor activity always "agree" on their bindings. For a predecessor activity x and successor activity y we need to see the following "pattern": $\langle \ldots, (x, \{\ldots\}, \{y, \ldots\}), \ldots, (y, \{x, \ldots\}, \{\ldots\}), \ldots\rangle$, i.e., an occurrence of activity x with y in its output binding needs to be followed by an occurrence of activity y, and an occurrence of activity y with x in its input binding needs to be preceded by an occurrence of activity x. To formalize the notion of a valid sequence, we first define the notion of *state*. States are represented by multi-sets of *obligations*, e.g., state

$[(a,b)^2,(a,c)]$ denotes the state where there are two pending activations of b by a and there is one pending activation of c by a. This means that b needs to happen twice while having a in its input binding and c needs to happen once while having a in its input binding.

Definition 3 (State). *Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $S = \mathbb{B}(A \times A)$ is the* state space *of C. $s \in S$ is a* state*, i.e., a multi-set of pending* obligations*. Function $\psi \in B^* \to S$ is defined inductively: $\psi(\langle\,\rangle) = [\,]$ and $\psi(\sigma \oplus (a, as^I, as^O)) = (\psi(\sigma) \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$ for any binding sequence $\sigma \oplus (a, as^I, as^O) \in B^*$.*[2] *$\psi(\sigma)$ is the state after executing binding sequence σ.*

Consider C-net C_{rfc} shown in Fig. 2. Initially there are no pending "obligations", i.e., no output bindings have been enacted without having corresponding input bindings. If activity binding $(a, \emptyset, \{b,d\})$ occurs, then $\psi(\langle(a,\emptyset,\{b,d\})\rangle) = \psi(\langle\,\rangle) \setminus (\emptyset \times \{a\}) \uplus (\{a\} \times \{b,d\}) = ([\,] \setminus [\,]) \uplus [(a,b),(a,d)] = [(a,b),(a,d)]$. State $[(a,b),(a,d)]$ denotes the obligation to execute both b and d using input bindings involving a. Input bindings remove pending obligations whereas output bindings create new obligations.

A *valid sequence* is a binding sequence that (1) starts with start activity a_i, (2) ends with end activity a_o, (3) only removes obligations that are pending, and (4) ends without any pending obligations. Consider, for example, the valid sequence $\sigma = \langle(a,\emptyset,\{b,d\}),(d,\{a\},\{e\}),(b,\{a\},\{e\}),(e,\{b,d\},\emptyset)\rangle$ for C-net C_{travel} in Fig. 1:

$$\begin{aligned}
\psi(\langle\,\rangle) &= [\,] \\
\psi(\langle(a,\emptyset,\{b,d\})\rangle) &= [(a,b),(a,d)] \\
\psi(\langle(a,\emptyset,\{b,d\}),(d,\{a\},\{e\})\rangle) &= [(a,b),(d,e)] \\
\psi(\langle(a,\emptyset,\{b,d\}),(d,\{a\},\{e\}),(b,\{a\},\{e\})\rangle) &= [(b,e),(d,e)] \\
\psi(\langle(a,\emptyset,\{b,d\}),(d,\{a\},\{e\}),(b,\{a\},\{e\}),(e,\{b,d\},\emptyset)\rangle) &= [\,]
\end{aligned}$$

Sequence σ indeed starts with start activity a, ends with end activity e, only removes obligations that are pending (i.e., for every input binding there was an earlier output binding), and ends without any pending obligations: $\psi(\sigma) = [\,]$.

Definition 4 (Valid). *Let $C = (A, a_i, a_o, D, I, O)$ be a C-net and $\sigma = \langle(a_1, as_1^I, as_1^O), (a_2, as_2^I, as_2^O), \ldots, (a_n, as_n^I, as_n^O)\rangle \in B^*$ be a binding sequence. σ is a* valid sequence *of C if and only if:*

- *$a_1 = a_i$, $a_n = a_o$, and $a_k \in A \setminus \{a_i, a_o\}$ for $1 < k < n$;*
- *$\psi(\sigma) = [\,]$; and*
- *for any non-empty prefix $\sigma' = \langle(a_1, as_1^I, as_1^O), \ldots, (a_k, as_k^I, as_k^O)\rangle$ $(1 \leq k \leq n)$: $(as_k^I \times \{a_k\}) \leq \psi(\sigma'')$ with $\sigma'' = \langle(a_1, as_1^I, as_1^O), \ldots, (a_{k-1}, as_{k-1}^I, as_{k-1}^O)\rangle$*

$V_{CN}(C)$ is the set of all valid sequences of C.

[2] $\oplus$ is used to concatenate an element to the end of a sequence, e.g., $\langle a,b,c\rangle \oplus d = \langle a,b,c,d\rangle$. $X \uplus Y$ is the union of two multi-sets. $X \setminus Y$ removes Y from X (difference of two multi-sets). Ordinary sets will be used as multi-sets throughout this paper.

The first requirement states that valid sequences start with a_i and end with a_o (a_i and a_o cannot appear in the middle of valid sequence). The second requirement states that at the end there should not be any pending obligations. (One can think of this as the constraint that no tokens left in the net.) The last requirement considers all non-empty prefixes of σ: $\langle (a_1, as_1^I, as_1^O), \ldots, (a_k, as_k^I, as_k^O) \rangle$. The last activity binding of the prefix (i.e., (a_k, as_k^I, as_k^O)) should only remove pending obligations, i.e., $(as_k^I \times \{a_k\}) \leq \psi(\sigma'')$ where $as_k^I \times \{a_k\}$ are the obligations to be removed and $\psi(\sigma'')$ are the pending obligations just before the occurrence of the k-th binding. (One can think of this as the constraint that one cannot consume tokens that have not been produced.)

The C-net in Fig. 1 has seven valid sequences: only b is executed ($\langle (a, \emptyset, \{b\}), (b, \{a\}, \{e\}), (e, \{b\}, \emptyset) \rangle$), only c is executed (besides a and e), b and d are executed (two possibilities), and b, c and d are executed (3 possibilities because b needs to occur before c). The C-net in Fig. 2 has infinitely many valid sequences because of the loop construct involving f.

For the semantics of a C-net we only consider valid sequences, i.e., *invalid sequences are not part of the behavior* described by the C-net. This means that C-nets do not use plain "token-game semantics" as employed in conventional languages like BPMN, Petri nets, EPCs, and YAWL. The semantics of C-nets are more declarative as they are defined over complete sequences rather than a local firing rule. Note that the semantics abstract from the moment of choice; pending obligations are not exposed to the environment and are not fixed during execution (i.e., all *valid* interpretations remain open).

2.3 Soundness

The notion of *soundness* has been defined for a variety of workflow and business process modeling notations (e.g., workflow nets as shown in Sect. 3.1). A process model is *sound* if it is free of deadlocks, livelocks, and other obvious anomalies. A similar notion can be defined for C-nets.

Definition 5 (Soundness of C-nets [2]). *A C-net $C = (A, a_i, a_o, D, I, O)$ is* sound *if (1) for all $a \in A$ and $as^I \in I(a)$ there exists a $\sigma \in V_{CN}(C)$ and $as^O \subseteq A$ such that $(a, as^I, as^O) \in \sigma$, and (2) for all $a \in A$ and $as^O \in O(a)$ there exists a $\sigma \in V_{CN}(C)$ and $as^I \subseteq A$ such that $(a, as^I, as^O) \in \sigma$.*

Since the semantics of C-nets already enforce "proper completion" and the "option to complete", we only need to make sure that there are valid sequences and that all parts of the C-net can potentially be activated by such a valid sequence. The C-nets C_{travel} and C_{rfc} in Figs. 1 and 2 are sound. Figure 3 shows two C-nets that are not sound. In Fig. 3(a), there are no valid sequences because none of output bindings of a matches any of the input bindings of e. For example, consider the binding sequence $\sigma = \langle (a, \emptyset, \{b\}), (b, \{a\}, \{e\}) \rangle$. Sequence σ cannot be extended into a valid sequence because $\psi(\sigma) = [(b, e)]$ and $\{b\} \notin I(e)$, i.e., the input bindings of e do not allow for just booking a flight whereas the output bindings of a do. In Fig. 3(b), there are valid sequences,

e.g., $\langle (a, \emptyset, \{c\}), (c, \{a\}, \{e\}), (e, \{c\}, \emptyset) \rangle$. However, not all bindings appear in one or more valid sequences. For example, the output binding $\{b\} \in O(a)$ does not appear in any valid sequence, i.e., after selecting just a flight the sequence cannot be completed properly. The input binding $\{c, d\} \in I(e)$ also does not appear in any valid sequence, i.e., the C-net suggests that only a car and hotel can be booked but there is no corresponding valid sequence.

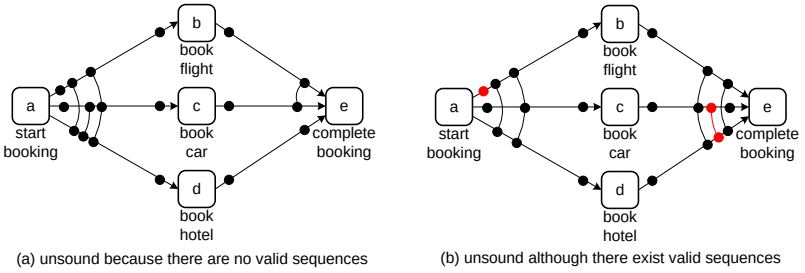


Fig. 3. Two C-nets that are not sound. The first net (a) does not allow for any valid sequence, i.e., $V_{CN}(C) = \emptyset$. The second net (b) has valid sequences but also shows input/output bindings that are not realizable (indicated in red)

Figure 4 shows another C-net. One of the valid binding sequences for this C-net is $\langle (a, \emptyset, \{b\}), (b, \{a\}, \{b, c\}), (b, \{b\}, \{c, d\}), (c, \{b\}, \{d\}), (c, \{b\}, \{d\}), (d, \{b, c\}, \{d\}), (d, \{c, d\}, \{e\}), (e, \{d\}, \emptyset) \rangle$, i.e., the sequence $\langle a, b, b, c, c, d, d, e \rangle$. This sequence covers all the bindings. Therefore, the C-net is sound. Examples of other valid sequences are $\langle a, b, c, d, e \rangle$, $\langle a, b, c, b, c, d, d, e \rangle$, and $\langle a, b, b, b, c, c, c, d, d, d, e \rangle$.

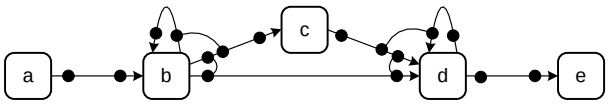


Fig. 4. A sound C-net for which there does not exist a WF-net having the same set of activity sequences

C-nets are particularly suitable for process mining given their declarative nature and expressiveness without introducing all kinds of additional model elements (places, conditions, events, gateways, etc.). Several process discovery use similar representations [7,8,9,10]. However, these models tend to use rather informal semantics; the model serves more like a "picture" showing dependencies rather than an end-to-end process model.

3 Relating C-nets and Petri Nets

To better understand the semantics of C-nets, we relate C-nets to Petri nets. We provide a mapping from WF-nets to C-nets and show that the resulting C-net is behaviorally equivalent to the original WF-net. We also provide a mapping from C-nets to WF-nets that over-approximates the behavior.

3.1 Petri Nets and WF-nets

We assume that the reader is familiar with Petri nets. Therefore, we just summarize the basic concepts and notations relevant for the two mappings.

Definition 6 (Petri net). *A* Petri net *is a triplet $N = (P, T, F)$ where P is a finite set of* places*, T is a finite set of* transitions *such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the* flow relation*. A* marked *Petri net is a pair (N, M), where $N = (P, T, F)$ is a Petri net and where $M \in \mathcal{B}(P)$ is a* multi-set *over P denoting the* marking *of the net.*

Petri nets are defined in the standard way. Markings, i.e., states of the net, are denoted as multi-sets. For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ (input nodes) and $x\bullet = \{y \mid (x, y) \in F\}$ (output nodes). A transition t is *enabled* if each of its input places $\bullet t$ contains at least one token. An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the input places $t\bullet$. Formally: $(M \setminus \bullet t) \uplus t\bullet$ is the marking resulting from firing enabled transition t in marking M.

A sequence $\sigma \in T^*$ is called a *firing sequence* of (N, M_0) if and only if, for some $n \in \{0, 1, \ldots\}$, there exist markings $M_1, \ldots, M_n$ and transitions $t_1, \ldots, t_n \in T$ such that $\sigma = \langle t_1 \ldots t_n \rangle$ and, for all i with $0 \leq i < n$, t_{i+1} is enabled in M_i and firing t_{i+1} results in marking M_{i+1}.

For business process modeling and process mining, often a restricted class of Petri nets is used: *Workflow nets (WF-nets)* [1,3]. The reason is that *process instances* have a clear starting and ending point. For example, a customer order, a patient treatment, a request for a mortgage, etc. all have a life-cycle with a well-defined start and end. Process instances are often referred to as *cases*. A WF-net describes the life-cycle of such cases.

Definition 7 (Workflow net [1]). *Petri net $N = (P, T, F)$ is a* workflow net *(WF-net) if and only if (1) P contains an input place p_i (also called source place) such that $\bullet p_i = \emptyset$, (2) P contains an output place p_o (also called sink place) such that $p_o\bullet = \emptyset$, and (3) every node is on a path from p_i to p_o.*

Cases start in the marking $[p_i]$ (one token in the unique source place) and ideally end in the marking $[p_o]$ (one token in the unique sink place). The WF-net should ensure that it is always possible to reach the final marking $[p_o]$. Moreover, a WF-net should not contain dead parts, i.e., parts that can never be activated. These requirements result in the classical definition of soundness for WF-nets.

Definition 8 (Soundness [1,3]). *Let $N = (P, T, F)$ be a WF-net with input place p_i and output place p_o. N is* sound *if and only if (1) for any marking reachable from $[p_i]$ it is possible to reach the marking $[p_o]$ (*option to complete*), and (2) $(N, [p_i])$ contains no dead transitions (*absence of dead parts*, i.e., for any $t \in T$, there is a firing sequence enabling t).*

We are interested in the set $V_{PN}(N)$ of all firing sequences that start in marking $[p_i]$ and end in marking $[p_o]$. Note that in a sound WF-net, all full firing sequences (i.e., firing sequences ending in a dead marking) are valid.

Definition 9 (Valid firing sequences). *Let $N = (P, T, F)$ be a WF-net. $V_{PN}(N) \subseteq T^*$ is the set of all* valid firing sequences*, i.e., firing sequences starting in marking $[p_i]$ and ending in marking $[p_o]$.*

At first sight, C-nets seem to be related to *zero-safe nets* [5]. The places in a zero-safe net are partitioned into *stable-places* and *zero-places*. Observable markings only mark stable-places, i.e., zero-places need to be empty. In-between observable markings zero-places may be temporarily marked. However, zero-places cannot be seen as bindings because the obligations between two activities may be non-local, i.e., an output binding may create the obligation to execute an activity occurring much later in the process.

3.2 Mapping WF-nets onto C-nets

Any sound WF-net can be transformed into an equivalent C-net by converting places into activities with XOR-join and XOR-split bindings. The idea is sketched in Fig. 5 and can be formalized as follows.

Definition 10 (Mapping I). *Let $N = (P, T, F)$ be a WF-net with input place p_i and output place p_o. $C_N = (A, a_i, a_o, D, I, O)$ is the corresponding C-net with $A = T \cup P$, $a_i = p_i$, $a_o = p_o$, $D = F$, $I(t) = \{\bullet t\}$ and $O(t) = \{t\bullet\}$ for $t \in T$, and $I(p) = \{\{t\} \mid t \in \bullet p\}$ and $O(p) = \{\{t\} \mid t \in p\bullet\}$ for $p \in P$.*

To relate valid firing sequences in WF-nets to valid binding sequences in C-nets, we define a generic projection operation. $\sigma \uparrow Y$ is the projection of some sequence $\sigma \in X^*$ onto some subset $Y \subseteq X$, i.e., elements of σ not in Y are removed. This operation can be generalized to sets of sequences, e.g., $\{\langle a, b, c, a, b, c, d\rangle, \langle b, b, d, e\rangle\} \uparrow \{a, b\} = \{\langle a, b, a, b\rangle, \langle b, b\rangle\}$.

Theorem 1. *Let $N = (P, T, F)$ be a sound WF-net having C_N as its corresponding C-net.*

- *For any valid firing sequence $\sigma_N \in V_{PN}(N)$, there exists a valid binding sequence $\sigma_C \in V_{CN}(C_N)$ such that $\alpha(\sigma_C) \uparrow T = \sigma_N$.*
- *For any valid binding sequence $\sigma_C \in V_{CN}(C_N)$, there exists a valid firing sequence $\sigma_N \in V_{PN}(N)$ such that $\alpha(\sigma_C) \uparrow T = \sigma_N$.*

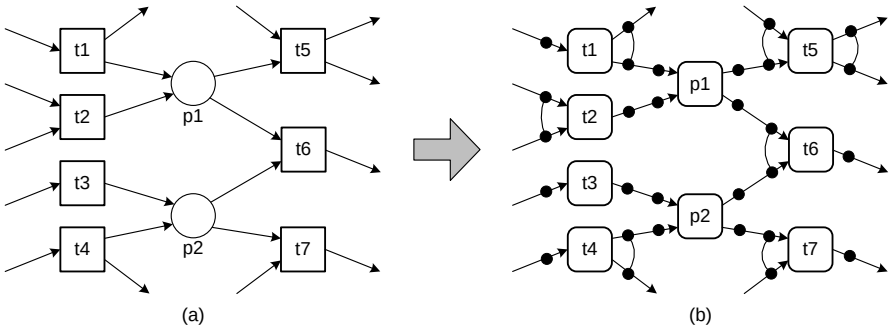


Fig. 5. Mapping a fragment of a WF-net (a) onto a C-net (b) using Definition 10

Proof. Let σ_N be a valid firing sequence of N. Replay σ_N on N while labeling each token with the name of the transition that produced it. Suppose that $t6$ in Fig. 5 fires while consuming a token from $p1$ produced by $t2$ and a token from $p2$ produced by $t3$. This occurrence of $t6$ corresponds to the subsequence $\langle \ldots, (p1, \{t2\}, \{t6\}), (p2, \{t3\}, \{t6\}), (t6, \{p1, p2\}, \{\ldots\}) \rangle$. This way it is possible to construct a valid binding sequence σ_C. Note that there may be multiple valid binding sequences corresponding to σ_N.

Let σ_C be a valid binding sequence. It is easy to see that σ_C can be replayed on the WF-net. In fact, one can simply abstract from "place activities" as these correspond to routing decisions not relevant for WF-nets (only the presence of a token matters not where the token came from). Therefore, each σ_C corresponds to a single σ_N. □

C-nets are at least as expressive as sound WF-nets because all valid firing sequences in N have a corresponding valid binding sequence in C_N and vice-versa. The reverse does not hold as is illustrated by Fig. 4. This model is unbounded and has infinitely many binding sequences. Since sound WF-nets are bounded [1,3], they can never mimic the behavior of the C-net in Fig. 4.

3.3 Mapping C-nets onto WF-nets

Figure 4 illustrates that WF-nets are not as expressive as C-net. Nevertheless, it is interesting to construct WF-nets that over-approximate the behavior of C-nets.

Definition 11 (Mapping II). *Let $C = (A, a_i, a_o, D, I, O)$ be a C-net. $N_C = (P, T, F)$ is the corresponding WF-net with $P = \{p^I_a \mid a \in A\} \cup \{p^O_a \mid a \in A\} \cup \{p^D_{(a_1,a_2)} \mid (a_1, a_2) \in D\}$, $T^I = \{a^I_X \mid a \in A \ \wedge \ X \in I(a) \ \wedge \ X \neq \emptyset\}$, $T^O = \{a^O_X \mid a \in A \ \wedge \ X \in O(a) \ \wedge \ X \neq \emptyset\}$, $T = A \cup T^I \cup T^O$, $F = \{(p^I_a, a) \mid a \in A\} \cup \{(a, p^O_a) \mid a \in A\} \cup \{(a^I_X, p^I_a) \mid a^I_X \in T^I\} \cup \{(p^O_a, a^O_X) \mid a^O_X \in T^O\} \cup \{(p^D_{(a_1,a)}, a^I_X) \mid a^I_X \in T^I \wedge a_1 \in X\} \cup \{(a^O_X, p^D_{(a,a_2)}) \mid a^O_X \in T^O \wedge a_2 \in X\}$.*

Figure 6 illustrates this construction. The black transitions correspond to silent transitions (often referred to as τ transitions). Since there is a unique start activity a_i, there is one source place $p_i = p^I_{a_i}$. Moreover, there is one sink place $p_o = p^O_{a_o}$ and all nodes are on a path from p_i to p_o. Therefore, N_C is indeed a WF-net.

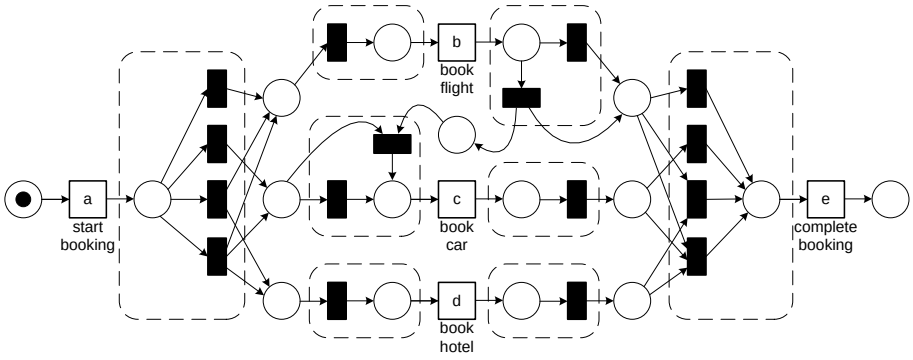


Fig. 6. A C-net transformed into a WF-net: every valid firing sequence of the WF-net corresponds to a valid sequence of the C-net C_{travel} shown in in Fig. 1 and vice versa

It is easy to see that Definition 11 is such that the WF-net can mimic any valid binding sequence. However, the corresponding WF-net does not need to be sound and may have a firing sequence that cannot be extended into a valid firing sequence.

Theorem 2. *Let $C = (A, a_i, a_o, D, I, O)$ be a C-net having N_C as its corresponding WF-net.*

- *For any valid binding sequence $\sigma_C \in V_{CN}(C)$, there exists a valid firing sequence $\sigma_N \in V_{PN}(N_C)$ such that $\alpha(\sigma_C) = \sigma_N \uparrow A$.*
- *For any valid firing sequence $\sigma_N \in V_{PN}(N_C)$, there exists a valid binding sequence $\sigma_C \in V_{CN}(C)$ such that $\alpha(\sigma_C) = \sigma_N \uparrow A$.*

Proof. It is easy to construct a valid firing sequence σ_N for any valid binding sequence σ_C. An activity binding (a, X, Y) in σ_C corresponds to the firing subsequence $\langle a^I_X, a, a^O_Y \rangle$ in σ_N. (For the start and end activity, a^I_X respectively a^O_Y are omitted.) The constructed sequence meets all requirements.

Let σ_N be a valid firing sequence. Consider the occurrence of a transition $a \in A$ in σ_N. Based on the structure of the WF-net it can be seen that a was preceded by a *corresponding* transition in T^I (unless $a = a_i$) and will be followed by a *corresponding* transition in T^O (unless $a = a_o$). The reason is that a has a dedicated input place (no other transition can consume from it) and a dedicated output place (no other transition can add tokens) and that after executing σ_N only $p^O_{a_o}$ is marked. Hence, for every occurrence of some transition $a \in A$ there

is a corresponding occurrence of a transition $a^I_X \in T^I$ and a corresponding occurrence of a transition $a^O_Y \in T^O$. This information can be used to construct $\sigma_C \in V_{CN}(C)$ such that $\alpha(\sigma_C) = \sigma_N \uparrow A$. □

The theorem shows that the expressiveness of C-nets is due its declarative semantics which considers only valid binding sequences (and not the notation itself). If one restricts WF-nets to valid firing sequences (and allows for silent transitions!), the same expressiveness is achieved.[3] Note that this is related to the notion of *relaxed soundness* [6]. In fact, a C-net C is sound if and only if the corresponding WF-net N_C is relaxed sound. In [6] it is shown that for some relaxed sound WF-nets a corresponding sound WF-net can be constructed.

4 Application of C-Nets in ProM

In the previous sections we introduced C-nets and related them to Petri nets. After these theoretical considerations, we briefly describe the way in which the *ProM framework* supports C-nets. ProM is an open-source process analysis tool with a pluggable architecture. Originally, the focus of ProM was exclusively on process mining. However, over time the scope of the system broadened to also include other types of analysis (e.g., verification). In the remainder, we provide a brief overview of ProM's functionality. Note that we show just a fraction of the hundreds of plug-ins available (cf. www.processmining.org).

4.1 Model Management and Conversion

ProM is able to load and save C-nets in a dedicated file format. Petri nets can be converted to C-nets using the construction of Definition 10. Similarly, it is possible to convert a C-net into a Petri net using the construction of Definition 11. Conversions to and from other formats (EPCs, BPMN, etc.) are being developed. These formats can already be converted to Petri nets thus enabling an indirect conversion from these formats to C-nets.

4.2 Model-Based Verification

ProM has extensive support for transition systems and Petri nets. Moreover, also Petri nets with reset and inhibitor arcs and specific subclasses such as WF-nets are supported. Typical Petri nets properties such as liveness, boundedness, etc. can be analyzed using various plug-ins. ProM also embeds the well-known LoLA (a Low Level Petri Net Analyzer) tool for more advanced forms of model-based analysis. There are also plug-ins analyzing structural properties of the net (invariants, traps, siphons, components, etc.). These plug-ins can be applied to WF-nets. Moreover, plug-ins like Woflan are able to verify soundness and diagnose errors.

The plug-in "Check Soundness of Causal Net" checks the property defined in Definition 5. Internally, the plug-in converts the model into a WF-net and then checks relaxed soundness.

[3] Expressiveness in terms matching sequences.

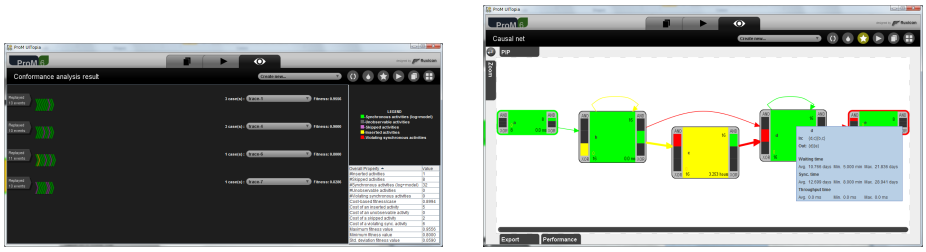
4.3 Process Discovery

One of the most challenging topics in process mining is the automated derivation of a model based on example traces [2]. The starting point for process discovery is an event log in MXML or XES format. ProM provides a wide variety of process discovery techniques, e.g., techniques based on state-based region theory, language-based region theory, genetic mining, fuzzy mining, folding of partial orders, or heuristic mining. The process discovery plug-ins in ProM typically produce a Petri net or a model close to C-nets [2,7,8,9,10]. Using the various conversion plug-ins such results can be mapped onto C-nets.

What is missing are dedicated process discovery techniques producing C-nets while exploiting the representational bias. This is a topic for further research.

4.4 Conformance Checking and Performance Analysis

Given an event log and a process model, it is possible to replay the log on the model. ProM provides several plug-ins that replay logs on Petri nets. An example, is the "Conformance Checker" plug-in [2].



(a) Conformance analysis (b) Performance analysis

Fig. 7. Two ProM plug-ins showing the results obtained through replaying the event log on a C-net

Recently, ProM started to support several plug-ins that replay logs on C-nets [4]. Figure 7(a) shows that ProM is able to discover deviations between a C-net and an event log. The plug-in indicates where deviations occur and what the overall fitness of the log is (using configurable cost functions). Most event logs contain timestamps. Therefore, replay can also be used to identify bottlenecks and to measure waiting and service times. Figure 7(b) shows the result of such analysis; the colors and numbers indicate different performance measurements.

5 Conclusion

This paper makes the case for Causal-nets (C-nets) in process mining. C-nets provide a better representational bias than conventional languages that are either too restrictive (e.g., OR-joins, unstructured loops, and skipping cannot be

expressed) or too liberal (in the sense that most models are incorrect). Key ingredients are (1) the notion of bindings allowing for any split and join behavior and (2) the semantic restriction to valid binding sequences.

We explored the basic properties of C-nets and analyzed their relation to Petri nets. Moreover, we described the degree of support provided by ProM. Model management, conversion, verification, process discovery, conformance checking, and performance analysis of C-nets are supported by ProM 6 which can be downloaded from www.processmining.org.

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. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.
3. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
4. A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011.
5. R. Bruni and U. Montanari. Zero-Safe Nets: Comparing the Collective and Individual Token Approaches. *Information and Computation*, 156(1-2):46–89, 2000.
6. 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.
7. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.
8. A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
9. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
10. A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). BETA Working Paper Series, WP 334, Eindhoven University of Technology, Eindhoven, 2010.
11. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.