

An SMT-based Discovery Algorithm for C-nets

Marc Solé^{1,2} and Josep Carmona²

¹ Computer Architecture Department, UPC msole@ac.upc.edu

² Software Department, UPC jcarmona@lsi.upc.edu

Abstract. Recently, *Causal nets* have been proposed as a suitable model for process discovery, due to their declarative semantics and the great expressiveness they possess. In this paper we propose an algorithm to discover a causal net from a set of traces. It is based on encoding the problem as a Satisfiability Modulo Theories (SMT) formula, and uses a binary search strategy to optimize the derived model. The method has been implemented in a prototype tool that interacts with an SMT solver. The experimental results obtained witness the capability of the approach to discover complex behavior in limited time.

1 Introduction

Process Mining [1] is a discipline that aims at using the data available in information systems to discover the processes taking place inside an organization, check their compliance and perform predictions. It is an evolving area which, although becoming crucial to support decision making, it still needs to settle down in terms of algorithmic and model support. One example of this is the vast amount of algorithms that exist for a wide range of models: Petri nets, Heuristic nets, Event-Driven Process Chains, Fuzzy models, among others [1].

Recently, a formalism called Causal nets (C-nets) [2] has been proposed as a suitable modeling language for process mining. It is a rather compact representation that allows expressing complex behavior that it is sometimes difficult to describe using other models. For instance consider the set of traces (*log*) in Fig. 1(a). In Fig. 1(b) we can see two Petri nets that are required to represent all the sequences without adding extra behavior. It is possible to use a single Petri net, instead of two, but then the use of *silent* transitions is needed, as shown in (c). On the other hand the equivalent C-net representation (d) is quite compact. The semantics of that C-net can be informally described as:

Activity a must be executed initially, followed either by b, c or b and c. Any of these three possibilities is followed by the execution of activity e.

Figure 2 (from [2]) shows a more meaningful example, describing a C-net that models the process of booking resources for a travel.

The *discovery problem* refers to obtaining a model (in some suitable formalism) that describes the behavior recorded in a log. To the best of our knowledge, there are few discovery algorithms for C-nets. One indirect method is to first

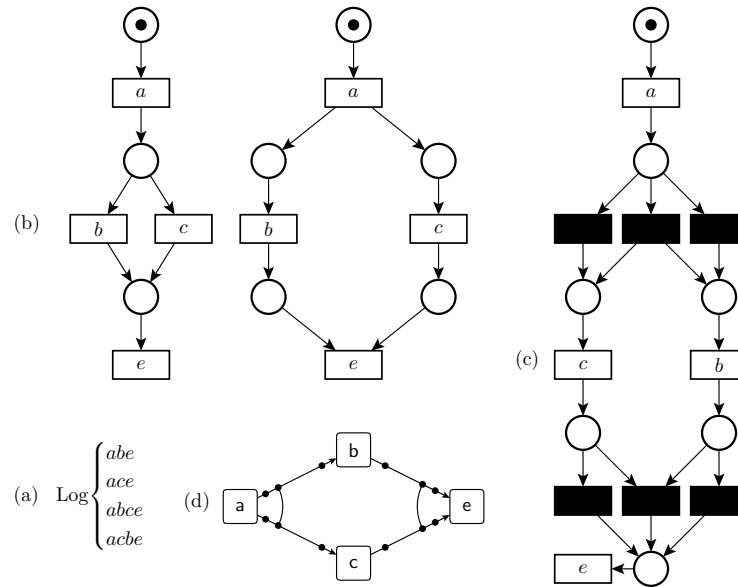


Fig. 1. (a) A log. (b) Set of two Petri nets describing the log. (c) Single Petri net describing the log with the help of silent transitions. (d) Causal net of the same log.

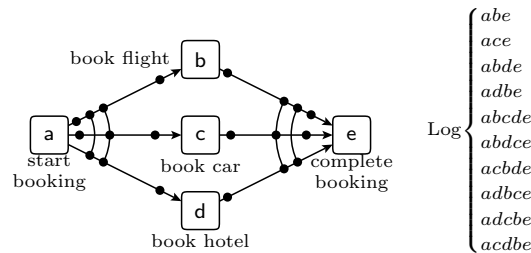


Fig. 2. Causal net C_{travel} from [2].

discover a Petri net and then convert it into a C-net as described in [2]. This strategy is very cumbersome since the conversion introduces a silent activity in the C-net for each place in the Petri net, thus increasing significantly the size of the C-net, although a very compact C-net representing the same language may exist. Another possibility is to use discovery algorithms for *flexible heuristic nets* [3], a model closely related to C-nets, or *heuristic nets* [4, 5] which can be viewed as a restricted subclass of C-nets. However, these two approaches cannot guarantee that the log is included in the language of the derived model.

This paper presents the first algorithm to discover a C-net from a log that guarantees that i) the language of the C-net includes the log, and ii) the C-net

has the minimal number of arcs. It is based on encoding the problem as an SMT formula, and using binary search to find a minimal C-net (in terms of number of arcs). A prototype tool which interfaces an SMT solver is reported, showing promising experimental results.

Organization of the paper: In Sect. 2 we give the formal definition of C-nets and we introduce some of the mathematical background used in the rest of the paper. Our approach to the discovery of C-nets is explained in Sect. 3 and experimentally tested in Sect. 4. Finally, Sect. 5 presents some future work and concludes this paper.

2 Background

2.1 Mathematical preliminaries

A multiset (or a bag) is a set in which elements of a set X can appear more than once, formally defined as a function $X \rightarrow \mathbb{N}$. We denote as $\mathbb{B}(X)$ the space of all multisets that can be created using the elements of X . Let $M_1, M_2 \in \mathbb{B}(X)$, we consider the following operations on multisets: sum $(M_1 + M_2)(x) = M_1(x) + M_2(x)$, subtraction $(M_1 - M_2)(x) = \max(0, M_1(x) - M_2(x))$ and inclusion $(M_1 \subseteq M_2) \Leftrightarrow \forall x \in X, M_1(x) \leq M_2(x)$.

A log L is a bag of sequences of activities. In this work we restrict the type of sequences that can form a log. In particular, we assume that all the sequences start with the same initial activity and end with the same final activity, and that these two special activities only appear once in every sequence. This assumption is without loss of generality, since any log can be easily converted by using two new activities that are properly inserted in each trace.

Given a finite sequence of elements $\sigma = e_1 e_2 \dots e_n$, its length is denoted $|\sigma| = n$, and its prefix sequence up to element i , with $i \leq n$, denoted σ_i , is $e_1 \dots e_i$. We define σ_0 as the empty sequence, denoted ϵ . Conversely, its suffix sequence after i , with $i < n$, denoted $\sigma_{i \rightarrow}$, is $e_{i+1} \dots e_n$. The alphabet of σ , denoted A_σ , is the set of elements in σ . We extend this notation to logs, so that A_L is the alphabet of the log L , i.e., $A_L = \bigcup_{\sigma \in L} A_\sigma$. Finally, the number of occurrences of a given element e in σ , i.e. $|\{e_i \mid e_i = e\}|$, is denoted as $\#(\sigma, e)$.

2.2 Causal nets (C-nets)

In this section we introduce the main model used along this paper.

Definition 1 (Causal net [2]). *A Causal net is a tuple $C = \langle A, a_s, a_e, I, O \rangle$, where A is a finite set of activities, $a_s \in A$ is the start activity, $a_e \in A$ is the end activity, and I (and O) are the set of possible input (output resp.) bindings per activity. Formally, both I and O are functions $A \rightarrow S_A$, where $S_A = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$, and satisfy the following conditions:*

$$- \{a_s\} = \{a \mid I(a) = \{\emptyset\}\} \text{ and } \{a_e\} = \{a \mid O(a) = \{\emptyset\}\}$$

- all the activities in the graph $(A, \text{arcs}(C))$ are on a path from a_s to a_e , where $\text{arcs}(C)$ is the dependency relation induced by I and O such that $\text{arcs}(C) = \{(a_1, a_2) \mid a_1 \in \bigcup_{X \in I(a_2)} X \wedge a_2 \in \bigcup_{Y \in O(a_1)} Y\}$.

Definition 1 slightly differs from the original one from [2], where the set $\text{arcs}(C)$ is explicitly defined in the tuple. The C-net of Fig. 1(d) is formally defined as $C = \langle \{a, b, c, e\}, a, e, I, O \rangle$, with $I(a) = \emptyset$, $O(a) = \{\{b\}, \{c\}, \{b, c\}\}$, $I(b) = \{\{a\}\}$, $O(b) = \{\{e\}\}$, $I(c) = \{\{a\}\}$, $O(c) = \{\{e\}\}$, $I(e) = \{\{b\}, \{c\}, \{b, c\}\}$ and $O(e) = \emptyset$. The dependency relation of C , which corresponds graphically to the arcs in the figure, in this case is: $\text{arcs}(C) = \{(a, b), (a, c), (b, e), (c, e)\}$. The activity bindings are denoted in the figure as dots in the arcs, e.g., $\{b\} \in O(a)$ is represented by the dot in the arc (a, b) that is next to activity a , while $\{a\} \in I(a)$ is the dot in arc (a, b) next to b . Non-singleton activity bindings are represented by arcs connecting the dots: $\{b, c\} \in O(a)$ is represented by the two dots in arcs (a, b) , (a, c) that are connected through an arc.

Definition 2 (Binding, Binding Sequence, Projection). *Given a C-net $\langle A, a_s, a_e, I, O \rangle$, the set B of activity bindings is $\{(a, S^I, S^O) \mid a \in A \wedge S^I \in I(a) \wedge S^O \in O(a)\}$. A binding sequence $\beta \in B^*$ is a sequence of activity bindings. By removing the input and output bindings from a binding sequence β , we do obtain an activity sequence denoted as σ_β .*

Two binding sequences of the C-net in Fig. 1(d) are: $\beta^1 = (a, \emptyset, \{b\})(b, \{a\}, \{e\})(e, \{b\}, \emptyset)$ and $\beta^2 = (a, \emptyset, \{b, c\})(c, \{a\}, \{e\})(e, \{c\}, \emptyset)$. The projection of β^1 is $\sigma_{\beta^1} = abe$.

The semantics of a C-net are achieved by selecting, among all the possible binding sequences, the ones satisfying certain properties. These sequences will form the set of *valid binding sequences* of the C-net, and their corresponding projection (see Def. 2) will define the language of the C-net. The next definition addresses this.

Definition 3 (State, Valid Binding Sequence, Language). *Given a C-net $C = \langle A, a_s, a_e, I, O \rangle$, its state space $S = \mathbb{B}(A \times A)$ is composed of states that represent multisets of pending obligations. Function $\psi \in B^* \rightarrow S$ is defined inductively: $\psi(\epsilon) = \emptyset$ and $\psi(\beta \cdot (a, S^I, S^O)) = \psi(\beta) - (S^I \times \{a\}) + (\{a\} \times S^O)$. The state $\psi(\beta)$ is the state of the C-net after the sequence of bindings β . The binding sequence $\beta = (a_1, S_1^I, S_1^O) \dots (a_n, S_n^I, S_n^O)$ is said to be valid if:*

1. $a_1 = a_s$, $a_n = a_e$ and $\forall k : 1 < k < n, a_k \in A \setminus \{a_s, a_e\}$
2. $\forall k : 1 \leq k \leq n, (S_k^I \times \{a_k\}) \subseteq \psi(\beta_{k-1})$
3. $\psi(\beta) = \emptyset$

The set of all valid binding sequences of C is denoted as $V_{CN}(C)$. The language of C , denoted $\mathcal{L}(C)$, is the set of activity sequences that correspond to a valid binding sequence of C , i.e., $\mathcal{L}(C) = \{\sigma_\beta \mid \beta \in V_{CN}(C)\}$.

For instance, in Fig. 1(d), β^1 is a valid binding sequence, while β^2 is not, since the final state is not empty (condition 3 is violated). The language of that C-net is $\{abe, ace, abce, acbe\}$. Similarly to *Workflow* Petri nets [6], C-nets have a notion of *soundness* [2]:

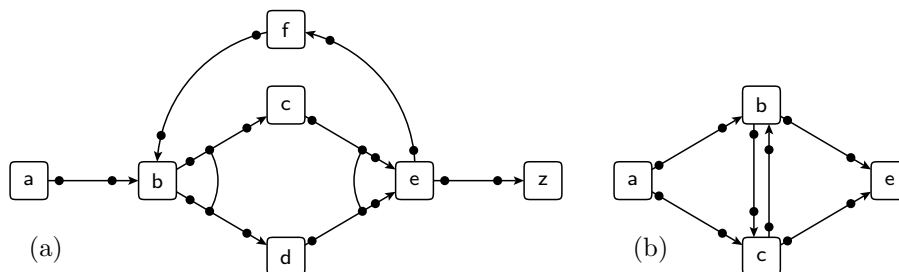


Fig. 3. (a) C-net mixing concurrent and exclusive behavior for activities c and d . (b) Immediately follows C-net of $\log L = \{abce, acbe\}$. Its language (using regular expressions) is $ab(cb)^*e \cup a(bc)^+e \cup ac(bc)^*e \cup a(cb)^+e$.

Definition 4 (Soundness). A C-net $C = \langle A, a_s, a_e, I, O \rangle$ is sound if (i) $\forall a \in A, \forall S \in I(a), \exists \beta \in V_{CN}(C) : (a, S, X) \in \beta$, and (ii) $\forall a \in A, \forall S \in O(a), \exists \beta \in V_{CN}(C) : (a, X, S) \in \beta$. That is, every input and output binding defined in C is used in at least one valid sequence.

Importantly, C-nets can naturally represent behavior that cannot be easily expressed in the Petri net notation unless unobservable (silent) transitions are used. Fig. 3(a) illustrates this point. In the C-net, activities c and d can occur concurrently or exclusively, even in different iterations of the loop created by activity f , e.g., $abcdez$ or $abdefbcez$. However, there is still another possibility that arises from combining the AND-split and the XOR-join: $abdceefbdfbcez$. Note that in this last trace, activity e could execute twice for a single b (although in the overall trace they execute the same number of times).

C-nets, contrary to Petri nets, have an “additive” nature. That is, while adding a place to a Petri net can only restrict behavior, adding an arc (or any other element) to a C-net can only add behavior. The “additive” nature of C-nets is formally defined with the help of Def. 5 and Property 1.

Definition 5. Given two C-nets $C_1 = \langle A_1, a_s^1, a_e^1, I_1, O_1 \rangle$ and $C_2 = \langle A_2, a_s^2, a_e^2, I_2, O_2 \rangle$, we say that C_1 is included in C_2 , denoted $C_1 \subseteq C_2$, if:

- $a_s^1 = a_s^2 \wedge a_e^1 = a_e^2$,
- $A_1 \subseteq A_2$, and
- $\forall a \in A_1, I_1(a) \subseteq I_2(a) \wedge O_1(a) \subseteq O_2(a)$

Property 1. Let C_1 and C_2 be two C-nets. If $C_1 \subseteq C_2$, then $V_{CN}(C_1) \subseteq V_{CN}(C_2)$, $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$ and $\text{arcs}(C_1) \subseteq \text{arcs}(C_2)$.

2.3 C-net discovery

Given a log L , the problem tackled in this paper is to derive a C-net C such that $\mathcal{L}(C) \supseteq L$ and C contains the minimal number of arcs. In Sect. 3 we present such a method together with heuristics to limit the language of the derived net.

Given the additive nature of C-nets, there is a simple method to generate a C-net that can replay all the traces in L . It is based on the *immediately follows* relation [6] between the activities in L , denoted $<_L$ and defined as $<_L = \{(a, b) \mid \exists \sigma = a_1 \dots a_n \in L : a_i = a \wedge a_{i+1} = b\}$.

Definition 6. *Given a log L , the immediately follows C-net of L , denoted $C_{IF}(L)$, is the C-net $\langle A, a_s, a_e, I, O \rangle$ such that: (i) $A = A_L$, (ii) $\forall \sigma = a_1 \dots a_n \in L, a_1 = a_s \wedge a_n = a_e$, (iii) $\forall a \in A, O(a) = \{\{b\} \mid a <_L b\} \wedge I(a) = \{\{b\} \mid b <_L a\}$.*

The immediately follows C-net can be computed in linear time with respect of the size of the log, but allows for many unobserved behavior, thus exhibiting a poor *precision* [7]. For instance consider the following log: $L = \{abce, acbe\}$. Activities b and c interleave, but if we build the immediately follows C-net (Fig. 3(b)) we can see that it allows for loops of arbitrary length involving these two activities, e.g., $abcbce$ or $acbce$, since $\mathcal{L}(C_{IF}(L)) = ab(cb)^*e \cup a(bc)^+e \cup ac(bc)^*e \cup a(cb)^+e$, which significantly differs from L .

3 Discovering strategies for C-nets based on SMT

3.1 Protobinding sequences of a log

In Sect. 2.2 we have seen first the definition of a C-net and then the definition of the valid sequences of bindings it can produce. To discover a C-net from a log, we follow the same path but in the opposite direction: we will define sequences of triples representing unrestricted bindings that satisfy some properties, and then we will show that given these sequences, it is easy to obtain a C-net C such that these sequences are actually valid sequences of bindings of C . Consequently, this transforms the discovery problem for C-nets into the problem of deriving these sequences of triples from the sequences in the log. Let us first formalize the concept of *protobinding*:

Definition 7 (Protobinding, Well-Formed Protobinding Sequence). *A triple (a, X, Y) is a protobinding if a is an element and both X and Y are sets. A sequence $\beta = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n)$ of protobindings is well-formed if it satisfies the following conditions:*

- (W1) $\forall i : 1 < i \leq n, X_i \supseteq \emptyset \wedge a_i \neq a_1$
- (W2) $\forall i : 1 \leq i < n, Y_i \supseteq \emptyset \wedge a_i \neq a_n$
- (W3) $X_1 = Y_n = \emptyset$
- (W4) $\forall i : 1 \leq i \leq n, \psi(\beta_{i-1}) \supseteq (X_i \times \{a_i\})$
- (W5) $\psi(\beta_n) = \emptyset$

Compared with the definition of binding (Def. 2), this is a weaker definition since it is no longer tied to a particular C-net. Given a set B of sequences of well-formed protobindings it is possible to characterize (with necessary conditions) all the C-nets such that their set of valid sequences of bindings contain the sequences of protobindings in B , as next lemma states.

Lemma 1. *Given a set of well-formed protobinding sequences B with identical initial and final activities a_s and a_e , respectively, and a C-net $C = \langle A, a_s, a_e, I, O \rangle$. The following conditions:*

- (N1) $A \supseteq \{a_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$
- (N2) $\forall a \in A, I(a) \supseteq \{X_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$
- (N3) $\forall a \in A, O(a) \supseteq \{Y_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$

hold if, and only if, $V_{CN}(C) \supseteq B$.

Proof. \Rightarrow Take any protobinding sequence $\beta \in B$, we will prove that since it is well-formed it is actually a valid binding sequence of C , thus $V_{CN}(C) \supseteq B$. By Def. 2, we know that to be a binding sequence (not necessarily valid) of C , every protobinding $(a_i, X_i, Y_i) \in \beta$ must satisfy that $a_i \in A$, $X_i \in I(a_i)$ and $Y_i \in O(a_i)$, which is trivially true given our definition of C . Thus, β is a binding sequence of C . We now prove that β is in fact also valid. To prove this we use the fact that β is well-formed (thus satisfies W1 to W5). Proving that $\beta = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n)$ is a valid binding sequence, we need to prove that $a_1 = a_s$, $a_n = a_e$ and $\forall k : 1 < k < n, a_k \in A \setminus \{a_s, a_e\}$. The first two conditions are satisfied because we require that all the sequences in B start with a_s and end with a_e . The third condition is guaranteed by W1 and W2. The remaining two conditions for a valid sequence are directly W4 and W5.

\Leftarrow If C is a C-net and $\mathcal{L}(C) \supseteq \{\sigma_\beta \mid \beta \in B\}$, then if N1 does not hold it exists some activity that appears in the sequences of B that cannot be executed by C . If N2 does not hold, then there is some sequence that is not possible because some X_i cannot be used by C , and the same applies for N3 and Y_i . \square

Creating a tuple $\langle A, a_s, a_e, I, O \rangle$ satisfying N1, N2 and N3 does not necessarily yield a C-net (for instance we could add activities to A that violate some of the conditions of Def. 1), since these are necessary but not sufficient conditions. To guarantee that a C-net is generated and it is *sound* (c.f., Def. 4), we restrict the conditions of the previous lemma in the following theorem:

Theorem 1. *Given a set of well-formed protobinding sequences B with identical initial and final activities a_s and a_e , respectively, the tuple $C = \langle A, a_s, a_e, I, O \rangle$ with:*

- (T1) $A = \{a_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$
- (T2) $\forall a \in A, I(a) = \{X_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$
- (T3) $\forall a \in A, O(a) = \{Y_i \mid \exists \beta \in B : (a_i, X_i, Y_i) \in \beta\}$

is a sound C-net such that $V_{CN}(C) \supseteq B$.

Proof. We will prove that C is a C-net, since then we can use Lemma 1 to prove that $V_{CN}(C) \supseteq B$. Proving that C is sound, once we know it is a C-net, is straightforward since every input and output binding in I and O appears in at least one of the sequences in B , thus in at least one valid binding sequence of C , thus C is sound. So we have only to prove that C is a C-net satisfying

Def. 1. First of all we have to prove that a_s is the only activity with empty input binding ($I(a_s) = \emptyset$) and a_e is the only activity with empty output binding ($O(a_e) = \emptyset$). Consider all sequences $\beta = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n) \in B$, by W3, we know that $\emptyset \in I(a_s)$ and $\emptyset \in O(a_e)$ (because $a_1 = a_s$ and $a_n = a_e$), and since these two activities are only executed once (due to W1 and W2) there is no other set in $I(a_s)$ and $O(a_e)$. Since the other activities $a \in A \setminus \{a_s, a_e\}$ are not executed at the beginning nor the end of β , also by W1 and W2 we know their X_i and Y_i sets are non-empty, thus they cannot have $I(a) = \{\emptyset\}$ nor $O(a) = \{\emptyset\}$. We must also prove that the I and O functions are defined over the powerset of A , *i.e.*, $\forall a \in A, \forall X \in I(a), X \subseteq A$. Because of W4 and W5 the sequence can only consume obligations previously produced, and all the obligations must be consumed. If some X_i in β contains activities not in A , it is impossible to satisfy W4 thus it would not be well-formed, which is a contradiction. Similarly, if some Y_i in β contains activities not in A , then the obligations created can never be consumed, violating W5. Finally, we have to prove that all the activities in the graph $(A, \text{arcs}(C))$ are on a path from a_s to a_e . Again by W4 and W5, since an activity a_i (different than a_s) can only execute when it has at least one obligation (a, a_i) for it in $\psi(\beta_{i-1})$ and $a \in X_i$, thus $(a, a_i) \in \text{arcs}(C)$, and the source of all this chain of obligations is a_s and ends in a_e or otherwise a_i (or some of its successors in the obligation chain) would have produced an obligation that nobody would have consumed, violating W5, then every activity is in a dependency chain between a_s and a_e . \square

The theorem allows an easy conversion from protobinding sequences to C-nets, so that the C-net discovery problem from a log L can be reduced to the following problem: given a log L , compute a well-formed protobinding sequence for each sequence in L . Since by definition all our sequences in the log have the same initial and final activities, then all the protobinding sequences will also have, thus we can use Theorem 1 to discover a C-net. Although the theorem does not consider all the C-nets whose valid binding sequences include the protobinding sequences B , it gives always the smallest C-net (in terms of valid binding sequences and also in terms of number of structural elements of the C-net) that can generate the sequences in B as next corollary states.

Corollary 1. *Given a set of well-formed protobinding sequences B with identical initial and final activities a_s and a_e , respectively, a C-net C such that $V_{CN}(C) \supseteq B$ and a C-net C_\perp satisfying T1, T2 and T3, then $C_\perp \subseteq C$.*

Proof. If C is a C-net and $V_{CN}(C) \supseteq B$, then by Lemma 1 we know it satisfies N1, N2 and N3. Since C_\perp satisfies T1, T2 and T3, by Def. 5 $C_\perp \subseteq C$, and this entails by Property 1 $V_{CN}(C_\perp) \subseteq V_{CN}(C)$. \square

In the next section we explain how we can encode as linear constraints the problem of computing the sequences of protobindings.

3.2 Encoding the problem as linear constraints

Given a sequence $\sigma = a_1 \dots a_n$ of a log L , it is trivial to build a protobinding sequence β_σ out of it as $\beta_\sigma = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n)$. The difficult part is

to ensure that β_σ is actually well-formed. We will encode the unknown X_i and Y_i sets using integer variables and then define the linear constraints that will guarantee that β_σ is well-formed. We start by delimiting the values that the X_i and Y_i unknowns can take using the following property:

Property 2. Let $\sigma = a_1 \dots a_n$ be a sequence of activities. Consider the protobinding sequence $\beta_\sigma = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n)$. If β_σ is well-formed, then $\forall i : 1 \leq i \leq n, X_i \subseteq A_{\sigma_{i-1}} \wedge Y_i \subseteq A_{\sigma_{i \rightarrow}}$.

Proof. If for some i , $X_i \not\subseteq A_{\sigma_{i-1}}$, then let $a \in X_i \setminus A_{\sigma_{i-1}}$. Now activity a_i is waiting for an obligation (a, a_i) that cannot have been produced (since $a \notin A_{\sigma_{i-1}}$). Thus $\psi(\beta_{i-1}) \not\subseteq (X_i \times \{a_i\})$ and β is not well-formed (violates W4), which is a contradiction. Similarly, if $Y_i \not\subseteq A_{\sigma_{i \rightarrow}}$, then it exists an activity a such that $a \in Y_i \setminus A_{\sigma_{i \rightarrow}}$. Now obligation (a_i, a) cannot be consumed (since $a \notin A_{\sigma_{i \rightarrow}}$) so $\psi(\beta_n) \neq \emptyset$ (violating W5), and again β is not well-formed. \square

To encode arithmetically the sets X_i and Y_i for each β_σ , we use an integer variable over the domain $\{0, 1\}$ (*i.e.*, a Boolean variable, although we treat it as an integer in this section) to encode the fact that a particular activity belongs to the set. In particular we use a variable $x_{\sigma, i, (a, a_i)}$ to indicate whether activity a belongs to X_i in β_σ or not. As usual when sets are encoded using characteristic functions we use the following semantics:

$$x_{\sigma, i, (a, a_i)} = \begin{cases} 1 & \text{if } a \in X_i \text{ in } \beta_\sigma \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, the variable $y_{\sigma, i, (a_i, a)}$ indicates if a belongs to Y_i in β_σ . Due to Property 2, the activity a for x variables can only be chosen among the alphabet of prefix σ_{i-1} , while in y variables it is restricted to the alphabet of the suffix of σ after a_i , *i.e.*, $A_{\sigma_{i \rightarrow}}$. We denote by X and Y the set of all x and all y variables, respectively.

We will now rewrite the conditions (W1, W2, W3, W4 and W5) of Def. 7 for a well-formed protobinding sequence $\beta_\sigma = (a_1, X_1, Y_1) \dots (a_n, X_n, Y_n)$ as inequalities using the X and Y variables.

Condition W1 In this case, part of the condition is already guaranteed, since our definition of log already assumes that the initial activity only appears once. Thus the condition simplifies to requiring that every X_i (except X_1) must be non-empty:

$$\forall i : 1 < i \leq n, \sum_{e \in A_{\sigma_{i-1}}} x_{\sigma, i, (e, a_i)} \geq 1 \quad (1)$$

Condition W2 This is the symmetrical case to W1 but with the Y_i sets. Since the uniqueness of the final activity is already guaranteed, we must only enforce that the Y_i sets (except Y_n) are non-empty:

$$\forall i : 1 \leq i < n, \sum_{e \in A_{\sigma_{i \rightarrow}}} y_{\sigma, i, (a_i, e)} \geq 1 \quad (2)$$

Condition W3 This needs no conversion, since we can directly assign the empty set to X_1 and Y_n . Note that the model does not even generate any variable in X or Y to represent these sets, since $A_{\sigma_0} = A_{\sigma_{n \rightarrow}} = \emptyset$.

Condition W4 This condition requires that the state of obligations after executing prefix β_{i-1} (i.e., $\psi(\beta_{i-1})$) contains, at least, the obligations in $(X_i \times \{a_i\})$. This is the same as requiring that the number of obligations of the type (e, a_i) in $\psi(\beta_{i-1})$ is larger or equal than the number of obligations (e, a_i) in $(X_i \times \{a_i\})$. Moreover, if a_i is the last occurrence of that activity, condition W5 applies instead, since there cannot be pending obligations in the final state, so the last occurrence of an activity must consume all the obligations for it. The number of such obligations in $\psi(\beta_{i-1})$ can be computed by summing the number of times the obligation has been produced minus the number of times it has been already consumed before the execution of a_i .

$$\forall i : (1 \leq i \leq n \wedge \exists j : (j > i \wedge a_j = a_i)), \forall e \in A_{\sigma_{i-1}},$$

$$\sum_{k:k < i \wedge a_k = e} y_{\sigma,k,(e,a_i)} - \sum_{m:m \leq i \wedge a_m = a_i} x_{\sigma,m,(e,a_i)} \geq 0 \quad (3)$$

Condition W5 To force that the final number of obligations must be zero we require that the number of (e, a_i) obligations is exactly zero after the last execution of a_i in the sequence. Since it is simply a stronger version of (3), it replaces (3) in the last execution of a_i .

$$\forall i : (1 \leq i \leq n \wedge \forall j (j > i \Rightarrow a_j \neq a_i)), \forall e \in A_{\sigma_{i-1}},$$

$$\sum_{k:k < i \wedge a_k = e} y_{\sigma,k,(e,a_i)} - \sum_{m:m \leq i \wedge a_m = a_i} x_{\sigma,m,(e,a_i)} = 0 \quad (4)$$

Definition 8 (Structural equations). *The set of structural equations for a C-net including the behavior of a log L , denoted $\text{structural_equations}(L)$, is the set of equations obtained by adding the equations from (1), (2), (3) and (4) for every $\sigma \in L$.*

Example 1. Let us see an example. Consider the sequence $\sigma_\beta = abcbe$, so that $\beta = (a_1, X_1, Y_1)(a_2, X_2, Y_2)(a_3, X_3, Y_3)(a_4, X_4, Y_4)(a_5, X_5, Y_5)$ with $a_1 = a$, $a_2 = b$, $a_3 = c$, $a_4 = b$, $a_5 = e$, and $X_1 = Y_5 = \emptyset$. Table 1 shows the structural equations for each prefix in the sequence.

Note that in this table some of the equations for $i = 1$ are empty since $A_{\sigma_0} = \emptyset$, a similar case to that of $i = 5$ and (2), because $A_{\sigma_{5 \rightarrow}} = \emptyset$. Moreover, (4) is used instead of (3) for $i \in \{3, 4, 5\}$ because these are the last executions of activities c , b and e , respectively.

In summary, by finding the satisfying assignments to the X and Y variables in the equations arising from a log, one can derive a C-net that includes the language of the log. In terms of complexity, the number of variables that each

$i = 1$	$A_{\sigma_0} = \emptyset, A_{\sigma_1 \rightarrow} = \{b, c, e\}, \sigma_1 = a$
(1)	–
(2)	$y_{\sigma,1,(a,b)} + y_{\sigma,1,(a,c)} + y_{\sigma,1,(a,e)} \geq 1$
(3)	–
$i = 2$	$A_{\sigma_1} = \{a\}, A_{\sigma_2 \rightarrow} = \{b, c, e\}, \sigma_2 = ab$
(1)	$x_{\sigma,2,(a,b)} \geq 1$
(2)	$y_{\sigma,2,(b,b)} + y_{\sigma,2,(b,c)} + y_{\sigma,2,(b,e)} \geq 1$
(3)	$y_{\sigma,1,(a,b)} - x_{\sigma,2,(a,b)} \geq 0$
$i = 3$	$A_{\sigma_2} = \{a, b\}, A_{\sigma_3 \rightarrow} = \{b, e\}, \sigma_3 = abc$
(1)	$x_{\sigma,3,(a,c)} + x_{\sigma,3,(b,c)} \geq 1$
(2)	$y_{\sigma,3,(c,b)} + y_{\sigma,3,(c,e)} \geq 1$
(4)	$y_{\sigma,1,(a,c)} - x_{\sigma,3,(a,c)} = 0$ and $y_{\sigma,2,(b,c)} - x_{\sigma,3,(b,c)} = 0$
$i = 4$	$A_{\sigma_3} = \{a, b, c\}, A_{\sigma_4 \rightarrow} = \{e\}, \sigma_4 = abcb$
(1)	$x_{\sigma,4,(a,b)} + x_{\sigma,4,(b,b)} + x_{\sigma,4,(c,b)} \geq 1$
(2)	$y_{\sigma,4,(b,e)} \geq 1$
(4)	$y_{\sigma,1,(a,b)} - x_{\sigma,2,(a,b)} - x_{\sigma,4,(a,b)} = 0, y_{\sigma,2,(b,b)} - x_{\sigma,4,(b,b)} = 0$ and $y_{\sigma,3,(c,b)} - x_{\sigma,4,(c,b)} = 0$
$i = 5$	$A_{\sigma_4} = \{a, b, c\}, A_{\sigma_5 \rightarrow} = \emptyset, \sigma_5 = abcbe$
(1)	$x_{\sigma,5,(a,e)} + x_{\sigma,5,(b,e)} + x_{\sigma,5,(c,e)} \geq 1$
(2)	–
(4)	$y_{\sigma,1,(a,e)} - x_{\sigma,5,(a,e)} = 0, y_{\sigma,2,(b,e)} - y_{\sigma,4,(b,e)} - x_{\sigma,5,(b,e)} = 0$ and $y_{\sigma,3,(c,e)} - x_{\sigma,5,(c,e)} = 0$

Table 1. Structural equations for sequence *abcbe*.

activity occurrence generates is $|A|$, thus for a sequence σ , the total number of variables generated is $|A| \cdot |\sigma|$. Hence, the total number of variables for a log L is $|A| \cdot \sum_{\sigma \in L} |\sigma|$, which is $\mathcal{O}(|L| \cdot |A| \cdot \max_{\sigma \in L} (|\sigma|))$. The number of equations for an activity $a_i \in \sigma$ is two ((1) and (2)) plus $|A_{\sigma_{i-1}}|$ for (3) and (4). Thus, for a sequence σ , the maximum number of equations is $\mathcal{O}(|\sigma| \cdot |A|)$ so for the whole log L is $\mathcal{O}(|L| \cdot |A| \cdot \max_{\sigma \in L} (|\sigma|))$, the same as the number of variables. Next sections illustrate how to algorithmically solve the discovery problem described in this section.

3.3 Solving linear constraints using SMT

The equations presented in previous section can be represented in different domains. In the algebraic domain, one option is to model equations (1)–(4) in a Integer Linear Programming (ILP) model (but with binary variables), and use one of the available solvers. However, such an option has an important drawback: the cost function used to minimize the solution to the problem must be linear. A possibility is to minimize the sum of all the X and Y variables. However, this will promote solutions like the immediately follows C-net, since in that C-net every activity (except the initial and final ones) always consumes one obligation and produces one obligation, thus it is not possible to have a C-net producing less obligations. Ideally we would like to express that we seek for a C-net as

simple as possible and, as we will see in Sect. 3.4, we can restrict its number of arcs. However, this requires an expression involving logical disjunctions. Although these type of constraints can be encoded as linear combinations³, they require the introduction of auxiliary variables and additional constraints.

An alternative will be to solve the problem in the Boolean domain. SMT solvers for the theory of quantifier-free bit-vector arithmetic [8], as we will see in this section, can also model equations (1)–(4) and have the advantage that they can also encode more easily the bound on the number of arcs in the C-net, as well as some other constraints (see Sect. 3.6). Since SMT solvers provide a higher degree of flexibility and our tests showed that in terms of running time ILP solvers and SMT solvers had a similar performance in our benchmarks, we have decided to use SMT to encode the problem.

Variables in X and Y are all Boolean, so to obtain a Boolean formula that represents the model is possible. Now let us show how (1), (2), (3) and (4) can be encoded as Boolean formulas. Equations (1) and (2) are trivial, since they correspond to a disjunction. For instance, the inequality (1): $\sum_{e \in A_{\sigma_{i-1}}} x_{\sigma,i,(e,a_i)} \geq 1$ can be rewritten as $\bigvee_{e \in A_{\sigma_{i-1}}} x_{\sigma,i,(e,a_i)} = 1$. Equations (3) and (4), are very similar, so we can simply focus on one of them. The key idea is to compute Boolean expressions that represent the individual bits of the sum or the subtraction of these Boolean variables. For (3) we then have to check the sign of the result (must be positive, so the most significant bit in two’s complement must be zero) while for (4) the result must be zero (so all the bits of the result have to be zero).

The translation process is quite involved (this is why automated tools are used for this task), but it can be quite straightforward for the most simple cases. For instance, going back to Example 1, consider the equation $y_{\sigma,1,(a,b)} - x_{\sigma,2,(a,b)} \geq 0$ ((3) for $i = 2$). Translated to a Boolean expression this is simply $y_{\sigma,1,(a,b)} \vee \overline{x_{\sigma,2,(a,b)}}$.

Once all formulas have been translated to the Boolean domain, they can be converted into CNF formulas and fed into a SAT solver. In this work (Sect. 4) we have used the STP solver [9] to convert our linear equations to CNF formulas.

3.4 Adding a cost function

It is possible to encode as an SMT formula an expression that bounds the number of arcs in the derived C-net. To accomplish this we can use any of the sets of Boolean variables. Without loss of generality, we use set X . For readability we introduce an auxiliary notation to denote all the variables in X that correspond to a given binding (a, b) in the sequences of a log L . Namely, $X_{(a,b)}(L) = \{x_{\sigma,i,(a,b)} \mid \exists \sigma = a_1 \dots a_{|\sigma|} \in L : a_i = b \wedge a \in A_{\sigma_{i-1}}\}$. We can now compute the number of arcs in the C-net obtained through T1, T2 and T3 (Theorem 1) using the following expression:

$$\text{number_of_arcs}(L) \stackrel{\text{def}}{=} \sum_{a \in A_L} \sum_{b \in A_L} \bigvee_{x \in X_{(a,b)}(L)} x$$

³ For instance $z = x \vee y$ is equivalent to $z \geq x$, $z \geq y$ and $z \leq x + y$.

Then, the equation bounding the number of arcs is:

$$\text{bound_arcs}(L, l) \stackrel{\text{def}}{=} \text{number_of_arcs}(L) \leq l \quad (5)$$

In Sect. 3.5 we will use this equation to find the C-net whose language includes the log L and has the minimum number of arcs. Since we will explore the solution space using a binary search strategy, we need to derive lower and upper bounds on the number of arcs that the C-net can have.

An upper bound can be obtained by computing the immediately follows C-net and counting its arcs. A possible lower bound can be the maximum between $|A_L| - 1$, which is the minimum number of arcs to guarantee that all the activities in the log are connected, and the bound obtained in the following lemma:

Lemma 2. *Let A_s be a set containing all the activities that appear in second position in some sequence of log L . Similarly, let A_e be a set containing the activities that appear in previous to the last position in some sequence. Any C-net C such that $L \subseteq \mathcal{L}(C)$ satisfies:*

$$\text{arcs}(C) \geq |A_s| + |A_e| + |A_L \setminus (A_s \cup A_e)| - 2 + \max(|A_s \setminus A_e|, |A_e \setminus A_s|)$$

Proof. C has an arc from initial activity a_s to all the activities in A_s . Similarly has an arc from every activity in A_e to the final activity a_e , otherwise there is a sequence in L that does not belong to $\mathcal{L}(C)$. This means we have already $|A_s| + |A_e|$ arcs in C . Now for activities in $A_s \cap A_e$ no further arc is mandatory, however for activities not in the intersection there must be a path from a_s to a_e (by C-net definition). Since one activity in $A_s \setminus A_e$ can be connected to another activity in $A_e \setminus A_s$, the difference set with maximum number of elements determines the number of additional arcs that have to be added (thus we must add $\max(|A_s \setminus A_e|, |A_e \setminus A_s|)$ to the number of arcs). Finally the activities not in A_s , A_e nor in $\{a_s, a_e\}$ can be arbitrarily placed, however the structure that yields the least number of additional arcs is to put them in a sequence in an already existing path from a_s to a_e , in which case one arc is added for each activity in this set. Hence $|A_L \setminus (A_s \cup A_e \cup \{a_s, a_e\})| = |A_L \setminus (A_s \cup A_e)| - 2$ arcs are added. \square

Depending on the characteristics of the log (mainly the sizes of the A_s and A_e sets), this bound might be more restrictive than using simply connectedness arguments (*i.e.*, the bound $|A_L| - 1$) thus using the largest of both values potentially decreases the number of SMT problems that have to be solved.

3.5 The algorithm

In Algorithm 1 we give the pseudocode of the proposed approach. The main idea is to build the structural equations mandatory to any C-net whose language includes a given log L , and then bound the number of arcs allowed in the solution. Following the outcome of the SMT solver, the bound is changed, so that we minimize the number of arcs using a binary search strategy. To obtain reasonable

Algorithm 1 Discover minimal C-net

```

1: procedure DISCOVERMINCNET( $L$ )
2:    $C = \langle A, a_s, a_e, I, O \rangle \leftarrow C_{\text{IF}}(L)$  ▷ See Sect. 2.3
3:    $A_s \leftarrow \{a \mid (a_s, a) \in \text{arcs}(C)\}$ 
4:    $A_e \leftarrow \{a \mid (a, a_e) \in \text{arcs}(C)\}$ 
5:    $\text{min} \leftarrow \max(|A|, |A_s| + |A_e| + |A \setminus (A_s \cup A_e)| - 2 + \max(|A_s \setminus A_e|, |A_e \setminus A_s|))$ 
6:    $\text{max} \leftarrow |\text{arcs}(C)| - 1$ 
7:    $E_s \leftarrow \text{structural\_equations}(L)$ 
8:   while  $\text{min} \leq \text{max}$  do
9:      $\text{avg} \leftarrow \lfloor (\text{min} + \text{max}) / 2 \rfloor$ 
10:     $E \leftarrow E_s \cup \{\text{bound\_arcs}(L, \text{avg})\}$  ▷ Add (5)
11:     $\text{feasible}, \text{solutions} \leftarrow \text{solve}(E)$  ▷ Call SMT solver
12:    if  $\text{feasible}$  then
13:       $C \leftarrow \text{extract\_cnet}(\text{solutions})$  ▷ Model feasible
14:       $\text{max} \leftarrow |\text{arcs}(C)| - 1$  ▷ Since  $|\text{arcs}(C)| \leq \text{avg}$ 
15:    else
16:       $\text{min} \leftarrow \text{avg} + 1$  ▷ Model unfeasible
17:    end if
18:  end while
19:  return  $C$ 
20: end procedure

```

initial bounds, we use Lemma 2 for a lower bound (line 5) and the number of arcs in the immediately follows C-net for the upper bound (line 6).

Note that, although the minimum number of arcs to guarantee that all activities are connected is $|A| - 1$, the minimum bound in the algorithm is set to $|A|$. This is because there is a single model that has $|A| - 1$ arcs, which corresponds to a sequence of activities. If this model is feasible, then it should have been already found in C_{IF} , thus $|\text{arcs}(C)| = |A| - 1$ and the algorithm would never enter the loop and return C_{IF} . On the other hand, if $|\text{arcs}(C)| > |A| - 1$, then there is no feasible model with just $|A| - 1$ arcs, thus the minimum search bound can be set to $|A|$.

The algorithm contains two calls to functions not yet introduced. One is function `solve(E)` which simply calls the SMT solver on the set of equations E and returns two values: *feasible* that is a Boolean value indicating whether the solver found a solution to the equations in E and *solutions* that contains the values of the X and Y variables in case the problem was feasible. The other function, `extract_cnet(solutions)`, simply builds a C-net out of the values of the variables in sets X and Y using the principles explained in Theorem 1.

Theorem 2. *Let C be the C-net returned by Algorithm 1 executed on a log L . The language of C includes L and there is no other C-net including L that has less arcs than C .*

Proof. Since the equations E_s represent all possible well-formed protobinding sequences of L , any valid solution is a set B of well-formed protobinding sequences of L . Using Theorem 1 on B (the `extract_cnet` function) we obtain

the C-net C whose set of valid binding sequences includes B (thus its language includes L) and has the smallest number of arcs (Corollary 1). Since we simply add a restriction (`bound_arcs(L, avg)`) on the maximum number of arcs that the sequences in B induce on C , by performing a binary search we guarantee that no other C-net whose language includes the L can have fewer arcs than C . \square

3.6 Encoding other types of constraints

The approach of Sect. 3.5 does not give any guarantee on the amount of additional behavior that the generated C-net might exhibit. For instance, consider the log $\{abcdez, abdcez\}$. The C-net whose language contains only this log is shown in Fig. 4(a). However there are other C-nets with six arcs that also contain the log (as well as some other sequences), like the one in (b). Ideally we would like to obtain the simplest C-net that adds the least amount of additional behavior. While restricting the language accepted by a Petri net is straightforward, the same operation in C-nets is much more difficult given their additive nature and the fact that their language is not prefix closed.

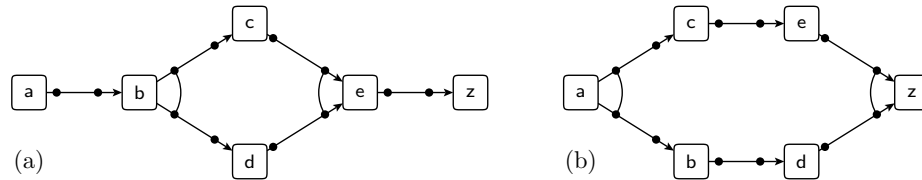


Fig. 4. Two possible C-nets with the minimum amount of arcs for log $\{abcdez, abdcez\}$.

The basic problem is that in C-nets we could only exclude complete sequences (notice that there are infinite potential sequences starting with the initial activity and ending with the final activity), rather than prefixes (known as *wrong continuations* [10] or *faulty words* [11]) as in Petri nets. Since it is not possible to exclude an infinite number of complete sequences, we have to resort to some heuristics to favor the selection of C-nets with a more restricted language. We will use three approaches: the first one penalizes the activities whose input binding set does not contain activities that are near enough in a sequence, the second limits the amount of different input and/or output bindings per activity, and the third restricts the set of activities for which an activity can generate or consume obligations. This latter approach is fundamental to tackle some of our largest benchmarks, since it can greatly reduce the amount of variables in the model to solve. Note that using the following heuristics the guarantee that no other C-net whose language includes the log can have fewer arcs is lost.

Restricting the first occurrence of activities To check whether two activities are nearby in a sequence, we take into account the other sequences in

the log and the position that the activity occupies in all other sequences that share a prefix with the current sequence. For instance in the log used in Fig. 4, $\{abcdez, abdcez\}$, activity d is executed in the fourth position in the first sequence, but there is another sequence sharing a prefix ab in which it appears in the third position. Consequently, we would penalize activity d if its input binding does not contain at least one of the activities after the first position. That is, if its input binding is $\{a\}$ we would add one to the penalty function. Summing all the penalties for the first occurrence of every activity in each sequence, we obtain an expression that can be bounded, similarly to (5), and added into the SMT problem. In our example, the C-net of 4(b) would have a penalty of one (because of activity c), while the C-net in (a) has a zero penalty.

Limiting the input/output bindings per activity The second approach involves limiting the number of input and/or output bindings per activity. Auxiliary variables are used for this task. We illustrate this point using variables X , since the strategy for the Y variables is identical. Assume that two input bindings are allowed for each activity. For each variable $x_{\sigma,i,(a,b)}$ involving an obligation (a,b) we generate one variable $i_{k,(a,b)}$ for each one of the k input bindings we allow. Since in this case $k = 2$, this means that we would have $i_{1,(a,b)}$ and $i_{2,(a,b)}$. Now for every input binding set in position i of sequence σ , we will enforce that $\bigvee_{1 \leq j \leq k} \left(\bigwedge_{a \in A_{\sigma_i}} x_{\sigma,i,(a,a_i)} = i_{j,(a,a_i)} \wedge \bigwedge_{a \notin A_{\sigma_i}} x_{\sigma,i,(a,a_i)} = 0 \right)$.

Limiting the obligation alphabet We have seen in Property 1 that the set of input bindings for an activity a_i in sequence σ is a subset of A_{σ_i} . However this allows for very long causal dependencies, that are rather unfrequent in most of the logs. We can simplify the C-net discovery problem by bounding this set using a window: we will only allow activity a_i to consume obligations generated by activities that are at most at distance w of any occurrence of a_i in any of the sequences of the log. Using a window size of one ($w = 1$) the number of variables can be dramatically reduced (the same principles are used to restrict the output binding sets), allowing the discovery of larger benchmarks. However, when using this approach complex causalities between activities can be missed.

4 Experiments

First of all, Table 2 describes some of the examples used in our experiments. We have used logs obtained by simulation of C-nets coming from [2], or that we have created to represent a variety of non-trivial behavior. Other benchmarks are logs introduced in [12], for which we have manually created a C-net (to set a target C-net to achieve) from a Petri net generated by a discovery tool using the theory of regions [11]. The table also includes the following information: $|L|$ is the number of distinct sequences in the log, $|\sigma_m|$ is the length of the largest sequence and $|A|$ is the size of the alphabet of activities.

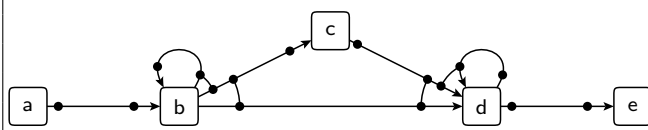
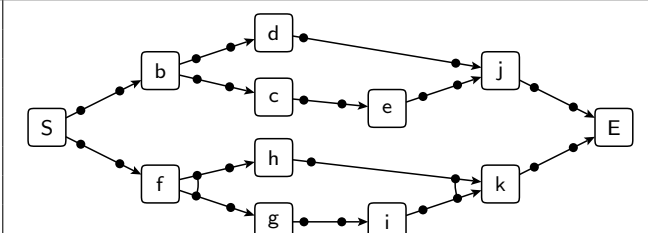
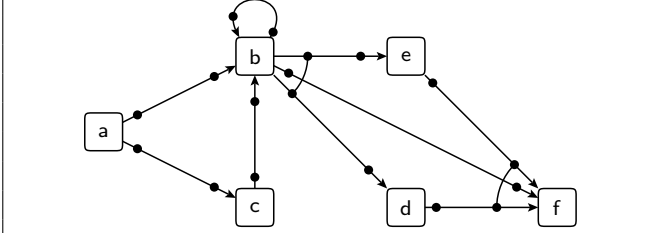
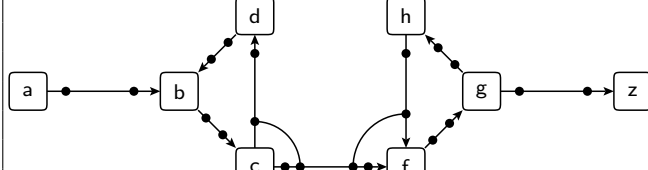
abcde abcbedde abbbccddde abbcdede abcbedde abcbbdcedde abbccdde abbcbddcedde	aalst2b [2] $ L = 10, \sigma_m = 5, A = 5$ 
abcdez abdefbcez abdceefdbfcez	mixedXorAnd $ L = 8, \sigma_m = 11, A = 5$ See Fig. 3(a)
SbcejE SbdjE SfghikE SfghikE SfhgikE	a12f0n00_5 [12] $ L = 5, \sigma_m = 7, A = 12$ 
abf acbdef abbedf abbbbedf acbedf acbbbedf acbbf abbbbdef abbdef acbbbdef acbf	optional1 $ L = 11, \sigma_m = 8, A = 6$ 
abcfgz abcdbcfghfgz abcfghdbcfgz abcfghdbcfgz abcfghbcfgz abcfghbcfgz abcdcbcfghfgz	cycles $ L = 7, \sigma_m = 18, A = 8$ 

Table 2. Logs used in the experiments

We have implemented Algorithm 1 in a prototype tool that uses the **STP** solver [9] as the underlying SMT solver. We have compared it with the *Flexible Heuristic Miner* (FHM) [3] which is able to discover a formalism similar to C-nets (called *flexible heuristic nets* from a log [3]). Table 3 shows the results on some small log examples with the following information: *arcs* is the number of arcs of the final C-net, *T* is the elapsed time (in seconds) required to complete the discovery process, *id* indicates if the obtained C-net was identical to the original one, in the case where the log originated from a C-net, or has the same language as a Petri net found using the theory of regions, *cf* is the *cost-based fitness per case* metric of [13] where 1.0 indicates that all sequences in the log belong to the language of the C-net, and the smaller the value is the less sequences are reproducible by the C-net, $|X \cup Y|$ is the number of Boolean variables used to encode the SMT problem, $|E|$ is the number of equations that the SMT problem contains, *bounds* is the initial range in the number of arcs where the binary search must take place, *it* is the number of iterations to obtain this C-net, and column *heur* indicates if some of the heuristics in Sect. 3.6 was used, where *f* refers to restricting the first occurrence of activities and *i* (*o*) to limiting the number of input (output) bindings per activity.

Benchmark	FHM				Algorithm 1									
	arcs	T	id	cf	$ X \cup Y $	$ E $	bounds	arcs	it	T	heur	id	cf	
aalst1 (Fig. 2)	6	0.1	n	0.71	156	147	[6, 11]	6	2	0.3	–	y	1.0	
aalst2b	7	0.0	n	0.24	156	147	[5, 9]	6	3	0.2	–	n	1.0	
mixedXorAnd	8	0.0	n	0.12	219	162	[7, 11]	8	3	0.2	–	y	1.0	
a12f0n00_5	14	0.2	n	0.87	176	143	[12, 17]	14	3	0.1	f	y	1.0	
optional1	6	0.0	n	0.27	413	264	[6, 10]	9	2	0.1	f,o	y	1.0	
cycles	9	0.1	n	0.09	839	542	[8, 17]	9	3	1.3	f,i	y	1.0	
a22f0n00_1	34	0.5	n	0.37	28898	18942	[22, 166]	≤ 39	≥ 4	$> 1h$	–	–	–	

Table 3. Results of discovery algorithm on small examples.

The results on these small benchmarks show that the approach is, in general, able to derive valuable C-nets. In fact the quality of the discovered nets is much better than the ones derived using FHM. For instance, the latter generates four C-nets with empty language. In contrast, Algorithm 1 always generates C-nets whose language contains the given log (fitness=1.0), not only this but also it rediscovers the original C-nets in most of the cases. However two logs are not successfully discovered: for the **aalst2b** benchmark, we obtain a C-net equal to the one in Table 2, but without the arc between *b* and *d*; on the other hand, the largest benchmark in this table (**a22f0n00_1** from [12]) could not be discovered in the one hour limit used in our experiments. Although it seems reasonable to assume that the large number of variables and equations is the responsible of this fact, a more careful evaluation shows that this is not the determinant factor. For instance, in Fig. 5, we can see the time used by the solver to solve the set of

equations, as the bound in the number of arcs allowed is reduced. Initially, the solver finds quickly a solution, but as the bound approaches the lower limit, the time needed grows exponentially. The reason is simple: the set of valid solutions diminishes as the bound also reduces, and the solver has to spend more time searching. Note that the set of equations to solve in all these cases is exactly the same (same variables, same equations) with the only exception that the constant used to bound the number of arcs is different.

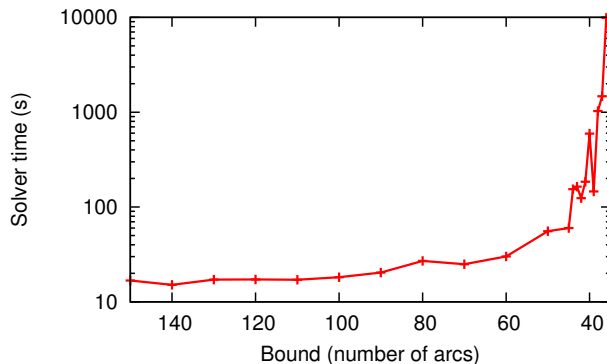


Fig. 5. `stp` solver times for the `a22f0n00_1` benchmark.

To be able to process larger benchmarks we have to resort to our last heuristic (limiting the obligation alphabet). Table 4 shows the results for our previous benchmarks as well as some larger examples also from [12]. In this case we have not used any other heuristic. Despite the fact that the original models were discovered only in three of the benchmarks, in each case the model found was actually the original but in which some additional input and output binding sets were present. This strongly suggests that a strategy that minimizes the number of such sets would greatly improve the results, as well as the overall readability of the derived C-nets.

5 Conclusion and Future Work

This paper has presented an algorithm to derive a C-net from a set of traces, which guarantees minimality in the number of arcs. As future work, we plan to incorporate in the algorithm new ideas on how to bound the language of the C-net obtained. Also, high-level strategies that can make the approach able to handle industrial examples will be considered in the future.

Benchmark	$ L $	$ \sigma_m $	$ A $	$ X \cup Y $	$ E $	bounds	arcs	it	T	id	cf
aalst1	10	5	5	136	137	[6, 11]	6	2	0.0	y	1.0
aalst2b	8	11	5	240	246	[5, 9]	6	3	0.1	n	1.0
mixedXorAnd	3	14	7	89	98	[7, 11]	8	3	0.0	y	1.0
a12f0n00_5	5	7	12	72	91	[12, 17]	14	3	0.0	y	1.0
optional1	11	8	6	229	220	[6, 10]	9	2	0.0	n	1.0
cycles	7	18	8	265	288	[8, 17]	9	3	0.1	y	1.0
a22f0n00_1	99	46	22	12827	10369	[22, 166]	34	7	9.3	n	1.0
a22f0n00_5	836	76	22	121281	97429	[22, 183]	34	7	264.9	n	1.0
a32f0n00_1	100	73	32	26378	19049	[32, 362]	46	8	35.4	n	1.0
a42f0n00_1	100	58	42	48432	31815	[42, 735]	63	9	248.4	n	1.0

Table 4. Results of discovery algorithm when heuristics to limit the number of variables are used (activity window of size 1).

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Van Der Aalst, W., Adriansyah, A., Van Dongen, B.: Causal nets: a modeling language tailored towards process discovery. In: CONCUR. (2011) 28–42
3. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (fhm). In: CIDM, IEEE (2011) 310–317
4. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: ICATPN. Volume 3536 of LNCS. (2005) 48–69
5. Weijters, A., van der Aalst, W., de Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology (2006)
6. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE TKDE **16**(9) (2004) 1128–1142
7. Munoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance. In: Business Process Management (BPM). (2010)
8. Jha, S., Limaye, R., Seshia, S.: Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: Computer Aided Verification. (2009) 668–674
9. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Computer Aided Verification. (2007) 524–536
10. Bergenthum, R., Desel, J., Lorenz, R., S.Mausser: Process mining based on regions of languages. In: Business Process Management (BPM). (2007) 375–383
11. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Theory and Practice of Software Development (TAPSOFT). Volume 915 of LNCS. (1995) 364–383
12. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Petri Nets. Volume 5062 of LNCS. (2008) 368–387
13. Adriansyah, A., van Dongen, B., van der Aalst, W.: Conformance checking using cost-based fitness analysis. In: Enterprise Distributed Object Computing Conference (EDOC). (2011) 55–64