Softw Syst Model (2018) 17:1055–1078
https://doi.org/10.1007/s10270-016-0536-y

CrossMark

REGULAR PAPER

Encoding process discovery problems in SMT

Marc Solé[1] · Josep Carmona[2]

Received: 9 October 2014 / Revised: 11 May 2016 / Accepted: 17 May 2016 / Published online: 6 June 2016
© Springer-Verlag Berlin Heidelberg 2016

Abstract *Information systems*, which are responsible for driving many processes in our lives (health care, the web, municipalities, commerce and business, among others), store information in the form of *logs* which is often left unused. *Process mining*, a discipline in between *data mining* and *software engineering*, proposes tailored algorithms to exploit the information stored in a log, in order to reason about the processes underlying an information system. A key challenge in process mining is *discovery*: Given a log, derive a formal process model that can be used afterward for a formal analysis. In this paper, we provide a general approach based on satisfiability modulo theories (SMT) as a solution for this challenging problem. By encoding the problem into the logical/arithmetic domains and using modern SMT engines, it is shown how two separate families of process models can be discovered. The theory of this paper is accompanied with a tool, and experimental results witness the significance of this novel view of the process discovery problem.

Keywords Process discovery · SMT application · Causal nets · Petri nets

Communicated by Dr. Daniel Varro.

✉ Josep Carmona
jcarmona@cs.upc.edu

Marc Solé
kwisath@gmail.com

[1] CA Strategic Research, CA Technologies, Barcelona, Spain

[2] Universitat Politècnica de Catalunya, Barcelona, Spain

1 Introduction

Nowadays, information systems are continuously monitored, producing a vast amount of data in form of logs that describe the execution of their main processes. One of the principal challenges is to use this data source in order to enhance an information system into several dimensions: correctness, performance, alignment with the specification, among others.

Process mining is a relatively novel discipline which has received a lot of attention in the last decade [1]. By using the logs as source of information, process mining techniques are meant to *discover*, *analyze* and *enhance* formal process models of an information system [2]. There is a certain connection between the well-established *data mining* field, which focuses on the analysis of data sets to obtain hidden relationships, and process mining, since some of the process mining algorithms are grounded on traditional data mining techniques. However, process mining focuses on processes underlying an information system (like the process of handling customer orders, or the process of treating a patient in a hospital), and therefore the problems tackled in these two fields are rather different.

Process discovery faces the following problem: to discover a formal process model (e.g., a Petri net [3], or an automaton) that adequately represents the traces in the log. The reader can refer to Figs. 1 and 2 to see toy examples of process discovery. Process discovery can be oriented to *control-flow* (discover the causal relationships between the activities), *data* (determine data patterns for several purposes) or *social* (find the structure of the human collaboration to carry out processes). In this paper, we focus on control-flow process discovery.

In the last decade, several algorithms for control-flow process discovery have appeared, most of them focused on

المنارة للاستشارات
www.manaraa.com

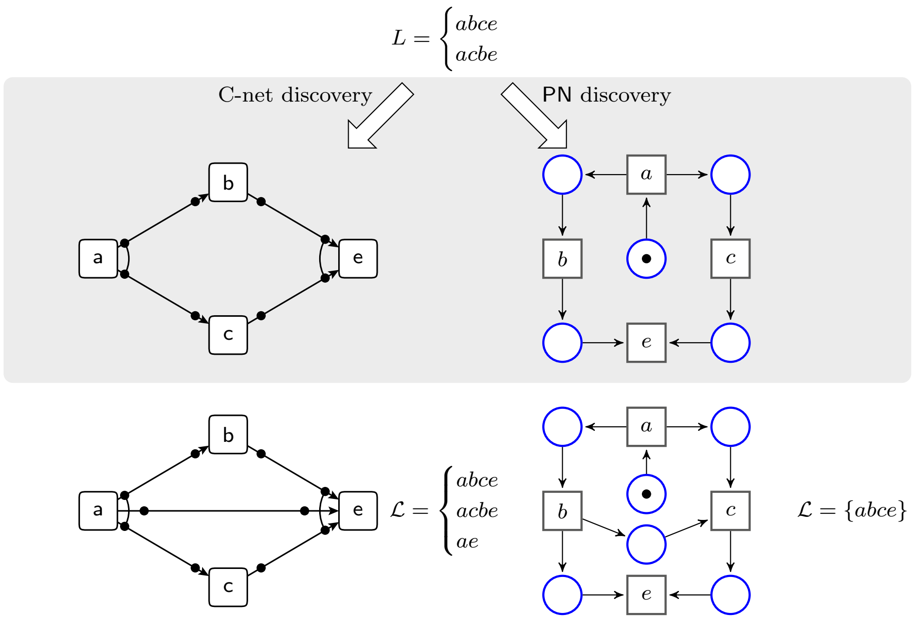

$L = \begin{cases} abce \\ acbe \end{cases}$
C-net discovery
PN discovery
a
b
c
e
$\mathcal{L} = \begin{cases} abce \\ acbe \\ ae \end{cases}$
$\mathcal{L} = \{abce\}$

Fig. 1 Process discovery aims at obtaining a model out of a log. (*Above*) In this example, two models, an additive one (a C-net on the *left*) and a restrictive one (a Petri net on the *right*), are generated from the same log. (*Below*) A model is additive if the addition of elements can only increase the language it represents; on the other hand, restrictive models contain elements (places in the case of Petri nets) whose addition can only restrict the language of the model

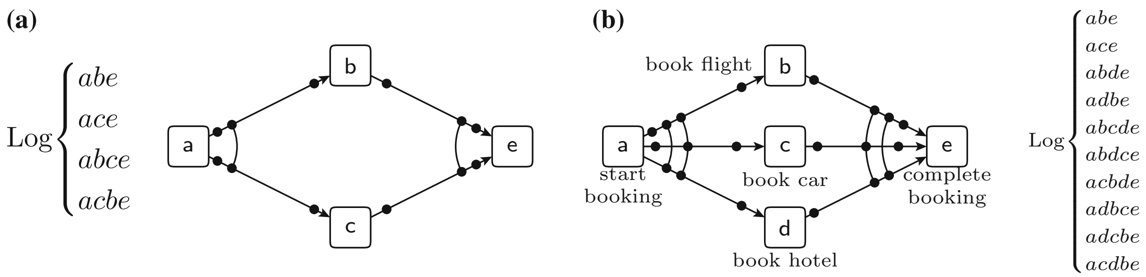


Fig. 2 **a** Causal net. **b** Causal net C_{travel} from [11]

the discovery of Petri nets. These algorithms have diverse assumptions, guarantees and complexity. In general, control-flow discovery algorithms can be split into lightweight methods that focus on restricted formalisms and complex methods that allow for general process models at the expense of a higher computational cost. The contributions of this paper can be categorized into the class of complex control-flow discovery algorithms, casting the discovery as an optimization problem. The techniques of this paper may be an alternative to the current complex techniques, encoding the discovery problem into a satisfiability formula whose satisfying assignments denote the discovered process models.

Satisfiability Modulo Theories (SMT) [4] is a decision problem for first-order logic formulas combined with background theories such as arithmetic, bit-vectors, arrays and uninterpreted functions. It has been successfully applied in several disciplines, including program verification [5], unit testing [6], interactive theorem provers [7], scheduling [8] and planning [9]. In the last decade, there has been an enormous progress in SAT engines, making it possible to apply them in industrial scenarios [10]. This is the driving force that has motivated the work presented in this paper.

A common ingredient in the aforementioned applications is the use of a *model* of an SMT/SAT formula to construct the solution, e.g., the problem of scheduling reduces to modeling the set of restrictions that a valid schedule should satisfy as an SMT formula. The same approach will be applied in this work: Log traces implicitly represent the causal relations of a potential process model, and the problem of process discovery is to derive a process model which satisfies all these implicit causal relations. To present the theory in a general setting, we will distinguish two types of process models: *additive models*, for which the addition of elements in the model implies the addition of behavior. Examples of additive models are *Causal nets* [11]. On the other hand, elements in a *restrictive model* may exclude certain behavior, thus restricting the language produced by the model. An example of restrictive models is Petri nets [3]. The distinction between additive and restrictive models allows us to consider the discovery problem of each class very differently,

Springer
المنارة للاستشارات
www.manaraa.com

describing generic and particular algorithms for each class. The main contribution of this work is the proposal of generic algorithms for these two classes of process models, and the proposal of techniques for encoding the discovery of Petri nets using SMT.

Until now, SMT techniques for software engineering have focused on a program as the main object of study: symbolic execution, model checking, static analysis and verification of programs [10]. This paper brings SMT to a different degree of abstraction, considering the use of formal process models in the life cycle of an information system as a main actor that needs to be obtained, analyzed and enhanced during the different stages of the design of a system.

The techniques of this paper are meant to provide fitting, precise and simple process models without any restriction of the behavior underlying the log. In contexts where *noise* may exist, we foresee the application of these techniques after noise filtering has been applied on the log. Likewise, when the size of the log prevents from applying the techniques of this paper right away, one may use decompositional approaches (e.g.clustering, projection) to derive tractable sublogs that can be handled by the techniques of this paper.

The contribution of this paper with respect to our previous work [12,13] is summarized as follows:

- The techniques in [12,13] only focus in C-nets, while in this work we propose generic algorithms for two classes of process models: additive and restrictive (Sect. 4).
- An SMT-encoding technique for the discovery of Petri nets, and its corresponding implementation and experimental evaluation (Sects. 6, 7).
- A complexity study of the encoding problems presented along the paper.
- A new variant of the tool from [12,13] is presented that uses an SMT solver under pseudo-Boolean constraints, and an experimental evaluation witnessing the improvement over the previous version is reported.

1.1 Organization of this paper

To illustrate the contribution of this paper, an example is provided in Sect. 2, together with a short overview of the current applications of process discovery. Then, in Sect. 3 the necessary preliminaries are briefly introduced. Generic algorithms for the discovery of additive/restrictive models are proposed in Sect. 4. Then, instantiations of these algorithms for the particular case of C-nets and Petri nets are described in Sect. 5 and Sect. 6, respectively. Sect. 7 summarizes a set of experiments that have been performed on the tool supporting the techniques of this paper. Then in Sect. 8 a discussion on related work is provided. Finally, Sect. 9 presents future work and concludes this paper.

2 Process discovery: applications and a motivating example

The growing field of process discovery has been already applied in several scenarios. Here we try to summarize six of them (the list is by no means exhaustive), providing examples of use of process discovery:

- Municipalities: Typical processes in a municipality are issuing building permits or the handling of invoices. Often, there exists no formal or complete definition of these processes, but process discovery can address this. This allows to analyze afterward the differences between organizations, and the degree of alignment between every case and the formal process. Several municipalities in Holland have applied process discovery to their logs [2].
- Health care: Information systems that monitor the processes within hospitals record every event that is produced. This allows, for instance, to have an accurate view of the typical paths followed by a particular group of patients [14,15]. Also, medical devices generate data that can also be analyzed from a process perspective.
- Web services: Service-oriented architecture (SOA) products like IBM WebSphere provide logging of the event information. Consequently, the logs produced can be provided to process discovery techniques in order to formally describe the execution of business processes, and determine its correctness [16].
- Chip manufacturing: ASML is the leading manufacturer of wafer scanners in the world. In [17], a case study shows the applicability of process discovery in this context.
- Auditing: The role of an auditor may change in the presence of process mining techniques, since many checkings can be done automatically and without the restriction to be applied to a small set of records [18].
- Software engineering: Apart from remarkable open-source/academic tools like ProM (Eindhoven University), several well-known software vendors are incorporating process discovery capabilities on their products: ARIS Process Performance Manager (Software AG), Comprehend (Open Connect), Discovery Analyst (StereoLOGIC), Flow (Fourspark), Futura Reflect (Perceptive Software), Interstage Automated Process Discovery (Fujitsu), OKT Process Mining suite (Exeura), Process Discovery Focus (Iontas/Verint), Disco (Fluxicon), Celonis and Minit (Gradient).

We informally describe with a toy example the problem of process discovery and the main differences between additive and restrictive models. Let us assume an information system coordinating the purchase of items in an online shop. We focus on four particular events of the purchase process: Activity *a* corresponds to a customer logging into the system,

المنارة للاستشارات
www.manaraa.com

b represents the fact that the customer places some online orders, c marks the finalization of a survey with his/her satisfaction with the company, while e represents the customer logout. By monitoring the system, the two traces *abce* and *acbe* have been recorded in the log L, as shown in Fig. 1. Informally, in the behavior represented the customer first enters into the system (a), then performs b and c (in any possible order) and finally exits the system (e).

The discipline of Process discovery aims at generating a process model from the traces contained in a log. In the context of this paper, a process model is considered to have two crucial characteristics: (1) a graphical description, to enable the visualization in a software engineering setting, and (2) a formal semantics, to allow for the unambiguous reasoning on the underlying behavior. The two classes of models present in the figure, namely C-nets (left) and Petri nets (right), are representatives of two broad families of process models: additive models and restrictive models, respectively. The formal semantics of these two models are described in the following sections, but can be intuitively understood as follows: In the C-net, the activity a can occur (since it is free from input obligations), generating obligations to occur to the adjacent activities b and c. Generating obligations is denoted by the dots associated with the arcs exiting from the activity. For the Petri net, the process has initially a token (shown as a dot) in the initial place (the circle beneath the a). When the transition a occurs, the token is removed and copied to the places on either side of a, enabling the transitions b and c. These models have been obtained from the log of the figure by using the techniques described in this paper.

The difference between an additive and a restrictive model can be seen by considering the slight modifications made in the structure of each one of the models discovered: The addition of an arc between event a and e in the discovered C-net gives rise to the C-net in the bottom-left corner. This addition incorporates the trace *ae* as a possible behavior (i.e., a customer is allowed to leave the system without neither purchasing anything nor filling the survey). In contrast, the new place connecting transitions b and c in the Petri net discovered produces the Petri net in the bottom-right corner of the figure. This net enforces the customer to first buy some goods and then take the survey, allowing only one of the possible traces in the discovered model.

3 Background

3.1 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 $X \to \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers. 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 $(M_1 \subseteq M_2) \Leftrightarrow \forall x \in X, M_1(x) \leq M_2(x)$. We say a multiset M is *k-bounded* if $\forall x \in X, M(x) \leq k$. As usual, sets will be considered as bags when necessary.

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. For instance, in the log of Fig. 1 all sequences start with activity a and end with activity e, and these activities appear only once in each sequence. This assumption is without loss of generality, since any log can be easily converted into this form by using two new activities that are properly inserted in each trace.

Given a finite sequence of elements $\sigma = e_1 e_2 \ldots e_n$, its length is denoted by $|\sigma| = n$, and the element at position i (e.g., e_i) is denoted by σ_i. Its prefix sequence up to element i (but not including it), with $i \leq n + 1$, denoted by $\sigma_{\leftarrow i}$, is $e_1 \ldots e_{i-1}$. We define $\sigma_{\leftarrow 1}$ as the empty sequence, denoted by ϵ. Conversely, its suffix sequence after i, with $i < n$, denoted by $\sigma_{i \rightarrow}$, is $e_{i+1} \ldots e_n$. We express the fact that an element e appears in sequence σ as $e \in \sigma$. The alphabet of σ, denoted by 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., $A_L = \bigcup_{\sigma \in L} A_\sigma$.

3.2 Process discovery

We assume a log L represents the footprints of the real process executions of a system S that is only (partially) visible through these runs. Process discovery techniques aim at extracting a process model M (e.g.a Petri net) from L with the goal of eliciting the process underlying in S. We denote $obs(M)$ as the set of traces underlying a model M. By relating the behaviors of L, $obs(M)$ and S, particular concepts can be defined [19]. A model M *fits* log L if $L \subseteq obs(M)$. A model is *precise* in describing a log L if $obs(M) \backslash L$ is small. A model M represents a *generalization* of log L with respect to system S if some behavior in $S \backslash L$ exists in $obs(M)$. Finally, a model M is *simple* when it has the minimal complexity in representing $obs(M)$, i.e., the well-known *Occam's razor principle*. It is widely acknowledged that the size of a process model is the most important simplicity indicator [2].

The problem of process discovery is solved by process discovery algorithms that are formally defined as functions that map L onto a process model M in such a way that some of the aforementioned metrics (fitness, precision, generalization and simplicity) are optimized. Unlike in several approaches in the literature [2], in this work we will take into

www.manaraa.com

account most of these factors when deriving the models. First, the methods can be used to derive fitting models. Second, we consider techniques to improve precision and generalization of the derived models (see for instance Sect. 4.1). Finally, we incorporate techniques to simplify process models (e.g.reducing arcs or bindings in a C-net) without severely penalizing the other quality metrics.

4 Generic algorithms for the discovery of additive and restrictive models based on SMT

Satisfiability modulo theories (SMT) is a decision problem for logical formulas with respect to combinations of background theories expressed in first-order logic with equality. Examples of theories are the theory of real numbers, the theory of integers and the theories of various data structures such as lists, arrays or bit-vectors. SMT is the problem of determining whether an instance formula is satisfiable.

This section describes the SMT-based generic algorithmic support for the two classes of process formalisms considered in this work: additive and restrictive models. As will become clear at the end of the section, this alternative on process formalisms is meaningful since the nature of discovery techniques required for each class is completely different.

4.1 Discovery of additive models

Structural elements in an additive model have the role of expressing the behavior allowed by the model. Hence, the more structural elements are present, the more behavior is described in the model. An example of an additive model is a grammar: The addition of a new production rule can only express more behavior accepted by the grammar. Another example of additive model is an automaton: The addition of arcs and states potentially increases the language. Techniques for discovery C-nets, an additive process formalism, are presented in Sect. 5.

For many additive models, it is possible to construct an SMT problem whose solution can be transformed into a model capable of describing a given log L (and possibly showing additional behavior). A general algorithm for such a task is shown in Algorithm 1 (for the particular case in which the generic cost function is minimized). The objective of this algorithm is to find the smallest model M with perfect fitness ($\mathcal{L}(M) \supseteq L$).

The fundamental idea is to encode the equations that guarantee that the sequences in the log L can be replayed by the derived model M (the `structural_equations` function, line 5), thus guaranteeing a perfect *fitness* of the model. An example of structural equations for a particular example of additive model is described in detail in Sect. 5. Then a cost function must be defined (in the algorithm, it is assumed to be an integer cost function), typically based on the number of elements that the model contains. There are two functions in Algorithm 1 related to the cost function: One is `cost` that returns the value of the cost function for a given model M; the other is `cost_function` which encodes as an SMT formula the computation of the cost function using the SMT variables appearing in `structural_equations`.

Algorithm 1 Discover optimized additive model

```
1:  function DISCOVERADDITIVEMODEL($L$)
2:      $M \leftarrow$ trivial_model($L$)
3:      $min \leftarrow$ cost_lower_bound($L$)        ▷ The cost of a solution is $\geq min$
4:      $max \leftarrow$ cost($M$) $- 1$
5:      $E_s \leftarrow$ structural_equations($L$)
6:      while $min \leq max$ do
7:          $avg \leftarrow \lfloor (min + max)/2 \rfloor$
8:          $E \leftarrow E_s \wedge$ (cost_function($L$) $\leq avg$)
9:          $feasible, solutions \leftarrow$ solve($E$)        ▷ Call SMT solver
10:         if $feasible$ then
11:             $M \leftarrow$ extract_model($solutions$)        ▷ Model feasible
12:             $max \leftarrow$ cost($M$) $- 1$        ▷ Since cost($M$) $\leq avg$
13:         else
14:             $min \leftarrow avg + 1$        ▷ Model unfeasible
15:         end if
16:     end while
17:     return $M$
18: end function
```

Since the latter function allows limiting the cost of the model found (if the SMT problem is feasible), using a binary search (lines 6–16) this cost function is minimized, thus yielding a model with the least number of elements considered by the function. The binary search uses the functions `solve` and `extract_model`. Function `solve`(E) calls the SMT solver on the set of equations E and returns two values: *feasible* and *solutions*. *feasible* is a Boolean value indicating whether the solver found a solution to the equations in E. *solutions* contains the values of the SMT variables in case the problem was feasible. On the other hand, function `extract_model`($solutions$) builds an additive model from the values of SMT variables where its language is guaranteed to include L.

Note that a binary search requires some initial bounds on the cost function. For this reason, a trivial (and typically large) model is initially built that can replay the log, to provide an initial upper bound (line 4). The initial lower bound (represented by the function `cost_lower_bound`) can be derived from the information in the log or some restriction of the model.

The minimization of model elements achieved with the binary search addresses another conformance factor, *simplicity* (see Sect. 3.2). In general, it is assumed that models with less elements are simpler. Thus, this algorithm, in the general case, guarantees that we obtain the simplest possible model (in terms of number of arcs) that provides complete fitness of the log.

المنارة للاستشارات
www.manaraa.com

This algorithm is valid for any additive model as long as the necessary restrictions can be encoded in a particular SMT domain for which there is a solver available. However, the usefulness of the algorithm may be hampered for many different factors, including:

- The complexity of the encoding of the structural equations or the cost function.
- The difficulty of defining a suitable cost function that promotes as many conformance factors (see Sect. 3.2) as possible.

For instance, let us consider an automaton, which is an additive model. If the cost function to be minimized is the number of arcs or states, the resulting optimal automaton will contain a single state and a self-loop for every symbol in the alphabet A_L of the log L; thus, the automaton will represent the language A_L^*. This is a trivial solution for which the SMT apparatus was unnecessary. However, as we see in this section, there exist additive models for which it is possible to define simple encodings and have useful cost functions.

Further uses of SMT in additive models: The flexibility of SMT problems can be used to tackle one of the most challenging problems for additive models: generate a model that explicitly forbids some behavior (thus improving precision if behavior not present in the log is forbidden). While the latter is straightforward for restrictive models, additive models must consider all their components and their interactions. For instance, in the case of a grammar that can produce a forbidden behavior, the question is how to modify the productions so that the behavior in the log can still be produced by the grammar, but, at the same time, the behavior we want to forbid cannot be generated. In this section, we outline a methodology that can be used in combination with an SMT solver to achieve this particular objective.

First of all we must find undesired behavior. This can be explicitly given as negative examples by the user in some cases, but frequently these counterexamples are not available. Assuming we want to restrict as much as possible the model to the behavior in the log, we consider as undesired behavior the one included in the model but not present in the log. The idea will be to find this behavior, determine whether there is enough evidence in the log so as to consider it undesired, and then forbid the generation of this behavior in the creation of the model.

Algorithm 2 Forbid behavior in an additive model

```
1:  function ForbidInAdditiveModel($M, L, t$)
2:    $E_s \leftarrow$ structural_equations($L$)
3:    $l \leftarrow 1$
4:    while $l \leq t$ do
5:      $\sigma \leftarrow$ forbidden_behavior($M, L, l$)
6:      if $\sigma = \epsilon$ then                                ▷ If no such $\sigma$ exists
7:        $l \leftarrow l + 1$                                  ▷ Increment length of $\sigma$
8:      else                                          ▷ $|\sigma| = l \wedge \sigma \in \mathcal{L}M \setminus L$
9:        if relevant($L, \sigma$) then
10:         $F \leftarrow$ elements($M, \sigma$)
11:         $E_s' \leftarrow E_s \wedge \bigvee_{e \in F} e? \notin M$     ▷ Forbid some element of $F$
12:         $feasible, M' \leftarrow$ binary_search($E_s'$)          ▷ Obtain minimal model satisfying $E_s'$
13:         if $feasible$ then
14:           $E_s \leftarrow E_s'$                         ▷ Update structural equations
15:           $M \leftarrow M'$                                   ▷ Update model
16:           $l \leftarrow 1$ ▷ To ensure new model forbids all lengths
17:         else
18:           $L \leftarrow L \cup \{\sigma\}$ ▷ Discard $E_s'$. Avoid finding again $\sigma$
19:         end if
20:       else
21:         $L \leftarrow L \cup \{\sigma\}$                       ▷ Avoid finding again $\sigma$
22:       end if
23:     end if
24:   end while
25:   return $M$
26: end function
```

A particular realization of this general concept is illustrated in Algorithm 2. The algorithm receives three parameters: the additive model M, the log L and a threshold t. The idea is that all sequences up to length t that can be generated by the model but do not appear in the log, i.e. $\mathcal{L}(M)\backslash L$, are considered as potentially undesired behavior[1]. The algorithm starts by looking for potentially undesired sequences of length l (line 5) using the function `forbidden_behavior`. This function returns the empty sequence ϵ if no sequence of length l exists in $\mathcal{L}(M)\backslash L$. In such a case, the length of the searched sequence l is incremented. Otherwise, we have found a sequence that we potentially have to forbid. The function `relevant` determines whether, considering the information of the log, the sequence is relevant enough to be forbidden or not.

If the sequence is not considered relevant, we add it to the log preventing that the algorithm finds the same sequence over and over again. Otherwise, we compute the set F of elements in the model M that are involved in the production of σ. This can be done by replaying the sequence on the model. For instance, in a grammar this set would be the set of productions of the grammar that are needed to produce the sequence (if more than one set is possible, only one of them is returned by this function). Then, the set of structural equations E_s is extended with an equation that forbids at least one of the elements required to produce σ. If several sets F exist, the algorithm will keep finding σ until all possible sets

[1] Notice that depending on the notion of valid sequence, the notion of undesired behavior may vary. For instance, for certain formalisms, only complete sequences (i.e., sequences from start to end) may be considered. For the sake of generality, we opt to abstracting from these matters in Algorithm 2.

المنارة للاستشارات
www.manaraa.com

F have been considered. Once the enriched set of structural equations E_s' has been computed, the model (if feasible) is minimized using the function `binary_search` that corresponds to lines 6–17 of Algorithm 1 substituting E_s by the parameter of the function, in this case E_s'. If the problem is feasible, then the changes in the structural equations and the model are accepted; otherwise, σ is added to L to prevent finding the sequence again, since we cannot forbid this behavior[2]. In the former case, notice that the length is reset to 1 in Step 16: To avoid that M' may incorporate forbidden behavior already removed in M, since M' and M can be drastically different. Therefore, the undesired behavior of any length below the threshold t must be tested each time a new model is computed. In any case, since the number of potential traces to forbid is finite (and depends on the maximal length t), Algorithm 2 terminates.

An implementation of this algorithm has demonstrated to be crucial for tackling particular discovery instances, as demonstrated in Sect. 7. Since this algorithm strongly relies on the notion of replay, Sect. 5.4 provides a detailed description for the particular case of C-nets.

4.2 Discovery of restrictive models

The structural elements of a restrictive model are meant to cut the set of potential behaviors. Therefore, adding a new element implies that some behavior is left out. An example of restrictive model is a linear programming model, where the addition of constraints clearly reduces the space of solutions of the model. In the context of process mining, Petri nets [3] are the representative restrictive model. A technique for the discovery of Petri nets is described in Sect. 6.

Since an element of a restrictive model may potentially remove behavior, the general approach for this class of systems is necessarily quite different. The SMT encoding for additive models is a juxtaposition of different subproblems, i.e., each sequence could constitute a single SMT problem and solved independently, and the union of all these solutions would still be a valid global solution to the whole set of sequences. The derived subproblems are put together to allow optimizing the number of elements in the model (e.g., minimizing the number of arcs in the C-net). In contrast, the SMT problem for restrictive models should consider all the sequences in the log, because we must ensure that the new element does not restrict any of the observed behavior. Thus, in this approach, several SMT problems are solved, and each one of these solutions corresponds to one element of the model. We iteratively discover new restrictive elements until we obtain some guarantee that no other restrictive element can be found that forbids some non-observed behavior.

Algorithm 3 Discover optimized restrictive model

```
1: function DISCOVERRESTRICTIVEMODEL($L$)
2:     $M \leftarrow$ empty_model()
3:     $E_s \leftarrow$ structural_equations($L$)
4:     while constrainable($M$, $L$) do
5:         $E \leftarrow E_s \wedge$ (new_element($E_s$, $M$))    ▷ Find element not previously found
6:         $feasible, solutions \leftarrow$ solve($E$)    ▷ Call SMT solver
7:         if $feasible$ then
8:             $M \leftarrow M \cup \{$extract_element($solutions$)$\}$    ▷ Add element to $M$
9:         end if
10:    end while
11:    return $M$
12: end function
```

Algorithm 3 shows a general strategy for deriving the most restrictive model. The basic idea is that we start with the empty model and we keep adding elements to it until no further restriction of the language is possible. Thus, this strategy clearly focuses on *fitness* (no observed behavior is left out) and *precision* (no other model can be built which has a smaller language). A particular instantiation of this algorithm is presented in Sect. 6. In detail, the algorithm is built on the following helper functions:

- `structural_equations` characterizes the constraints that derived model elements must satisfy in order to not forbid valid sequences in the log.
- `constrainable` tests whether new elements can be added to further restrict the model while accepting the behavior from the log.
- `new_element` provides further constraints that enforce the structural equations used so far in order to guide the search for new model elements.
- `solve` effectively determines whereas solutions exists after the aforementioned enforcing of the structural equations.
- `extract_element` extracts one solution to be added to the model when the SMT instance is feasible.

As with additive models, this algorithm is useful as long as some conditions are met. For instance, the number of possible elements in a model must be finite, and we must be able to determine whether a model can be further restricted (the function `constrainable` in line 4 must be computable). Not all restrictive models have this property; however, Petri nets have a well-founded theory that can be used to compute such a termination criterion.

5 Discovering strategies for C-nets based on SMT

Recently, a formalism called Causal nets (C-nets) [11] has been proposed as a suitable modeling language for process

[2] In this case if previous iterations of the algorithm were only due to forbidding σ on other parts of the model, these modifications could in principle be rolled back.

المنارة للاستشارات
www.manaraa.com

mining. It is an additive model that allows expressing complex behavior that is sometimes difficult to describe using other models. Unlike grammars, C-nets have a graphical counterpart which makes them attractive in the context of process mining.

Let us first describe with the help of a couple of simple examples the semantics of C-nets. Figure 2a shows a log and a C-net whose language is exactly the set of traces in the log. The semantics of the C-net can be informally described as:

Activity a must be executed initially, since no obligations (input arcs with dots) exist for a. It can generate obligations to either (1) activity b, or (2) activity c or (3) activities b and c. Any of the these three possibilities requires the execution of the corresponding activities, consuming the obligation(s) from activity a and generating obligation(s) to activity e. The final execution of e will empty the set of obligations and therefore will lead to a valid trace.

Figure 2b (from [11]) shows a more meaningful example, describing a C-net that models the process of booking resources for travel. By considering the informal semantics described in the C-net of Fig. 2a, we let as an exercise for the reader to check whether only the traces listed in the log belong to the language of the C-net. The following section provides the formal definition of C-nets.

5.1 Causal nets (C-nets)

Definition 1 (*Causal net* [11]) A C-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:

- $\{a_s\} = \{a \mid I(a) = \{\emptyset\}\}$ and $\{a_e\} = \{a \mid O(a) = \{\emptyset\}\}$
- all the activities in the graph $(A, \mathrm{arcs(C)})$ are on a path from a_s to a_e, where arcs(C) is the *dependency relation induced by I and O* such that $\mathrm{arcs(C)} = \{(\mathrm{a}_1, \mathrm{a}_2) \mid \mathrm{a}_1 \in \bigcup_{\mathrm{X} \in \mathrm{I(a_2)}} \mathrm{X} \wedge \mathrm{a}_2 \in \bigcup_{\mathrm{Y} \in \mathrm{O(a_1)}} \mathrm{Y}\}$.

Definition 1 slightly differs from the original one from [11], where the set arcs(C) is explicitly defined in the tuple. The C-net of Fig. 2a 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: $\mathrm{arcs(C)} = \{(\mathrm{a, b}), (\mathrm{a, c}), (\mathrm{b, e}), (\mathrm{c, e})\}$. 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 $\{a\} \in I(b)$ is the dot in arc (a, b) next to b. Non-singleton activity bindings are represented by circular segments connecting the dots: $\{b, c\} \in O(a)$ is represented by the two dots in arcs (a, b), (a, c) that are connected through a circular segment.

Definition 2 (*Binding, binding sequence, activity projection*) Given a C-net $\langle A, a_s, a_e, I, O\rangle$, $B = \{(a, S^I, S^O) | a \in A \wedge S^I \in I(a) \wedge S^O \in O(a)\}$ is the set of activity bindings. A binding sequence $\beta \in B^*$ is a sequence of activity bindings. Given a binding sequence $\beta = (a_1, S^I_1, S^O_1) \ldots (a_{|\beta|}, S^I_{|\beta|}, S^O_{|\beta|})$, its activity projection is the activity sequence denoted by $\sigma_\beta = a_1 \ldots a_{|\beta|}$. Bindings from a binding sequence β, we obtain an activity sequence denoted as σ_β.

Two binding sequences of the C-net in Fig. 2a 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 Definition 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 are bags of *obligations* (activity 2-tuples). An obligation (a, b) expresses that activity a has executed and expects b to execute. When this obligation is satisfied, it is removed from the state and thus a state informally represents the bag of pending (i.e., not yet satisfied) obligations. The state reached by the C-net after the execution of a binding sequence β is defined with the help of a function ψ: It maps sequences of bindings (formally B^*, where B is the set of bindings of Definition 2) to the state space S. Function $\psi : B^* \rightarrow S$ 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 binding sequence $\beta = (a_1, S^I_1, S^O_1) \ldots (a_{|\beta|}, S^I_{|\beta|}, S^O_{|\beta|})$ is said to be *valid* if the following conditions hold:

1. $a_1 = a_s, a_{|\beta|} = a_e$ and $\forall k : 1 < k < |\beta|, a_k \in A \backslash \{a_s, a_e\}$
2. $\forall k : 1 \leq k \leq |\beta|, (S^I_k \times \{a_k\}) \subseteq \psi(\beta_{\leftarrow k})$
3. $\psi(\beta) = \emptyset$

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

For instance, in Fig. 2a, 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 $\{abe, ace, abce, acbe\}$.

5.2 C-net discovery

Given a log L, the problem tackled in this section is to derive a C-net C that addresses satisfactorily the factors described

www.manaraa.com

in Sect. 3.2. Concretely, we tackle *fitness* by guaranteeing that all the sequences of the log belong to the language of the model, *simplicity* by minimizing the structural elements of the net and *precision* because by removing unnecessary structural elements we also restrict the language of the model. We now present a method to accomplish this, based on encoding the discovery problem as an SMT instance.

5.2.1 Protobinding sequences of a log

In Sect. 5.1, 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. Then, we will show that given these sequences, it is possible 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 4 (*Protobinding, well-formed protobinding seq.*) A triple (a, X, Y) is a *protobinding* if a is an element and both X and Y are sets. A sequence $\beta = (a_1, X_1, Y_1)\dots(a_{|\beta|}, X_{|\beta|}, Y_{|\beta|})$ of protobindings is *well-formed* if it satisfies the following conditions:

(W1) $\forall i : 1 < i \le |\beta|, X_i \neq \emptyset \wedge a_i \neq a_1$
(W2) $\forall i : 1 \le i < |\beta|, Y_i \neq \emptyset \wedge a_i \neq a_{|\beta|}$
(W3) $X_1 = Y_{|\beta|} = \emptyset$
(W4) $\forall i : 1 \le i \le |\beta|, \psi(\beta_{\leftarrow i}) \supseteq (X_i \times \{a_i\})$
(W5) $\psi(\beta) = \emptyset$

Given a set B of well-formed sequences of protobindings, it is possible to characterize the C-nets such that their set of valid sequences of bindings contain the sequences of protobindings in B, as the next theorem states.

Theorem 1 *([12]) 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) $A = \{a \mid \exists \beta \in B : (a, X, Y) \in \beta\}$
(T2) $\forall a \in A,\ I(a) = \{X \mid \exists \beta \in B, \exists Y : (a, X, Y) \in \beta\}$
(T3) $\forall a \in A,\ O(a) = \{Y \mid \exists \beta \in B, \exists X : (a, X, Y) \in \beta\}$

is a C-net such that $V(C) \supseteq B$.

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 sequences in the log have the same initial and final activities, 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 was proven in [12] that 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.

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

5.2.2 Encoding the problem as linear constraints

Given a sequence σ of a log L, it is trivial to build a protobinding sequence β_σ out of it as $\beta_\sigma = (\sigma_1, X_1, Y_1)\dots(\sigma_{|\sigma|}, X_{|\sigma|}, Y_{|\sigma|})$. The difficult part is to ensure that β_σ is actually well formed. We will encode the unknown X_i (input bindings) and Y_i (output bindings) 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 1 ([12]) Let σ be a sequence of activities. Consider the protobinding sequence $\beta_\sigma = (\sigma_1, X_1, Y_1)\dots(\sigma_{|\sigma|}, X_{|\sigma|}, Y_{|\sigma|})$. If β_σ is well-formed, then $\forall i : 1 \le i \le |\sigma|, X_i \subseteq A_{\sigma_{\leftarrow i}} \wedge Y_i \subseteq A_{\sigma_{i\rightarrow}}$.

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 $x_{\sigma,i,(a,\sigma_i)}$ to indicate whether activity a belongs to X_i in β_σ or not. Note that the subscript contains one redundant element (σ_i) that we keep for readability. The other elements are necessary: σ allows us to distinguish the variables assigned to different sequences, i avoids confusion between variables when the same activity appears in different positions of σ and a is required to identify the obligation consumed. As usual when sets are encoded using characteristic functions we use the following semantics:

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

Similarly, the variable $y_{\sigma,i,(\sigma_i,a)}$ indicates if a belongs to Y_i in β_σ. Due to Property 1, the activity a for variables in X_i can only be chosen among the alphabet of prefix $\sigma_{\leftarrow i}$, i.e., $A_{\sigma_{\leftarrow i}}$, 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 $\mathcal{X}$ and $\mathcal{Y}$ the set of all x and all y variables, respectively.

We will now rewrite the conditions (W1,W2,W3,W4 and W5) of Definition 4 for describing a well-formed protobinding sequence $\beta_\sigma = (\sigma_1, X_1, Y_1)\dots(\sigma_{|\sigma|}, X_{|\sigma|}, Y_{|\sigma|})$ as inequalities using the $\mathcal{X}$ and $\mathcal{Y}$ variables.

المنارة للاستشارات
www.manaraa.com

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:

$$\boxed{\forall i : 1 < i \leq |\sigma|, \quad \sum_{a \in A_{\sigma_{\leftarrow i}}} x_{\sigma,i,(a,\sigma_i)} \geq 1} \tag{1}$$

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 $Y_{|\sigma|}$) are non-empty:

$$\boxed{\forall i : 1 \leq i < |\sigma|, \quad \sum_{a \in A_{\sigma_{i \rightarrow}}} y_{\sigma,i,(\sigma_i,a)} \geq 1} \tag{2}$$

Condition W3 This needs no conversion, since we can directly assign the empty set to X_1 and $Y_{|\sigma|}$. Note that the model does not even generate any variable in $\mathcal{X}$ or $\mathcal{Y}$ to represent these sets, since $A_{\sigma_{\leftarrow 1}} = A_{\sigma_{|\sigma| \rightarrow}} = \emptyset$.

Condition W4 This condition requires that the state of obligations after executing prefix $\beta_{\leftarrow i}$ (i.e., $\psi(\beta_{\leftarrow i})$) contains, at least, the obligations in $(X_i \times \{\sigma_i\})$. This is the same as requiring that the number of obligations of the type (a, σ_i) in $\psi(\beta_{\leftarrow i})$ is larger or equal than the number of obligations (a, σ_i) in $(X_i \times \{\sigma_i\})$. Moreover, if σ_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 $\psi(\beta_{\leftarrow i})$ 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 σ_i.

$$\boxed{\forall i : \left(1 \leq i \leq |\sigma| \wedge \exists j : \left(j > i \wedge \sigma_j = \sigma_i\right)\right), \forall a \in A_{\sigma_{\leftarrow i}}, \quad \sum_{k:k<i \wedge \sigma_k = a} y_{\sigma,k,(a,\sigma_i)} - \sum_{m:m \leq i \wedge \sigma_m = \sigma_i} x_{\sigma,m,(a,\sigma_i)} \geq 0} \tag{3}$$

where the first term in the subtraction from Eq. (3) describes the obligations generated up to a given point, whereas the second term considers the obligations consumed.

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

$$\boxed{\forall i : \left(1 \leq i \leq |\sigma| \wedge \forall j \left(j > i \Rightarrow \sigma_j \neq \sigma_i\right)\right), \forall a \in A_{\sigma_{\leftarrow i}}, \quad \sum_{k:k<i \wedge \sigma_k = a} y_{\sigma,k,(a,\sigma_i)} - \sum_{m:m \leq i \wedge \sigma_m = \sigma_i} x_{\sigma,m,(a,\sigma_i)} = 0} \tag{4}$$

Definition 5 (*Structural equations*) The set of equations for a C-net including the behavior of a log L, denoted by *structural_equations*(L), is the set obtained by joining the set of Eqs. (1), (2), (3) and (4) for every $\sigma \in L$.

Example 1 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 $A_{\sigma_{\leftarrow 1}} = \emptyset$, 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 $\mathcal{X}$ and $\mathcal{Y}$ variables in the equations arising from a log, one can derive a C-net that includes the language of the log[3]. In terms of complexity, the number of variables that each 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 $\mathcal{O}\left(|L| \cdot |A| \cdot \max_{\sigma \in L}\left(|\sigma|\right)\right)$.

Depending on the input formula, SMT solvers either convert it into a SAT problem or can use tailored strategies for the non-Boolean parts present in the formula. Given the mixture of linear and Boolean equations that form the problem described above, we use the following convention when computing the number of formulas that form the SMT problem:

[3] Remarkably, the SMT technique proposed can be applied individually to every trace of the log, which allows to independently solve the problem when complexity issues may arise.

المنارة للاستشارات
www.manaraa.com

Table 1 Structural equations for sequence *abcbe*

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

We give both the number of linear equations and Boolean disjunctive clauses that form each equation. To count the disjunctive clauses we assume that every linear equation appearing in a Boolean formula is substituted by a dummy Boolean variable, and then the Boolean formula is expressed in CNF. Using this definition, the number of equations is summarized in the following table:

Equation in SMT problem	# Linear equations (per σ_i)	# Disjunctive clauses (per σ_i)
(1)	1	–
(2)	1	–
(3) and (4)	$\lvert A_{\sigma_{i-1}}\rvert$	–

Thus for an activity $\sigma_i \in \sigma$ we have $|A_{\sigma_{i-1}}| + 2$ linear equations. So for a sequence σ, the maximum number of equations is $\mathcal{O}(|A| \cdot |\sigma|)$. Therefore, the whole log L requires $\mathcal{O}(|L| \cdot |A| \cdot \max_{\sigma \in L}(|\sigma|))$ equations, the same as the number of variables. Next sections illustrate how to algorithmically solve the discovery problem described in this section.

5.2.3 Solving linear constraints using SMT

The main goal of this section is to show how to solve the problem of discovery in the SMT domain. SMT solvers for the theory of quantifier-free bit-vector arithmetic [20] can model Eqs. (1)–(4). Additionally, they can naturally encode the bound on the number of arcs in the C-net, as well as some other constraints (for instance, the heuristics for limiting the number of input/output bindings per activity or limiting the earliest time an activity can be executed as presented in [12], or the formulas of Sect. 5.4).

Variables in $\mathcal{X}$ and $\mathcal{Y}$ are all Boolean, so obtaining 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): $\sum_{a\in A_{\sigma_{\leftarrow i}}} x_{\sigma,i,(a,\sigma_i)} \geq 1$ can be rewritten as $\bigvee_{a\in A_{\sigma_{\leftarrow i}}} x_{\sigma,i,(a,\sigma_i)} = 1$. Equations (3) and (4) are pseudo-Boolean (i.e.they are lineal combinations of binary variables) which can be also expressed using the quantifier-free bit-vector arithmetic (for instance, encoding them using adders).

Note that these equations (ignoring the formulas used by the heuristics) can be also solved by pseudo-Boolean solvers, so in the experiments (Sect. 7) we will also compare the strategy of using the quantifier-free bit-vector arithmetic with them, when this is possible (if only heuristics that can be expressed as pseudo-Boolean formulas are used).

5.2.4 Adding a cost function

Due to the additive nature of C-nets, reducing the number of arcs tends to restrict the language of the net. Fortunately, it

المنارة للاستشارات
www.manaraa.com

is possible to encode an expression as an SMT formula that bounds the number of arcs in the derived C-net. To accomplish this, we can use any of the sets $\mathcal{X}$ or $\mathcal{Y}$. Without loss of generality, we use set $\mathcal{X}$. For readability we introduce an auxiliary notation to denote the subset of variables in $\mathcal{X}$ that correspond to a given binding (a, b) in the sequences of a log L. Namely, $\mathcal{X}_{(a,b)}(L) = \{x_{\sigma,i,(a,b)} \mid \exists \sigma \in L : \sigma_i = b \wedge a \in A_{\sigma_{\leftarrow i}}\}$. We can now characterize in an SMT formula the number of arcs in the C-net obtained through T1, T2 and T3 (Theorem 1) using the following pseudo-Boolean expression (in which we abuse notation so that the logical Or is interpreted as binary value that can added in the summation):

$$\boxed{\text{number\_of\_arcs}(L) \stackrel{\text{def}}{=} \sum_{a \in A_L} \sum_{b \in A_L} \bigvee\nolimits_{x \in \mathcal{X}_{(a,b)}(L)} x}$$

Then, the equation bounding the number of arcs is:

$$\boxed{\text{bound\_arcs}(L, l) \stackrel{\text{def}}{=} \text{number\_of\_arcs}(L) \leq l} \tag{5}$$

In terms of complexity, we have the following number of equations:

Equation in SMT problem	# Linear equations	# Disjunctive clauses
(5)	1	$\lvert A\rvert \cdot \sum_{\sigma \in L} \lvert\sigma\rvert$

In Sect. 5.3, we 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 counting the arcs of a trivial C-net that includes the language of the log (the "immediately follows" C-net of [12]). Given a log L, the "immediately follows" C-net can replay all the sequences in L and is based on the *immediately follows* relation [21] between the activities in L, denoted by $<_L$ and defined as $<_L = \{(\sigma_i, \sigma_{i+1}) \mid \exists \sigma \in L \wedge 1 \leq i < |\sigma|\}$.

Definition 6 (*Immediately follows C-net*) Given a log L, the *"immediately follows" C-net of* L, denoted by $C_{\mathrm{IF}}(L)$, is the C-net $\langle A, a_s, a_e, I, O\rangle$ such that: (i) $A = A_L$, (ii) $\forall \sigma \in L, \sigma_1 = a_s \wedge \sigma_{|\sigma|} = a_e$, (iii) $\forall a \in A, O(a) = \{\{b\} \mid a <_L b\} \wedge I(a) = \{\{b\} \mid b <_L a\}$. Trivially, $\mathcal{L}C_{\mathrm{IF}}(L) \supseteq L$.

In Fig. 3, we can see the "immediately follows" C-net of the log in Fig. 1 ($\{abce, acbe\}$). It is easy to check that the language of the C-net includes the two sequences of the log and many more.

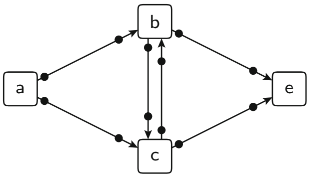


Fig. 3 "Immediately follows" C-net of the log in Fig. 1 ($\{abce, acbe\}$). New traces like *abcbce* are valid in this C-net

A possible lower bound is given by $|A_L| - 1$, which is the minimum number of arcs to guarantee that all the activities in the log are connected, although tighter lower bound can be given in some scenarios (see [12] for the details).

5.3 The algorithm

In Algorithm 4, we give the pseudocode of the proposed approach (notice that it is an instantiation of the general algorithm described in Sect. 4.1). The main idea is to build the structural equations mandatory to any C-net whose language includes a given log L (thus guaranteeing fitness) 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, thus increasing the simplicity of the model. Once the number of arcs cannot be further decreased, a final step removes all redundant bindings.

Algorithm 4 Discover minimal C-net

```
 1: function DISCOVERMINCNET($L$)
 2:     $C = \langle A, a_s, a_e, I, O\rangle \leftarrow C_{\mathrm{IF}}(L)$                ▷ See Sect. 5.2
 3:     $min \leftarrow |A|$
 4:     $max \leftarrow |\text{arcs(C)}| - 1$
 5:     $E_s \leftarrow \text{structural\_equations}(L)$
 6:     while $min \leq max$ do
 7:         $avg \leftarrow \lfloor (min + max)/2 \rfloor$
 8:         $E \leftarrow E_s \wedge \text{bound\_arcs}(L, avg)$               ▷ Add (5)
 9:         $feasible, solutions \leftarrow \text{solve}(E)$          ▷ Call SMT solver
10:         if $feasible$ then
11:             $C \leftarrow \text{extract\_cnet}(solutions)$          ▷ Model feasible
12:             $max \leftarrow |\text{arcs(C)}| - 1$           ▷ Since $|\text{arcs(C)}| \leq avg$
13:         else
14:             $min \leftarrow avg + 1$                       ▷ Model unfeasible
15:         end if
16:     end while
17:     $C \leftarrow \text{binding\_minimization}(C)$                ▷ See Sect. 5.4
18:     return $C$
19: end function
```

To obtain reasonable initial bounds for the binary search, we use the connectivity argument of Sect. 5.2.4 for a lower bound (line 3) requiring at least as many arcs as the number of different activities and the number of arcs in the "immediately follows" C-net for the upper bound (line 4)[4].

[4] Although the minimum number of arcs required to guarantee that all activities are connected is $|A| - 1$, the minimum bound in the algorithm

المنارة للاستشارات
www.manaraa.com

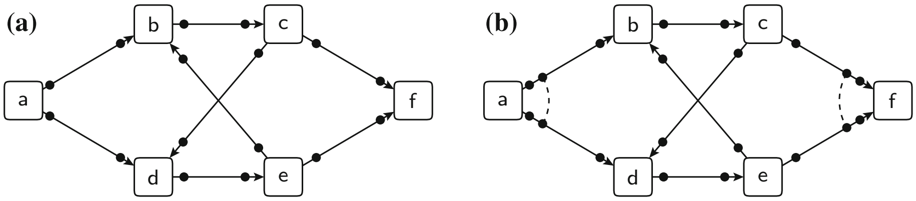


Fig. 4 Redundant bindings in a C-net discovered using an SMT-based approach. The `binding_minimization` algorithm (Sect. 5.4) is able to generate (**a**) starting from (**b**)

The algorithm contains three calls to functions that either are specializations of functions in Algorithm 1 or have not been yet introduced. One is function `solve`(E) which 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 $\mathcal{X}$ and $\mathcal{Y}$ variables in case the problem was feasible. The second function, `extract_cnet`($\mathit{solutions}$), simply builds a C-net from the values of the variables in sets $\mathcal{X}$ and $\mathcal{Y}$ using the principles explained in Theorem 1. Finally, the function `binding_minimization`, explained in Sect. 5.4, removes the largest set of input/output bindings that are redundant. We illustrate with an example how redundant bindings might appear in the C-nets produced before this function is called.

Example 2 Consider the log $L = \{abcf, adef, abcdef, adebcf, abcdebcf\}$ which can be described by the C-net of Fig. 4a. However, a possible output for the SMT-based approach is given in Fig. 4b which has two additional bindings, marked with dashed arcs. In particular, the model in Fig. 4b allows the sequences of activities *bc* and *de* to interleave. This is a valid possibility in the sequences *abcdef* and *adebcf* of L. Since both models have the same number of arcs, the algorithm does not prefer one C-net over the other. In this example, it is clear that the C-net in Fig. 4b can be improved by minimizing the number of bindings in the model.

Since the minimization of bindings is achieved by solving a related but somewhat different SMT problem, we will explain the details of this technique in the next section. The following theorem is the main result of this section:

Theorem 2 *Let C be the C-net returned by Algorithm 4 executed on a log L. The language of C includes L, there is no other C-net including L that has less arcs than C, and C contains no redundant binding.*

Proof In [12], it was already proved that an algorithm equal to Algorithm 4 but without the call to the `binding_minimization` function yields a C-net C' such that the language of C' includes L and there is no other C-net including L that has less arcs than C'. Since the function `binding_minimization` removes all redundant bindings from C', we obtain a C-net C that still satisfies the two previous conditions but contains no redundant binding. □

5.4 C-net replay and binding minimization

Given a particular activity sequence and a C-net, the replay problem is to find the valid binding sequence of the net whose projection is the activity sequence. The problem can be generalized to sets of activity sequences (i.e., a log). This is a relevant problem, since it allows determining whether a particular activity sequence belongs to the language of a C-net, which is a fundamental knowledge to effectively use the model. As we see in this section, the replay problem for C-nets is much more complex than in other models, but can be solved using an SMT approach. Moreover, the SMT equations can be later reused for other interesting applications, like removing redundant bindings. In this section, a brief informal description of C-net replay and binding minimization is provided, which is developed further in [13].

The replay of a log L in a given C-net C can be expressed also as an SMT problem, by using the structural equations of L: In structural_equations(L) some variables are removed, to reflect that some dependencies between activities are no longer possible. This is because they do not appear in C (we call these equations the *skeleton of* C). Additionally, a set of equations is incorporated that restricts the possible assignments of the $\mathcal{X}$ and $\mathcal{Y}$ variables to the set of input and output bindings in C. Although alternative replay methods based on exploring the state space of bindings are possible, this approach has some advantages. First of all, only exponential techniques are known for the problem of C-net replay (although the problem is known to be in NP) [5]. Second, *the replay problem of all the log can be solved in a single SMT problem instance* (which is NP-complete). Finally, the SMT-

Footnote 4 continued
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 $C_{\mathrm{IF}}(L)$, thus $|\mathrm{arcs(C)}| = |A| - 1$ and the algorithm would never enter the loop and return $C_{\mathrm{IF}}(L)$. On the other hand, if $|\mathrm{arcs(C)}| > |A| - 1$, then there is no feasible model with just $|A| - 1$ arcs; thus, the minimum search bound can be set to $|A|$.

[5] One can notice this with the simple example of Fig. 2b: To replay the occurrence of activity a, the three output bindings should be considered as potential successor states, in general to proceed with the replay any of them can be combined with the occurrences of the sequent activities, which in turn may introduce new output binding possibilities.

المنارة للاستشارات
www.manaraa.com

based replay will be the basis to minimize the number of bindings of a C-net.

Formally, given a C-net $C = \langle A, a_s, a_e, I, O\rangle$ and a log L, we denote the skeleton of C as *skeleton*(C, L), which is defined as:

$$\boxed{\text{skeleton}(C, L) \overset{\text{def}}{=} \text{structural\_equations}(L) \wedge \bigwedge\nolimits_{x \in \mathcal{X}_{(a,b)} : (a,b) \notin \text{arcs(C)}} \overline{x} \wedge \bigwedge\nolimits_{y \in \mathcal{Y}_{(a,b)} : (a,b) \notin \text{arcs(C)}} \overline{y}}$$

Basically, in the formula above we set all the variables representing arcs not found in C to false, hence invalidating these arcs to be used in the replay.

Similarly, the equations that restrict the choices of input/output bindings, denoted as *restrict_choices*(C, L), correspond to:

$$\boxed{\text{restrict\_choices}(C, L) \overset{\text{def}}{=} \bigwedge\nolimits_{\sigma \in L} \left(\bigwedge\nolimits_{1 < i \leq |\sigma|} \bigvee\nolimits_{S \in I(\sigma_i)} X_i = S \wedge \bigwedge\nolimits_{1 \leq i < |\sigma|} \bigvee\nolimits_{S \in O(\sigma_i)} Y_i = S\right)}$$

These equations enforce that the input and output bindings can only be the ones present in C-net C. We define the replay SMT problem *replay*(C, L) as:

$$\boxed{\text{replay}(C, L) \overset{\text{def}}{=} \text{skeleton}(C, L) \wedge \text{restrict\_choices}(C, L)}$$

The solution to this SMT problem is the set of values of the $\mathcal{X}$ and $\mathcal{Y}$ variables from which the X_i and Y_i sets can be reconstructed. This means that from each sequence σ in the log, we can obtain a valid binding sequence β of C-net C such that $\text{act}(\beta) = \sigma$.

Theorem 3 ([13]) *The equations replay*(C, L) *have a solution if, and only if, every sequence σ in L is replayable by C.*

In terms of the number of equations required by replay (C, L) consider first the predicate skeleton(C, L). The first approximation would be to consider it as the same asymptotic number of equations as structural_equations(L), i.e.$\mathcal{O}(|L| \cdot |A_L| \cdot \max_{\sigma \in L}(|\sigma|))$ (see Sect. 5.2.3). However, since only the obligations for which an arc exists in C are allowed, the number of equations can be directly reduced, instead of explicitly negating the variables representing arcs not found in C. This reduction contributes to further reduce the amount of variables that each activity generates to the incoming/outgoing arcs that it has on the C-net C. If we denote the set of incoming/outgoing arcs of activity a in C as arcs(C, a), the number of equations in formula skeleton (C, L) is $\mathcal{O}(|L| \cdot \max_{\forall a \in A_L}(|\text{arcs(C, a)}|) \cdot \max_{\sigma \in \text{L}}(|\sigma|))$.

On the other hand, the predicate restrict_choices(C, L) has a set of equalities for each input/output binding. The number of terms required to express this set equality (e.g., $X_i = S$) is restricted by the corresponding arcs in C and thus $\max_{\forall a \in A_L}(|\text{arcs(C, a)}|)$ in the worst case. Given that the number of set equalities depends on the number of input/output bindings, we obtain $\mathcal{O}(|L| \cdot \max_{\sigma \in L}(|\sigma|) \cdot \max_{\forall a \in A_L}(|I(a)| + |O(a)|) \cdot \max_{\forall a \in A_L}(|\text{arcs(C, a)}|))$.

Equation in SMT problem	# Linear equations	# Disjunctive clauses
skeleton(C, L)	$\mathcal{O}(\lvert L\rvert \cdot \max_{\forall a \in A_L}(\lvert\text{arcs (C, a)}\rvert) \cdot \max_{\sigma \in \text{L}}(\lvert\sigma\rvert))$	–
restrict_ choices(C, L)	$\mathcal{O}(\lvert L\rvert \cdot \max_{\sigma \in L}(\lvert\sigma\rvert) \cdot \max_{\forall a \in A_L}(\lvert I(a)\rvert + \lvert O(a)\rvert) \cdot \max_{\forall a \in A_L}(\lvert\text{arcs(C, a)}\rvert))$	$\mathcal{O}(\lvert L\rvert \cdot \max_{\sigma \in L}(\lvert\sigma\rvert) \cdot \max_{\forall a \in A_L}(\lvert I(a)\rvert + \lvert O(a)\rvert) \cdot \max_{\forall a \in A_L}(\lvert\text{arcs(C, a)}\rvert))$

The mechanism by which input and output bindings of a C-net can be minimized is closely related to the SMT-based replay. The basic idea is to build an SMT replay problem in which we add an additional equation, enforcing that at least a given number of bindings are not used during the C-net replay. In other words, given an l, the replay problem then becomes: *is it possible to replay the net without using at least l of its bindings?*. Once we know how to establish this bound on the number of unused bindings, by performing a binary search we can maximize them, thus minimizing the number of required C-net bindings.

Formally, the quantity to maximize, expressed as a pseudo-Boolean formula (in which the logical And of Boolean variables is treated as an integer binary variable that can be added in the summation) is:

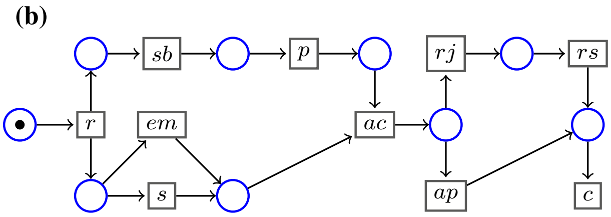
$$\text{unused}(C, L) \overset{\text{def}}{=} \sum_{a \in A} \sum_{S \in (I(a) \cup O(a))} \bigwedge_{\sigma \in L} \bigwedge_{\sigma_i = a} (X_i \neq S \vee Y_i \neq S)$$

المنارة للاستشارات
www.manaraa.com

Fig. 5 Petri net discovery: **a** log, **b** discovered Petri net

(a)

1	r, s, sb, p, ac, ap, c
2	r, sb, em, p, ac, ap, c
3	r, sb, p, em, ac, rj, rs, c
4	r, em, sb, p, ac, ap, c
5	r, sb, s, p, ac, rj, rs, c
6	r, sb,p, s, ac, ap, c
7	r, sb, p, em, ac, ap, c



where the condition $\bigwedge_{\sigma \in L} \bigwedge_{\sigma_i = a} X_i \neq S$ expresses the fact that a particular input binding S of C does not appear in any of the valid binding sequences replayed. The sum of all these conditions (including the symmetrical conditions on the output bindings) gives the number of unused bindings during the replay. Thus, given a (lower) limit l on the number of unused bindings, the SMT problem built is:

$$\boxed{\text{min\_unused}(C, L, l) \overset{\text{def}}{=} \text{replay}(C, L) \wedge (\text{unused}(C, L) \geq l)}$$

To perform a binary search, we must provide a range of possible values for the parameter l. The lower bound of this range is clearly zero, since it is possible that the C-net requires all its bindings. On the other hand, if C contains n bindings, it is possible to give a tighter upper bound than simply n. In particular, any activity that is not the initial nor the final one, must have at least one input and one output bindings, while the initial (final) activity must have at least one output (input) binding. Thus, if C contains $|A|$ activities, this means that at least $(|A| - 2) \cdot 2 + 2$ bindings are required, so $n - 2\,|A| + 2$ is a valid upper bound. This upper bound can be further improved with the information obtained during the creation of the formula unused(C, L) as explained in [13].

5.5 A note on the selection of the SMT domain

The encoding presented in this section can be represented in domains different from SMT. In the algebraic domain, one option is to model Eqs. (1)–(4) from Sect. 5.2.2 in an 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 $\mathcal{X}$ and $\mathcal{Y}$ 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. The approach for minimizing arcs described in Sect. 5.2.4 requires expressions involving logical disjunctions, which poses certain problems for ILP formulations, requiring the introduction of auxiliary variables and additional constraints[6].

In conclusion, SMT solvers provide a higher degree of flexibility than ILP. Moreover, our tests showed that in terms of run time they had a similar or better performance than ILP solvers in our benchmarks.

6 Discovering strategies for Petri nets based on SMT

6.1 Petri nets, transition systems and the theory of regions

In this section, we provide the background necessary to understand the technique for discovery of Petri nets based on SMT, which is presented in Sect. 6.2. A simple example of Petri net discovery is illustrated in Fig. 5. The technique is grounded in the *theory of regions* [22], a theory that appeared in the early nineties to provide a correspondence between an automaton and a Petri net. For the sake of brevity, we have chosen to present only the necessary ingredients of this theory in this section. For a detailed description on how the theory of regions can be applied to derive Petri nets in a general setting, the reader can refer to [23,24].

The starting point of the algorithms presented in this section is an automaton whose language contains all the traces described in a log. This is without loss of generality, since there are linear algorithms to convert a log into an automaton [25]. For instance, we can see a log in Fig. 6 together with two possible transformations into a transition system (a type of automaton, see the formal definition below) that always produce acyclic automata, although more sophisticated techniques exist that can produce more compact (and non necessarily acyclic) transformations [26,27]. The following definition formalizes the type of automata the proposed algorithms consider:

Definition 7 (*Transition system*) A *transition system* (TS) is defined as a tuple $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$, where S is a set of *states*, Σ is an alphabet of *events*, $T \subseteq S \times \Sigma \times S$ is a set of *(labeled) transitions* or *arcs* and $\mathsf{s}_0 \in S$ is the *initial state*.

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

المنارة للاستشارات
www.manaraa.com

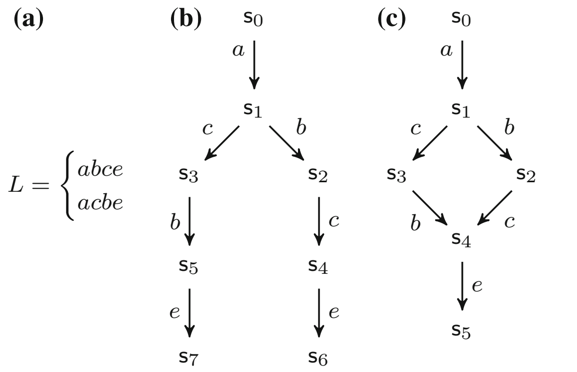


Fig. 6 **a** A log. **b** Transformation of the log into a transition system, merging equal prefixes of sequences. **c** Transformation of the log into a transition system, merging prefixes with the same number of the same activities

We use $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}'$ as a shortcut for $(\mathsf{s}, \mathsf{e}, \mathsf{s}') \in T$, and we denote its transitive closure as $\xrightarrow{*}$. A state s' is said to be *reachable from state* s if $\mathsf{s} \xrightarrow{*} \mathsf{s}'$. We extend the notation to arc sequences, i.e., $\mathsf{s}_1 \xrightarrow{\sigma} \mathsf{s}_{n+1}$ if $\sigma = \mathsf{e}_1 \dots \mathsf{e}_n$ and $\forall_{1 \le i \le n} (\mathsf{s}_i, \mathsf{e}_i, \mathsf{s}_{i+1}) \in T$. The *language* of a TS U, $\mathcal{L}U$, is the set of arc sequences feasible from the initial state.

Definition 8 (*Petri net* [3]) A Petri net (PN) is a tuple (P, T, W, M_0) where the sets P and T represent finite sets of places and transitions, respectively, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the weighted flow relation. The initial marking $M_0 \in \mathbb{N}^P$ defines the initial state of the system.

In a PN N, a transition $t \in T$ is *enabled* in a certain marking M if $\forall p \in P : M(p) \ge W(p, t)$ holds. Firing an enabled transition t in M leads to the marking M' defined by $M'(p) = M(p) - W(p, t) + W(t, p)$, for each $p \in P$, and is denoted by $M \xrightarrow{t} M'$. The set of all markings reachable from the initial marking M_0 is called its Reachability Set. We say that N is k-bounded if, for all reachable marking M and all place p, it holds that $M(p) \le k$. The *Reachability Graph* of N, denoted by $\mathsf{RG}(N)$, is a transition system in which the set of states is the Reachability Set, the events are the transitions of the net and an arc (M_1, t, M_2) exists if and only if $M_1 \xrightarrow{t} M_2$. We use $\mathcal{L}(N)$ as a shortcut for $\mathcal{L}(\mathsf{RG}(N))$.

To obtain a suitable implementation of the `constrainable` function of Algorithm 3 for the Petri nets, we must resort to the theory of regions, which are multisets over the states of a transition system that satisfy some conditions. To specify these conditions, we need the concept of gradient.

Definition 9 (*Gradient in a* TS) Let $\langle S, \Sigma, T, \mathsf{s}_0 \rangle$ be a TS. Given a multiset of states r and an arc $\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T$, its *gradient* is defined as $\delta_r(\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}') = r(\mathsf{s}') - r(\mathsf{s})$. If all the arcs of an event $\mathsf{e} \in \Sigma$ have the same gradient, we say that the event e has *constant gradient*, whose value is denoted as $\delta_r(\mathsf{e})$.

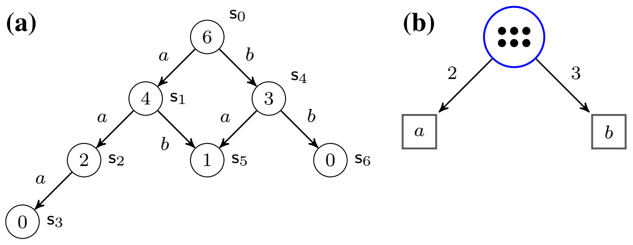


Fig. 7 **a** Transition system for the log $\{aaa, ab, ba, bb\}$ and one of its regions r: $r(\mathsf{s}_0) = 6, r(\mathsf{s}_1) = 4, \dots, r(\mathsf{s}_6) = 0$ in which the gradients of the events are $\delta_r(\mathsf{a}) = -2$ and $\delta_r(\mathsf{b}) = -3$. **b** Corresponding feasible place in the Petri net

Definition 10 (*Region*) A *region* r is a multiset of states defined in a TS, in which all the events have constant gradient.

Figure 7a shows a TS with a multiset defined over the states of the system. The numbers within the states correspond to the multiplicity of the multiset r shown, e.g., $r(\mathsf{s}_0) = 6$. Multiset r is a region because both events a and b have constant gradient, i.e., $\delta_r(\mathsf{a}) = -2$ and $\delta_r(\mathsf{b}) = -3$. It is easy to check that these gradients are constant: For instance, every arc of event a has a difference of -2 between the target state and the source state of every arc, e.g., $r(\mathsf{s}_1) - r(\mathsf{s}_0) = 4 - 6 = -2$, $r(\mathsf{s}_5) - r(\mathsf{s}_4) = 1 - 3 = -2$, etc.

There is a direct correspondence between regions and the *feasible places* of a PN with respect to the language of a TS.

Definition 11 (*Feasible place*) Given a TS $U = \langle S, \Sigma, T, \mathsf{s}_0 \rangle$ and a PN $N_p = \langle \{p\}, \Sigma, W, M_0 \rangle$, we say that place p is *feasible* (w.r.t. $\mathcal{L}(U)$) if $\mathcal{L}(N_p) \supseteq \mathcal{L}(U)$. Every region r of U corresponds to a feasible place p such that $M_0(p) = r(\mathsf{s}_0)$, and $W(p, \mathsf{e}) = -\delta_r(\mathsf{e})$ if $\delta_r(\mathsf{e}) < 0$, and $W(\mathsf{e}, p) = \delta_r(\mathsf{e})$ otherwise.

The gradient of the region describes the flow relation of the corresponding place, and the multiplicity of the initial state indicates the number of initial tokens [24]. Figure 7b shows the feasible place corresponding to the region shown in Fig. 7a.

Intuitively, feasible places are places that can be used to construct a PN that will always include the language of a TS U, since the addition of a feasible place p with language $\mathcal{L}(N_p)$ to a PN N yields a net N' such that $\mathcal{L}(N') = \mathcal{L}(N) \cap \mathcal{L}(N_p)$, and all feasible places p satisfy $\mathcal{L}(N_p) \supseteq \mathcal{L}(U)$.

6.2 Discovery of Petri nets

Although there can be large number of regions in a given transition system (in the worst case, an exponential number with respect to the set of states of the TS), it was proved in [28] that the set of minimal regions is enough to constrain the language of the net as much as possible while including

المنارة للاستشارات
www.manaraa.com

the language of the transition system. For this reason, minimal regions are relevant in Petri net discovery for obtaining precise models. Formally, a *minimal region* r is a region such that, for all regions r', $r' \subseteq r \Rightarrow r' = \mathbf{0}$, where $\mathbf{0}$ denotes the 0-bounded region. That is, it does not have a proper subregion different than the 0-bounded region. This section focuses on how to instruct SMT to compute the set of minimal regions of a given TS.

Given a TS $U = \langle S, \Sigma, T, \mathsf{s}_0\rangle$, a k-bounded region r of U can be encoded in SMT as follows: $\forall \mathsf{s} \in S$, variable m_s represents the value of $r(\mathsf{s})$. Similarly, the gradients of r are encoded using a variable δ_e for each event $\mathsf{e} \in \Sigma$.

The set of equations describing all the k-bounded regions of a TS $U = \langle S, \Sigma, T, \mathsf{s}_0\rangle$, denoted by *k_bounded_regions* (U, k), can be further encoded in SMT:

- $\forall \mathsf{s} \in S,\ m_s \leq k$, i.e., all multiplicities are k-bounded, and
- $\forall \mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T,\ \delta_e = m_{s'} - m_s$, i.e., all gradients are constant.

$$\boxed{\text{k\_bounded\_regions}(U, k) \overset{\text{def}}{=} \bigwedge_{\mathsf{s} \in S} m_s \leq k \wedge \bigwedge_{\mathsf{s} \xrightarrow{\mathsf{e}} \mathsf{s}' \in T} \delta_e = m_{s'} - m_s} \qquad (6)$$

As we have seen in the general algorithm (Algorithm 3), we need a way to compute new elements (regions in this case) at each iteration of the algorithm (general method `new_element` in Algorithm 3). In our case, given that we are only interested in minimal regions, non-minimal regions need to be discarded. We can prune non-minimal regions with the help of two formulas: *proper_subregion*(r), which guarantees that the region found is a proper subregion of r and $\neg$*superregions*(R) which prevents finding a region that includes ($\subseteq$) any of the regions in a set R (this set will contain the regions currently in the model).

The set of equations describing the regions that are proper subregions of a region r, denoted by *proper_subregion*(r), is:

- $\bigwedge_{\mathsf{s} \in S} m_s \leq r(\mathsf{s})$, i.e., all multiplicities are smaller than or equal to the ones in r, and
- $\bigvee_{\mathsf{s} \in S} m_s < r(\mathsf{s})$, i.e., at least one multiplicity is strictly smaller.

$$\boxed{\text{proper\_subregion}(r) \overset{\text{def}}{=} \bigwedge_{\mathsf{s} \in S} m_s \leq r(\mathsf{s}) \wedge \bigvee_{\mathsf{s} \in S} m_s < r(\mathsf{s})} \qquad (7)$$

Conversely, the equations describing the regions that are superregions of a region r, denoted by *superregion*(r) are $\bigwedge_{\mathsf{s} \in S} m_s \geq r(\mathsf{s})$, i.e., all multiplicities are greater than or equal to the multiplicities in r.

$$\boxed{\text{superregion}(r) \overset{\text{def}}{=} \bigwedge_{\mathsf{s} \in S} m_s \geq r(\mathsf{s})} \qquad (8)$$

Given a set R of regions, it is easy to generate the equations describing the regions that are superregions of at least one region in R as $\bigvee_{r \in M}$ superregion(r). We denote this set of equations as *superregions*(R).

$$\boxed{\text{superregions}(R) \overset{\text{def}}{=} \bigvee_{r \in R} \text{superregion}(r)} \qquad (9)$$

With all the previous equations, it is possible to devise an algorithm (see Algorithm 5) to find all minimal regions of a transition system U (thus generating the most restrictive Petri net, using the k-bounded regions in the transition system, that contains the language of U) [28]. With respect to the general algorithm (Algorithm 3), one should notice that Algorithm 5 is computing a finite set of regions R which will be then translated to a Petri net using a standard construction from [28], shown at the end of this section. Algorithm 5 contains some optimizations to increase the efficiency of the approach by avoiding visiting particular solutions of the region space more than once. In any case, still several parts can be identified with respect to the general Algorithm 3:

- The `structural_equations` method is split into lines 5, 8 and 11 where the combined characterization of regions is conducted.
- An emptiness test to the stack of regions s in Algorithm 5 implements the `constrainable` generic method, since only when no more regions exist in s we can certify that the model derived cannot be constrained further.
- The `extract_element` function has been renamed to `extract_region`, since we obtain a region r from the solutions of a feasible SMT problem where for all states s we have $r(\mathsf{s}) = m_s$.

The algorithm receives two parameters, the transition system U and the desired region bound. The main idea is to explore the region space from the largest region (region $\mathbf{k}$, a multiset[7] in which all the multiplicities are equal to k) finding each time a proper subregion of the last region found

[7] This is indeed a region, since the gradient of every event is constant and equal to zero.

المنارة للاستشارات
www.manaraa.com

Algorithm 5 Discover all minimal k-bounded regions of TSU

```
1: function DISCOVERMINIMALREGIONS($U, k$)
2:     $R \leftarrow \emptyset$                                  ▷ Set of minimal regions
3:     $s \leftarrow$ empty_stack()                     ▷ Stack of regions to explore
4:     push($s$, $\mathbf{k}$)                    ▷ Start with the largest $k$-bounded region
5:     $E_1 \leftarrow$ k_bounded_regions($U, k$)       ▷ Equations describing all
           $k$-bounded regions of $U$
6:     while ¬empty($s$) do
7:         $r \leftarrow$ top($s$)
8:         $E_2 \leftarrow$ ¬superregions($R$)              ▷ Eqs. forbidding all
               superregions of regions in $R$
9:         $region\_found \leftarrow False$
10:        repeat
11:            $E_3 \leftarrow$ proper_subregion($r$)        ▷ Eqs. describing all
                   proper subregions of $r$
12:            $feasible, solutions \leftarrow$ solve($E_1 \wedge E_2 \wedge E_3$)   ▷ Call
                   SMT solver
13:            if $feasible$ then
14:                $r \leftarrow$ extract_region($solutions$)    ▷ Model feasible
15:                push($s, r$)
16:                $region\_found \leftarrow True$
17:            end if
18:        until ¬$feasible$
19:        pop($s$)
20:        if $region\_found$ then
21:            $R \leftarrow R \cup \{r\}$                      ▷ Add minimal region $r$
22:        end if
23:    end while
24:    return $R$
25: end function
```

that is not a superregion of the minimal regions found so far (the ones in R). Once this descending chain cannot be further continued, a minimal k-bounded region of U has been found, which is added to the set R. Then the algorithm backtracks to the previous region and asks for a proper subregion that is not a superregion of any of the regions in the updated set R. Once again when this chain of regions is exhausted we have found another minimal region of U. This iteration in lines 10–18 from Algorithm 5 is not explicit in Algorithm 3 and should be understood as a necessary minimality test preceding the method `extract_region`[8]. This process continues until all minimal regions have been found, as next theorem states.

Theorem 4 *The set R returned by Algorithm 5 executed on a TS U contains all the k-bounded minimal regions of U.*

Proof Assume R does not contain some k-bounded minimal region r of U. Since r is k-bounded, it must satisfy the equations in E_1. Thus, if r was discarded by Algorithm 5, it must be because either it did not satisfy E_2, or it has a proper subregion. In the former case, then it is a superregion of at least one region in R; thus, r is not minimal. In the latter case, since the algorithm iterates until $\neg feasible$ is true, a contradiction is reached. □

We illustrate how this algorithm operates with the following example:

Example 3 Consider the TS shown in Fig. 8b. If we use Algorithm 5 to compute its 1-bounded minimal regions, the first steps of the sequence of SMT problems solved are shown in Fig. 9. Since in this case $k = 1$, the algorithm starts with region **1**, denoted as r_a. Smaller subregions are then found (r_b and r_c) until at step 3 the problem becomes unfeasible. At this point, r_c is known to be a minimal region (note this corresponds to region r_0 in Fig. 8b) and thus added to R. Then the algorithm backtracks to the last step that yield a satisfiable problem (step 1) and asks for a subregion of r_b that is not a superregion of r_c. This produces region r_d which is in fact minimal (corresponds to region r_1 in Fig. 8b). The process will continue until the last minimal region, r_2 in Fig. 8b, is found and the SMT problem built from the root node (step 0) becomes unfeasible. An example of the SMT problem to solve to obtain step 4 is given in Table 2.

Since after the execution of the algorithm, R contains all the minimal k-bounded regions, a PN built using these regions (with Algorithm 6) will yield a PN such that no other net built from k-bounded regions can have a smaller language [28].

In terms of complexity of the SMT problem solved in each iteration, notice that the equations describing regions should contain at least one variable per state in the transition system and should also encode the gradients for each event. Next table shows a detailed count for each one of the predicates used in Algorithm 5:

Equation in SMT problem	# Linear equations	# Disjunctive clauses
k_bounded_regions(U, k)	$\vert S\vert + \vert T\vert$	–
proper_subregion(r)	$2\vert S\vert$	1
¬supperregion(R)	$\vert R\vert \cdot \vert S\vert$	$\vert R\vert$

Notice that apart from the complexity of the encoding shown above, the worst-case complexity of Algorithm 5 is exponential in the number of states $|S|$, due to the exploration in the space of k-bounded multisets of S.

Finally, once the set of k-bounded minimal regions are computed with Algorithm 5, the Petri net can be derived. Algorithm 6 shows how to build a PN that includes the language of a TS given a set of regions R of the transition system.

[8] Testing minimality of model elements is a feature not considered in Algorithm 3, and this is the reason why the generic algorithm (Algorithm 3) and the instantiation (Algorithm 5) have a different structure.

المنارة للاستشارات
www.manaraa.com

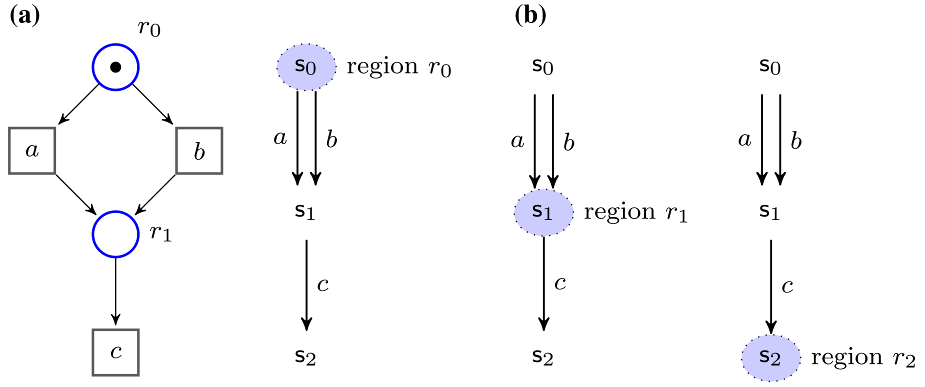


Fig. 8 **a** A PN. **b** Its reachability graph (three copies), in which three regions, which form the set of minimal regions, have been *shaded* (e.g., region r_0 is $r_0(s_0) = 1, r_0(s_1) = 0, r_0(s_2) = 0$)

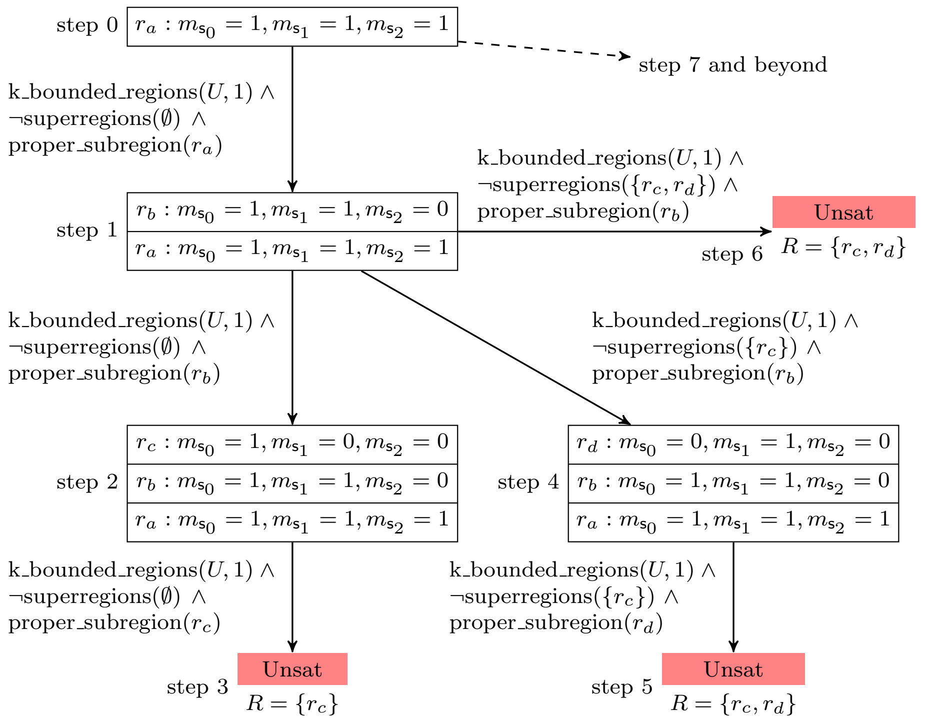

step 0
$r_a : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 1$
step 7 and beyond
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\emptyset) \wedge$
proper_subregion(r_a)
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\{r_c, r_d\}) \wedge$
proper_subregion(r_b)
step 1
$r_b : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 0$
$r_a : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 1$
Unsat
step 6
$R = \{r_c, r_d\}$
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\emptyset) \wedge$
proper_subregion(r_b)
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\{r_c\}) \wedge$
proper_subregion(r_b)
step 2
$r_c : m_{s_0} = 1, m_{s_1} = 0, m_{s_2} = 0$
$r_b : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 0$
$r_a : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 1$
step 4
$r_d : m_{s_0} = 0, m_{s_1} = 1, m_{s_2} = 0$
$r_b : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 0$
$r_a : m_{s_0} = 1, m_{s_1} = 1, m_{s_2} = 1$
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\emptyset) \wedge$
proper_subregion(r_c)
k_bounded_regions$(U, 1) \wedge$
¬superregions$(\{r_c\}) \wedge$
proper_subregion(r_d)
step 3
Unsat
$R = \{r_c\}$
step 5
Unsat
$R = \{r_c, r_d\}$

Fig. 9 Several steps of the execution of Algorithm 5 on the TS U of Fig. 8b with $k = 1$. Each step shows the contents of the stack s if the SMT problem was feasible; otherwise, it shows the set of minimal regions found so far. The label on each arc indicates which SMT problem is solved in each case

Table 2 Equations used to obtain step 4 in Fig. 9

	Equations
k_bounded_regions$(U, 1)$	$m_{s_0} \le 1 \wedge m_{s_1} \le 1 \wedge m_{s_2} \le 1 \wedge$ $\delta_a = m_{s_1} - m_{s_0} \wedge \delta_b = m_{s_1} - m_{s_0} \wedge \delta_c = m_{s_2} - m_{s_1}$
¬superregions$(\{r_c\})$	$\neg(m_{s_0} \ge 1 \wedge m_{s_1} \ge 0 \wedge m_{s_2} \ge 0)$
proper_subregion(r_d)	$m_{s_0} \le 1 \wedge m_{s_1} \le 1 \wedge m_{s_2} \le 0 \wedge$ $(m_{s_0} < 1 \vee m_{s_1} < 1 \vee m_{s_2} < 0)$

7 Experiments

We have implemented Algorithms 4 and 5 in a prototype tool that uses the STP solver [29] as the underlying SMT solver[9].

In this section, we evaluate the capacity of these algorithms to derive valuable models from several logs. For the particular case of Algorithm 4 in which only pseudo-Boolean formulas appear, we have also compared with the CLASP solver [30]. Note that in this latter case the solver admits a cost function to minimize and thus no binary search is required to minimize the number of arcs.

[9] STP translates the SMT formula to a SAT formula and then uses the miniSAT solver, but any other SAT (or incremental SAT) tool can be used as backend.

المنارة للاستشارات

www.manaraa.com

Algorithm 6 Build a PNfrom a set regions R of TSU [28]

```
function DISCOVERPNFROMREGIONS($U = \langle S, \Sigma, T, s_0\rangle$, $R$)
    $W \leftarrow \emptyset$                          ▷ Initially empty flow relation
    for $r \in R$ do
        for $t \in \Sigma$ do
            if $\delta_r(t) < 0$ then
                $W(r, t) \leftarrow -\delta_r(t)$
                $W(t, r) \leftarrow 0$
            else
                $W(r, t) \leftarrow 0$
                $W(t, r) \leftarrow \delta_r(t)$
            end if
        end for
        $M_0(r) \leftarrow r(s_0)$
    end for
    return $\langle R, \Sigma, W, M_0\rangle$ ▷ Each region is a place, each event is a
        transition
end function
```

Table 3 shows the results for Algorithm 4 on some small log examples from [12] with the following information:

- $|L|$ is the number of distinct sequences in the log,
- $|\sigma_m|$ is the length of the largest sequence,
- $|A_L|$ is the size of the alphabet of activities,
- *arcs* is the number of arcs of the final C-net,
- *time* is the elapsed time (in seconds) required to complete the discovery process using STP,
- *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,
- $|\mathcal{X} \cup \mathcal{Y}|$ is the number of Boolean variables used to encode the SMT problem,
- $|E|$ is the number of linear 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,
- column *heur* indicates if some of the heuristics defined in [12] 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,
- column *CLASP* shows the time required by the CLASP solver. In some cases this solver, which only accepts pseudo-Boolean formulas, could not be used because some of the formulas were not pseudo-Boolean. This is indicated by a "–" in the table.

The results on these small benchmarks show that the approach is, in general, able to derive valuable C-nets. It also shows that the techniques presented in this paper should be optimized in order to be able to deal with large inputs, as we will report below. Algorithm 4 always generates C-nets whose language contains the given log (fitness=1.0); moreover, it also 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 original one, but without an arc; on the other hand, the largest benchmark in this table (a22f0n00_1 from [31]) could not be discovered within the one hour limit used in our experiments. In addition to the number of variables and equations, other factors may contribute for a solver to find a solution fast: the asymmetry between deciding whereas a formula satisfiable or unsatisfiable, the dependence to the number of arcs in the solution, among others. This should be explored in future work to instruct better the usage of SMT solvers.

To be able to process larger benchmarks, we have to resort to a simplification heuristic (limiting the obligation alphabet [12]) that only allows arcs between activities that appear in some sequence of the log at most a distance d, where d is a parameter called the size of the activity window. Table 4 shows the results for our previous benchmarks as well as some larger examples also from [31]. In this case, we have not used any other heuristic. Despite this fact, the original models were discovered in all cases, but one benchmark. The aalst2b benchmark is a difficult one for an arc minimizing strategy, since the model contains one arc more than the minimum number of arcs to include the language of the log. For this example, we had the more complex Algorithm 2, which was capable of deriving the correct net in about 5 s.

In terms of comparisons between STP and CLASP, the latter is an order of magnitude faster in general, except in the largest benchmark. This suggests that if the problem can be kept in the pseudo-Boolean domain, it is usually a better strategy to use a specific solver for this class than a more general solver.

To test the validity of our approach for Petri net discovery (Sect. 6), we have compared Algorithm 5 with a state-of-the-art discovery algorithm grounded also on the theory of regions but using ILP techniques. In particular, we have compared with the ILP plug-in in the ProM suite that uses the Jsolve library for ILP solving. Since Algorithm 5 works on a transition system and we initially have a log, a transformation must be performed to use the algorithm. In this case, we have used the *common final marking* transformation of [26].

Table 5 shows the results for both tools. Column *P/F* indicates the number of places and the number of arcs of the resulting net, respectively. Column *time* was the running time to obtain the net and ETC is a metric [32] describing the precision of the net. This metric quantifies how much behavior not seen in the log is present in the model, where value 1.0 indicates that no additional behavior is present, while values near 0.0 show that the model included lots of additional behavior. The ETC metric is computed by counting model deviations with respect to the log behavior, i.e., for each reachable state s in the model that is reached by a sequence σ that is a prefix of some trace in the log, the ratio $\frac{|obs(\sigma)|}{|allow(s)|}$ is computed,

www.manaraa.com

Table 3 Results of discovery algorithm on small examples

Benchmark	$\|L\|$	$\|\sigma_m\|$	$\|A_L\|$	$\|\mathcal{X} \cup \mathcal{Y}\|$	$\|E\|$	bounds	arcs	it	time	heur	id	CLASP
aalst1 (Fig. 2b)	10	5	5	156	147	[6, 11]	6	2	0.3	–	y	0.0
aalst2b	8	11	5	156	147	[5, 9]	6	3	0.2	–	n	0.0
mixedXorAnd	3	14	7	219	162	[7, 11]	8	3	0.2	–	y	0.0
a12f0n00_5	5	7	12	176	143	[12, 17]	14	3	0.1	f	y	–
optional1	11	8	6	413	264	[6, 10]	9	2	0.1	f,o	y	–
cycles	7	18	8	839	542	[8, 17]	9	3	1.3	f,i	y	–
a22f0n00_1	99	46	22	28,898	18,942	[22, 166]	$\leq$39	$\geq$4	>1 h	–	n	>1 h

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

Benchmark	$\|L\|$	$\|\sigma_m\|$	$\|A_L\|$	$\|\mathcal{X} \cup \mathcal{Y}\|$	$\|E\|$	bounds	arcs	it	time	id	CLASP
aalst1	10	5	5	136	137	[6, 11]	6	2	0.0	y	0.0
aalst2b	8	11	5	240	246	[5, 9]	6	3	0.1	n	0.0
mixedXorAnd	3	14	7	89	98	[7, 11]	8	3	0.2	y	0.0
a12f0n00_5	5	7	12	72	91	[12, 17]	14	3	0.1	y	0.0
optional1	11	8	6	229	220	[6, 10]	9	2	0.1	y	0.0
cycles	7	18	8	265	288	[8, 17]	9	3	0.2	y	0.0
a22f0n00_1	99	46	22	12,827	10,369	[22, 166]	34	7	10.4	y	0.8
a22f0n00_5	836	76	22	12,1281	97,429	[22, 183]	34	7	284.9	y	21.7
a32f0n00_1	100	73	32	26,378	19,049	[32, 362]	46	8	36.9	y	3.8
a42f0n00_1	100	58	42	48,432	31,815	[42, 735]	62	9	251.9	y	309.7

Table 5 Results of the PN discovery algorithm

Log	$\|L\|$	$\|\sigma_m\|$	$\|A_L\|$	`ILP`			Algorithm 5		
				P/F	time	ETC	P/F	time	ETC
a32_1	100	73	32	31/73	25	0.52	32/75	26	0.52
a32_5	900	102	32	31/73	112	0.59	31/73	35	0.59
t32_1	200	360	33	30/72	288	0.37	31/74	63	0.37
t32_5	1800	379	33	30/72	9208	0.39	30/72	84	0.39
a42_1	100	58	42	44/109	154	0.35	52/134	175	0.37
a42_5	900	78	42	44/101	1557	0.41	46/107	721	0.41

where $obs(\sigma)$ ($allow(s)$) is the set of activities observed in the log after σ (allowed in the model at state s).

8 Related work

Process discovery is a vivid area, which has produced several techniques in the last decade. In this section, we focus on the related work for the particular models that we are considering in this paper: C-nets and Petri nets. The reader can find a complete overview of process discovery in [2].

Being a rather novel formalism, there is only one approach in the literature for the discovery of C-nets: the *flexible heuristics miner* [33]. This technique is a light-weight method that additionally can deal with noise. However, an artifact of this is that the derivation of fitting models is not guaranteed, which may represent a problem in several contexts.

In the case of Petri nets, the first algorithm for process discovery in the literature was the α-algorithm [21], which is based on detecting ordering relations in the log. Although having low complexity, the α-algorithm can only discover a very restricted class of behaviors. To surpass this limitation, several extensions have been presented in the literature [34–36]. In general, these techniques are not general enough for capturing all necessary constructs in process models [37].

A recent technique that is guided toward the discovery of block-structured models and has low complexity has been presented in [38]. However, this technique is guided toward

المنارة للاستشارات
www.manaraa.com

a particular class of Petri nets (workflow and sound), describing a very restricted type of behaviors.

Evolutionary approaches have been also proposed to derive unrestricted [39] or block-structured [40] Petri nets. However, evolutionary methods unfortunately can have problems in dealing with inputs of medium/large size.

Finally, approaches closer to our work are grounded on the *theory of regions* [22], to tackle the process discovery problem as we do in Sect. 6: The works [24,31,41,42] can discover unrestricted models but may have difficulties in handling large specifications similarly to the techniques proposed in this paper. The techniques of this paper show promising results, as reported in the previous section, where a comparison with the techniques in [31] is provided.

In summary, techniques for discovery in the literature can be split into light-weight and complex. The technique of this paper falls in the latter class. The advantages of our technique stem from the fact that by casting the problem as an optimization instance, different versions can be obtained by manipulating the constraints/cost function. This distinguishing feature may be explored to further improve the search for solutions.

9 Conclusions, open problems and extensions

This paper has presented general algorithms to discover process models from logs, for two families of processes: additive and restrictive models. Two relevant representatives of these families have been used to develop this general framework, and the experimental results show that SMT techniques can be competitive with respect to other state-of-the-art discovery techniques. However, we believe there is much room for improvement given that the discovery algorithms presented did only use off-the-shelf SMT solvers, but most of the equations were unchanged between different runs of the solver. Clearly we would benefit from solvers capable of presolving sets of equations or incrementally add/remove equations without having to recompute everything from scratch.

On the other hand, some requirements could be relaxed: For instance, it might be interesting to instruct the SMT solver to satisfy as many structural equations (that guarantee fitness) as possible but allowing some of them not to be satisfied (so the overall problem is unsatisfiable) in order to achieve better results in the cost functions. That is sacrificing fitness to obtain a simpler model, for instance. In this regard, techniques like Max-SAT [43] might prove a valuable option.

Acknowledgements This work has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under Grant COMMAS (Ref. TIN2013-46181-C2-1-R).

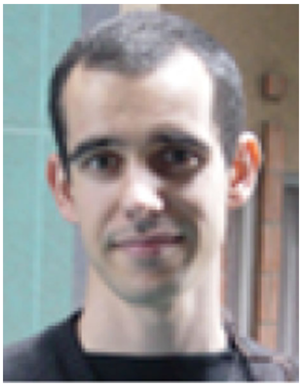
References

1. van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, R.P.J.C., van den Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B.F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D.R., Gaaloul, W., van Geffen, F., Goel, S., Günther, C., Guzzo, A., Harmon, P., ter Hofstede, A., Hoogland, J., Ingvaldsen, J.E., Kato, K., Kuhn, R., Kumar, A., La Rosa, M., Maggi, F., Malerba, D., Mans, R.S., Manuel, A., McCreesh, M., Mello, P., Mendling, J., Montali, M., Motahari-Nezhad, H.R., zur Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Pérez, H.S., Pérez, R.S., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., Wynn, M.: IEEE task force on process mining: process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops (1), Lecture Notes in Business Information Processing, vol. 99, pp. 169–194. Springer, New York (2011)
2. van der Aalst, W.M.P.: Process Mining—Discovery, Conformance and Enhancement of Business Processes. Springer, New York (2011)
3. Murata, T.: Petri Nets: properties, analysis and applications. In: Proceedings of the IEEE, pp. 541–580 (1989)
4. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). J. ACM **53**(6), 937–977 (2006)
5. Srivastava, S., Gulwani, S., Foster, J.S.: VS3: SMT solvers for program verification. In: Bouajjani, A., Maler, O. (eds.) CAV, Lecture Notes in Computer Science, vol. 5643, pp. 702–708. Springer, New York (2009)
6. Tillmann, N., Schulte, W.: Unit tests reloaded: parameterized unit testing with symbolic execution. IEEE Softw. **23**(4), 38–47 (2006)
7. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS, Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer, New York (2008)
8. Metzner, A., Fränzle, M., Herde, C., Stierand, I.: Scheduling distributed real-time systems by satisfiability checking. In: Ng, J.K., Sha, L., Lee, V.C.S., Takashio, K., Ryu, M., Ni, L. (eds.) RTCSA, pp. 409–415. IEEE Computer Society, Los Alamitos (2005)
9. Wolfman, S.A., Weld, D.S.: The LPSAT engine & its application to resource planning. In: Dean, T. (ed.) IJCAI, pp. 310–317. Morgan Kaufmann, Los Altos (1999)
10. de Moura, L.M., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Commun. ACM **54**(9), 69–77 (2011)
11. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Causal nets: a modeling language tailored towards process discovery. In: Katoen, J-.P., König, B. (eds.) CONCUR 2011 – Concurrency Theory, Lecture Notes in Computer Science, vol. 6901, pp. 28–42. Springer, Berlin Heidelberg (2011) CONCUR, pp. 28–42 (2011)
12. Solé, M., Carmona, J.: An SMT-based discovery algorithm for C-nets. In: Petri Nets, LNCS, vol. 7347, pp. 51–71 (2012)
13. Solé, M., Carmona, J.: Amending C-net discovery algorithms. In: S.Y. Shin, J.C. Maldonado (eds.) Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18–22, 2013, pp. 1418–1425. ACM (2013). doi:10.1145/2480362.2480628. http://doi.acm.org/10.1145/2480362.2480628
14. Bose, R.P.J.C., van der Aalst, W.M.P.: Analysis of patient treatment procedures. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops (1), Lecture Notes in Business

www.manaraa.com

Information Processing, vol. 99, pp. 165–166. Springer, New York (2011)

15. R.S, Mans, Schonenberg, H., Song, M., van der Aalst, W.M.P., Bakker, P.J.M.: Application of process mining in healthcare—a case study in a Dutch hospital. In: Fred, A.L.N., Filipe, J., Gamboa, H. (eds.) BIOSTEC (Selected Papers), Communications in Computer and Information Science, vol. 25, pp. 425–438. Springer, New York (2008)
16. van der Aalst, W.M.P., Verbeek, H.M.W.E.: Process mining in web services: the websphere case. IEEE Data Eng. Bull. **31**(3), 45–48 (2008)
17. Rozinat, A., de Jong, I.S.M., Günther, C.W., van der Aalst, W.M.P.: Process mining applied to the test process of wafer scanners in ASML. IEEE Trans. Syst. Man Cybern. Part C **39**(4), 474–479 (2009)
18. van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Verdonk, M.: Auditing 2.0: using process mining to support tomorrow's auditor. IEEE Comput. **43**(3), 90–93 (2010)
19. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Coop. Inf. Syst. 23(1) (2014). doi:10.1142/S0218843014400012
20. Jha, S., Limaye, R., Seshia, S.: Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, pp. 668–674. Springer, New York (2009)
21. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE TKDE **16**(9), 1128–1142 (2004)
22. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-structures. Part I, II. Acta Inf. **27**, 315–368 (1990)
23. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) Petri Nets, LNCS **1491**, pp. 529–586. Springer, New York (1998)
24. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Trans. Comput. **59**(3), 371–384 (2010)
25. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B., Kindler, E., Günther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2009)
26. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) Petri Nets, LNCS **6128**, pp. 226–245. Springer, New York (2010)
27. Solé, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1), 192–205 (2013). doi:10.1109/TKDE.2011.192
28. Bernardinello, L.: Synthesis of net systems. In: Marsan, M.A. (ed.) Application and Theory of Petri Nets, LNCS, vol. 691, pp. 89–105. Springer, New York (1993)
29. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, pp. 524–536. Springer, New York (2007)
30. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: a conflict-driven answer set solver. In: Baral, C., Brewka, G., Schlipf, J.S. (eds.) LPNMR, Lecture Notes in Computer Science. Lecture Notes in Computer Science, vol. 4483, pp. 260–265. Springer, New York (2007)
31. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) ATPN, pp. 368–387. Springer, New York (2008)
32. Munoz-Gama, J., Carmona, J.: A Fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) Business Process Management (BPM). Springer, New York. (2010)
33. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Chawla, N., King,I., Sperduti, A. (eds.) CIDM, pp. 310–317. IEEE, Los Alamitos (2011)
34. Guo, Q., Wen, L., Wang, J., Yan, Z., Yu, P.S.: Mining invisible tasks in non-free-choice constructs. In: Proceedings of Business Process Management—13th International Conference, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, pp. 109–125 (2015)
35. de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Workflow mining: current status and future directions. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS/DOA/ODBASE, pp. 389–406. Springer, New York (2003)
36. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2), 145–180 (2007)
37. van Dongen, B.F., de Medeiros, A.K.A., Wen, L.: Process mining: overview and outlook of petri net discovery algorithms. Trans. Petri Nets Other Models of Concurr. **2**, 225–242 (2009)
38. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs—a constructive approach. In: Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24–28, 2013. Proceedings, pp. 311–329 (2013)
39. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: Ciardo, G., Darondeau, P. (eds.) CATPN, LNCS, vol. 3536, pp. 48–69. Springer, New York (2005)
40. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10–15, 2012, pp. 1–8 (2012). doi:10.1109/CEC.2012.6256458
41. Bergenthum, R., Desel, J., Lorenz, R., S.Mauser: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management, pp. 375–383. Springer, New York (2007)
42. Solé, M., Carmona, J.: Light region-based techniques for process discovery. Fundam. Inform. **113**(3–4), 343–376 (2011)
43. Argelich, J., Manyà, F.: Exact max-sat solvers for over-constrained problems. J. Heuristics **12**, 375–392 (2006)



Marc Solé holds a Ph.D. (2009) in Computer Science by the Technical University of Catalonia (UPC), where he is also an Assistant Professor in the Computer Science Faculty of Barcelona. His Ph.D. thesis was focused on automated formal verification of timed concurrent systems. His research interests include formal verification, process mining, data anonymization and UAS.

المنارة للاستشارات
www.manaraa.com

Josep Carmona received the MS and Ph.D. degrees in Computer Science from the Technical University of Catalonia, in 1999 and 2004, respectively. He is an Associate Professor in the Software Department of the same university. His research interests include formal methods, concurrent systems, and process and data mining. He has co-authored more than 50 research papers in conferences and journals. In 2009, he received the best paper award at the International Conference on Application of Concurrency to System Design.

Springer
للاستشارات المنارة
www.manaraa.com

Software & Systems Modeling is a copyright of Springer, 2018. All Rights Reserved.

المنارة للاستشارات

www.manaraa.com