Proceedings of the Federated Conference on Computer Science and Information Systems pp. 445–453
DOI: 10.15439/2018F131
ISSN 2300-5963 ACSIS, Vol. 15

Generation of Synthetic Business Process Traces using Constraint Programming

Piotr Wiśniewski, Krzysztof Kluza, Antoni Ligęza
AGH University of Science and Technology
al. A. Mickiewicza 30, 30-059 Krakow, Poland
E-mail: {wpiotr,kluza,ligeza}@agh.edu.pl

Anna Suchenia
Cracow University of Technology
ul. Warszawska 24, 31-155 Kraków, Poland
Email: asuchenia@pk.edu.pl

***Abstract*—Juxtapositioning manually created business process models with diagrams generated using process discovery algorithms exposes high complexity of the latter. As a consequence, their formal verification requires significant computational resources due to a large state space. Nevertheless, an analysis of the generated model is needed to assure its correctness and the ability to represent source data. As a solution to this problem, we present an approach for constraint-based generation of a complete workflow log for a given BPMN model. In this paper, we propose a method to extract directed subgraphs representing token flows in the process together with a set of predefined constraints. Likewise, in the case of process simulation, these constraints ensure the correctness of the generated traces. Ultimately, the obtained results can be compared to the original workflow log used for process discovery in order to verify the obtained model.**

***Index Terms*—business process management, process verification, workflow logs, constraint programming**

I. Introduction

BUSINESS process models aim to represent knowledge about workflows which take place in an organization. Such chains of different activities are represented by graphical diagrams that hold information about their dependencies, execution conditions, alternative flows and other properties included in the used modeling notation. Creation of a process model may be performed manually by a process designer who builds the diagram in a graphical editor or prepares other representations such as UML activity diagram [1], structured text [2], natural text description [3] or a spreadsheet representation [4]. Another way to design a model is to discover it from event logs generated by existing IT systems [5], [6].

Although process mining appears to be a convenient technique which does not require much effort in the phase of process modeling, the discovered workflow models can suffer from various defects. Such flaws can be caused either by the imperfection of the selected algorithm [7] or by corrupted logs [8]. Raw data recorded from real system may be characterized by missing events, imprecise or incorrect data, as well as irrelevant artifacts.

Several methods were developed to evaluate discovered models. They include simulations performed on a generated Petri Net which represents the workflow [9], application of evaluation metrics [10], as well as validating models against temporal logic formulae [11]. In order to improve data analyzed by a process miner, the technique of real-time log monitoring can be applied [12]. Another related approach consists in generating synthetic traces based on a structured declarative model [13].

The existing methods can be used to improve quality of process mining from various perspectives, however they tend to operate on different workflow representations, such as raw event data or Petri Nets, without considering the output model. The purpose of the proposed method is to evaluate the created business process model and verify its behavior by generating its admissible execution traces. The synthetic log can be then compared to the set of real execution sequences obtained from a computer system in order to indicate areas of imperfection and take corrective actions, such as changing the mining algorithm or refining log data. A schematic illustration of our approach is presented in Figure 1.

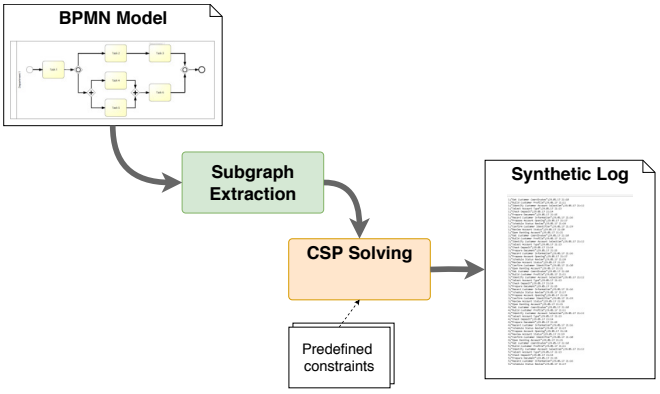

Figure 1. Overview of the proposed approach.

This paper is organized as follows: Section II provides an overview of Business Process Management which includes its definition and phases. Section III describes the concept of a BPM lifecycle and its aspects. A definition of workflow log was presented in Section IV and followed by a brief description of process mining, where logs are used to extract process knowledge, which was provided in Section V. Our main contribution is included in Sections VI and VII where we proposed methods to determine the number of traces in a process log, as well as presented a constraint-based model to generate a synthetic set of traces. Concluding remarks and plans for future works were summarized in Section IX.

II. Business Process Management

Business Process Management (BPM) [14] is a modern approach to improving organization's workflow, which focuses on reengineering of processes to obtain optimization of procedures, increase efficiency and effectiveness by constant process improvement.

The key aspect of BPM is a Business Process (BP). Although there is no single definition of a Business Process, the existing definitions have many things in common [15], [16], [17], [18]. A BP is usually described as a collection of related activities which transform different kinds of clearly specified inputs to produce a customer value, mainly considered as products or services and organizational goals, as output.

Different definitions emphasize various aspects of such defined processes. Davenport described a process highlighting the importance of producing an output for a customer – how work is done [15]. Definition of Eriksson and Penker emphasizes how work is performed rather than describing products or services, results of a process [19]. Jacobson, in turn, underlined that a process should be customer-oriented and meet an individual customer's needs [20]. A wider conceptualization of process was presented by Melao and Pidd [21]. They gave four perspectives on business process to understand BPs more fully. In their approach, BPs can be seen as either deterministic machines, complex dynamic systems, interacting feedback loops or social constructs.

Thus, Business Process Management requires a specification of many aspects, such as goals, inputs, outputs, used resources, activities and their order, impact to other organizational units, customers and owners for each of managed processes to enable real benefits. It unifies the previously distinct disciplines such as Process Modeling, Simulation, Workflow, Enterprise Application Integration (EAI), and Business-to-Business (B2B) integration into a single standard [22].

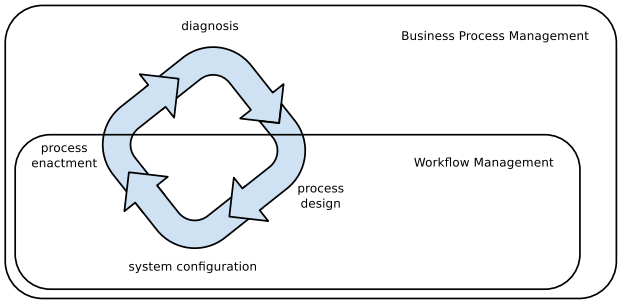

Figure 2. Comparison of Workflow Management and Business Process Management (based on [23]).

Therefore, BPM is often considered as either a legacy or the next step after workflows. Workflow Management Coalition (WfMC) [24] defines a workflow [25] in terms of automation of a business process during which documents, information or tasks are passed from one participant to another for action, according to a defined set of procedural rules. According to van der Aalst et al. [23], BPM is a broader term than Workflow Management (WFM). BPM supports business processes using methods, techniques, and software in designing, enacting, controlling, and analyzing processes involving humans, organizations, applications, documents and other information sources. It is restricted to operational processes, thus it excludes processes that cannot be made explicit.

A simple approach to process management distinguishes four phases of supporting processes [23]:

1) *process design*, in which the process is designed or redesigned,
2) *system configuration*, in which the design is implemented by configuring process management system,
3) *process enactment*, in which the operational business process is executed using the configured system,
4) *diagnosis*, in which the process is analyzed or verified to identify things that can be improved.

The relationship between WFM and BPM is presented in Figure 2. As one can observe, BPM extends the traditional WFM approach. In the case of the WFM systems, they do not support diagnosis phase, and such features as simulation, verification or validation of process designs.

III. BPM Lifecycle

Although many aspects of BPM have been debated in literature, one of the fundamental BPM issues is a repeating sequence of steps, the so-called Business Process Management Lifecycle (see Figure 3). The main idea behind the BPM lifecycle is to manage and improve BPs over business changes. Due to the use of clearly defined phases, BPM enables the continuous maintenance and the evolution of processes. During iterations, such parameters of business processes as cost, time, quality of output or customer satisfaction can be improved causing an improvement of the whole process.

Thus, BPM is in fact the application of the management cycle to organization's business processes [26]. The BPM lifecycle starts with specification of organizational and process goals as well as an assessment of environmental factors having an effect on the organization BPs. In the next process design phase, the organization processes are to be identified or redesigned. In this phase, the particular process details should be specified, and the proper variables that will influence the process design should be identified as well. During the next phase the previously specified process models are implemented in the environment, usually manually via procedure handbooks or using BPM or workflow software. Finally, the implemented process can be instantiated and executed. During execution, the performance is monitored in order to control and improve the process. Data produced during the process enactment and monitoring phases, aggregated from multiple process instances, can be used in the evaluation phase, whose purpose is to formulate the results suitable for process improvement.

Our area of research focuses on analyzing process models discovered from event logs and verifying them in terms of admissible execution sequences. Therefore it covers the evaluation phase and its side activities such as auditing event logs as well as providing measures of improvement for the whole process.

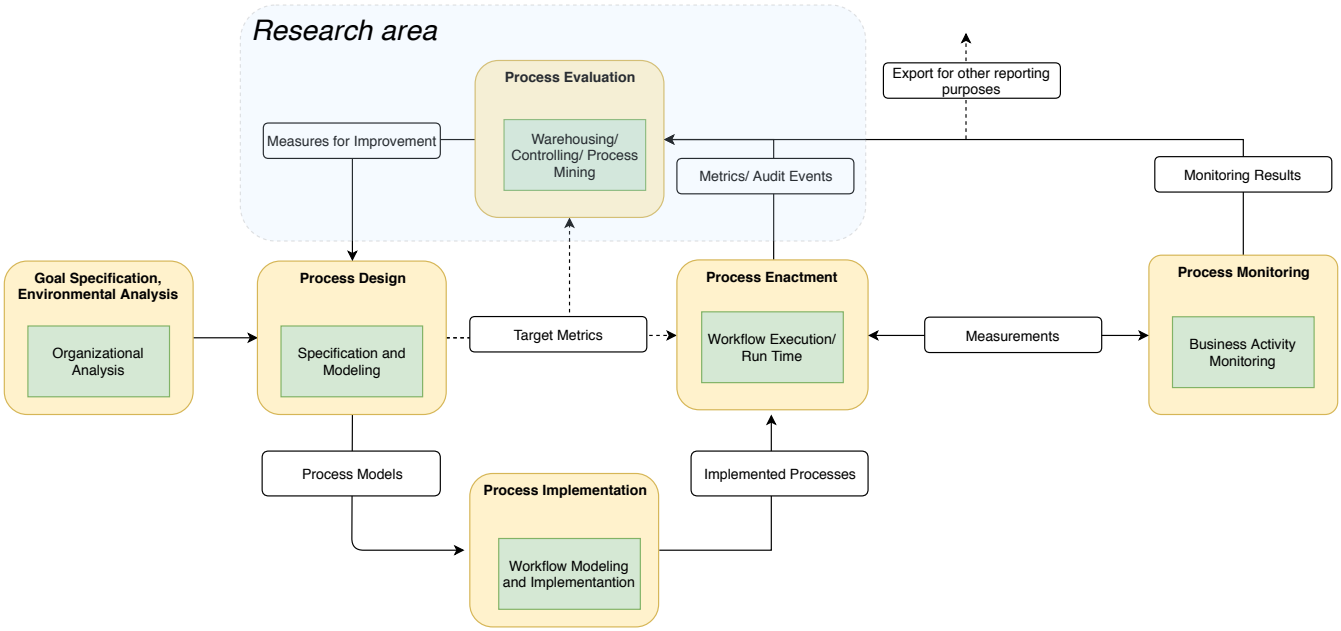

Figure 3. Business Process Management Lifecycle with the indication of our area of research (based on [26]).

IV. Workflow Log

In every business process, regardless whether it is executed manually or by an IT system, completion of each activity should be recorded with a proper timestamp. Such a record is often referred to as a log event [27] and may include information about a person or unit performing the task, as well as its cost and used resources. A set of log events ordered by their completion timestamps is called a workflow trace:

$$\sigma = \{\alpha^{(1)}, \alpha^{(2)}, ...\alpha^{(K)}\}, \tag{1}$$

where α is a log event and K represents the length of the trace.

One of the most important features of an event trace is that all recorded activities are ordered chronologically. In other words, even if two different activities are executed in parallel, their accomplishment time differs and it is always possible to distinguish the one which was completed first. However, as business processes are repeated many times, the order of their completion may differ depending on the instance of the process. In addition, processes may contain alternative gateways that, based on a logical condition, determine which task should be executed and which should remain unused. Therefore, in order to gather the information about the whole process, one should record a workflow trace for a number of times to ensure, that all or nearly all the possible execution sequences were collected. A set of workflow traces is called a workflow log:

$$W = \{\sigma_1, \sigma_2, ...\sigma_L\}, \tag{2}$$

where σ_i are separate workflow traces, also referred to as cases or workflow instances and L is the number of recorded traces.

A workflow log can be considered complete if it covers all the possible execution sequences of the process. In the case when activities are executed in loops, the number of possible traces may be infinite. Therefore, we weaken this requirement to a notion of sufficient completeness explained in Definition 1. It limits the required number of traces to those where the number of repetitions for each log event is equal to the number of cycles which include the corresponding activity.

Definition 1. (Sufficiently complete workflow log) Let G_P be a business process graph [28] representing the analyzed process and S_C be a set of all simple cycles in G_P. Function $C_C(\tau)$ determines the number of occurrences of the vertex representing activity τ in S_C. Workflow log W is sufficiently complete if it contains all the possible execution sequences where the number of occurrences for each activity τ is lower than or equal to $C_C(\tau) + 1$.

V. Process Mining

Process mining is an area of research which focuses on extracting knowledge from event logs [29] which were described in details in Section IV. One of the challenging tasks within process mining is process discovery [30] which includes algorithms able to generate process models in a flexible way, and in some cases without the need of any human actions. Although process discovery methods can produce syntactically correct workflow nets, the result is a general process model which is not directly applicable for execution in a runtime BPMN environment. BPMN diagrams can be obtained directly from event logs [31]. However, they require significant enhancements to be suitable for execution. Such modifications can be based on decision mining [32] which extends the process model by providing conditions for alternative or exclusive gateways.

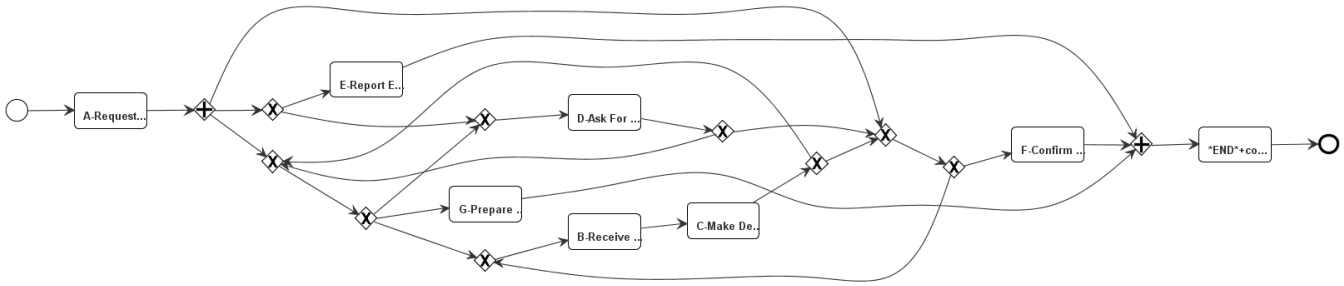

Figure 4. Result of applying a process mining algorithm to a workflow log of 36 traces.

Regarding process execution, the generated model still needs to be validated in terms of structural anomalies which may result in wrong dynamic behavior of a process [33].

Figure 4 presents a BPMN model discovered using ILP Miner [34] based on 36 different execution traces. In this example, the number of parallel and exclusive gateways (11) is higher than the number of activities (8). This implies a large number of routings and may possibly result in various exceptions during execution. Several complexity metrics exist which evaluate the complication level of business process models [35]. From a runtime point of view, the control-flow perspective should be considered. Cardoso et al. [36] propose a set of such metrics of which two can be applied in this case:

- Control Flow Complexity (CFC) which can be calculated as a sum of states induced by all the split gateways. Given n_{out} as the number of outputs of a gateway, each exclusive (XOR) split induces n_{out} states while a parallel split corresponds only to one state, as all the output branches are always used.
- Coefficient of Network Complexity (CNC) which is a quotient of the number of arcs (sequence flows) and the number of all activities, joins and splits in the process.

In the example presented in Figure 4, the Control Flow Complexity is equal to 12 and the Coefficient of Network Complexity has a value of 1.47. It is worth noting that values of these metrics for a simple workflow without any branching elements are equal to 0 and $(n_a + 1)/n_a$, where n_a is the number of activities, respectively.

VI. Determining the Number of Traces in a Process

The first step towards the estimation of the number of distinct traces is based on a business process model. A sequential workflow consisting of one start event, one end event and no gateways produces only one trace. A trivial example of such a process model is shown in Figure 5. However, models representing a simple workflow are rarely used in practice. According to the survey [37], in 90% of BPMN models the number of gateways varies between 5 and 15.

Figure 5. Simple sequential workflow.

In order to analyze business process model from an execution perspective, its token flow must be considered. It is assumed that the start event of a process generates a token which runs through the whole workflow to be consumed by an end event [38]. Although business processes can contain multiple start events, it is not regarded as a good practice as it remains unclear in the BPMN specification whether all the start events should occur before the process execution [39]. When determining a sufficiently complete log we refer to best practices of process modeling [40] following the statement that a well designed process model should contain only one start event which creates exactly one token.

On the other hand, multiple end events are a common practice in business process models, as they may represent different final states (e.g. goal and error states). However, only one of the end events is triggered in a single process instance. It may consume one or more tokens, depending on the number of incoming sequence flows.

Token flow in a business process is managed by logical gateways which determine branching flows. A single token created at the beginning of each process instance can be processed differently depending on the type of a gateway. The following actions are possible for a split gateway with n output branches:

- An exclusive (XOR) gateway places the token in one of the output sequence flows where the corresponding condition is met. As a result n different actions are possible.
- An inclusive (OR) gateway multiplies the incoming token by the number of conditions met. This action is followed by placing the created tokens on the corresponding sequence flows. As at least one output branch must be active, 2^{n-1} actions are possible.
- A parallel (AND) gateway multiplies the incoming token by the number of output branches. This result in only one possible action.

Since a single sequence flow should not transfer more than one token at a time, join gateways are responsible for merging multiple sequence flows into one output branch. An exclusive merge gateway only receives a token from one of its incoming branches and passes it to its output. A more complex situation occurs in case of a synchronization of multiple sequence flows which may be done in the following way:

- an inclusive merge consumes all the tokens created by its corresponding split gateways,
- a parallel merge consumes the tokens for all its input branches.

Each of the synchronizations result in creation of a token and passing it to the outgoing sequence flow except the situation when a parallel merge is declared implicitly by multiple sequence flows leading to an end event.

Since according to the best practices OR gateways should be avoided in BPMN modeling [40] and regarding the fact that in most cases they can be replaced by a sequence of exclusive and parallel gateways [41], this type of routing object was not taken into further analysis. As a consequence, the only flow objects in a well modeled process where token creation or consumption occurs are parallel gateways.

Figure 6 represents a simple BPMN model with one parallel and one exclusive gateway. As stated before in this section, an XOR split gateway generates as many possible states as is the number of its outgoing sequence flows. Thus, in this case there will be two possible execution sequences of the SESE (Single Entry Single Exit) block determined by two exclusive gateways: one sub-trace consisting of task D and one empty sequence. Although the presence of a single AND gateway induces one state, the generated tokens represent separate sequences of activities. As a result they can be executed independently until they reach a synchronization element represented by a parallel merge. Therefore, the first SESE block which follows the start event may also be executed in two ways, namely $\{A, B\}$ and $\{B, A\}$. Besides these two process fragments the remaining activity, denoted as C is executed in every process instance. This analysis leads to a conclusion that the number of execution traces in such a process is a multiplication of the corresponding values calculated for its SESE blocks.

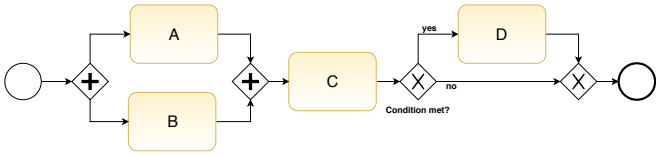

Figure 6. Example process with two basic types of gateways.

As stated before in this section, sequence flows followed by parallel gateways are executed independently. Therefore, if each branch within a SESE block delimited by AND gateways contains exactly one activity, these activities can be executed in any order. The example presented in Figure 6 illustrates a general rule for determining the number of admissible traces in a BPMN model which is expressed in Theorem 1.

Theorem 1. Let $\mathbb{P}$ be a BPMN process model containing a set of exclusive split gateways $\mathbb{G}_{XOR} = \{g_{X1}, g_{X2}, ...g_{Xk}\}$, a set of parallel split gateways $\mathbb{G}_{AND} = \{g_{A1}, g_{A2}, ...g_{Al}\}$ and two corresponding sets of merge gateways, namely $\mathbb{M}_{XOR}$ and $\mathbb{M}_{AND}$. If $\mathbb{P}$ consists of k SESE blocks determined by XOR gateways and l SESE blocks determined by AND gateways where each of l sequence flows contains exactly one activity, then the number of sequence flows in sufficiently complete workflow log W_{SC} can be expressed by Formula 3.

$$|W_{SC}| = \prod_{i=1}^{k} n(g_{Xi}) \cdot \prod_{i=1}^{l} n(g_{Ai})! \tag{3}$$

where $n(g)$ determines the number of outgoing sequence flows of a split gateway.

Proof. Let us assume that each of k exclusive gateways in $\mathbb{P}$ has exactly n outgoing flows. Knowing that every SESE element in a process model can be reduced to a subprocess which is a single BPMN activity [28] and that every XOR gateway allows for n actions, there are n possible states of every k subprocess in $\mathbb{P}$. This implies that the number of all admissible states is equal to n^k. Since the number of outgoing flows may vary for different gateways, $n(g_{Xi})$ has to be multiplied k times. □

VII. Constraint-Based Log Generation

The method presented in Section VI refers to these BPMN diagrams where SESE gateway structures are easily distinguishable. However, in automatically generated process models, as shown in Figure 4, gateways can be nested and the number of merge gateways does not have to match the number of splits. As a result, the number of traces is hardly calculable using analytical methods. Figure 7 shows a simple process model with nested gateways where Formula 3 cannot be applied.

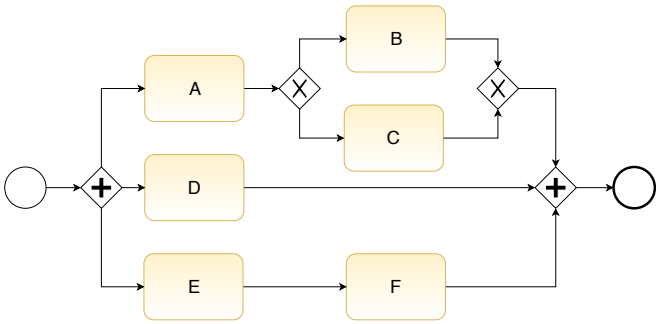

Figure 7. Example process with nested gateway structures.

The BPMN model presented in Figure 7 consist of six tasks formed in a parallel block with three branches. In business process models without loops, the upper limit of the number of traces is equal to the factorial of the number of activities. This would occur if all the tasks in the process were executed independently. In this case, however, there are following constraints which limit the set of possible sequences:

1) A must occur before B or C.
2) B is executed if and only if C is not.
3) E must occur before F.

In the analyzed example such constraints are easily identifiable and they can be expressed in a temporal logic [42]. However, dynamic generation of constraints for a complex

process model requires checking of all the dependencies between each pair of tasks which is an exponential problem. As a solution to this issue, we propose to extract process subgraphs from the model in a such manner that each of the graphs will represent the flow of a single token. It was stated in Section VI that each process instance ends with a single end event. As a consequence, all the generated tokens have to be consumed either by one of the parallel join gateways or by an event. Thus, the method can be applied to those SESE blocks in the process which start with a parallel split and whose last flow object is either a parallel join or an end event with multiple incoming sequence flows.

Let us denote the set of subgraphs as $S_G = \{s_1, s_2, ...s_q\}$. In the model presented in Figure 7, there are three tokens generated at a parallel split gateway and all of them are synchronized by a parallel merge before the end event. As a result, the extraction will provide three subgraphs, each representing tasks on a single branch (see Figure 8).

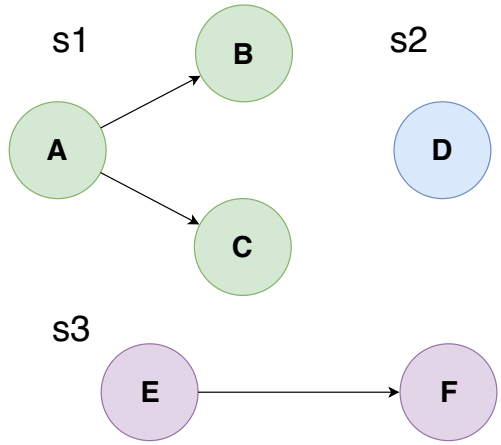

Figure 8. Token flow subgraphs extracted from a business process model.

In order to determine admissible workflow traces, a path should be calculated in every subgraph. The beginning of such a path is a vertex with no incoming edges. A path should end if a vertex with an outdegree equal to zero is reached. In the next step, all these paths are merged to vector β of length $p \cdot q$ where p is the overall number of tasks in the analyzed SESE block and q is the number of extracted subgraphs.

In Constraint Satisfaction Problems (CSP), a state which represents a set of elements must satisfy a collection of finite constraints over variables [43], [44]. CSPs can be solved using the constraint programming technique whose applications include design and modeling [45], as well as planning and scheduling [46]. If the problem is not over-constrained, a CSP algorithm always finds all the admissible solutions for finite domains [44]. Solving a Constraint Satisfaction Problem consists in general of four following steps:

1) Ordering the decision variables according to preference criteria.
2) Assigning values to each of the variables with respect to their domains.
3) Verifying if any of the constraints is violated at any step of the solving process.
4) If a complete or partial constraint-violating assignment is found, backtracking is enforced, and a succeeding set of values is assigned.

In this case, to calculate a process trace, we developed a constraint-based model in MiniZinc environment which consists of the following core elements:

1) Input data:
   - a list of tasks in the analyzed block,
   - vector e_x of maximum numbers of executions for each task (1 by default),
   - adjacency matrices $A_1, ...A_q$ for each subgraph.
2) Decision variables:
   - a subgraph trace matrix S_t of size $q \times p$,
   - a vector of merged traces β,
   - a vector γ of size p representing a workflow trace.

In order to improve the understandability of the code, we define two custom predicates which are further used in the predefined constraints:

- connected – returns true if two tasks are connected in the subgraph represented by its adjacency matrix,

```
predicate connected(src, dest, adj)
    = (adj[src, dest] == 1);
```

- same_block – returns true if a pair of indices of vector β is in the same subgraph trace.

```
predicate same_block(idx1, idx2)
    = (idx1 > 0 /\ idx2 > 0
        /\ idx1 div p == idx2 div p
    );
```

Constraints included in the model can be divided into three main groups, depending on a decision variable to which they are related. Let us briefly present these constraints along with their simplified representations in the MiniZinc language:

1) Subgraph traces:
   - the count of occurrences for each task should be lower then or equal to the input value,

```
forall(i in 1..p, j in 1..q)(
    count_geq(row(S_t,j), i, e_x[i])
);
```

   - the value 0 in the event log represents an idle task,

```
idle_task = 0;
```

   - the last log event should not be preceded by an idle task,

```
forall(i in 1..q)(
    count_neq(row(S_t,i), idle_task,
      last_process_index+1)
);
```

- all tasks after the last log event should be idle,

```
forall(i in 1..p, j in 1..q)(
    if i > last_task_index[j]
    then S_t[j,i] == idle_task
    else S_t[j,i] != idle_task
    endif
);
```

- the first element of a trace should be represented by a subgraph vertex without any incoming edges,

```
forall(i in 1..p, j in 1..q)(
  if S_t[j,1] == i
  then count(row(A_j, i),-1,0)
  endif
);
```

- the last log event should be represented as a vertex without outgoing edges,

```
forall(i in 1..p, j in 1..q)(
  if s_t[j,1] == i
  then count(row(A_j, i),1,0)
  endif
);
```

- if one task directly follows another in a trace, then it is connected by a directed edge in the corresponding subgraph.

```
forall(i in 1..q, j in
  1..last_process_index) (
  connected(S_t[i,j],S_t[i,j+1], A_i)
    \/ (S_t[i,j] == 0
        /\ S_t[i,j+1] == 0)
    \/ (j == last_task_index[1]
        /\ S_t[i,j+1] == 0)
);
```

2) Merged vector β:
   - the vector β is a concatenation of trace matrix rows.

```
forall(i in 1..last_process_index, j
  in 1..q)(
  if i < last_task_index[j]
  then beta[(j-1)*last_process_index
    + i] = S_t[j,i]
  else beta[(j-1)*last_process_index
    + i] = 0
  endif
);
```

3) Final trace vector:
   - the vector γ holds non-zero indices of β,

```
forall(i in gamma_indices)(
  if gamma[i] > 0 then
    beta[gamma[i]] != 0
  else beta[gamma[i]] == 0
  endif
);
```

- all the elements of γ are different except zero,

```
alldifferent_except_0(gamma);
```

- all the elements following the last non-zero element of γ are equal to zero,

```
forall(i in gamma_indices)(
  if i > last_gamma_index then
    gamma[i] == 0
  else gamma[i] != 0
  endif
);
```

- for each pair of tasks in γ if the pair is in the same row of matrix S_t then these task should be ordered in the same way as in S_t.

```
forall(i in gamma_indices, j in
  gamma_indices)(
  if i != j /\ gamma[i] != 0 /\
    gamma[j] != 0 /\
    same_block(gamma[i], gamma[j])
    then
    if i > j then
      gamma[i] > gamma[j]
    else
      gamma[i] < gamma[j]
    endif
  endif
);
```

In order to run the MiniZinc solver two files are needed:

- the model file *trace_id.mzn* which contains definitions of decision variables, predicates and constraints,
- the data file *subgraphs.dzn* where activity names, their maximum number of executions and subgraph adjacency matrices are defined.

For the workflow trace generation the search goal should be set for constraint satisfaction by using the statement `solve satisfy`.

The analyzed example model contains 6 tasks and 3 subgraphs, then $p = 6$ and $q = 3$. Formula 4 presents an example subgraph trace matrix S_t for subgraphs shown in Figure 8. Creation of vector β consists in merging all the subgraph traces (see Formula 5). Indices of β are used to generate trace vector γ (see Formula 6).

$$S_t = \begin{bmatrix} A & B & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 \\ E & F & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

$$\beta = [A\ B\ 0\ 0\ 0\ 0\ D\ 0\ 0\ 0\ 0\ 0\ E\ F\ 0\ 0\ 0\ 0] \tag{5}$$

$$\gamma = [13\ 7\ 1\ 2\ 14\ 0] \tag{6}$$

The resulting trace is an ordered set (see Formula 1) determined by non-zero elements of β whose indices are ordered by values of γ. Its value for the running example represented by the BPMN model in Figure 7 and trace matrix S_t defined by Formula 4 is shown in Formula 7.

$$\sigma = \{E, D, A, B, F\} \tag{7}$$

In order to generate all the admissible task sequences the solver should be set to print all solutions. Execution of the model results in a sufficiently complete log of a SESE process block. If a process contains multiple token split blocks then all of them should be handled separately.

VIII. Log-Based Verification of Process Models

Several metrics were proposed to analyze results of process discovery algorithms, namely: replay fitness, simplicity, precision, and generalization [47]. Since we tend to compare two sufficiently complete workflow logs without analyzing the graphical layout of the generated model, the following quality measures have been considered and adopted for the purpose of log comparison:

- model fitness – the percentage of traces from the original log which were generated based on the discovered model,
- execution precision – the percentage of generated workflow traces that are allowed in the original log.

It is worth noting that values for both metrics should be calculated in order to validate the resulting model. To illustrate this problem, let us analyze the example shown in Figure 6. Its original complete log can be easily determined analytically:

$$\begin{aligned} W_C = \{&\{A, B, C\}, \{B, A, C\}, \\ &\{A, B, C, D\}, \{B, A, C, D\}\}. \end{aligned} \tag{8}$$

Now let us assume that this log was used to discover a BPMN model whose traces were then generated using the constraint-based approach. The synthetic log W_S was determined as follows:

$$W_S = \{\{A, B, C, D\}, \{B, A, C, D\}, \{A, C, B, D\}\}. \tag{9}$$

Traces where activity D does not occur were not present in W_S. Thus, only half of the original traces are reproduced by the model which results in a model fitness equal to 50%. On the other hand, trace $\{A, C, B, D\}$ is not an element of W_C. In this case the execution precision will be equal to 66,67%, as only two synthetic traces out of three can be found in the original workflow log.

The application of the proposed method to the example process model presented in Figure 4 resulted in generation of 11820 distinct workflow traces. Table I presents results of a log-based verification performed for the example BPMN model.

Table I
Evaluation of the example BPMN model.

Parameter	Value
Number of original traces	36
Number of synthetic traces	11820
Traces not generated	0
Model fitness	100%
Synthetic traces not included in the log	11784
Execution precision	0.03%

The results show that the discovered process model is characterized by low execution precision (0.03%). It means that the selected process mining algorithm has a tendency to generalize, i.e. it allows for much more behavior than included in the original workflow log. Therefore, the next step of the verification process should be to check if the synthetic traces not included in the original log can be allowed in the real process. If not, then the choice of the process mining algorithm should be reconsidered in order to provide more accurate process representations.

IX. Conclusions

In the paper, we presented a novel constraint-based algorithm which results in generation of a sufficiently complete workflow log for a given business process model. The proposed approach may serve as an additional tool to verify BPMN diagrams generated using process mining techniques. Comparison of the real execution log with a synthetic one helps to choose the most suitable discovery algorithm for the analyzed process or gives clues to the user how the model can be enhanced manually.

As future works, we plan to develop an automated tool for a comparison of two workflow logs which will be able to identify flaws occurring as a result of process discovery. Such a solution could be also used as a decision support system that, based on a re-created log, provides advice to process designers which mining algorithm to use.

References

[1] J. R. Nawrocki, T. Nedza, M. Ochodek, and L. Olek, "Describing business processes with use cases," in *BIS*, 2006, pp. 13–27.

[2] K. Kluza and K. Honkisz, "From SBVR to BPMN and DMN models. proposal of translation from rules to process and decision models," in *Artificial Intelligence and Soft Computing: 15th International Conference, ICAISC 2016, Zakopane, Poland, June 12-16, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds. Springer International Publishing, 2016, vol. 9693, pp. 453–462.

[3] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *Advanced Information Systems Engineering*. Springer, 2011, pp. 482–496.

[4] K. Kluza and P. Wiśniewski, "Spreadsheet-based business process modeling," in *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*. IEEE, 2016, pp. 1355–1358.

[5] A. A. Kalenkova, W. M. van der Aalst, I. A. Lomazova, and V. A. Rubin, "Process mining using BPMN: relating event logs and process models," *Software & Systems Modeling*, vol. 16, no. 4, pp. 1019–1048, 2017.

[6] W. M. Van Der Aalst, "A general divide and conquer approach for process mining," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. IEEE, 2013, pp. 1–10.

[7] A. Rozinat, A. K. A. de Medeiros, C. W. Günther, A. Weijters, and W. M. van der Aalst, "The need for a process mining evaluation framework in research and practice," in *International Conference on Business Process Management*. Springer, 2007, pp. 84–89.

[8] S. Suriadi, R. Andrews, A. H. ter Hofstede, and M. T. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.

[9] A. Rozinat and R. Mans, "Mining cpn models: discovering process models with data from event logs," in *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*. Citeseer, 2006.

[10] A. Rozinat, A. A. De Medeiros, C. W. Günther, A. Weijters, and W. M. Van der Aalst, "Towards an evaluation framework for process mining algorithms," *BPM Center Report BPM-07-06, BPMcenter. org*, vol. 123, p. 142, 2007.

[11] W. M. van der Aalst, H. De Beer, and B. F. van Dongen, "Process mining and verification of properties: An approach based on temporal logic," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2005, pp. 130–147.

[12] D. Loreti, F. Chesani, A. Ciampolini, and P. Mello, "A distributed approach to compliance monitoring of business process event streams," *Future Generation Computer Systems*, 2018.

[13] F. Chesani, A. Ciampolini, D. Loreti, and P. Mello, "Abduction for generating synthetic traces," in *International Conference on Business Process Management*. Springer, 2017, pp. 151–159.

[14] M. Weske, *Business Process Management: Concepts, Languages, Architectures 2nd Edition*. Springer, 2012.

[15] T. H. Davenport, *Process Innovation: Reengineering Work Through Information Technology*. Boston, MA, USA: Harvard Business School Press, 1993.

[16] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. New York, NY, USA: Harper Business, 1993.

[17] S. A. White and D. Miers, *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Lighthouse Point, Florida, USA: Future Strategies Inc., 2008.

[18] A. Lindsay, A. Dawns, and K. Lunn, "Business processes - attempts to find a definition," *Information and Software Technology*, vol. 45, no. 15, pp. 1015–1019, December 2003, elsevier.

[19] H.-E. Eriksson and M. Penker, *Business Modeling with UML: Business Patterns at Work*. Wiley, 2000.

[20] I. Jacobson, M. Ericsson, and A. Jacobson, *The object advantage: business process reengineering with object technology*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1994.

[21] N. Melao and M. Pidd, "A conceptual framework for understanding business processes and business process modelling," *Information Systems Journal*, vol. 10, no. 2, pp. 105–129, 2000.

[22] M. Owen and J. Raj, "BPMN and Business Process Management. Introduction to the new business process modeling standard." OMG, Tech. Rep., 2006, www.bpmn.org.

[23] W. van der Aalst, "Business process management: a personal view," *Business Process Management Journal*, vol. 10, no. 2, 2004.

[24] WfMC, "Workfow Management Coalition," http://www.wfmc.org/.

[25] P. Lawrence, Ed., *Workflow Handbook*. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[26] M. zur Muehlen and D. T.-Y. Ho, "Risk management in the BPM lifecycle," in *Business Process Management Workshops*, 2005, pp. 454–466.

[27] W. Van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[28] P. Wiśniewski, "Decomposition of business process models into reusable sub-diagrams," in *ITM Web of Conferences*, vol. 15. EDP Sciences, 2017, p. 01002.

[29] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs *et al.*, "Process mining manifesto," in *International Conference on Business Process Management*. Springer, 2011, pp. 169–194.

[30] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[31] A. A. Kalenkova, M. de Leoni, and W. M. van der Aalst, "Discovering, analyzing and enhancing BPMN models using ProM?" in *Business Process Management-12th International Conference, BPM*, 2014, pp. 7–11.

[32] A. Rozinat and W. M. van der Aalst, "Decision mining in prom," in *Business Process Management*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4102, pp. 420–425.

[33] A. Suchenia (Mroczek), P. Wiśniewski, and A. Ligęza, "Overview of verification tools for business process models," in *Communication Papers of the 2017 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. Maciaszek, and M. Paprzycki, Eds., vol. 13. PTI, 2017, pp. 295–302. [Online]. Available: http://dx.doi.org/10.15439/2017F308

[34] J. M. E. Van der Werf, B. F. van Dongen, C. A. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *International conference on applications and theory of petri nets*. Springer, 2008, pp. 368–387.

[35] K. Kluza and G. J. Nalepa, "Proposal of square metrics for measuring business process model complexity," in *Proceedings of the Federated Conference on Computer Science and Information Systems – FedCSIS 2012, Wroclaw, Poland, 9-12 September 2012*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2012, pp. 919–922. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6354395

[36] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers, "A discourse on complexity of process models," in *Proceedings of the 2006 international conference on Business Process Management Workshops, Vienna, Austria*, ser. BPM'06, S. D. e. a. J. Eder, Ed. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 117–128.

[37] M. Chinosi and A. Trombetta, "BPMN: An introduction to the standard," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[38] V. S. W. Lam, "A precise execution semantics for BPMN," *IAENG International Journal of Computer Science (IJCS)*, vol. 39, no. 1, pp. 20–33, 2012.

[39] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[40] J. Mendling, H. A. Reijers, and W. M. van der Aalst, "Seven process modeling guidelines (7pmg)," *Information and Software Technology*, vol. 52, no. 2, pp. 127–136, 2010.

[41] C. Favre and H. Völzer, "The difficulty of replacing an inclusive or-join," in *International Conference on Business Process Management*. Springer, 2012, pp. 156–171.

[42] R. Klimek, L. Faber, and M. Kisiel-Dorohinicki, "Verifying data integration agents with deduction-based models," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. IEEE, 2013, pp. 1029–1035.

[43] P. Sitek and J. Wikarek, "A hybrid method for modeling and solving constrained search problems," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. IEEE, 2013, pp. 385–392.

[44] A. Ligęza, "Models and tools for improving efficiency in constraint logic programming," *Decision Making in Manufacturing and Services*, vol. 5, no. 1, pp. 69–78, 2011. [Online]. Available: https://journals.agh.edu.pl/dmms/article/view/537

[45] P. Wiśniewski, K. Kluza, M. Ślażyński, and A. Ligęza, "Constraint-based composition of business process models," in *International Conference on Business Process Management*. Springer, 2017, pp. 133–141.

[46] P. van Beek and X. Chen, "Cplan: A constraint programming approach to planning," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, ser. AAAI '99/IAAI '99. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1999, pp. 585–590. [Online]. Available: http://dl.acm.org/citation.cfm?id=315149.315406

[47] J. C. Buijs, B. F. Van Dongen, and W. M. van Der Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2012, pp. 305–322.