BePT: A Process Translator for Sharing Process Models

Chen Qian  
Tsinghua University  
Haidian, Beijing, China  
qc16@mails.tsinghua.com

Lijie Wen*  
Tsinghua University  
Haidian, Beijing, China  
wenlj@tsinghua.edu.cn

Ahhil Kumar  
Penn State University  
University Park, PA, USA  
akhillkumar@psu.edu

ABSTRACT

Sharing process models on the web has emerged as a widely used concept. Users can collect and share their experimental process models with others. However, some users always feel confused about the shared process models for lack of necessary guidelines or instructions. Therefore, several process translators have been proposed to explain the semantics of process models in natural language (NL) in order to extract more value from process repositories. We find that previous studies suffer from information loss and generate semantically erroneous descriptions that diverge from original model behaviors. In this paper, we propose a novel process translator named BePT (Behavior-based Process Translator) based on the encoder-decoder paradigm, encoding a process model into a middle representation and decoding the representation into a NL text. The theoretical analysis demonstrates that BePT satisfies behavior correctness, behavior completeness and description minimality. The qualitative and quantitative experiments show that BePT outperforms the state-of-the-art methods in terms of capability, detailedness, consistency, understandability and reproducibility.

KEYWORDS

Process Model, Model Behavior, Translator, Natural Language

1 INTRODUCTION

A process consists of a series of interrelated tasks. Its graphical description is called process model [37]. Over the past decade, a specific kind of process model - scientific workflow - has been established as a valuable means for scientists to create reproducible experiments [36]. Several scientific workflow management systems (SWFM) have become freely available, easing scientific models’ creation, management and execution [36]. However, creating scientific models using SWFMs is still a laborious task and complex enough to impede non computer-savvy researchers from using these tools [3]. Therefore, cloud repositories emerged to allow sharing of process models, thus facilitating their reuse and repurpose [11, 12, 36]. As shown in Figure 1, the model developers use modeling tools such as PIPE\(^1\), WoPED\(^2\) and Signavio\(^3\) to build and manage process models which can be hosted in the cloud, then run as a service or be downloaded to users’ local workspaces. Popular examples of such scientific model platforms or repositories include myExperiment, Galaxy, Kepler, CrowdLabs, Taverna, VisTrails, e-BioFlow, e-Science and SHIWA [11–13, 16, 36]. As for users, reusing shared models from public repositories is much more cost-effective than creating, testing and tuning a new one.

Figure 1: The Model Sharing Scenario: The users collect the shared models developed by the third-party developers.

However, those models are difficult to reuse since they lack necessary NL guidelines or instructions to explain the steps, jump conditions and related resources [10, 20, 21, 33]. For example, the repository offered with myExperiment currently contains more than 3910 process models from various disciplines including bioinformatics, astrophysics, earth sciences or particle physics [36], but only 1288 out of them have NL documents\(^4\), which shows the gap between the shared models and their NL descriptions. This real-world scenario illustrates that the cloud platforms do not have effective means to address this translation problem [33, 43], i.e., automatically translating the semantics of process models in NL, thus making it challenging for users to reuse the shared models. Now that means to translate a process model are able to help users understand models and improve shared models’ reusability [10], a growing interest in exploring automatic process translators - process to text (P2T) techniques - has emerged.

Related Work. The structure-based approach [20] was proposed to generate the text of a process model. It first extracts language information before annotating each label [19, 22, 23]. Then it generates the annotated tree structure before traversing it by a depth first search. Once sentence generation is triggered, it employs NL tools to generate corresponding NL sentences [18]. This work solved the annotation problem, but it only works for structured models and ignores unstructured parts. Another approach [21] was subsequently proposed to handle unstructured parts. It recursively extracts the longest path of a model on the unstructured parts to linearize each activity. However, it only works on some patterns and is hard to be extended to much more complex situations. Along this line, another structure-based method was proposed [33] which can handle more elements and complex patterns. It first preprocesses a model by trivially reversing loop edges and splitting multiple-entry-multiple-exit gateways. Then, it employs heuristic rules to match every source gateway node with the defined goals. Next, it unfolds the original model based on those matched goals. Finally, it generates the texts of the unfolded models. Although this structure-based method maintains good paragraph indentations, it neglects the behavior correctness and completeness. Other "to-text"
works that take BPMN [26], EPC [1], UML [28], image [41] or video [24] as inputs are difficult to be applied into the process-related scenarios or are not for translation purpose. Hence, our motivation is to design a novel process translator.

We define our problem as follows: given a process model, our approach is going to generate the textual descriptions for the semantics of the model. Note that although there are many process modeling languages, we choose Petri nets as our process modeling language [38] because of: 1) their formal semantics; 2) an abundance of analysis techniques and tools; 3) ease of transformation from/to other process modeling languages. Hence, a process model in any other language can be easily converted into a Petri net.

Our approach - BePT - first embeds the structure information and the language information of a Petri net into a tree representation, and then linearizes the tree representation by extracting its behavior paths. Finally, it generates sentences for each sorted path. The theoretical analysis and the conducted experiments demonstrate that BePT satisfies three desirable properties and outperforms the state-of-the-art P2T approaches in several dimensions. To summarize, our contributions are listed as follows:

1) **Behavior based:** To the best of our knowledge, this work is the first attempt that fully considers model behaviors in process translation. BePT avoids the semantic error problem. The theoretical analysis proves its three key properties: behavior correctness, behavior completeness and description minimality.

2) **Powerful capability:** BePT can handle more model patterns including easy structured parts and complex unstructured parts. Besides, the problematic modeling features can be detected and appended as auxiliary warning messages to the final output.

3) **Better expressiveness:** BePT linearizes each element on a behavior-unfolded graph, organizes the text in an intuitive way into various templates and grammar patterns. The corresponding experiments show that it is more expressive in terms of detailedness, consistency and understandability.

4) **Better reproducibility:** Due to the clean organization, detailedness and understandability, users can better understand the generated texts from which they can reproduce the original models.

5) **More accurate statistics:** The experiments are conducted on ten-time larger (compared with previous works) datasets collected from industry and academic fields to better reveal the statistical characteristics.

6) **Tool development:** An easy-to-use web tool with a proper GUI has been implemented and we make it publicly available. It can be used as an independent software, and also be integrated into existing systems or platforms.

The rest of the paper is organized as follows. Section 2 introduces background knowledge. Section 3 illustrates a motivating example. Section 4 describes BePT and Section 5 evaluates current P2T methods before Section 6 concludes the paper.

2 **PRELIMINARIES**

Before going further into the main idea, we introduce some background knowledge: Petri net [25, 30, 31], Refined Process Structure Tree (RPST) [40], Complete Finite Prefix (CFP) [8, 9, 27] and Deep Syntactic Tree (DSynT) [2, 18]. These four concepts are respectively used for process modeling, structure analysis, behavior unfolding and sentence generation.

2.1 **Petri Net**

**Definition 1 (Petri Net, Net System, Boundary node).** A Petri net \( N \) is a tuple \((P, T, F)\), where \( P \) is a finite set of places, \( T \) is a finite set of transitions, \( F \subseteq (P \times T) \cup (T \times P) \) is a set of directed arcs. A marking of \( N \), denoted \( M \), is a bag of tokens over \( P \). A net system \( S = (N, M) \) is a Petri net \( N \) with an initial marking \( M \). The input set and output set of a node \( n \) are respectively denoted as \( *n = \{ x | (x, n) \in F \} \) and \( n* = \{ x | (n, x) \in F \} \). The source set and sink set of a net \( N \) are respectively denoted as \( *N = \{ x \in P \cup T | x \in \emptyset \} \) and \( N* = \{ x \in P \cup T | x* \in \emptyset \} \). These boundary elements \(*N* = N \cup N*\) are called boundary nodes of \( N \).

**Definition 2 (Firing Sequence, TAR, Trace).** Let \( S = (N, M) \) be a net system with \( N = (P, T, F) \). A transition \( t \in T \) can be fired under a marking \( M \), denoted \( (N, M)(t) \), iff each \( p \in *t \) contains at least one token. After \( t \) fires, the marking \( M \) changes to \( M \setminus \{ t \} \cup \{ t* \} \) (Firing Rule). A sequence of transitions \( \sigma = t_1 t_2 \cdots t_n \in T^* \) is called a firing sequence (iff \( (N, M)(t_1)(N, M_1)(t_2) \cdots (N, M_n) \) holds). Any transition pair that fires continguously \( (t_i < t_{i+1} \) is called a transition adjacency relation (TAR). A firing sequence \( \sigma \) is a trace of \( S \) iff the tokens completely flow from all source(s) to sink(s).

**Example 1.** Figure 2(a) shows a real-life bioinformatic process model expressed by Petri net. \( P_a \) contains one token so that the current marking \( M = \{ 1, 0, 0, 0 \} \) (over \( P_a, P_b, P_c, P_d \)). According to the firing rule, each node in the input set of \( T_a \) \((T_a = \{ P_a \})\) contains at least one token so that \( T_a \) can be fired. After firing \( T_a \), the marking becomes \( M^\ast T_a \cup T_a^\ast \), i.e., \([0, 1, 0, 0]\). The TAR set of \( N_1 \) is \( \{ T_a < T_b, T_a < T_c, T_b < T_d, T_c < T_d \} \). The trace set of \( N_1 \) is \( \{ T_a T_b T_d, T_a T_c T_d \} \).

2.2 **Refined Process Structure Tree (RPST)**

**Definition 3 (Component, RPST, Structured, Unstructured).** A process component is a sub-graph of a process model with a single entry and a single exit (SESE), and it does not overlap with any other component. The RPST of a process model is the set of all the process components. Let \( C = \bigcup_{i=1}^{n} \{ c_i \} \) be a set of components of a process model. \( C \) is a trivial component iff \( C \) only contains a single arc; \( C \) is a polygon component iff the exit node of \( c_i \) is the entry node of \( c_{i+1} \); \( C \) is a bond component iff all sub-components share same boundary nodes; Otherwise, \( C \) is a rigid component. A rigid component is a region of a process model that captures arbitrary structure. Hence, if a model contains no rigid components, we say it is structured, otherwise it is unstructured.

**Example 2.** The colored backgrounds in Figure 2(a) demonstrate the decomposed components which naturally form a tree structure - RPST - shown in Figure 2(b). The whole net (polygon) can be decomposed into three first-layer SESE components (\( P^1, P^2, P^3 \)), and these three components can be decomposed into second-layer components (\( a, b, P^3, P^4, g, h \)). The recursive decomposition ends at a single arc (trivial).

2.3 **Complete Finite Prefix (CFP)**

**Definition 4 (Cut-off transition, mutual, CFP).** A branching process \( O = (P, T, F) \) is a completely fired graph of a Petri net...
satisfying that 1) \(|p| \leq 1, \forall p \in P; 2)\) no element is in conflict with itself; 3) for each \(x\), the set \(\{y \in P \cup T | y < x\}\) is finite. The mapping function \(h\) maps each CFP element to the corresponding element in the original net. If two nodes \(p_1, p_2\) in CFP satisfy \(h(p_1) = h(p_2)\), we say they are mutual (places) to each other.

A transition \(t\) is a cut-off transition if there exists another transition \(t'\) such that \(\text{Cut}(\{t\}) = \text{Cut}(\{t'\})\) where \(\{t\}\) denotes a set of transitions of \(t\) satisfying TAR closure (\(\forall e \in T : e < t \Rightarrow e \in \{t\}\) and \(\text{Cut}(\{t\}) = h(\{a\} \cup \{b\})\)). A CFP is the greatest backward closed subnet of a branching process containing no transitions after any cut-off transition.

**Example 3.** Figure 2(c) shows the branching process of \(N_1\) (including the light-gray part). Since each original node corresponds to one or more CFP nodes, thus we append “\(c\)” to number each CFP node. As \(h(P_{c1}) = h(P_{c2}) = P_c\) so that \(P_{c1}\) and \(P_{c2}\) are mutual (places). In \(N_1\), \(\text{Cut}(\{P_{b1}\}) = \text{Cut}(\{P_{c1}\}) = P_c\) so that \(P_{c1}\) is a cut-off transition (transitions after \(P_{c1}\) are cut). The cut graph is CFP of \(N_1\) (excluding the light-gray part).

In particular, we emphasize two reasons for using the CFP concept \([8, 9, 27]\): 1) the CFP contains no false behaviors of its original model; 2) the CFP is a complete and minimal behavior-unfolded graph of the original model.

### 2.4 Deep Syntactic Tree (DSynT)

A DSynT is a dependency representation of a sentence. In a DSynT, each node carries a verb or noun decorated with meta information such as the tense of the main verb or the number of nouns etc, and each edge can denote three dependencies - subject (I), object (II), modifier (ATTR) - between two adjacency nodes.

**Example 4.** Figure 2(d) shows the DSynT of \(T_a\) in \(N_1\). The main verb “extract” is decorated by class “verb” and the voice “active”. The subject and the object of “extract” are “experiment” (assigned by the model developer) and “genes”. This DSynT represents the dependency relations of the sentence “the experimenter extracts the genes”.

### 3 A MOTIVATING EXAMPLE

In this part, we use a motivating example to illustrate some issues that exist in previous P2T techniques. We name three existing process translators as Leo \([20]\), Hen \([21]\), Goun \([33]\) respectively. Leo text, Hen text and Goun text respectively denote the generated NL text via the corresponding translator. Figure 3 shows two motivating examples. Due to the limited space, we only present the structural skeleton.

For ease of representation, we employ “•” to mark each paragraph, the underline format “\(T\)” represents the subject-predicate-object sentence for a single activity \(T\), and the right arrow “\(\rightarrow\)” represents the conjunction adverb.

As for \(N_2\), Leo text, Hen text and Goun text are almost the same:

*One of the following branches is executed:*

- \(T_d \rightarrow T_e\).
- \(T_h \rightarrow T_e\).

*When all of the above branches are executed,...*

Note that \(T_a\) and \(T_h\) in \(N_2\) are in a conflict relation (starting from the same place), but finally turn into a concurrent relation (ending at the same transition). Therefore, \(T_d\) is a deadlocked activity according to the firing rule, i.e., \(T_d\) will never fire. Thus, \(N_2\) is incorrectly modeled. But in the above generated text, \(T_d \rightarrow T_e\) and \(T_h \rightarrow T_e\) imply that \(T_c\) can be fired after \(T_a\) or \(T_h\). Besides, the singular adverb “is” contradicts with the plural adverb “are” in the text. This is the so-called semantic error problem \([33]\), i.e., misleading descriptions diverging from the semantics of the original model. As for \(N_3\), Leo and Hen fail to handle it since parallel behaviors (caused by transitions with multiple outgoing arcs) in a rigid region have no longest node-by-node path, whereas Goun utilizes a structure-based unfolding strategy. Goun text also contains diverging descriptions: “\(T_d \rightarrow T_e\) → \(T_h \rightarrow T_e\)”, which will never occur according to the firing rule. Thus, \(N_3\) also reveals the semantic error problem.

These two examples show that state-of-the-art methods - Leo, Hen, Goun - not only cannot identify the modeling features, but also cannot guarantee the correctness of model semantics in the generated descriptions \([33]\), i.e., semantic errors exist. Therefore, we investigated the literature and tried to break through the main bottlenecks. Here, we list some challenges to be solved:

C1 How to analyze and decompose the structure of a complex model, such as an unstructured or multi-layered one?

C2 For each element, how to analyze the language pattern of a short label, extract the main linguistic information and create semantically correct descriptions?

C3 How to transform a non-linear process model into linear representations, especially when it contains complex patterns?
4 OUR METHOD
To solve the six challenges (C1-C6) listed in the previous section, we propose BePT which is built on the encoder-decoder framework inspired from machine translation systems [5, 6, 35]. The encoder creates an intermediate tree representation from the original model and the decoder generates the NL descriptions from it. Figure 4 presents a high-level framework of BePT, including four main phases: Structure Embedding, Language Embedding, Text Planning and Sentence Planning [20, 21, 33]:

1) **Structure Embedding** (C1): Embedding the structure information of the original model into the intermediate representation.

2) **Language Embedding** (C2): Embedding the language information of the original model into the intermediate representation.

3) **Text Planning** (C3, C4): Linearizing the non-linear tree representation by extracting its behavior paths (defined below). This phase is the most important and challenging.

4) **Sentence Planning** (C5, C6): Generating NL text by employing pre-defined language templates and NL tools.

4.1 Structure Embedding
We take a simplified model $N_s$ shown in Figure 5, as our running example due to its complexity and representativeness. A complex sub-component (any structure is possible) in the original model is replaced by the black single activity $T_e$. The simplified model $N_s$ is also complex since it contains a main path and two loops.

We employ a simplification algorithm from [33] to replace each sub-model with a single activity to obtain a simplified but behavior-equivalent one because a model containing many sub-models may complicate the behavior extraction [33]. In the meantime, the simplification operation causes no information loss [33] since the simplified part will be visited in the deeper recursion. We emphasize that this simplification step is easy and extremely necessary for behavior correctness (see Appendix A).

Next, we analyze its structural skeleton and then create the RPST of $N_s$. Finally, we embed its structure information - RPST - into a tree representation (as shown in the upper part of Figure 6).

![Figure 5: A simplified model ($N_s$). The original complex component (any structure is possible) is simplified by the black element (a single activity).](image)

4.2 Language Embedding
4.2.1 Extract Linguistic Information. This step sets out to recognize NL labels and extract the main linguistic information [19, 22, 23]. For each NL label, we first examine prepositions and conjunctions. If prepositions or conjunctions are found, respective flags are set to true. Then we check if the label starts with a gerund. If the first word of the label has an “ing” suffix, it is verified as a gerund verb phrase (e.g., “extracting gene”). Next, WordNet [29] is used to learn if the first word is a verb. If so, the algorithm refers it to a verb phrase style (e.g., “extract gene”). In the opposite case, the algorithm proceeds to check prepositions in the label. A label containing prepositions the first of which is “of” is qualified as a noun phrase with “of” prepositional phrase (e.g., “creation of database”). If the label is categorized to none of the enumerated styles, the algorithm refers it to a noun phrase style (e.g., “gene extraction”). Finally, we similarly categorize each activity label into four labeling styles (gerund verb phrase, verb phrase, noun phrase, noun phrase with “of” prepositional phrase).

Lastly, we extract the linguistic information - role, action and objects - depending on which pattern it triggers. For example, in $N_s$, the label of $T_d$ triggers a verb phrase style. Accordingly, the action lemma “remove” and the noun lemma “impurity” are extracted.

4.2.2 Create DSynTs. Once this main linguistic information is extracted, we create a DSynT for each label by assigning the main verb and main nouns including other associated meta information [2, 18] (as shown in the lower part of Figure 6).

For better representation, we concatenate each DSynT root node to its corresponding RPST leaf node, and we call this concatenated tree RPST-DSynTs (RDT). The RDT of $N_s$ is shown in Figure 6.

So thus far, we have embedded the structural information (RPST) and the linguistic information (DSynTs) of the original process model into the intermediate representation RDT. Then, it is passed to the decoder phase.

4.3 Text planning
The biggest gap between a model and a text is that a model contains sequential and concurrent semantics [14], while a text only contains sequential sentences. Thus, this step focuses on transforming a non-linear model into its linear representations.

In order to maintain behavior correctness, we first create the CFP of the original model because a CFP is a complete and minimal
behavior-unfolded graph of the original model [8, 9, 27]. Figure 7 shows the CFP of $N_r$. According to Definition 4, $T_{d1}$ and $T_{c1}$ are two cut-off transitions, thus, no transitions follow them.

Besides, we introduce a basic concept: shadow place. Shadow places (SP) are those places that are: 1) mutual with CFP boundary places or 2) mapped to the boundary places of the original model.

**Example 5.** In Figure 7, the five colored places are shadow places of $N_r$ ($P_{d1}, P_{c1}, P_{d2}, P_{a1}, P_{d1}, P_{a2}, P_{p1}, P_{d2}$) as $P_{d1}, P_{c1}, P_{d2}, P_{a1}, P_{d2}$ are mutual with the CFP boundary places, and $P_{d1} = h(P_{a1}, P_{d2})$ is mapped to the boundary places of the original model $N_r$. Intuitively, a shadow place represents the repetition of a boundary place in the original model or its CFP.

4.3.1 **Behavior Segment.** Since we have obtained the behavior-unfolded graph, i.e., CFP, now, we define (behavior) segments which capture the minimal behavioral characteristics of a CFP.

**Definition 5 (Behavior Segment).** Given a net $N = (P, T, F)$ and its CFP $\mathcal{N}$, a behavior segment $S \equiv (P', T', F')$ is a connected sub-model of $\mathcal{N}$ satisfying:

1) $S \subseteq SP(\mathcal{N}) \land P' \subset S \cup \mathcal{N} \land F' \cap (S \cup \mathcal{N}) = \emptyset$, i.e., all boundary nodes are shadow places and all other places are not (SP-bounded).
2) If each place in $h(S)$ contains one token, after firing all transitions in $h(T)$, each place in $h(S)$ contains just one token while other places in $h(N)$ are empty (repaly-hold).

**Example 6.** According to Definition 5, if we put $h(P_{a1}) = P_a$ (in $N_r$) a token, $T_a$ (in $N_r$) can be fired, and after this firing, only $P_b$ (in $N_r$) contains a token. Therefore, the sub-model containing nodes $P_{a1}, T_{a1}, P_{b1}$ (in $N_r$) and their adjacency arcs is a behavior segment. All behavior segments of $N_r$ are shown in Figure 9(a) (careful readers might have realized that these four segments belong to sequential structures, i.e., all segments contain only SESE nodes. However, a behavior segment can be a non-sequential structure, i.e., containing multiple incoming or multiple outgoing nodes. For example, the behavior segment of $N_3$ in Figure 3 is homogenous to $N_3$ itself, containing four multiple incoming or multiple outgoing nodes).

4.3.2 **Linking Rule.** Behavior segments capture the minimal behavioral characteristics of a CFP. In order to portray the complete characteristics, we link these segments to obtain all possible behavior paths by applying the linking rule below.

**Definition 6 (Linking Rule).** For two segments $S_i = (P_i, T_i, F_i)$ and $S_j = (P_j, T_j, F_j)$, if $h(S_i) \supset h(S_j)$ we say they are linkable. If two places $p_i \in S_i \land p_j \in S_j$ are mutual, we say $p_i$ is the joint place of $p_j$ denoted as $J(p_i, p_j) = p_i$ where $J$ is the joint function. When $n \not\in S_j$, $J(n) = n$. The linked segment of two linkable segments $S_i, S_j$ is $(P_i, T_i, F_i) \land S_j$:

1) $P_i = P_i \cup (P_j \setminus S_j)$, i.e., the places of a linked segment consist of all places in $S_i$ and all non-entry places in $S_j$.
2) $T_i = T_i \cup T_j$, i.e., the transitions of a linked segment consist of all transitions in $S_i$ and $S_j$.
3) $F_i = \{(J(u), J(v))|(u, v) \in F_i \lor F_j\}$, i.e., the arcs of a linked segment are the $J$-replaced arcs of $S_i$ and $S_j$.

Similarly, $S_1, S_2, \ldots, S_n$ denotes the recursive linking of two segments $S_1, S_2, \ldots, S_{n-1}$ and $S_n$. The graphical explanation of the linking rule is shown in Figure 8.

**Figure 8:** The graphical explanation of linking two segments $S_i, S_j$. The joint nodes are shown in red/blue color.

4.3.3 **Behavior Path.** According to the linking rule, we can obtain all linked segments. However, a linked segment might involve infinite linking due to concurrent and loop behaviors [14]. Hence, we use truncation conditions to avoid infinite linking, which leads to the definition of a (behavior) path. Behavior paths capture complete behavioral characteristics of a CFP.

**Definition 7 (Behavior Path).** A segment $P = S_1, S_2, \ldots, S_n$ of $N$ is a behavior path iff one of the following conditions holds:

1) $P \equiv *N \land P \subseteq N^*$, i.e., $P$ starts from the entry of $N$ and ends at one of the exits of $N$.
2) $h(P) = h(P^*)$, i.e., $P$ starts from a shadow node (set) and ends at this node (set), i.e., loop structure.

**Example 7.** Take Figure 9(a) as an example. Since $h(S_1) = h(P_{a1}) = h(P_{a2}) \supset h(*S_1) = h(P_{a1}) = h(P_{a2})$, it follows that $S_3$ and $S_1$ are linkable with $J(P_{a1}) = P_{a1}$, and $h(S_1, S_1) = h(S_3, S_1) = h(P_b)$. Thus, the linked segment $S_3, S_1$ is a behavior path $(P_4$ in Figure 9(b)). Partial behavior paths of $N_r$ are shown in Figure 9(b).

After defining behavior path, we can extract all behavior paths from any process model. However, to linearize each path, we employ heuristic strategies (path-level linearization) corresponding to a user’s understanding way, presented in Algorithm 1. A path starting...
from source node(s) holds higher priority, ending in sink node(s) holds lower priority (Line 2-3), and a shortest path first strategy (Line 4-5) is used.

**Algorithm 1: Path-Linearization Algorithm**

| Input: | N: A CFP; P: The behavior path set of N. |
| Output: | The sorted behavior paths \( \mathcal{L}^P \). |
| 1 | foreach \( P_i = (T_i, T_j, F_i) \), \( P_j = (T_j, T_k, F_j) \) in \( P \) do |
| 2 | if \( P_i \subseteq \mathbb{N} \land P_j \subseteq \mathbb{N} \lor \bar{P}_i \subseteq \mathbb{N} \land \bar{P}_j \subseteq \mathbb{N} \) then |
| 3 | set \( P_i > P_j \) |
| 4 | else if \( |P_i \cup T_j| < |P_j \cup T_j| \) then |
| 5 | set \( P_j > P_i \) |
| 6 | return \( \mathcal{L}^P \) |

The node-level linearization is realized by recursion of behavior paths. Each path is recognized as a polygon component and then put into the recursive algorithm - BePT. The end point is a non-decomposable trivial component. When encountering a gateway node (split or join node), the corresponding DSynT (a predefined language template) is retrieved from RDT or pre-defined XML format files. When encountering a SESE node, the corresponding DSynT is extracted from the embedded RDT. After obtaining all DSynTs, the sentence planning phase is triggered.

### 4.4 Sentence planning

Sentence planning sets out to generate a sentence for each node. The main idea here is to utilize a DSynT to create a NL sentence [2, 18, 21]. The generation task is divided into two levels: template sentence and activity sentence generation.

- **Template sentences** focus on describing the behavioral information related to the non-terminal RPST nodes. We provide 28 language template DSynTs (including split, join, deal transition, deadlock [37] etc.) to represent corresponding semantic(s). Choosing which template depends on three parameters [20, 21, 33]: 1) the existence of a gateway label; 2) the gateway type; 3) the number of outgoing arcs. For instance, for a place with multiple outgoing arcs, the corresponding template sentence “One of the branches is executed” will be retrieved and output.

- **Activity sentences** focus on describing a single activity related to the terminal (leaf) RPST nodes. RDT representation has embedded all DSynT messages, thus, for each activity, we can directly access its DSynT from RDT.

After preparing all DSynTs in the text planning phase, we employ three steps to optimize the expression before final generation:

1. Checking whether each DSynT lacks necessary grammar meta-information to guarantee its grammatical correctness.
2. Pruning redundant TARs to ensure the selected TARs will not be repeated (Pruning Rule). For example, \( T_a < T_b \) derived by \( P_2 \) or \( P_4 \) in Figure 9(b) is a redundant TAR because it has been concluded in \( P_1 \).
3. Refining the DSynT messages containing the same linguistic component between two consecutive sentences and making use of three aggregation strategies: role aggregation, action aggregation and object aggregation [21, 33].

After expression optimization, we employ the DSynT-based realizer RealPro [18] to realize sentence generation. RealPro requires a DSynT as input and outputs a grammatically correct sentence [2]. In a loop, every DSynT is passed to the realizer. The resulting NL sentence is then added to the final output text. After all sentences have been generated, the final text is presented to the end user.

**Example 8.** The generated text of \( N^* \) in Figure 5 is as follows (other state-of-the-art methods cannot handle this model):

1. **The following main branches are executed:**
2. **The experimenter extracts the genes. Then, he sequences the DNA. Subsequently, the experimenter records the data.**
3. **Attention, there are two loops which may conditionally occur:**
4. **After sequencing DNA, the experimenter can also remove impurities if it is not clean. Then, he continues extracting genes.**
5. **After recording the data, there is a series of activities that need to be finished before DNA sequencing:**
6. **…”**
7. **Once the data is enough, the process ends.”**

Template sentences (1, 3, 7) describe where the process starts, splits, joins and ends. Activity sentences (2, 4, 5) describe each sorted behavior path. The paragraph placeholder (6) can be flexibly replaced according to the sub-text of the simplified component \( T_C \). We can see that BePT first describes the main path \( (T_a \rightarrow T_b \rightarrow T_c) \) before two possible loops \( (T_a \rightarrow T_b \rightarrow T_d \rightarrow T_e) \) and \( (T_c \rightarrow T_b \rightarrow T_e) \). These three paragraphs of the generated text correspond to three correct firing sequences of the

![Figure 9: The behavior segments and the partial behavior paths of N_p.](image-url)
original model, the generated text contains just enough descriptions to reproduce the original model without redundant descriptions.

4.5 Property Analysis

We emphasize BePT’s three strong properties - correctness, completeness and minimality. Specifically, given a net system $S = (N, M)$ and its TAR set $T(S)$. The behavior path set $P$ of $N$ by the linking rule (Definition 6) satisfies: 1) **behavior correctness**, $\forall P \in P \Rightarrow T(P) \subseteq T(S)$; 2) **behavior completeness**, $\forall \tau \in T(S) \Rightarrow \exists P \in P, \tau \in T(P)$; 3) **description minimality**, each TAR $T(S)$ by the pruning rule is described only once in the final text. Please see Appendices A, B and C for detailed proofs.

5 EVALUATION

We have conducted extensive qualitative and quantitative experiments. In this section, we report the experimental results to answer the following research questions:

**RQ1** Capability: Can BePT handle more complex model patterns?  
**RQ2** Detailedness: How much information does BePT provide?  
**RQ3** Consistency: Is BePT text consistent to the original model?  
**RQ4** Understandability: Is BePT text easier to understand?  
**RQ5** Reproducibility: Can the original model be reproduced only from its generated text?

5.1 Experimental Setup

In this part, we describe our experimental datasets, the baselines and the experiment settings.

5.1.1 Datasets. We collected and tested on seven publicly accessible datasets: SAP, DG, TC, SPM, IBM, GPM, BAI [20, 21, 32, 33]. Among them, SAP, DG, TC, IBM are from industry (enterprises etc.) and SPM, GPM, BAI are from academic areas (literatures, online tutorials, books etc.). The characteristics of the seven datasets are summarized in Table 1 (sorted by the decreasing ratio of structured models SMR). There are a total of 389 process models consisting of real-life enterprise models (87.15%) and synthetic models (12.85%). The number of transitions varies from 1 to 145 and the depth of RPSTs varies from 1 to 12. The statistical data is fully skewed due to the different areas, amounts and model structures.

5.1.2 Baseline Methods. We compared our proposed process translator BePT with the following three state-of-the-art methods:

- **Leo** [20]. It is the first structured-based method focusing mainly on structured components: trivial, bond and polygon.
- **Hen** [21]. It is the extended version of Leo focusing mainly on rigid components with longest-first strategy.
- **Goun** [33]. It is a state-of-the-art structured-based method focusing mainly on unfolding model structure without considering its behaviors.

5.1.3 Parameter Settings. We implemented BePT based on jBPT. An easy-to-use version of BePT is also publicly available. We include an editable parameter for defining the size of a paragraph and predefine this parameter with a value of 75 words. Once this threshold is reached, we use a change of the performing role or an intermediate activity as indicator and respectively introduce a new paragraph. Besides, we use the default language grammar style of subject-predicate-object and object-be-predicated-by-subject to express a sentence [20, 21, 33]. Finally, we set all parameters valid for all methods, i.e., to generate intact textual descriptions without any reduction.

5.2 Results

5.2.1 Capability (RQ1).

As discussed earlier, a rigid is a region that captures arbitrary model structure. Thus, these seven datasets are representative enough as the SMR varies from 100% (structured easy models) to 28.95% (unstructured complex models). We analyzed and compared all process models. Table 2 reports their handling capabilities w.r.t some representative complex patterns [25, 33].

First, we can see that BePT shows the best handling capabilities. Among the 14 patterns, BePT can handle them all, which is better than Goun that can handle 9 patterns. Second, four methods can handle structured models well, while Goun and BePT can handle unstructured models, and BePT can further provide extra helpful messages. Third, the R and the Extra parts show that BePT can handle rigid of arbitrary complexity even the model is unsymmetrical, non-free-choice or multi-layered. From these results we can conclude that the behavior-based method BePT is the best one.

5.2.2 Detailedness (RQ2).

In the sentence planning phase, BePT checks the grammatical correctness of each DSynT so that the generated text can accord with correct English grammar, including various conjunctions, adverbial clauses and transitional sentences. Here, instead of comparing the grammatical correctness, we summarize the structural characteristics of all generated texts in Table 3.

A general observation is that BePT texts are longer than the other texts. Leo texts, Hen texts, Goun texts contain an average of 66.7, 78.0 and 79.3 word length and 17.2, 19.6, 19.9 sentence length respectively, while BePT texts include an average of 115.3 words and 22.3 sentences. However, this does not imply that BePT texts are verbose, using longer sentence to describe the same content. Rather, the main reason is that Leo, Hen, Goun ignore some modeling-level messages related to soundness and safety [37, 39], but BePT supplements them.

Therefore, we conclude that BePT generates more detailed messages on account of additional useful information. Certainly, although all parameters are set to be valid in this experiment, BePT is actually configurable, i.e., users can set parameters to determine whether to generate these complementary details or not.

5.2.3 Consistency (RQ3).

We always hold the belief that the textual paragraphs will hugely influence readability since paragraph indentation can reflect the number of components, the modeling depth of each activity, etc. Considering the generated text of the motivating example in Section 3, if the text contains no paragraph indentation (each paragraph starts from the bullet point "•"), it will be much harder to fully reproduce the model semantics [21, 33].
In this part, we aim at the detection of structural consistency between a process model and its corresponding textual descriptions. This task requires an alignment of a model and a text, i.e., activities in the texts need to be related to model elements and vice versa [7, 42]. For an activity $T$, its modeling depth $md(T)$ is the RPST depth of $T$, and its description depth $dd(T)$ is how deep it is indented in a text. For the activity set of a model, the modeling depth distribution is denoted as $X = \{md(T_1), md(T_2), \ldots, md(T_n)\}$ and the description depth distribution is denoted as $Y = \{dd(T_1), dd(T_2), \ldots, dd(T_n)\}$. We employ a correlation coefficient to evaluate the consistency between two distributions of $X$ and $Y$ as follows:

$$\rho(X, Y) = \frac{E[(X - E(X))(Y - E(Y))]}{\sqrt{D(X) \cdot D(Y)}} \in [-1.0, 1.0]$$  

where $E$ is the expectation function and $D$ is the variance function. The value of the $\rho(X, Y)$ function ranges from -1.0 (negatively related) to 1.0 (positively related).

Figure 10 shows the consistency results of four P2T methods. First, BePT gets the highest consistency value in every dataset, meaning that BePT positively follows the depth distribution of original models to the maximum extent. Notice that all methods obtain 1.00 consistency on the SAP dataset since all SAP models are structured. However, on the SPM dataset, BePT achieves 0.86 consistency, while the other methods are only at around 0.25. The main reason is that SPM contains plenty of close-to-structured rigid which directly reflects the other methods’ drawbacks. Second, with lower SMR, the consistency performance rapidly decreases. The most obvious updates occur in GPM and BAI where Leo, Hen and Goun even get negative coefficient values, which demonstrates that they negatively relate the distribution of the original models even causing the opposite distribution, while BePT obtains 0.42 and 0.25 which shows that BePT is still positively related even while facing unstructured situations. Hence, we conclude that BePT texts conform better to the original models.

### Table 1: Statistics of the evaluation datasets.

| Source | Type     | N    | SMR ↓ | Place min ave max | Transition min ave max | Arc min ave max | RPST depth min ave max |
|--------|----------|------|-------|-------------------|------------------------|-----------------|------------------------|
| SAP    | Industry | 72   | 100.00% |
| DG     | Industry | 38   | 94.74%  |
| TC     | Industry | 49   | 81.63%  |
| SPM    | Academic | 14   | 57.00%  |
| IBM    | Industry | 142  | 53.00%  |
| GPM    | Academic | 36   | 42.00%  |
| BAI    | Academic | 38   | 28.95%  |

Table 2: The handling capabilities of four P2T methods w.r.t. some representative patterns.

| Type | Pattern                  | Leo | Hen | Goun | BePT |
|------|--------------------------|-----|-----|------|------|
| T, B, P | Trivial                  | ✓   | ✓   | ✓    | ✓    |
|        | Polygon                  | ✓   | ✓   | ✓    | ✓    |
|        | Easy Bond                | ✓   | ✓   | ✓    | ✓    |
|        | Unsymmetrical Bond       |     |     | ✓    | ✓    |
| R     | Place Rigid              | ✓   | ✓   | ✓    | ✓    |
|        | Transition Rigid         | ✓   | ✓   | ✓    | ✓    |
|        | Mix Rigid                |     |     | ✓    | ✓    |
|        | Intersectant Loop        | ✓   | ✓   | ✓    | ✓    |
|        | Non-free-choice Construct| ✓   | ✓   | ✓    | ✓    |
|        | Invisible or Duplicated Task| | | | |
|        | Multi-layered Embedded   | ✓   | ✓   | ✓    | ✓    |
| Extra | Modeling Information     | ✓   | ✓   | ✓    | ✓    |
|        | Multi-layered Paragraph  | ✓   | ✓   | ✓    | ✓    |

Table 3: Average number of words and sentences per text. Red numbers denote the maximum and green numbers denote the minimum per dataset.

| Words/Text | Leo | Hen | Goun | BePT |
|------------|-----|-----|------|------|
| SAP        | 38.0| 38.0| 38.1 | 38.1 |
| DG         | 74.0| 79.7| 79.6 | 85.3 |
| TC         | 99.2| 110.8| 112.4 | 135.0 |
| SPM        | 41.5| 54.1| 55.6 | 100.9 |
| IBM        | 140.2| 180.7| 182.9 | 191.9 |
| GPM        | 38.3| 50.8| 53.8 | 147.0 |
| BAI        | 25.7| 31.7| 32.6 | 111.3 |
| Total      | 66.7| 78.0| 79.3 | 115.3 |

| Sentences/Text | Leo | Hen | Goun | BePT |
|----------------|-----|-----|------|------|
| 6.0            | 6.0 | 6.2 | 6.2  |
| 13.0           | 15.0| 15.0| 15.7 |
| 12.2           | 15.5| 15.7| 18.7 |
| 5.8            | 7.9 | 8.1 | 11.1 |
| 74.2           | 80.7| 81.7| 86.2 |
| 6.2            | 7.5 | 7.9 | 16.2 |
| 2.7            | 4.4 | 4.6 | 15.7 |
| 17.2           | 19.6| 19.9| 22.3 |

Figure 10: The consistency distribution. The red color denotes the positive coefficient while the blue color denotes the negative coefficient.
be reproduced, we can believe that the text $T$ contains enough information to reproduce the original model, i.e., excellent reproducibility.

We evaluate the P2T performance using the $F_1$ measure (harmonic average of recall and precision) which is inspired from data mining field [15]:

$$F_1 = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \in [0.0, 1.0]$$

where $\beta$ is the balance weight. In our experiments, equal weights ($\beta = 1.0$) are assigned to balance recall and precision. The higher the $F_1$ is, the better the reproducibility is.

**Structural Reproducibility.** Figure 12 shows the results of four dimensions (place, transition, gateway, element).

![Figure 11: The graphical representation of information gain line and the perplexity distributions.
](image)

![Figure 12: The $F_1$ measures on structural dimensions.
](image)

where $T'$ is the described activity set and $U'$ is the neglected activity set. This formula employs the information entropy $|T'| \log_2 |T'|$ to describe the confusion of all activities in a paragraph. Its exponent value has a same magnitude of $|T'|$. We notice that if any activity cannot be generated in the text, the text system should reduce the understandability value with the original model, i.e., improve the perplexity of the text system, hence, it multiplies $|U'|$.

When describing single paragraph $S_1$, the information gain [15] of the text system is $\psi(S_1)$. After describing paragraph $S_2$, the information gain changes to $\psi(S_1) + \psi(S_2)$. Similarly, after describing all paragraphs, the information gain is $\Sigma_{i=1}^{n} \psi(S_i)$. These values are mapped to $n$ points $(i, \Sigma_{k=1}^{i} \psi(S_k))_{i=1}^{n}$ shown in Figure 11(a). We call the broken line linking all points information gain line $IGL((M, T), S)$. Then, we can define the perplexity of the text system $T$ (the integral of all sentence perplexity):

$$\psi(S_i) = e^{\log_2 |T'| \cdot |U'|}$$

$$\text{perplexity}(\langle M, T \rangle) = \int_0^n IGL((M, T), s)ds, s \in \mathbb{R}$$

$IGL(S)$ intuitively measures whether the model-text pair system is understandable. We calculated this metric for each datasets and reported the results.

Figure 11b shows the perplexity results. We can see that BePT achieves the lowest perplexity in all datasets, i.e., best understandability. On average, the perplexity has been reduced from $10^2.74$ to $10^0.98$. This results also show that the perplexity trend is Leo $\geq$ Hen $\geq$ Goun $\geq$ BePT, i.e., the understandability trend is Leo $\leq$ Hen $\leq$ Goun $\leq$ BePT.

5.2.5 Reproducibility (RQ5). This part evaluates the reproducibility of the generated text, i.e., did the original model be reproduced from the generated text?

For each model-text pair $\langle M, T \rangle$, we manually back-translate (extract) the process model from the generated text and compare the elements between the original and the extracted models. All back-translators are provided only the generated texts without them knowing any information of the original models. They reproduce the original models from the texts according to their own understanding. After translation, we evaluate the structural and behavioral reproducibility between the original model and the extracted one. If an isomorphic model $M$ can be reproduced, we can believe that the text $T$ contains enough information to reproduce the original model, i.e., excellent reproducibility.

We evaluate the P2T performance using the $F_1$ measure (harmonic average of recall and precision) which is inspired from data mining field [15]:

$$F_1 = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \in [0.0, 1.0]$$

where $\beta$ is the balance weight. In our experiments, equal weights ($\beta = 1.0$) are assigned to balance recall and precision. The higher the $F_1$ is, the better the reproducibility is.

**Structural Reproducibility.** Figure 12 shows the results of four dimensions (place, transition, gateway, element).

![Figure 12: The $F_1$ measures on structural dimensions.
](image)

First, we can see that the $F_1$ value of the four methods falls from 100% to a lower value w.r.t. decreasing SMR. For GPM and BAI datasets, Leo achieves only around 40%. The low-value cases significantly affect the ability to understand or reproduce the original model, and it reflects the general risk that humans may miss
elements when describing a model, i.e., they lose around 60% information. Still, Hen achieves around 90% while BePT hits 100%, i.e., Goun and BePT lose least information. We can conclude that, among the four P2T methods, BePT achieves highest reproducibility, followed by Goun and then Hen. The structural reproducibility performance also shows the trend, Leo ≤ Hen ≤ Goun ≤ BePT.

Behavioral Reproducibility. Behavioral reproducibility aims to evaluate the extent of correctly expressed behavior, i.e., how many correct behaviors are expressed in the generated texts. We also use $F_1$ to evaluate behavioral performance. In this part, we use TAR (local) and trace (global) to reflect the model behaviors. As trace behaviors exist space explosion problem, thus, for trace $F$-measure, we only evaluate these models without loop behavior.

Figure 13 shows the results for the behavior dimension (TAR, trace). The results show that BePT outperforms Leo, Hen and Goun significantly in terms of both TAR and trace performance. Leo performance falls sharply with decreasing SMR, while Hen and Goun drop more gently than Leo and they achieve around 70% on BAI for trace $F_1$. BePT gets the highest $F_1$ of around 100% for both TAR and trace measures, and BePT also produces a distinct improvement on TAR and trace $F_1$ over other methods. From these two performance results, we can conclude that BePT showcases the best reproducibility over the state-of-the-art P2T methods.

6 CONCLUSION AND FUTURE WORK

We present a behavior-based process translator. It first embed RDT and decode it by extracting the behavior paths. Then, we use NL tools to generate the textual descriptions. Our experiments show the significant improvements on capability, detailedness, consistency, understandability and reproducibility. This approach can unlock the hidden value that lies in large process repositories in the cloud, and make them more reusable.

Furthermore, empirical testing shows that BePT handles models within 1836ms on average (efficiency). Besides, BePT is model-independent and language-independent (expandability). The adaptation to other modeling languages requires model transformation and the replacement of corresponding elements, while adaptation to other languages only requires the replacement of three resources: the NL analysis, the DSynT templates and the realizer. BePT not only can be applied in cloud-sharing scenarios but also in engineering analysis [4], or integrated into real-life intelligent devices.

We also list some potential limitations of this study. Above all, when the model is unsound, BePT informs the user that the model contains non-sound or wrong parts but without giving any correction advice. Another drawback concerns manual extraction of the NL text because of the limited number of the participants. We cannot guarantee that each extraction rule for a generated text is identical. Thus, generating the correction advice and automatic reverse translation would also be of interest in future studies.

APPENDICES

A THE PROOF OF BEHAVIOR CORRECTNESS

**Property 1.** Given a net system $S = (N, M)$ and its TAR set $T(S)$. The behavior path set $P$ of its CFP $N$ by the linking rule (Definition 6) satisfies behavior correctness, $\forall P \in T(S)$.

**Proof.** 1. Given two Petri nets $N_i = (P_i, T_i, F_i), N_j = (P_j, T_j, F_j)$, we assume $P = iS_1, S_2, \ldots, S_{n+1}$. Then, consider two situations: a) inside a single segment; b) between the linking of two segments:

   a) The initial (default) marking $S$ is also the initial marking of $h(S_1)$, i.e., the marking $S^*_{\tau_1}$ is reachable. According to the definition of behavior segment, $S^*_{\tau_1}$ is reachable from $S^*_{\tau_1}$, and the firing rule guarantees $T(S_{\tau_1}) \subseteq T(S_1)$. After executing $S^*_{\tau_1}$, $S^*_{\tau_2}$ is reachable as $S^*_{\tau_1} \subseteq S_2$, so that $T(S_2) \subseteq T(S)$ holds. Similarly, $iS_1, S_2, \ldots, S_{n+1}$, $iS_1, S_2, \ldots, S_{n+1}$ holds. We can use the notation $T(S_{j1} S_{j2} \ldots S_{jn+1})$ to denote the TAR set in the joint points, i.e., $T(S_{j1} S_{j2} \ldots S_{jn+1}) = \{a < b \mid a e S_{j1} \land b \in (S_{j1} S_{j2} \ldots S_{jn+1})\}$. Since $S^*_{\tau_1} \subseteq S_{\tau_1}$ guarantees that $(S_{\tau_1})^*$ can be fired after firing $(S_{\tau_1})^*$, i.e., $T(S_{j1} S_{j2} \ldots S_{jn+1}) \subseteq T(S)$. Therefore, $T(S_{j1} S_{j2} \ldots S_{jn+1}) = T(S_{j1}) \cup T(S_{j2}) \cup \ldots \cup T(S_{jn+1}) \subseteq T(S)$, $i, j = 1, 2, \ldots, n+1$.

   b) For two segments $S_i, S_{i+1}, i = 1, 2, \ldots, n-1$, we use the notation $T(S_i S_{i+1})$ to denote the TAR set in the joint points, i.e., $T(S_i S_{i+1}) = \{a < b \mid a e S_i \land b \in (S_i S_{i+1})\}$. Since $S_i \supseteq S_{i+1}$ guarantees that $(S_{i+1})^*$ can be fired after firing $(S_i)^*$, i.e., $T(S_i S_{i+1}) \subseteq T(S)$. According to the above two points, we can conclude that $\forall P \in P \Rightarrow T(P) = T(S_1) \cup T(S_2) \cup \ldots \cup T(S_n) \cup T(S_1 S_2) \cup T(S_2 S_3) \ldots \cup T(S_{n-1} S_n) \subseteq T(S)$.

B THE PROOF OF BEHAVIOR COMPLETENESS

**Property 2.** Given a net system $S = (N, M)$ and its TAR set $T(S)$. The behavior path set $P$ of its CFP $N$ by the linking rule (Definition 6) satisfies behavior completeness, $\forall \tau \in T(S) \Rightarrow \exists P \in P, \tau \in T(P)$.

**Proof.** 2. For any TAR $\tau = a < b \in T(S)$, the place set $a^* \cap b$ is denoted as $P$. The sub-model $(P, \{a, b\}, \{a \times P \cup P \times b\})$ is denoted as $N$. We use $N_i \propto N_j$ to denote $P_i \subseteq P_j \land T_i \subseteq T_j \land F_i \subseteq F_j$, i.e., $N_i$ is a sub-model of $N_j$. Then, consider the following situations:

   a) When $\forall P \in P, P \notin SP(N)$, there is no $P \in P$ that can be the boundary node of a segment according to Definition 5 ($SP$-bounded). Hence, $N$ can only exist in the middle of a segment, i.e., $\exists S_i, P_j \Rightarrow N \propto S_i P_j \notin P \Rightarrow \exists \tau \in T(N) \subseteq T(P)$.  

   b) When $\forall P \in P, P \in SP(N)$, $P$ is split, being the sink set of a certain segment $S_i$ and the source set of a certain segment $S_j$, $SP$-bounded, i.e., $P = S^*_{\tau} \supseteq S$ always holds. Hence, $\exists S_i, S_j, P_k \Rightarrow N \propto S_i S_j P_k \notin P \Rightarrow \exists \tau \in T(N) \subseteq T(P)$.  

   c) When $\exists P_1, P_2 \in P, P_1 \notin SP(N), P_2 \in SP(N)$, there is no $P \in P$ can be the boundary node of a segment, or it contradicts Definition 5 (reply-hold). Hence, $N$ can only exist in the middle of a segment, i.e., $\exists S_i, P_j \Rightarrow N \propto S_i P_j \notin P \Rightarrow \exists \tau \in T(N) \subseteq T(P)$.  

   d) When $P = \emptyset$, i.e., $a$ and $b$ are in a concurrent relation. There always exists a concurrent split transition $t$. According to Definition
C THE PROOF OF DESCRIPTION MINIMALITY

Property 3. For a net system $S = (N,M)$, the pruned TARs $\mathcal{T}(S)$ by the Pruning Rule satisfies description minimality.

Proof 3. According to Appendices B, for any TAR $\tau$, it can always be derived from a certain behavior path, i.e., $\forall \tau \in \mathcal{T}(S) \Rightarrow \exists \mathcal{P} \in \mathcal{P}, \tau \in \mathcal{T}(\mathcal{P}).$ Hence, for two TARs $\tau_1, \tau_2$ of the model with $\tau_1 \in \mathcal{T}(\mathcal{P}_1) \wedge \tau_2 \in \mathcal{T}(\mathcal{P}_2)$, $\tau_1 \neq \tau_2$. If $\tau_1 \neq \tau_2$, $\{\tau_1, \tau_2\} \subseteq \mathcal{T}(S)$ always holds, while $\{\tau_1\} \subseteq \mathcal{T}(S)$ always holds if $\tau_1 = \tau_2$. Therefore, the pruning rule always keeps TARs appearing at the first time, i.e., $\mathcal{T}(S)$ satisfies behavior minimality.

REFERENCES

[1] Banu Aysolmaz, Henri Leopold, Hajo A. Reijers, and Onur Demirörs. 2018. A semi-automated approach for generating natural language requirements documents based on business process models. Information and Software Technology 93 (2018), 1–29. https://doi.org/10.1016/j.infsof.2017.08.009

[2] Miguel Ballesteros, Bernd Bohnet, Simon Mille, and Leo Wanner. 2014. Deep Syntactic Parsing. In Proceedings of CoLING 2014, the 25th International Conference on Computational Linguistics. Technical Papers. Dublin City University and Association for Computational Linguistics, 1402–1413. http://www aclweb.org/anthology/C14-1133

[3] Sarah Cohen Boulakia and Ulf Leser. 2011. Search, adapt, and reuse: the future of the original model with $\tau_1 \in \mathcal{T}(\mathcal{P}_1) \wedge \tau_2 \in \mathcal{T}(\mathcal{P}_2)$, $\tau_1 \neq \tau_2$. If $\tau_1 \neq \tau_2$, $\{\tau_1, \tau_2\} \subseteq \mathcal{T}(S)$ always holds, while $\{\tau_1\} \subseteq \mathcal{T}(S)$ always holds if $\tau_1 = \tau_2$. Therefore, the pruning rule always keeps TARs appearing at the first time, i.e., $\mathcal{T}(S)$ satisfies behavior minimality.

REFERENCES

[1] Banu Aysolmaz, Henri Leopold, Hajo A. Reijers, and Onur Demirörs. 2018. A semi-automated approach for generating natural language requirements documents based on business process models. Information and Software Technology 93 (2018), 1–29. https://doi.org/10.1016/j.infsof.2017.08.009

[2] Miguel Ballesteros, Bernd Bohnet, Simon Mille, and Leo Wanner. 2014. Deep Syntactic Parsing. In Proceedings of CoLING 2014, the 25th International Conference on Computational Linguistics. Technical Papers. Dublin City University and Association for Computational Linguistics, 1402–1413. http://www aclweb.org/anthology/C14-1133

[3] Sarah Cohen Boulakia and Ulf Leser. 2011. Search, adapt, and reuse: the future of the original model with $\tau_1 \in \mathcal{T}(\mathcal{P}_1) \wedge \tau_2 \in \mathcal{T}(\mathcal{P}_2)$, $\tau_1 \neq \tau_2$. If $\tau_1 \neq \tau_2$, $\{\tau_1, \tau_2\} \subseteq \mathcal{T}(S)$ always holds, while $\{\tau_1\} \subseteq \mathcal{T}(S)$ always holds if $\tau_1 = \tau_2$. Therefore, the pruning rule always keeps TARs appearing at the first time, i.e., $\mathcal{T}(S)$ satisfies behavior minimality.

REFERENCES
[40] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. 2008. The Refined Process Structure Tree. In Business Process Management, Marlon Dumas, Manfred Reichert, and Ming-Chien Shan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 100–115.

[41] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015), 3156–3164.

[42] M. Weidlich, J. Mendling, and M. Weske. 2011. Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. IEEE Transactions on Software Engineering 37, 3 (2011), 410–429. https://doi.org/10.1109/TSE.2010.96

[43] Huijun Wu, Chen Wang, Jie Yin, Kai Lu, and Liming Zhu. 2018. Sharing Deep Neural Network Models with Interpretation. In Proceedings of the 2018 World Wide Web Conference (WWW ’18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 177–186. https://doi.org/10.1145/3178876.3185995