Bridging Domain Knowledge and Process Discovery Using Large Language Models (prompts and models)

Ali Norouzifar[1], Humam Kourani[2,1], Marcus Dees[3], and Wil van der Aalst[1]

[1] RWTH University, Aachen, Germany
{ali.norouzifar, wvdaalst}@pads.rwth-aachen.de
[2] Fraunhofer FIT, Sankt Augustin, Germany
humam.kourani@fit.fraunhofer.de
[3] UWV Employee Insurance Agency, Amsterdam, Netherlands
Marcus.Dees@uwv.nl

1 Prompts

Process discovery is a type of process mining aiming to represent the process model explaining how the process is executed. Our focus is on the discovery of an imperative process model. Recently, we developed a framework that allows for some declarative constraints as input to discover better imperative models. Declarative models are close to the natural language, however for a domain expert, they might not be familiar. Therefore, we need your help to translate the process description given as a text to the declarative constraints we need for the discovery of imperative process models.
Consider only the following declarative constraint definitions where a, b, c, and d are activities and template example-template(x,y) specifies a template named "example-template" and x and y can be on of the activities from the set of all activities. Do not reason based on the template names and only use explanations to understand the meaning of specific templates:

- at-most(a): a occurs at most once. Some examples satisfying this constraint:
  1. The process starts with activity b for setup, followed by a for approval, then c for data collection, and ends with d for implementation. Optional execution of b can occur again before c.
  2. The workflow begins with c for initial data collection, followed by d for setup, then a for mandatory approval, and concludes with b for detailed planning.
  3. The process initiates with d for setup, followed by b for planning, c for data analysis, and finally a for essential approval. Optional execution of c may occur again after b.

Some examples violating this constraint:

1. The process starts with a for initial approval, followed by b for planning, then loops back to a for re-approval, and ends with c for data analysis and d for implementation.
2. Begin with c for data collection, followed by a for approval, then b for planning, another a for re-approval, and ends with d for execution.
3. The workflow involves d for setup, followed by parallel execution of a for initial approval and b for planning, then a second occurrence of a for final approval, and c for final data analysis.

– existence(a): a occurs at least once, which means the existance of activity a is mandatory. Some examples satisfying this constraint:
  1. The process starts with activity b for initial setup, followed by a for mandatory approval, then c for data collection, and finally d for implementation. Optional execution of c can occur again before d.
  2. The workflow begins with d for setup, followed by a for critical approval, then parallel execution of b for planning and c for analysis, and concludes with d for finalization.
  3. Activity c starts with data gathering, then loops to b for planning, followed by a for essential approval, and ends with d for implementation.
  4. all the customers start the process with activity a

  Some examples violating this constraint:
  1. The process starts with b for initial tasks, followed by c for data analysis, and concludes with d for implementation. Optional execution of c can occur again before d.
  2. Begin with d for setup, followed by b for detailed planning, then c for further analysis, and ends with d for final execution. 3 The workflow involves c for initial data collection, followed by parallel execution of b for planning and d for implementation, and ends with another round of c.
  3. Either one of the activities a or b occurs. Each trace can only have a or b.
– response(a,b): If a occurs, then b occurs after a. Some examples satisfying this constraint:
  1. The process starts with c for data collection, then a for approval, followed by b for planning, and finally d for implementation. Optional execution of c may occur again before d.
  2. Activity d begins with preliminary tasks, then a for approval, followed by parallel execution of b for planning and c for analysis, ending with d for finalization.
  3. The workflow initiates with a for approval, loops back to a second approval if needed, then proceeds to b for detailed planning, and concludes with c for execution.

4. all the occurrences of activity a should be followed by activity b

Some examples violating this constraint:

1. The process starts with b for initial setup, followed by c for data analysis, then a for approval, and ends with d for implementation.
2. Begin with c for data gathering, followed by a for approval, then d for execution, and optionally b for post-implementation review.
3. The process involves d for setup, followed by a for approval, and then c for analysis. An optional activity b for final planning may or may not occur.

– precedence(a,b): b occurs only if preceded by a. Some examples satisfying this constraint:

1. The process begins with activity a for approval, followed by c for data collection, and then b for planning. Optional execution of d can occur anytime before or after b.
2. Activity d initiates the setup, then a for mandatory approval, followed by parallel execution of c for analysis and b for planning, and concludes with d for finalization.
3. The workflow starts with c for initial tasks, followed by a for approval, then b for detailed planning, and ends with a loop back to c for further analysis if needed.
4. after execution of activity a, some cases will continue with activity b and some cases will continue with activity c

Some examples violating this constraint:

1. The process starts with c for data gathering, followed by b for planning, then a for approval, and ends with d for implementation.
2. Begin with d for initial setup, followed by parallel execution of b for planning and c for analysis, and finally a for approval.
3. The workflow involves d for setup, followed by c for data collection, b for planning, and ends with an optional activity a for final approval.

– co-existence(a,b): a and b occur together. Some examples satisfying this constraint:

1. The process starts with b for initial setup, followed by a for approval, then c for data collection, and ends with d for implementation. Optional execution of b can occur again before c.
2. Activity a begins with approval, followed by parallel execution of b for planning and c for data analysis, and concludes with d for finalization.
3. The workflow starts with d for setup, followed by a for approval, b for planning, and ends with c for execution. Optional execution of c may occur again after b.

Some examples violating this constraint:

1. The process begins with c for data collection, followed by d for setup, then b for planning, and ends with another round of c.

2. Start with d for setup, followed by c for data analysis, then a for approval, and ends with c for final review.
3. The workflow involves c for initial data gathering, followed by parallel execution of d for setup and b for planning, and concludes with c for finalization.

– not-co-existence(a,b): a and b never occur together. Some examples satisfying this constraint:
1. The process starts with a for approval, followed by c for data collection, then d for implementation. Optional execution of c can occur again before d.
2. Begin with b for initial setup, followed by c for analysis, then d for execution. An optional activity c can repeat after d.
3. The workflow starts with c for data gathering, then moves to d for setup, followed by optional repetitions of c.
4. in our process activities a and b cannot occur together
5. some cases have activity a and the other cases have activity b
6. if activity a does not occur, then activity b, and c will occur

Some examples violating this constraint:
1. The process begins with a for approval, followed by b for planning, then c for data collection, and ends with d for implementation.
2. Start with c for data analysis, then a for approval, followed by d for execution, and ends with b for final review.
3. The workflow involves d for setup, followed by parallel execution of a for approval and b for planning, and concludes with c for finalization.

– not-succession(a,b): b cannot occur after a. Some examples satisfying this constraint:
1. The process starts with a for approval, followed by c for data collection, and ends with d for implementation. Optional execution of c can occur again before d.
2. Begin with b for initial setup, followed by d for setup, then a for approval, and finally c for analysis.
3. The workflow starts with c for data gathering, followed by a for approval, then d for execution, and optional repetitions of c.

Some examples violating this constraint:
1. The process begins with a for approval, followed by b for planning, then c for data collection, and ends with d for implementation.
2. Start with c for data analysis, then a for approval, followed by d for execution, and ends with b for final review.
3. The workflow involves d for setup, followed by parallel execution of a for approval and c for analysis, then b for planning, and concludes with c for finalization.

– responded-existence(a,b): If a occurs in the trace, then b occurs as well. Some examples satisfying this constraint:

1. The process starts with b for initial setup, followed by a for approval, then c for data collection, and ends with d for implementation. Optional execution of b can occur again before c.
2. Begin with a for initial approval, followed by d for setup, then b for planning, and finally c for analysis.
3. The workflow starts with c for data gathering, followed by a for approval, then d for execution, and optional repetitions of b for planning.

Some examples violating this constraint:

1. The process begins with a for approval, followed by c for data collection, then d for implementation. Optional execution of c can occur again before d.
2. Start with c for data analysis, followed by a for approval, then d for execution, and ends with c for final review.
3. The workflow involves d for setup, followed by a for approval, then parallel execution of c for analysis and b for planning, and concludes with c for finalization.

Some more instructions:

– It is not possible to generate constraints like response(a, (b or c)). The first and second elements must be a single activity

For each task, I provide the set of activity labels that exist in the process with a short description. Then, I present a text written by a process expert and want you to translate it to declarative constraints and write it in a plaintext block.

Sure, please provide the activity labels along with their descriptions and the text written by the process expert so that I can help you translate it into declarative constraints.

Here is the list of activities:
Receive Claim, Start Claim, Block Claim 1, Block Claim 2, Block Claim 3, Correct Claim, Unblock Claim 1, Unblock Claim 3, Reject Claim, Receive Objection 1, Receive Objection 2, Accept Claim, Payment Order, Execute Payment, Withdraw Claim, Repayment
Now extract declarative rules and write them in formal language from this process description:
"Our process is designed to handle claim applications within an insurance agency, where applicants request various types of benefits, and our system processes these applications through a structured workflow. There are three types of blocking mechanisms: Block Claim 1, Block Claim 2, and Block Claim 3, each with corresponding resolutions: Unblock Claim 1, Unblock Claim 2, and

Unblock Claim 3. The process begins once a claim is received. Block Claim 1 occurs when initial issues are detected, leading to necessary corrections before the claim is unblocked and accepted, resulting in the client receiving between one to three payments. Block Claim 2 is applied when the claim is deemed ineligible for receiving any payments, resulting in rejection and potentially an objection from the client. Block Claim 3 is triggered if there is missing information, such as incorrect bank account details, after a claim is accepted; the client is notified to provide the correct details, and the claim is unblocked upon receipt of this information. Additionally, clients may file an objection after receiving payments, leading to the withdrawal of the claim and repayment of the received money to the agency."
Consider this text and ask me questions in order to extract highly reliable declarative constraints.

To accurately extract highly reliable declarative constraints from the provided process description, I need to clarify a few details about the sequence and exclusivity of some activities:

1. Can "Block Claim 1," "Block Claim 2," and "Block Claim 3" occur in the same process instance, or are they mutually exclusive?
2. Are the corresponding unblocking activities ("Unblock Claim 1," "Unblock Claim 2," and "Unblock Claim 3") strictly linked to their respective blocking activities, or can there be other sequences?
3. Is there any restriction on the order of "Receive Objection 1" and "Receive Objection 2," or can they occur independently of each other?
4. Can "Correct Claim" occur multiple times, or is it limited to once per process instance?
5. Are there any specific constraints related to the sequence of "Withdraw Claim" and "Repayment" activities?
6. Is "Accept Claim" always followed by "Payment Order" and "Execute Payment," or can these activities occur in different orders or independently?
Once I have these details, I can provide a set of precise declarative constraints for the process.

1. Some cases may have both Block Claim 1 and Block Claim 3. But Block Claim 2 can never occur in the trace with Block Claim 1 or Block Claim 3.
2. Each blocking mechanism (Block Claim 1, Block Claim 2, Block Claim 3) has a corresponding unblocking mechanism (Unblock Claim 1, Unblock Claim 2, Unblock Claim 3), and each pair (Block, Unblock) occurs in a strict sequence.
3. Receive Objection 2 only occurs in the rejected cases and Receive Objection 1 only occurs in the accepted cases after they receive some payments. In general, Receive Objection 1 and Receive Objection 2 cannot occur together.

4. Correct Claim only occurs once and is only relevant for Block Claim 1.
5. Withdraw Claim should always be followed by Repayment.
6. Payment Order and Execute Payment only occur when the claim is accepted.

Based on the clarified details, here are the declarative constraints for the process:
not-co-existence(Block Claim 2, Block Claim 1)
not-co-existence(Block Claim 2, Block Claim 3)
co-existence(Block Claim 1, Unblock Claim 1)
co-existence(Block Claim 2, Unblock Claim 2)
co-existence(Block Claim 3, Unblock Claim 3)
precedence(Block Claim 1, Unblock Claim 1)
precedence(Block Claim 2, Unblock Claim 2)
precedence(Block Claim 3, Unblock Claim 3)
not-co-existence(Receive Objection 1, Receive Objection 2)
precedence(Reject Claim, Receive Objection 2)
precedence(Payment Order, Receive Objection 1)
at-most(Correct Claim)
precedence(Block Claim 1, Correct Claim)
precedence(Correct Claim, Unblock Claim 1)
response(Withdraw Claim, Repayment)
responded-existence(Accept Claim, Payment Order)
responded-existence(Payment Order, Execute Payment)
These constraints ensure that the process adheres to the specified rules and behaviors, creating a robust framework for discovering imperative process models.

The discovered process model is interesting but we observe some issues. For example, Receive Objection 1 should occur after all the payments are executed. This time of objection can occur at most one time per claim. Withdraw Claim can not be followed by another payment. Usually, after the payments are executed, the applicant has the option to send an objection, withdraw the claim, and repay the received benefits. Withdraw Claim only occurs at most once per claim.

Thank you for the additional details. Based on the new information, here are the updated and additional declarative constraints for the process:
not-co-existence(Block Claim 2, Block Claim 1)
not-co-existence(Block Claim 2, Block Claim 3)
co-existence(Block Claim 1, Unblock Claim 1)
co-existence(Block Claim 2, Unblock Claim 2)
co-existence(Block Claim 3, Unblock Claim 3)
precedence(Block Claim 1, Unblock Claim 1)
precedence(Block Claim 2, Unblock Claim 2)
precedence(Block Claim 3, Unblock Claim 3)

not-co-existence(Receive Objection 1, Receive Objection 2)
precedence(Reject Claim, Receive Objection 2)
precedence(Payment Order, Receive Objection 1)
at-most(Correct Claim)
precedence(Block Claim 1, Correct Claim)
precedence(Correct Claim, Unblock Claim 1)
response(Withdraw Claim, Repayment)
responded-existence(Accept Claim, Payment Order)
responded-existence(Payment Order, Execute Payment)
precedence(Execute Payment, Receive Objection 1)
at-most(Receive Objection 1)
not-succession(Withdraw Claim, Payment Order)
not-succession(Withdraw Claim, Execute Payment)
at-most(Withdraw Claim)
These constraints incorporate the new requirements and correct the issues you observed.

2 Discovered Process Models

2.1 Discovered model with using IMf with $f = 0.2$

2.2 Discovered model using IMr with $sup = 0.2$ without LLMs

2.3 Discovered model using IMr with $sup = 0.2$ and the rules provided by ChatGPT

2.4 Discovered model using IMr with $sup = 0.2$ and the rules provided by ChatGPT after incorporating the domain expert feedback

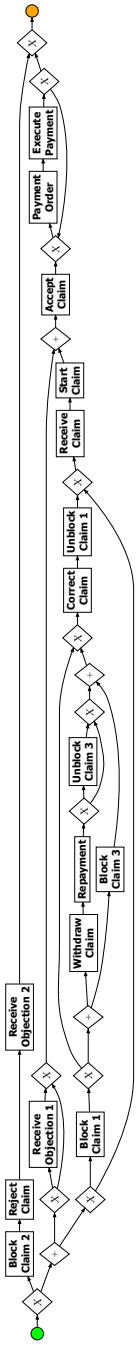

Fig. 1: Discovered model with using IMf with $f = 0.2$.

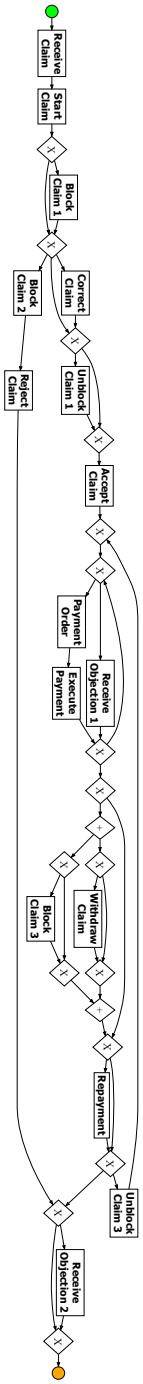

Fig. 2: Discovered model using IMr with $sup = 0.2$ without LLMs.

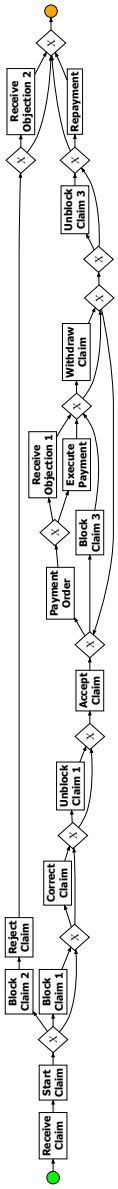

Fig. 3: Discovered model using IMr with $sup = 0.2$ and the rules provided by ChatGPT.

Fig. 4: Discovered model using IMr with $sup = 0.2$ and the rules provided by ChatGPT after incorporating the domain expert feedback.