**Process Mining And Simulation**

**Assignment No 3&4**

**Course Instructor**

Madam Behjat Zuhaira

**Group Members**

Abdullah Daoud……………………………………………………….(22I-2626)

Faizan Rasheed………………………………………………………..(22I-2734)

Dawood Hussain……………………………………………………….(22I-2410)

Talha Mukhtiar………………………………………………………..(22I-2720)

**Section**

SE-E

**Date**

**Wednesday, November 27, 2024**

**Fall 2024**

**Department of Software Engineering**

FAST – National University of Computer & Emerging Sciences

Islamabad Campus

Table of Contents

[1.](#_heading=h.30j0zll) Overview: 3

[1. Preprocessing Steps 3](#_heading=h.1fob9te)

[2. Design Choices 3](#_heading=h.3znysh7)

[3. Event Log Generator 3](#_heading=h.2et92p0)

[4. Alpha Algorithm 3](#_heading=h.tyjcwt)

[5. Visualization 4](#_heading=h.3dy6vkm)

[6. Integration 4](#_heading=h.1t3h5sf)

[2. Process Description and Model Input 4](#_heading=h.4d34og8)

[3. Model Outputs 5](#_heading=h.2s8eyo1)

[4. Fitness Analysis 5](#_heading=h.17dp8vu)

[5. Precision Analysis 6](#_heading=h.3rdcrjn)

[6. Accuracy and Generalization 6](#_heading=h.26in1rg)

[7. Challenges 7](#_heading=h.lnxbz9)

[8. Model Evaluation 7](#_heading=h.35nkun2)

[9. Recommendations for Improvement 7](#_heading=h.1ksv4uv)

[Conclusion 8](#_heading=h.44sinio)

# Overview:

### 1. Preprocessing Steps

This section would describe the steps taken before generating traces and applying the Alpha Algorithm. It should include:

* Parsing the process description into a structured format suitable for generating traces.
* Introducing mechanisms to simulate noise and uncommon paths while preserving model integrity.
* Ensuring the input parameters (noise frequency, uncommon paths, and trace counts) are validated for consistency

### 2. Design Choices

This section should explain the decisions made during the design and implementation phases:

* Selection of input parameters like noise frequency and uncommon path frequency to reflect real-world scenarios.
* Use of frozensets for representing transitions to maintain data immutability.
* Incorporation of probabilistic execution (e.g., XOR and AND splits) to handle diverse process pathways.
* Visualization tool (Graphviz) to enhance model interpretability and usability.

### 3. Event Log Generator

The event log generator uses user input for noise frequency, uncommon path frequency, process description, and custom noise/uncommon steps. It then parses the process and generates traces accordingly.

### 4. Alpha Algorithm

The Alpha Algorithm implementation follows standard steps for mining Petri nets:

1. Calculates direct succession and footprint matrix.
2. Extracts unique traces and frequencies.
3. Finds maximal pairs for places in the Petri net.

### 5. Visualization

The visualization with Graphviz is a great addition for intuitive representation. It ensures that places, transitions, and arcs are visually distinguishable.

### 6. Integration

We successfully integrated the event log generation with the Alpha Algorithm pipeline:

* Generating traces using noise and uncommon paths.
* Parsing logs and applying process mining.
* Building and visualizing the Petri net.

# 2. Process Description and Model Input

* **Process Description Provided**:

Start With(Receive Order), Then(Process Payment), Parallel(Verify Payment, Prepare Package),

And(Generate Shipping Label, Confirm Delivery), XOR(Dispatch by Courier, Enable In-store Pickup)

This process includes:

* + **Linear sequence**: Receive Order → Process Payment.
  + **Parallel execution**: Verify Payment and Prepare Package.
  + **AND-split**: Generate Shipping Label and Confirm Delivery.
  + **XOR-split**: Dispatch by Courier OR Enable In-store Pickup.
* **Input Parameters**:
  + Noise frequency: **20%** (0.2).
  + Frequency of uncommon paths: **10%** (0.1).
  + Noise step: **SysErr**.
  + Uncommon step: **ManualOverride**.
  + Number of traces to generate: **70**.
* **Challenges in the Description**:
  + Handling of **parallel tasks** (e.g., Verify Payment, Prepare Package).
  + Proper integration of **noise** (SysErr) and **uncommon paths** (ManualOverride) while maintaining model readability.
  + Accurately modeling **XOR-splits** and **AND-splits** with probabilistic execution.

# 3. Model Outputs

The generated Petri Net structure includes maximal sets and transitions, represented by **frozensets**:

* **Key observations**:
  + Atomic steps like {Verify Payment}, {Prepare Package}, and {SysErr} are present as standalone transitions.
  + Steps related to concurrency (e.g., {Verify Payment, Prepare Package}) and splits (e.g., {Generate Shipping Label, Confirm Delivery}) are represented appropriately.
  + Noise (SysErr) and uncommon steps (ManualOverride) are incorporated into the model.
* **Interpretation of Outputs**:
  + The Petri Net model successfully represents the process description, including:
    - Sequential dependencies: {Receive Order} → {Process Payment}.
    - Concurrent activities: {Verify Payment} and {Prepare Package}.
    - Probabilistic splits (e.g., XOR with Dispatch by Courier or Enable In-store Pickup).

# 4. Fitness Analysis

Fitness measures how well the model reproduces the observed traces.

* **Positive Observations**:
  + The sequential and concurrent components align with the process description.
  + Noise (SysErr) and uncommon paths (ManualOverride) are introduced, reflecting real-world deviations.
* **Challenges**:
  + Overfitting risk due to **noise**: Traces with SysErr might distort the model by introducing unexplainable transitions.
  + Handling of **parallelism**: While {Verify Payment} and {Prepare Package} appear as distinct frozensets, their exact alignment in traces must be verified.
* **Conclusion on Fitness**:
  + The model exhibits **good fitness** but might struggle with traces containing noisy or uncommon events, potentially leading to misalignment.

# 5. Precision Analysis

Precision assesses how well the model avoids overgeneralization by restricting behavior to observed traces.

* **Positive Observations**:
  + XOR and AND splits are explicitly defined, ensuring accurate modeling of exclusive and concurrent activities.
  + Uncommon paths (e.g., ManualOverride) are isolated, preventing excessive generalization.
* **Challenges**:
  + Overgeneralization risk due to added noise (SysErr) and probabilistic paths. For example:
    - A 20% noise frequency might allow transitions like {SysErr} to appear excessively, diluting precision.
  + Lack of strict constraints between steps could permit behaviors not observed in the actual traces.
* **Conclusion on Precision**:
  + The model maintains **moderate precision** but could be improved by minimizing overgeneralization caused by noise.

# 6. Accuracy and Generalization

Accuracy combines fitness and precision to evaluate the model's ability to balance observed behavior with generalization.

* **Strengths**:
  + The inclusion of concurrency (Verify Payment and Prepare Package) and probabilistic paths makes the model robust for real-world scenarios.
  + Explicit representation of splits ensures the model is adaptable to various execution sequences.
* **Weaknesses**:
  + Excessive noise and uncommon events might reduce the model's ability to generalize effectively.
  + XOR and AND transitions require fine-tuning to ensure correct probabilities are reflected in generated traces.
* **Overall Accuracy**:
  + The model demonstrates **high accuracy** in representing the described process but could benefit from reduced noise and stricter constraints on uncommon paths.

# 7. Challenges

1. **Concurrency Handling**:
   * Modeling parallel steps (e.g., Verify Payment and Prepare Package) without causing ambiguity in execution order.
2. **Noise and Uncommon Path Integration**:
   * Maintaining model interpretability while incorporating noisy (SysErr) and uncommon (ManualOverride) transitions.
3. **Split Representation**:
   * Accurately capturing AND and XOR splits with probabilistic execution.
4. **Scalability**:
   * As the process complexity increases, managing transitions and maintaining fitness becomes more challenging.

# 8. Model Evaluation

This section should evaluate the generated Petri Net model:

* Analysis of fitness, precision, accuracy, and generalization metrics to ensure the model aligns with observed traces.
* Discussion on overfitting due to noise and strategies to mitigate it.
* Evaluation of concurrency representation and split accuracy to reflect real-world processes.
* Recommendations based on evaluation outcomes to improve model fidelity.

# 9. Recommendations for Improvement

1. **Optimize Noise Frequency**:
   * Reduce noise frequency to prevent overfitting and improve precision.
2. **Refine Path Probabilities**:
   * Assign more realistic probabilities to XOR and AND splits to better reflect execution likelihoods.
3. **Concurrency Representation**:
   * Introduce explicit constraints or markers for parallel tasks to improve clarity.
4. **Validation Against Real Traces**:
   * Compare the model's output with actual event logs to identify misalignments and refine transitions.
5. **Visual Enhancements**:
   * Use annotations or labels to clearly represent noisy and uncommon steps in the Petri Net.

# Conclusion

The Petri Net model demonstrates strong fitness and moderate precision, resulting in high overall accuracy. Challenges such as noise, uncommon paths, and concurrency handling are well-addressed but could benefit from refinement. With adjustments to noise frequency and probabilistic splits, the model could achieve even greater precision and generalization.